Dissertation

Exact and non-exact procedures for solving the Respone Time Variability Problem (RTVP)

by

Alberto García Villoria

Thesis advisors:

Dr. Albert Corominas
Dr. Rafael Pastor

Submitted to the Institut d'Organització i Control de Sistemes Industrial in partial fulfillment of the requierements for the degree of

Doctor of Philosophy

at the

Universitat Politècnica de Catalunya

Barcelona, Spain. April 2010.

Abstract

When a resource must be shared between competing demands (of products, clients, jobs, etc.) that require regular attention, it is important to schedule the access right to the resource in some fair manner so that each product, client or job receives a share of the resource that is proportional to its demand relative to the total of the competing demands. These types of sequencing problems can be generalized under the following scheme. Given n symbols, each one with demand d_i (i = 1,...,n), a *fair* or *regular* sequence must be built in which each symbol appears d_i times. There is not a universal definition of fairness, as several reasonable metrics to measure it can be defined according to the specific considered problem.

In the Response Time Variability Problem (RTVP), the unfairness or the irregularity of a sequence is measured by the sum, for all symbols, of their variabilities in the positions at which the copies of each symbol are sequenced. Thus, the objective of the RTVP is to find the sequence that minimises the total variability. In other words, the RTVP objective is to minimise the variability in the instants at which products, clients or jobs receive the necessary resource.

This problem appears in a broad range of real-world areas. Applications include sequencing of mixed-model assembly lines under just-in-time (JIT), resource allocation in computer multi-threaded systems such as operating systems, network servers and media-based applications, periodic machine maintenance, waste collection, scheduling commercial videotapes for television and designing of salespeople's routes with multiple visits, among others. In some of these problems the regularity is not a property desirable by itself, but it helps to minimise costs. In fact, when the costs are proportional to the square of the distances, the problem of minimising costs and the RTVP are equivalent.

The RTVP is very hard to be solved (it has been demonstrated that it is NP-hard). The size of the RTVP instances that can be solved optimally with the best exact method existing in the literature has a practical limit of 40 units. On the other hand, the non-exact methods proposed in the literature to solve larger instances are simple heuristics that obtains solutions quickly, but the quality of the obtained solutions can be improved. Thus, the solution methods existing in the literature are not enough to solve the RTVP.

The main objective of this thesis is to improve the resolution of the RTVP. This objective is split in the two following sub-objectives: 1) to increase the size of the RTVP instances that can be solved optimally in a practical computing time; and 2) to obtain efficiently near-optimal solutions for larger instances. Moreover, the thesis has the following two secondary objectives: a) to research the use of metaheuristics under the scheme of hyper-heuristics, and b) to design a systematic, hands-off procedure to set the suitable values of the algorithm parameters.

To achieve the aforementioned objectives, several procedures have been developed. To solve the RTVP an exact procedure based on the branch and bound technique has been designed and the size of the instances that can be solved in a practical time has been increased to 55 units. For larger instances, heuristic, heuristic, metaheuristic and hyper-heuristic procedures have been designed, which can obtain optimal or near-optimal solutions quickly. Moreover, a systematic, hands-off fine-tuning method that takes

advantage of the two existing ones (Nelder & Mead algorithm and CALIBRA) has been proposed.

Contents

Acknowledgments

Scientific/Technical support

I would like to thank Albert Corominas and Rafael Pastor for their guidance, comments, advices and corrections during the development of this thesis. Their support and confidence on me have been a constant that I really appreciate. I would like also to thank Ernest Benedito, Anna Coves, Laia Ferrer, Amaia Lusa and Jordi Olivella, stuff of the Enginyeria d'Organització i Logística Industrial (EOLI) research group, part of the Universitat Politècnica de Catalunya (UPC), for his helping in several technical questions.

Part of the thesis work has been done during two research internships at the Centre of Logistics & Heuristic Optimisation (CLHO) and the Centre Génie Industriel et Informatique (G2I), respectively. I am very grateful to Said Salhi for his helping and animosity during the research and development of heuristic and hyper-heuristic algorithms for the problem of this thesis. I am also very grateful to the ideas contributed by Alexandre Dolgui, Xavier Delorme and Wieslaw Kubiak when dealing the thesis problem by means of the branch and bound technique; I think that the given ideas have been the cornerstone of the achieved success.

Financial support

The development of this work has also been carried out with the partial financial support of the Spanish government, by means of the projects:

- DPI2004-03472 with title *Design and balanced of assembly lines in realistic environments* (DELIMER), 2005 to 2007.
  Amount awarded: 74,060€
  Principal investigator: Rafael Pastor.

- DPI2007-61905 with title *Design and balanced of production and assembly lines in realistic environments* (DELIPYMER), 2008 to 2010.
  Amount awarded: 99,099€
  Principal investigator: Rafael Pastor.

An internship grant –Estades de recerca fora de Catalunya BE-DGR-2008 (3,300.00 €)– supported by the Agència de Gestió d'Ajuts Universitaris i de Recerca (AGAUR) of the Generalitat de Catalunya has been given for the internship at the CLHO, Kent Business School, University of Kent, from 01/25/2009 to 04/25/2009.

Two financial helps –Mobilitat externa del PDI 2008 (1,400.00 € each one)– supported by the Universitat Politècnica de Catalunya (UPC) have been given for the aforementioned internship at CLHO and for the intenrship at G2I, l'Ecole Nationale Supérieure des Mines de Saint-Etienne, from 05/01/2009 to 07/31/2009, respectively.

Acronyms and terminology

Next the main acronyms used over the thesis text are listed:

ACO:	Ant Colony Optimisation
B&B:	Branch and Bound
CE:	Cross-Entropy
EM:	Electromagnetism-like Mechanism
GA:	Genetic Algorithm
GRASP:	Greedy Randomized Adaptive Search Procedure
JIT:	Just-In-Time
MILP:	Mixed-Integer Linear Programming
MS:	Multi-Start
N&M:	The Nelder and Mead algorithm
PSC:	Psychoclonal
PSO:	Particle Swarm Optimisation
RTV:	Response Time Variability (metric)
RTVP:	Response Time Variability Problem
RVNS:	Reduced VNS
SA:	Simulated Annealing
TS:	Tabu Search
VNS:	Variable Neighbourhood Search

The following terminology is used when referring the response time variability problem:

n :	Number of symbols to be sequenced (i = 1,...,n)
d_i :	Number of copies to be sequenced of symbol i
$D = \sum_{i=1}^{n} d_i$:	Total number of copies to be sequenced
$\overline{t_i} = D/d_i$:	Average or ideal distance between two consecutive copies of symbol i
t_k^i :	Distance between the positions in which the copies k and k + 1 of symbol i are found (i = 1,...,n, k = 1,...,d_i - 1)
$t_{d_i}^i$:	Distance between the last copy of symbol i in the preceding cycle and the first copy of the same symbol in the current symbol.
RTV	$= \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \overline{t_i})^2$

I. Preface

This doctoral thesis deals with the hard sequencing problem known as response time variability problem (RTVP). The Institute of Industrial and Control Engineering (IOC), in which this thesis has been developed, met and was interested in the regularity problem of the production scheduling of mixed-model assembly lines. There is not a universal definition of regularity, as several reasonable metrics can be defined. In a visit of Prof. Wieslaw Kubiak of the Memorial University of Newfoundland, he proposed the *response time variability* metric. During the development of the thesis, it has been brought to our notice that the same problem may appear in a wide range of environments such as computer systems, maintenance problems, announcement broadcasting and salespeople's routing, among others. In all these cases there may be the necessity of obtaining *fair* or *regular* sequences. In the RTVP, the response time variability metric is used to measure the fairness or regularity of a sequence.

The main objective of this thesis is to improve the solution of the RTVP, which is a NP-hard problem, by means of exact and non-exact methods. Moreover, two other additional objectives are pursued: to propose a systematic, hand-off procedure to fine-tuning the algorithms and to contribute in the research of the use of meta-heuristics under the hyper-heuristic scheme.

The thesis is presented in the form of a compendium of published or accepted articles (together with other papers under the format of articles in review process, conference communications and technical reports), which is taken under the doctorate studies regulations of the Universitat Politècnica de Catalunya (UPC). This work is organised as follows. Chapter II introduces the RTVP and real-life contexts in which may appear. Chapter III reviews the state of the art of the problem. Chapter IV justifies the thesis and its objectives are proposed. The procedures to achieve the objectives are explained in Chapter V and the obtained results are discussed in Chapter VI. Finally, the conclusions are given in Chapter VII. The four articles that have published or accepted in journals included in the JCR index are annexed in Annex A1; other papers as four articles submitted to journals included in the JCR index that are in process of review, four articles published in other international journals, three communications to international conferences and three technical reports are annexed in Annex A2.

II. Introduction to the RTVP

II.1. Classification and formulation of the problem

The concept of *fair sequence* has emerged independently from scheduling problems in diverse environments. The common aim of these scheduling problems, as defined in Kubiak (2004), is to build a fair sequence using *n* symbols, where symbol *i* ($i = 1,...,n$) must be copied d_i times in the sequence. The fair sequence is the one which allocates a fair share of positions to each symbol *i* in any subsequence. This *fair* or *ideal* share of positions allocated to symbol *i* in a subsequence of length *k* is proportional to the relative importance (d_i) of symbol *i* with respect to the total copies of competing symbols (equal to $\sum_{i=1..n} d_i$). There is not a universal definition of fairness, as several reasonable metrics can be defined according to the specific problem considered. For a detailed introduction to fair sequences, it is recommended the book by Kubiak (2009).

The family of fair sequencing problems can be classified according to the following characteristics (León *et al*., 2003):

- *Cyclic vs Non-cyclic*. The problem is cyclic if the sequence is the same for all cycles and the distance, for each symbol *i*, between the first copy of *i* in a cycle and the last copy of *i* in the preceding cycle is considered.
- *Distance-constrained vs Not distance-constrained*. The problem is distance-constrained if the distance between two consecutive copies of the same symbol has an upper bound and/or a lower bound.
- *Optimality vs. Feasibility*. If the aim is to find a solution that optimises an objective function then we look for optimality. Instead, if the aim is to find a feasible solution, then we look for feasibility.

The RTVP is a fair sequencing problem which is cyclic, not distance-constrained and its aim is to optimise an objective function. Its formulation is the following. Let *n* be the number of symbols to be sequenced, where symbol *i* ($i = 1,...,n$) is to be copied d_i times in the sequence, and let *D* be the total number of copies ($\sum_{i=1..n} d_i$). Let *s* be a solution of an instance in the RTVP that consists of a circular sequence of copies ($s = s_1 s_2 \dots s_D$), where s_j is the copy sequenced in position *j* of sequence *s*. For each symbol *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the copies $k + 1$ and *k* of symbol *i* are found. We consider the distance between two consecutive positions to be equal to 1. Since the sequence is circular, position 1 comes immediately after the last position *D*; therefore, $t_{d_i}^i$ is the distance between the first copy of symbol *i* in a cycle and the last copy of the same symbol in the preceding cycle. For all symbol *i* in which $d_i = 1$, t_1^i is equal to $\bar{t}_i$. Let $\bar{t}_i$ be the average or ideal distance between two consecutive copies of symbol *i* ($\bar{t}_i = D / d_i$). The aim is to minimise the metric Response Time Variability (RTV), which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2 \qquad (1)$$

The RTV metric is a weighted variance with weights equal to d_i. That is, $RTV = \sum_{i=1..n} d_i \cdot Var_i$, where $Var_i = {}^{1}\!/_{d_i} \cdot \sum_{k=1..d_i} \left(t_k^i - \bar{t}_i\right)^2$. Thus, the distance between any two consecutive copies of the same symbol should be as regular as possible (ideally constant).

It is worth to note that since the average distance $\bar{t}_i$ is equal to t_1^i for all symbol i such that $d_i = 1$, these symbols do not intervene in the computation of the RTV metric. That is, for all these symbols $Var_i = 0$.

The RTVP has been proved to be NP-hard (Corominas *et al*., 2007).

Example

As an illustration, consider the following example. Let $n = 3$ with symbols *A*, *B* and *C*. Also consider $d_A = 2$, $d_B = 3$ and $d_C = 7$. Thus, $D = 12$, $\bar{t}_A = 6$, $\bar{t}_B = 4$ and $\bar{t}_C = 1.71$. Any sequence that contains symbol i exactly d_i times is a feasible solution. For instance, a feasible solution is shown in Figure 1a.

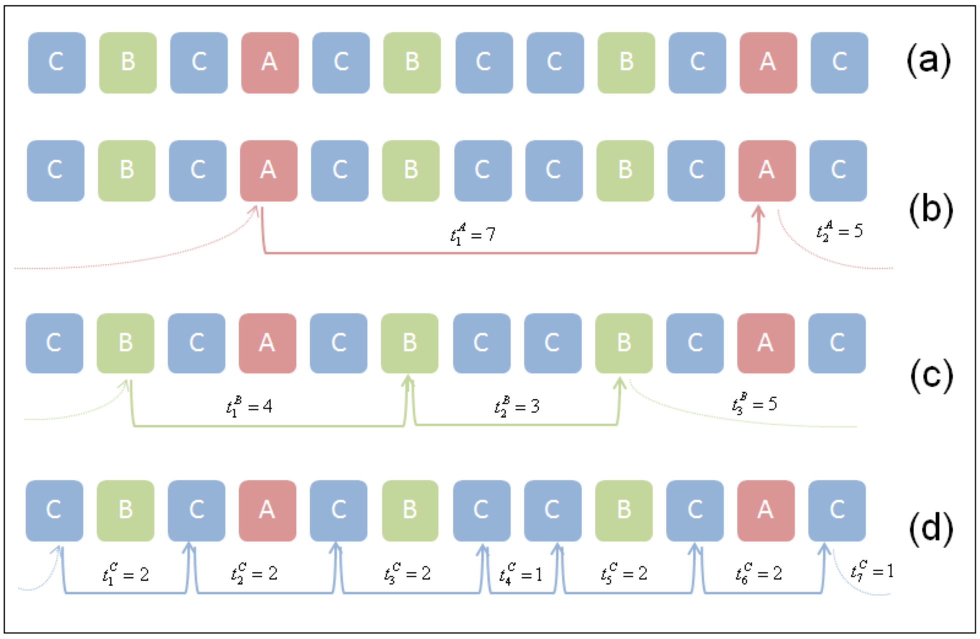


Figure 1. A feasible solution of an RTVP instance

The distances between the copies of symbol A are $t_1^A = 7$ and $t_2^A = 5$ (Figure 1b); the distances between the copies of symbol B are $t_1^B = 4$, $t_2^B = 3$ and $t_3^B = 5$ (Figure 1c); and the distances between the copies of symbol C are $t_1^C = 2$, $t_2^C = 2$, $t_3^C = 2$, $t_4^C = 1$, $t_5^C = 2$, $t_6^C = 2$ and $t_7^C = 1$ (Figure 1d). Therefore, the RTV value of this solution is the following:

$$\begin{aligned}RTV = &\left[(7-6)^2+(5-6)^2\right]+\\ &\left[(4-4)^2+(3-4)^2+(5-4)^2\right]+\\ &\left[(2-1.71)^2+(2-1.71)^2+(2-1.71)^2+(1-1.71)^2+(2-1.71)^2+(2-1.71)^2+(1-1.71)^2\right]\\ &=2+2+1.43=5.43\end{aligned}$$

II.2. Application in real-world contexts

When a resource must be shared between competing demands that require regular attention, it is important to schedule the access right to the resource in some fair manner so that each demand receives a share of the resource that is proportional to its demand relative to the competing demands (Herrmann, 2009). The objective in the RTVP is to minimise variability in the time between the instants at which that products, clients or jobs receive the necessary resources.

In the RTVP formulation introduced in the previous subsection, a symbol represents a product, client or job that demands the resource; a position of the solution sequence represents a times slot in which the symbol sequenced has access to the resource; and the number of times that each symbol i has to occur in the sequence (d_i) represents the number of time slots that each symbol has right. It is assumed that all time slots are the same amount of time. Thus, we can ignore time and consider only the positions in the sequence.

This problem appears in a broad range of real-world areas. Applications include sequencing of mixed-model assembly lines under just-in-time (JIT), resource allocation in computer multi-threaded systems such as operating systems, network servers and media-based applications, periodic machine maintenance, waste collection, scheduling commercial videotapes for television and designing of salespeople's routes with multiple visits, among others.

In some of these problems the regularity is not a property desirable by itself, but it helps to minimise costs. In fact, when the costs are proportional to the square of the distances, the problem of minimising costs and the RTVP are equivalent as follows:

$$RTV \quad =\sum_{i=1}^{n}\sum_{k=1}^{d_i}(t_k^i-\overline{t}_i)^2 \quad =\sum_{i=1}^{n}\sum_{k=1}^{d_i}\left(t_k^i\right)^2+\sum_{i=1}^{n}\sum_{k=1}^{d_i}\left(\overline{t}_i\right)^2-\sum_{i=1}^{n}\left(2\cdot\overline{t}_i\cdot\sum_{k=1}^{d_i}t_k^i\right) \quad =\sum_{i=1}^{n}\sum_{k=1}^{d_i}\left(t_k^i\right)^2+$$
$$\sum_{i=1}^{n}\sum_{k=1}^{d_i}\left(\overline{t}_i\right)^2-\sum_{i=1}^{n}2\cdot\overline{t}_i\cdot D$$

Since $\sum_{i=1}^{n}\sum_{k=1}^{d_i}\left(\overline{t}_i\right)^2$ and $\sum_{i=1}^{n}2\cdot\overline{t}_i\cdot D$ are constants, the problem of minimising *RTV* is equivalent to minimising $\sum_{i=1}^{n}\sum_{k=1}^{d_i}\left(t_k^i\right)^2$.

II.2.1. RTVP in the context of mixed-model, just-in-time, assembly lines

One of the first situations in which the idea of the fair, regular sequences appeared was the sequencing of mixed-model assembly lines at Toyota Motor Corporation under the just-in-time (JIT) production system.

Mixed-model assembly lines are production lines that are able to produce small lots (ideally of size one) of different models with negligible costs when changing over one model to another. The effective utilization of mixed-model lines requires the solution of the following two problems which depends on the company objectives (Korkmazel and Meral, 2001):

1. Line design and balancing together with cycle times and sequence of workstations.
2. Determination of the sequence of models to be produced.

This thesis deals with the second goal. Since Toyota Motor Corporation popularized the just-in-time (JIT) production systems, the problem of sequencing on mixed-model assembly lines has acquired high relevance. One of the most important JIT objectives is to get rid of all kinds of waste and inefficiency and, according to Toyota, the main waste is due to inventories. To reduce inventories, JIT production systems require producing only the necessary components in the necessary quantities at the necessary time. Because JIT is a pull production environment, the production schedule is focused on sequencing the models in the final assembly process.

There are two typical possible goals in JIT systems when the final sequence is being determined (Monden, 1983):

1. Smoothing the workload of the stations.
2. Keeping a regular rate of usage of every component used by the line.

The first goal is required when some models need more processing time than the cycle time in some stations. Although assembly lines have usually the flexibility to adjust to this situation without slowing down or stopping, if many units of models that require much time are successively sequenced then delays, line stoppages or incomplete work will occur. On the other hand, the second goal is vital to reduce component inventories. Inventory is needed to face the fluctuations of the demand of the components consumed by the line. The more regular the usage rate of the components, the smaller the inventory (ideally, no inventory would be needed with a constant usage rate). According to Monden (1983), the first goal is important, but the second goal is the cornerstone of JIT production systems. The problem of minimising variations in the usage rate of components is known as Output Rate Variation (ORV) problem (Kubiak, 1993).

An approximation for solving the ORV problem, which is NP-hard (Kubiak, 1993), is considering only the demand rates for the models (Miltenburg, 1989; Kubiak, 1993; Bautista *et al.*, 1995) and obtaining a sequence that minimises variations in the production rate of the models. This problem is known as the Product Rate Variation (PRV) problem (Kubiak, 1993). Note that when models require exclusive components, the PRV and the ORV problems are exactly equivalent. Anyway, the PRV problem can be considered as a problem independent of the ORV problem.

Miltenburg (1989) proposed a nonlinear formulation of the PRV problem. Consider n models with demand of units for each model i equal to d_i ($i = 1,\dots,n$). The total number of units to be produced is D ($D = \sum_{i=1..n} d_i$). The time required to produce each unit (regardless of the model) is constant; therefore, if we take the cycle time of the line as the unit of time, it can be considered that one model unit is produced per time unit. Thus, the production horizon is equal to D. The ideal production rate of model i for each time period k ($k = 1,\dots,D$) is r_i ($r_i = d_i / D$). Let x_{ik} be the total produced units of model i up to period k. Miltenburg suggested the following objective functions to be minimised:

$$\sum_{k=1}^{D}\sum_{i=1}^{n}\left({x_{ik}}/{k} - r_i \right)^2 \tag{2}$$

$$\sum_{k=1}^{D}\sum_{i=1}^{n}\left(x_{ik} - kr_i \right)^2 \tag{3}$$

$$\sum_{k=1}^{D}\sum_{i=1}^{n}\left| {x_{ik}}/{k} - r_i \right| \tag{4}$$

$$\sum_{k=1}^{D}\sum_{i=1}^{n}\left| x_{ik} - kr_i \right| \tag{5}$$

Originally, Miltenburg (1989) focused the regularity of the PRV problem on the variability of the production rate of the models. But the PRV problem can be generalized for that concerning the regularity of appearance of the models in the line (Bautista *et al*., 1997). For instance, Inman and Bulfin (1991) propose another objective function in which is minimised variations with respect ideal production due dates for each unit. It is worth to say that the distances between any two consecutive due dates defined by Inman and Bulfin for units of the same model are equal. The RTVP is very related with the PRV problem under the production context since the RTV measures the regularity in terms of the variability, for each model *i*, of the distances between the appearance of two consecutive units of model *i* (Equation 1).

Although the PRV problem has been usually discussed in the literature in terms of regular production rate (Miltenburg, 1989; Kubiak, 1993), feedback received from the manufacturing industry suggests that a good mixed-model sequence is one in which the distances between units of the same model are as regular as possible. Moreover, one drawback of the Miltenburg problem is that, on the contrary of the RTVP, it takes the positions of the models with only one unit to be produced into account although the positions of these models are irrelevant for the regularity of the consumption rates.

Example

The following example is to illustrate how the regularity in the production sequencing has effect on the necessary inventory. Consider the following example shown in Table 1 (inspired in one example given in Bautista *et al*., 1995). In the daily production, 10 units of model M1 and 10 units of model M2 have to be produced. Let's assume that 1 unit of model M1 consumes 1 unit of component C1, and 1 unit of M2 consumes 1 unit of component C2. Thus, the daily consumptions of C1 and C2 are 10 units, respectively, and their ideal consumption rates are 0.5 units per cycle.

Table 1. Example of a daily production

Model / Comp.	C1	C2	Program
M1	1	0	10
M2	0	1	10
Consumption	10	10	20
Rate	0.5	0.5	

Figure 2 shows the most irregular production sequence in terms of the RTV metric; that is, sequencing the 10 units of one model and then sequencing the other 10 units of the another model. We can see that the units C1 are consumed in the half of the horizon. Thus, the production of the components should be double than their average consumption rate during 10 cycles followed of 10 cycles of no components production. Another alternative (shown in Figure 2) is to produce the components at their average rate, but a coupling inventory is needed. The inventory will have, theoretically, between 0 and 5 units during the horizon. The same occurs to the units C2 but with a shift of 10 cycles.

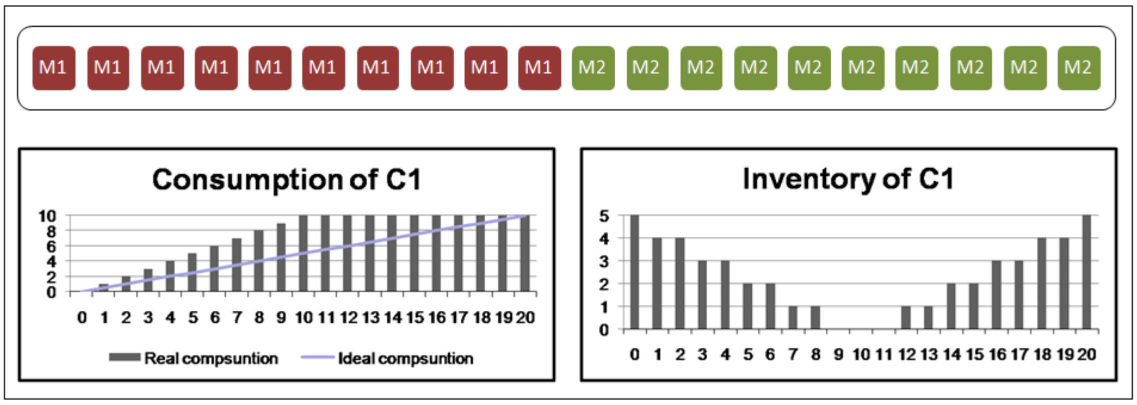


Figure 2. Manufacturing lots of 10 units: (10·M1 + 10·M2)

On other hand Figure 3 shows the most regular sequence in terms of the RTV metric; that is, a sequence in which the units of each model are allocated alternatively. The consumption of the component C1 is very similar to the production rate (analogously with the component C2). Thus, no inventory is needed theoretically.

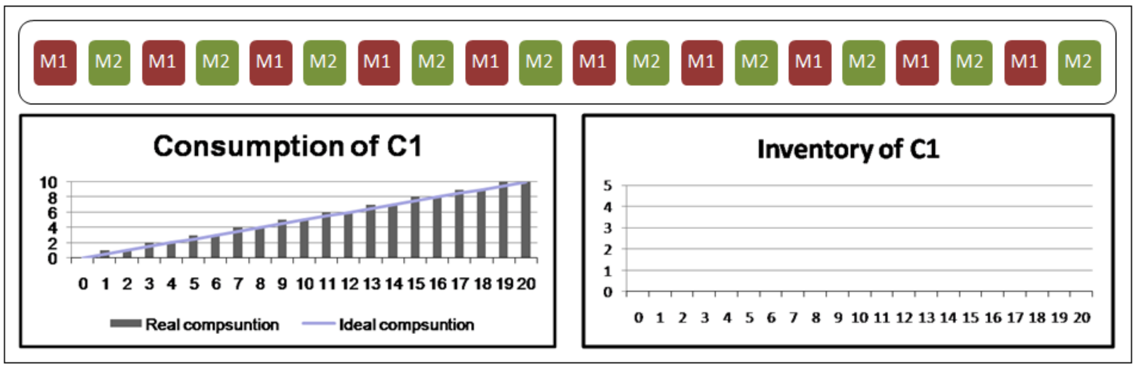


Figure 3. Manufacturing lots of 1 unit: 10·times (M1 + M2)

II.2.2. RTVP in the context of computer multithreaded systems

The need of fair sequencing also appeared in multithreaded computer systems (Waldspurger and Weihl, 1994 and 1995; Dong *et al.*, 1998; Bar-Noy *et al.*, 2002). Multithreaded systems (operating systems, network servers, media-based applications, etc.) are computer systems that do different tasks to attend to the requests of client programs that take place concurrently. These systems need to manage the scarce resource in order to service the requests of *n* clients. For example, multimedia systems should avoid presenting video frames too early or too late, which would result in jagged motion perceptions (Corominas *et al.*, 2007).

The first articles in which this problem is solved as a RTVP are Waldspurger and Weihl (1994, 1995). The resource is allocated in discrete time slots (authors refer to the duration of a standard time slice as a *quantum*). Resource rights are represented by *tickets* and each client *i* has a given number d_i of tickets. Thus, a client with twice as many tickets as another will receive twice as much of a resource in a given time interval. Waldspurger and Weihl define the response time as the elapsed time from a client's completion of one quantum up to including its completion of next. Since the quantum duration is fixed, this definition is equivalent to the number of quanta between a client's two consecutive quantum allocations plus one. The authors suggested the RTV metric to evaluate the fairness of a sequence (that is, the variability of the response times for each client).

II.2.3. Two case studies of the RTVP

Two case studies of RTVP applications were reported in the literature.

Hermann (2007, 2009) came up with the RTVP while working with a healthcare facility that needed to schedule the collection of waste from waste collection rooms throughout the building. Based on data about how often a waste collector had to visit each room and in view of the fact that different rooms require a different number of visits per shift, the facility manager wanted these visits to occur as regular as possible so that excessive waste would not collected in any room. For instance, if a room needed four visits per eight-hour shift, it would ideally be visited every two hours.

A study by Bollapragada *et al*. (2004) was motivated by a problem faced by the National Broadcasting Company (BNC), which is one of the main American firms in the television industry. Major advertisers buy hundreds of slots from the BNC to air commercials. The advertisers request that the airings of their commercials are as evenly spaced as possible over the broadcast season. The problem solved finally is not the RTVP, but a non-cycling variant. This study is continued in Brusco (2008).

II.2.4. RTVP in other contexts

Other contexts in which the RTVP appears are the periodic machine maintenance problem (Wei and Liu, 1983; Anily *et al.*, 1998) as well as other distance-constrained problems (e.g., see Han *et al.*, 1996). Although the main objective of the distance-constrained problem and the RTVP is to find a sequence as regular as possible, the advantage of the RTVP is that it will always come up with a feasible solution, contrary to the distance-constrained problem.

The RTVP can also be applied in the design of sales catalogues (problem introduced in Bollapragada *et al.*, 2004), in the scheduling of display advertisements on dynamic billboards at sport stadia and in the design of salespeople's routes with multiple visits, in which the visits to the same client should be as spaced as possible among the temporal horizon.

III. State of the art

Although the RTVP is in general NP-hard, the two-symbol case can be optimally solved with a polynomial algorithm proposed in Corominas *et al.* (2007). For a general case, Corominas *et al.* (2007) proposed a mixed-integer linear programming (MILP) model whose practical limit to obtain optimal solutions is 25 copies to be sequenced. Corominas *et al.* (2010) proposed an improved MILP model and increased the practical limit for obtaining optimal solutions from 25 to 40 copies to be sequenced.

For solving largest instances, heuristic methods have been proposed. This problem has been first time solved in Waldspurger and Weihl (1994) using a method that authors called *lottery scheduling*. This method is based on generating a solution at random as follows. For each position of the sequence, the symbol to be sequenced is chosen at random and the probability of each symbol is equal to the number of copies of this symbol that remain to be sequenced divided by the total number of copies that remain to be sequenced. The same authors proposed a greedy heuristic method that they called *stride scheduling* (Waldspurger and Weihl, 1995) that obtains better results than the lottery scheduling method. However, the stride scheduling method is, in fact, Jefferson’s method originally designed to solve the apportionment problem (Balinski and Young, 1982; Kubiak, 2004). The relation between fair sequences and the apportionment problem was first time introduced in Bautista *et al.* (1996).

In Corominas *et al.* (2007) five heuristics were proposed to solve the RTVP: the bottleneck algorithm used in Moreno (2002) to solve the *minmax* PRV problem, random generation, two classical parametric methods for solving the apportionment problem known as Webster’s method and Jefferson’s method (Balinski and Shahidi, 1998) and a new heuristic called Insertion method by the authors; moreover, a local search procedure is applied to the solutions obtained with the five heuristics. Parametric methods are defined as follows. Let x_{ik} be the number of copies of symbol i that have been already sequenced in the sequence of length k (assume x_{i0} = 0); the symbol to be sequenced in position k + 1 is $i^* = \arg\max_i \left\{ d_i / \left(x_{ik} + \delta \right) \right\}$, where $\delta \in \left(0,1\right]$. Webster’s and Jefferson’s methods are parametric methods that use a δ value equal to 0.5 and 1, respectively. Insertion method is a recursive heuristic based on grouping symbols into fictitious symbols until only two fictitious symbols remains and then solving optimally the two-symbol case.

Other seven heuristics for the RTVP were proposed in Corominas *et al.* (2009) together with twelve local search procedures. Six of these seven heuristics are three variants of Webster's method, one variant of Jefferson's method and two variants of the Insertion method. Another proposed heuristic is a greedy heuristic based, at each position k (k = $0 ... D-2$) of the sequence, on comparing for each symbol the cost of allocating a copy of it to position k + 1 and the cost of allocating it to position k + 2. The twelve local search procedures result from combining three neighbourhoods, two rules for replacing the current solution with a new one and two stopping rules. The three neighbourhoods are: 1) swapping two consecutive copies, 2) swapping any pair of copies, and 3) a copy of a symbol i is removed from the position it occupies and inserted between a pair of consecutive positions provided that there is no another copy of i between the initial position of the unit and the position in which is inserted. The two rules for replacing the current solution are: 1) being replaced with the first neighbour that is better that current solution, and 2) being replaced with the best neighbour, provided it is better than the current solution. Finally, the two stopping rules are: 1) that there is not a neighbour better than the current solution, or a neighbour for which the net improvement is 0 and, to avoid cycling, such that the maximum distance is not increased for either of two symbols being exchanged and at least one of the maximum distances actually decreases, and 2) similar to the preceding stopping rule but differs only in that considers also as candidates the neighbours for which the net improvement is 0 and such that the minimum distance does not decrease for either of the two symbols being exchanged and, moreover, at least one of the minimum distances actually increases.

In Herrmann (2007) was proposed an aggregation method based on grouping iteratively the symbols with the same number of copies to be sequenced into fictitious symbols and then applying a parametric method. The aggregation idea is extended in Herrmann (2009).

To solve the non-cycling variant of the RTVP in which the television advertising slots are scheduled, Bollapragada *et al*. (2004) developed two MILP models, a branch and bound (B&B) algorithm and four heuristics. Later, Brusco (2008) propose an enhanced B&B algorithm and a simulated annealing (SA) algorithm.

A lower bound of the RTVP has been proposed in Corominas *et al*. (2007). However, this lower bound is, in general, not enough accurate and is not a good quality indicator of the solution obtained by a heuristic method.

IV. Justification and objectives

The size of the RTVP instances which can be solved optimally with the best exact method has a practical limit of 40 copies. On the other hand, the non-exact methods proposed in the literature to solve larger instances are simple heuristics that obtains solutions quickly, but the quality of the obtained solutions can be improved. Thus, the solution methods existing in the literature are not enough to solve the RTVP.

On the other hand, the RTVP and variants have emerged recently and independently from different environments. This situation complicates to researchers and, especially, practitioners of a certain area to know the literature existing about the RTVP. Thus, it is observed in the state of the art that the researches and practitioners start from the scratch when he/she needs to solve the RTVP or variants in his/her work context. This thesis and the articles and communications that have been derived may help to announce the problem and, therefore, to unify the efforts to solve it.

The main objective of this thesis is to improve the resolution of the RTVP. This objective is split in the two following sub-objectives: 1) to increase the size of the RTVP instances that can be solved optimally in a practical computing time; and 2) to obtain efficiently near-optimal solutions for larger instances. To achieve this objective, several methods to solve the RTVP have been developed.

Moreover, the thesis has other two secondary objectives.

Hyper-heuristics are an emerging methodology in search and optimisation which can be defined as “heuristics to choose heuristics” (Burke *et al.*, 2003; Ross, 2005). Hyper-heuristics apply the right heuristic during the problem solving process, according to the current state of the solution. Thus, an intelligent application of different heuristics at different times in the search could lead to better performance than the application of individual heuristics. This innovative methodology has been applied to solve several optimisation problems: among others, timetabling, space allocation, flow-shop, job-shop, bin-packing and vehicle routing problems. All hyper-heuristics proposed in the literature work using simple heuristics, but very little work has still been done using metaheuristics (Burke and Kendall, 2005). We propose in this thesis to develop several hyper-heuristic algorithms to solve the RTVP that will use more complex, metaheuristic methods. A first secondary objective is to contribute in the new research field thanks to the development of the proposed hyper-heuristic algorithms.

The proposed methods have a set of parameters that need to be fine-tuned before the execution. The values of the parameters have usually a strong influence in the performance of algorithms. In non-exact methods, the performance is usually referred to the quality of the solution obtained; instead, in exact methods, the performance is usually referred to the computing time spent. Although the parameter values are extremely important because of the performance of the algorithm is very sensitive to them, the selection of parameter values is usually not enough justified (Eiben *et al.*, 1999; Adenso-Díaz and Laguna, 2006). Adenso-Díaz and Laguna (2006) reported that “about 10% of the total time dedicated to designing and testing of a new heuristic or metaheuristic is spent on development, and the remaining 90% is consumed fine-tuning parameters”. This statement can be also extended to the development of exact and hyper-heuristic methods. The another secondary objective of this thesis is to propose a systematic procedure for fine-tuning algorithms able to find good parameter values which needs of little human intervention and to be applied in the designed methods.

V. Solution procedures

V.1. Introduction

Different techniques and approaches have been applied in this thesis to solve the RTVP. An exact method based on the B&B technique has been designed in order to increase the size of the instances that can be optimally solved in a practical time with respect to the best MILP model proposed in the literature.

However, efficient non-exact methods are still needed for solving large, real-life instances. We have proposed heuristic, metaheuristic and hyper-heuristic methods.

An adaptive heuristic that incorporates a look-ahead strategy has been developed to obtain quite good solutions and very quick.

More complex procedures as metaheuristics have been considered to obtain better solutions. Some of the classical metaheuristics that have shown its efficiency to solve combinatorial problems are multi-start (MS), greedy randomized adaptive search procedure (GRASP), genetic algorithm (GA), simulated annealing (SA), tabu search (TS), ant colony optimization (ACO), particle swarm optimisation (PSO) and variable neighbourhood search (VNS). These metaheuristics are usually compiled in specialized handbooks (Glover and Kochenberger, 2003; Burke and Kendall, 2005). Moreover, new and promising metaheuristics have been proposed during the last decade: among others, cross-entropy method (CE), electromagnetism-like mechanism (EM) and psychoclonal (PSC). Algorithms based on all the mentioned metaheuristics have been developed.

Moreover, several hyper-heuristic approaches have been developed and some novel research has been done about the inclusion of metaheuristics in the hyper-heuristic scheme. The proposed hyper-heuristic methods have been tested solving the RTVP.

The fine-tuning of the parameters of the proposed B&B and metaheuristic methods have done in a systematic way using CALIBRA (Adenso-Díaz and Laguna, 2006). CALIBRA is a methods specially designed for fine-tuning the parameters of algorithms. On the other hand, another possible choice for fine-tuning the algorithm parameters is the Nelder and Mead algorithm (N&M) (Nelder and Mead, 1965). We have proposed a fine-tuning procedure that takes advantage of the characteristics of CALIBRA and N&M.

V.2. An exact algorithm

The best exact method proposed in the literature to solve the RTVP is a MILP model (Corominas *et al.*, 2010). MILP models use general software to solve problems; in Corominas *et al*. (2010), the ILOG CPLEX 9.0 optimiser is used. The disadvantage of the MILP approach is that general software is used to solve the MILP model and it cannot take advantage of all characteristics of the problem.

In order to solve optimally larger instances in a practical time, we propose to develop an exact algorithm based on the B&B technique. We have analysed the characteristics of the problem to propose a specially designed B&B algorithm. In particular, we have tried to avoid exploring dominated and equivalent solutions as much as possible.

The proposed B&B algorithm and the results of a computational experiment are reported in ***A branch and bound approach for the response time variability problem*** (García-Villoria *et al.*, 2009a)[*].

V.3. Non-exact methods

V.3.1. An heuristic algorithm

Several quick heuristics have been proposed in the literature to solve the RTVP. In order to improve the obtained solutions using very little computing time, a new heuristic has been designed.

The proposed heuristic allocates, at each iteration, a symbol into the first free position of the sequence (let the position be called *p*). The main idea behind the heuristic is to choose the symbol whose distance between position *p* and its last sequenced copy is the most similar to its ideal distance. The ideal distance of each symbol is not static but it changes dynamically according to the current state of the partial constructed sequence. Moreover, a look ahead strategy is used in the decisions made by the algorithm.

The heuristic is used to solve the RTVP and also to solve a variant of the problem that we called *minmax* RTVP. In this variant, the objective is to minimise the maximum absolute discrepancy in the distances between any two consecutive copies of the same symbol. The design of the heuristic and the results of the computational experiment are reported in ***An adaptive-based heuristic for the Response Time Variability Problem*** (Salhi and García-Villoria, 2009).

V.3.2. Metaheuristic algorithms

Metaheuristics are one of the most practical, non-exact approaches to solve hard optimisation problems. As it is pointed in the Metaheuristic Network (http://www.metaheuristics.net), "although metaheuristics are widely used techniques, the how and why they work effectively for specific problems and for others not, is still not well understood". Choosing the most suitable metaheuristic, or metaheuristic components, to use when a new problem is attacked is a very interesting question that remains still open. Given the lack of guidelines, the performance assessment of a metaheuristic for solving a problem is best carried out by experimentation (Chiarandini *et al.*, 2007).

We have designed one CE algorithm (***Solving the Response Time Variability Problem by means of the Cross-Entropy Method*** (García-Villoria *et al.*, 2010)), three MS, two

[*] The title of the works derived from the thesis are referenced using bold, italic letters during the remaining document.

GRASP and fourteen PSO algorithms (***Solving the Response Time Variability Problem by means of metaheuristics*** (García *et al*., 2006), ***Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics*** (Corominas *et al*., 2008), ***A Parametric Multi-start Algorithm for Solving the Response Time Variability Problem*** (Corominas *et al*., 2010), ***Introducing dynamic diversity in a discrete Particle Swarm Optimization*** (García-Villoria and Pastor, 2009a)), one ACO algorithm (***Using an Ant Colony System to solve the Response Time Variability Problem*** (Corominas *et al*., 2009a)), one EM algorithm (***Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism*** (García-Villoria and Pastor, 2009b)), one PSC algorithm (***Solving the Response Time Variability Problem by means of a psychoclonal approach*** (García-Villoria and Pastor, 2008)), one GA (***Solving the Response Time Variability Problem by means of a genetic algorithm*** (García-Villoria and Pastor, 2010)), two VNS algorithms (***Solving the Response Time Variable Problem by means of a Variable Neighbourhood Search Algorithm*** (Corominas *et al*., 2009b)), two TS algorithms (***Using Tabu Search for the Response Time Variability Problem*** (Corominas *et al*., 2009c), ***Resolución del response time variability problem mediante tabu search*** (Corominas *et al*., 2009d)), three hybrid algorithms (***Metaheuristic algorithms hybridized with variable neighbourhood search for solving the response time variability problem*** (Corominas *et al*., 2009e)) and three SA-based algorithms (***An enhanced metaheuristic for solving the response time variability problem*** (Corominas *et al*., 2009f)).

V.3.3. Hyper-heuristic algorithms

Hyper-heuristics are an emerging methodology in search and optimisation. A short definition of hyper-heuristic methods is “heuristics to choose heuristics”. Hyper-heuristics apply the right heuristic during the problem solving process, according to the current state of the solution. Hyper-heuristics operate indirectly on the solutions by choosing the (meta)heuristic to be applied. They thus operate at a higher level than classical heuristics and metaheuristics. In fact, hyper-heuristics have only access to a set of low-level (meta)heuristics that are applied to the current solution.

Hyper-heuristics can be divided into two categories: *constructive* hyper-heuristics and *improvement* hyper-heuristics. Constructive hyper-heuristics use a set of constructive heuristics as low-level heuristics, in order to construct a full solution. In contrast, improvement hyper-heuristics start from a complete initial solution and then improve on it, using either simple refinement heuristics or even more sophisticated, but time-consuming, metaheuristics.

We have proposed four constructive hyper-heuristics that use simple greedy heuristics as low-level heuristics and three improvement hyper-heuristics that use local search procedures as local search procedures. Both approaches are usual in the hyper-heuristic literature. Moreover, we have proposed other three constructive hyper-heuristics that use metaheuristics as low-level heuristics. To the best of our knowledge, there are not such studies reported in the literature. We believe this is mainly because of the excessive computational time that it may require. We have therefore put forward a mechanism on how to deal with this issue. We have proposed appropriate schemes on how to select the low-level metaheuristics based on the regular use of learning and launching stages at each cycle of the search. The goal is to control the computational burden while guiding the search toward good solutions

All hyper-heuristic algorithms have been tested solving the RTVP. The research, proposals and results are reported in ***Hyper-heuristic Approaches for the Response Time Variability Problem*** (García-Villoria *et al*., 2009b)

V.4. Fine-tuning

We have observed in the literature that the selection of the parameter values is usually not enough justified (even sometimes the parameter values used in the computational experiment are missing), as Adenso-Díaz and Laguna (2006) and Eiben *et al*. (1999) have also pointed.

We are aware of the difficulty of fine-tuning an algorithm due to the common non-linear interdependence between the parameters and, in the case of stochastic algorithms as most of metaheuristics, each execution may provide a different solution. Anyway, nowadays there are hands-off tools that can provide right parameter values in a reasonable computing time as, for example, CALIBRA and N&M.

CALIBRA has been specifically designed for fine-tuning algorithms and it is based on using conjointly Taguchi's fractional factorial experimental designs and a local search procedure (Adenso-Díaz and Laguna, 2006). On the other hand, N&M (Nelder and Mead, 1965) is a general direct optimisation algorithm (i.e., it only uses the values of the function). The disadvantage of using N&M to fine-tune the parameters of an algorithm is that good parameters values are needed at the beginning, which are not usually available.

Thus, CALIBRA have been used to fine-tune the parameters of the proposed B&B and metaheuristic methods. The systematic use of a hands-off procedure as CALIBRA to find the parameter values not only allows to find right parameter values but also to do a fair comparison between the results obtained with the metaheuristic methods.

Moreover, we have proposed a new hands-off, systematic fine-tuning procedure for fine-tuning metaheuristics that takes the advantages of CALIBRA and N&M. In the case that more fine-tuning time is available, the CALIBRA authors suggest applying again CALIBRA in a narrow range around the obtained parameter values. However, we suggest applying N&M instead. The explanation of this new procedure and the computational experiment to validate the proposal is reported in the article ***A systematic procedure based on CALIBRA and the Nelder & Mead algorithm for fine-tuning metaheuristics*** (Corominas *et al*., 2009g).

VI. Discussion of the results

VI.1. Exact solution of the RTVP

To fine-tune and test the B&B algorithm, 30 training and 320 test instances are used (available at https://www.ioc.upc.edu/EOLI/research/). The first 120 test instances are the same instances used to test the best MILP model (Corominas *et al.*, 2010). The instances were generated as follows. D was randomly selected with a discrete uniform distribution between 20 and 30, between 30 and 35, between 35 and 40, between 40 and 45, between 45 and 50, between 50 and 55, between 55 and 60 and between 60 and 65 for instances 1 to 40, 41 to 80, 81 to 120, 121 to 160, 161 to 200, 201 to 240, 241 to 280 and 281 to 320, respectively. For instances 1 to 40, n and d_i were randomly selected with a discrete uniform distribution between 3 and $\lceil D/2 \rceil$ and between 1 and $\lceil (D-n+1)/2 \rceil$ (with $\sum_{i=1}^{n} d_i = D$), respectively. For instances 41 to 320, n and d_i were randomly selected with a discrete uniform distribution between 3 and 12 and between 1 and $\lceil (D-n+1)/2.5 \rceil$ (with $\sum_{i=1}^{n} d_i = D$), respectively.

The B&B algorithm was coded and run under Java 2 Platform Standard Edition (J2SE) 1.4.2.14 and the computational experiment was carried out on a PC 3.00 GHz Intel Pentium IV with 1.5 GB of RAM.

Table 2 summarises the results obtained with a maximum calculation time of 10,000 seconds for each instance. The columns *#Opt* and *#Fea* show the number of instances that have been optimally solved and the number of instances in which a solution has been found but its optimality has not been demonstrated, respectively. The average computing time (in seconds) is shown in parentheses.

Table 2. Comparison between the best MILP and the proposed B&B algorithm

	#Opt	***#Fea***
MILP	114 (278 s.)	6 (10,000 s.)
B&B	114 (7.47 s.) + 6 (316.21 s.)	0

The best MILP model optimally solved 114 of the first 120 test instances whereas the proposed B&B algorithm solved all 120 instances. Moreover, the computing time has been considerably reduced; the MILP model needs an average time of 278 seconds to solve the 114 instances versus the average time of 7.47 seconds needed by the B&B algorithm to solve the same 114 instances.

Table 3 shows the results obtained for instances 1 to 320 with a maximum calculation time of 10,000 seconds for each instance. Column D shows the range size of the instances, column T shows the average time (in seconds) to solve an instance, column T_{S0} shows the time (in seconds) to obtain the initial solution, column *RTV* shows the average of the best RTV values found and column *#Opt* shows the number of instances that have been solved optimally.

Table 3. Results obtained with the B&B method

Instances	*D*	*T*	T_{S0}	*RTV*	*#Opt*
1-40	20-30	2.15	2.08	6.23	40
41-80	30-35	5.89	2.83	9.24	40
81-120	35-40	60.69	3.05	13.47	40
121-160	40-45	785.98	2.08	14.43	38
161-200	45-50	1,589.13	2.83	16.49	37
201-240	50-55	2,973.90	3.06	18.51	34
241-280	55-60	5,090.25	3.60	20.48	23
281-320	60-65	5,910.49	4.00	24.87	18

Between 40 and 45 copies, 45 and 50 copies and 50 and 55 copies the B&B algorithm solves the 95%, 92.5% and 85% of instances, respectively. For larger instances, the number of solved instances decreases quickly. However, the algorithm is still able to solve around 50% of instances that have between 55 and 65 copies to be sequenced.

Thus, we can say that the B&B algorithm is able to solve optimally in a practical time instances up to 55 copies to be sequenced (that is, the size of the instances that can be optimally solved has been increased 37.5% with respect to the best exact method published in the literature). Not only larger instances can be optimally solved but also it is useful to find new optimal solutions of the RTVP that can be used to compare the results obtained with heuristic and metaheuristic methods.

VI.2. Non-exact solution of the RTVP

To test the algorithms, a set of benchmark instances is needed. Because there was not any benchmark set published in the literature, a set of 740 testing instances generated at random has been used together with a set of 60 training instances to fine-tune the parameters of the algorithms (all instances can be found at https://www.ioc.upc.edu/EOLI/research/). These instances were grouped into four classes (from *CAT1* to *CAT4* with 15 training instances and 185 test instances in each class) according to their size. The instances were generated using the random values of *D* (number of copies) and *n* (number of symbols) shown in Table 4. For all instances and for each symbol $i = 1,\ldots,n$, a random value of d_i (number of copies of symbol i) is between 1 and $\lceil (D-n+1)/2.5 \rceil$ such that $\sum_{i=1..n} d_i = D$.

Table 4. Uniform distributions for generating the *D* and *n* values

	CAT1	*CAT2*	*CAT3*	*CAT4*
D	U(25, 50)	U(50, 100)	U(100, 200)	U(200, 500)
n	U(3, 15)	U(3, 30)	U(3, 65)	U(3, 150)

All algorithms were coded and run under Java 2 Platform Standard Edition (J2SE) 1.4.2.14 and all computational experiments were carried out on a 3.4 GHz Pentium IV with 1.5 GB of RAM.

In most of the metaheuristics, one or more of the following three neighbourhoods are used: 1) interchanging each pair of two consecutive copies of the sequence that represents the current solution (*N1*), 2) interchanging each pair of consecutive or no-consecutive copies of the sequence (*N2*), and 3) inserting each copy in each position of the sequence (*N3*).

VI.2.1. The heuristic algorithm

The proposed heuristic (*ENH-H*) is compared with the five best existing heuristics (Corominas *et al*., 2009). Those are known as *Oc*, *AWe/dg*, *We/dg*, *Je/dg* and *In*.

In the computational experiments all 800 testing and training instances are solved (since the heuristics have not parameters, training instances are not needed for the fine-tune). The results are analysed by considering all instances as well as each class of instances (*CAT1* to *CAT4*). The average RTV values of the solutions obtained with all heuristics are given in Table 5.

Table 5. Average RTV values obtained by the classical heuristics

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
ENH-H	**144.30**	**26.96**	**60.85**	**135.45**	**353.92**
Oc	215.61	28.96	74.20	198.61	560.68
Awe/dg	405.88	47.03	120.32	349.13	1,107.03
We/dg	434.56	50.93	129.62	376.27	1,181.43
Je/dg	594.51	57.52	164.19	499.72	1,656.61
In	778.51	121.16	308.45	658.21	2,026.21

We can see in Table 5 that *Oc* was the best existing heuristic in the literature. This observation is valid for the overall RTV averages as well as in each class of instances (*CAT1* to *CAT4*). On the other hand, our heuristic (*ENH-H*) obtains, on average, better solutions than *Oc*. If we consider the results by class, *ENH-H* is 6.91%, 17.99%, 31.80% and 36.88% better than *Oc* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. Thus, the results point that the larger the instance, the more competitive our heuristic. Moreover, the design *ENH-H* is simpler and is much faster than *Oc*. On average, *ENH-H* requires only 1.82 milliseconds to solve an instance, whereas *Oc* needs 1,479.99 milliseconds (i.e., nearly 810 times slower).

VI.2.2. The metaheuristic algorithms

In this thesis 34 metaheuristic-based algorithms has been designed and used to solve the RTVP. All these metaheuristic algorithms, except one MS algorithm, have been tested on the same set of 740 test instances introduced in Chapter VI.2.

The RTV averages of the solutions obtained for 50 and 1,000 seconds of computing time are shown in Table 6 and Table 7, respectively.

Table 6. RTV averages obtained for 50 computing seconds

		GLOBAL	***CAT1***	***CAT2***	***CAT3***	***CAT4***
CE(a)		52,920.08	21.16	106.15	2,809.81	208,743.18
Multi-start	*MS-1*(b)	21,390.39	12.08	44.36	226.90	85,278.25
	MS-2(c)	2,106.01	11.56	38.02	154.82	8,219.65
GRASP	Webster(b)	14,168.83	15.47	88.48	510.44	56,060.92
	Greedy(c)	2,308.69	13.00	60.45	270.93	8,890.37
PSO	*PSO-M1F*(b)	8,502.83	66.45	424.59	3,000.52	30,519.76
	PSO-M1T(b)	13,457.60	66.83	509.89	4,335.87	48,917.80
	PSO-M2F(b)	10,778.40	83.14	604.27	4,488.44	37,937.76
	PSO-M2T(b)	8,629.03	80.93	517.05	3,888.79	30,029.34
	DPSOpoi-c$_p$dyn(d)	4,625.54	16.42	51.34	610.34	17,824.04
	PSO-c$_3$dyn(d)	6,986.05	15.72	57.10	1,261.81	26,609.56
	PSOCB(d)	8,316.51	73.79	433.98	3,106.96	29,651.33
	DPSOvel(d)	8,686.47	19.28	179.60	2,287.05	32,259.96
	CPSO(d)	8,774.06	74.51	478.13	3,478.72	31,064.89
	DPSOpoi(d)	8,792.70	17.14	50.50	810.58	34,292.58
	PSO-c$_3$dyn'(d)	11,133.09	146.77	804.12	5,251.08	38,330.39
	PSOPC(d)	14,579.82	82.03	563.05	4,021.67	53,652.54
	PSO-c$_3$stat(d)	18,707.12	40.41	853.26	7,959.23	65,975.58
	PSOPC'(d)	19,626.03	145.26	1,178.29	9,086.24	68,094.33
ACO(e)		1,651.48	10.92	36.83	504.84	6,053.31
EM(f)		3,747.05	19.14	54.54	260.79	14,653.72
PSC(g)		235.68	14.92	44.25	137.07	746.50
GA(h)		186.94	11.65	29.41	84.54	622.16
VNS	*$RVNS_{(1,2,3)}$*(i)	63.96	10.73	23.69	51.80	169.64
	$RVNS_{(2,3)}$(i)	86.78	10.63	23.23	53.39	259.86
TS	*TS_{N2}*(j)	202.42	10.30	22.40	109.38	667.59
	TS_{N3}(k)	210.47	10.26	22.56	73.26	735.78
VNS hybrids	*TS+VNS*(l)	71.57	10.38	24.00	53.99	197.90
	MS+VNS(l)	62.17	10.24	21.23	47.46	169.76
	PSO+VNS(l)	60.03	10.47	22.42	49.37	157.86
SA	*SA_{N1}*(m)	50.87	10.26	21.67	44.57	126.98
	$MS+SA_{N1}$(m)	51.84	10.24	21.19	43.57	132.35
	$MS+SA_{N1,2,3}$(m)	73.12	10.24	21.52	47.37	213.37

(a) García-Villoria *et al.*, 2010; (b) García *et al.*, 2006; (c) Corominas *et al.*, 2008; (d) García-Villoria and Pastor, 2009a; (e) Corominas *et al.*, 2009a; (f) García-Villoria and Pastor, 2009b; (g) García-Villoria and Pastor, 2008; (h) García-Villoria and Pastor, 2010; (i) Corominas *et al.*, 2009b; (j) Corominas *et al.*, 2009c; (k) Corominas *et al.*, 2009d; (l) Corominas *et al.*, 2009e; (m) Corominas *et al.*, 2009f

The results point that simple metaheuristics based on replacing the current solution by one of its neighbours selected at random from one or more neighbourhoods, SA and RVNS, can work better than more complex algorithms like CE, PSO, ACO, EM and GA, for instance. Moreover, the hybridization of SA and VNS with a simple exploration mechanism like MS helps them to improve their performance.

Table 7. RTV averages obtained for 1,000 computing seconds

		GLOBAL	***CAT1***	***CAT2***	***CAT3***	***CAT4***
CE		*	*	*	*	*
Multi-start	*MS-1*	1,378.58	10.93	35.48	160.67	5,307.25
	MS-2	169.25	10.51	31.21	123.27	512.02
GRASP	Webster	1,495.12	13.59	75.08	428.86	5,462.95
	Greedy	301.90	11.56	50.45	227.50	918.10
PSO	*PSO-M1F*	6,619.34	66.45	424.54	3,000.52	22,985.85
	PSO-M1T	*	*	*	*	*
	PSO-M2F	*	*	*	*	*
	PSO-M2T	*	*	*	*	*
	DPSOpoi-c_p*dyn*	1,537.34	14.35	46.55	143.95	5,944.51
	PSO-c_3*dyn*	1,980.20	14.63	46.13	142.58	7,717.47
	PSOCB	3,696.44	13.83	42.18	391.54	14,338.20
	DPSOvel	4,312.31	17.75	84.17	1,036.87	16,110.42
	CPSO	6,731.24	73.79	433.98	3,106.96	23,310.24
	DPSOpoi	7,746.85	74.51	478.13	3,478.72	26,956.02
	PSO-c_3*dyn'*	8,838.70	82.03	563.05	4,021.67	30,688.03
	PSOPC	11,133.09	146.77	804.12	5,251.08	38,330.39
	PSO-c_3*stat*	16,212.08	16.75	592.64	6,520.72	57,718.22
	PSOPC'	18,495.01	138.76	1,056.59	8,414.15	64,370.53
ACO		1,208.81	10.46	31.17	337.31	4,456.32
EM		330.29	18.64	52.97	157.20	1,092.36
PSC		161.60	14.90	36.90	122.38	469.23
GA		106.68	10.92	27.00	74.86	313.92
VNS	$RVNS_{(1,2,3)}$	62.24	10.73	23.29	51.40	163.15
	$RVNS_{(2,3)}$	62.06	10.63	23.19	51.46	162.95
TS	TS_{N2}	113.31	10.24	21.46	106.21	315.33
	TS_{N3}	78.62	10.24	21.16	48.12	234.96
VNS hybrids	*TS+VNS*	55.05	10.24	22.48	47.66	139.84
	MS+VNS	54.95	10.24	20.94	43.26	145.35
	PSO+VNS	55.86	10.45	22.00	46.80	144.22
SA	SA_{N1}	50.75	10.26	21.67	44.55	126.54
	$MS+SA_{N1}$	46.60	10.24	20.92	40.33	114.91
	$MS+SA_{N1,2,3}$	61.92	10.24	20.95	42.99	173.51

(*) The computational experiment has not been done.

For little computing time (50 seconds), the five best metaheuristic algorithms according to the overall RTV value of their obtained solutions are, in the following order, SA_{N1}, $MS+SA_{N1}$, *PSO+VNS*, *MS+VNS* and $RVNS_{(1,2,3)}$. On the other hand, if more computing time is available (1,000 seconds) the best metaheuristic algorithms are, in the following order, $MS+SA_{N1}$, SA_{N1}, *MS+VNS*, *TS+VNS* and *PSO+VNS*. After 1,000 computing seconds, $MS+SA_{N1}$ is able to obtain an RTV average 8.18%, 15.20%, 15.35% and 16.58% better than SA_{N1}, *MS+VNS*, *TS+VNS* and *PSO+VNS*, respectively. Moreover, $MS+SA_{N1}$ converges very fast and it is able to obtain a better RTV average with 50 computing seconds (51.84) than the averages obtained with the third, four and fifth best methods (*MS+VNS*, *TS+VNS* and *PSO+VNS*, respectively) with 1,000 computing seconds (54.95, 55.05 and 55.86, respectively). Figure 4 shows the evolution of the RTV averages over the computing time.

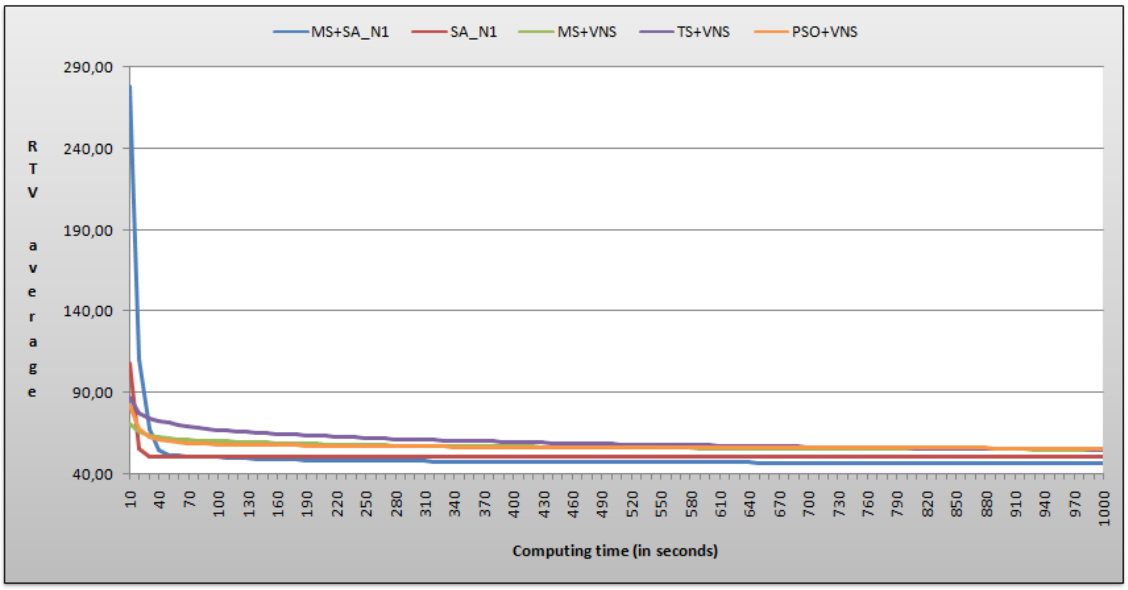


Figure 4. Average of the RTV values obtained over the computing time

Observing the RTV average by class, we can see that $MS+SA_{N1}$ is also the best for all classes (*CAT1* to *CAT4*) for 1,000 computing seconds. For *CAT1* instances, $MS+SA_{N1}$ obtains an optimal solution for all 185 instances (as it will be explained later in this chapter). For *CAT2* instances, the RTV average is 0.10% better than the average obtained with the second best algorithm for these instances (*MS+VNS*). For *CAT3* instances, the RTV average is 6.19% better than the average obtained with the second best algorithm for these instances ($MS+SA_{N1,2,3}$). Finally, for *CAT4* instances, the RTV average is 9.19% better than the average obtained with the second best algorithm for these instances (SA_{N1}).

To sum up, $MS+SA_{N1}$ is the best choice to solve the RTVP because is the algorithm that obtains the best solutions, on average, independently of the size of the instance to be solved. Moreover, this algorithm is able to obtain better solutions than other algorithms very quick.

Thus, the best solutions, on average, are obtained by $MS+SA_{N1}$ among all metaheuristic methods (and also all hyper-heuristic methods, as we can see in Chapter VI.2.3) designed to solve the RTVP. However, is the quality of these solutions good? To answer this question, we have tried find the optimal solutions by means of the proposed B&B algorithm but only the smallest instances (*CAT1* instances) were optimally solved. For the remaining instances, the lower bound (LB) proposed in Corominas *et al*. (2007) is used. Table 8 shows the averages of the LBs ($\overline{LB}$), the average of the optimal RTV values ($\overline{RTV^*}$) for the *CAT1* instances and the averages obtained with $MS+SA_{N1}$ ($\overline{RTV}$) with 1,000 computing seconds.

Table 8. Averages of the optimal RTV values and the RTV lower bounds

	CAT1	*CAT2*	*CAT3*	*CAT4*
$\overline{LB}$	5.35	10.95	21.15	48.15
$\overline{RTV^*}$	10.24	*	*	*
$\overline{RTV}$	10.24	20.92	40.33	114.91

For all 185 *CAT1* instances, $MS{+}SA_{N1}$ achieves the optimal solutions. We can see in Table 8 that the LB calculated as proposed in Corominas *et al*. (2007) is not accurate. For the smallest instances, the ratio between $\overline{RTV^*}$ and $\overline{LB}$ is 1.914. It could seem reasonable to assume that this ratio will remain equal or increase for larger instances. Thus, if we assume that the ratio remains equal, a more accurate estimation of the averages of the optimal values for *CAT2*, *CAT3* and *CAT4* instances are obtained by multiplying their $\overline{LB}$ by 1.914; that is, 20.96, 40.48 and 92.16 for *CAT2*, *CAT3* and *CAT4* instances, respectively. According to this assumption, we could say that the solutions obtained by the hybrid algorithms for *CAT2* and *CAT3* instances are very good.

VI.2.3. The hyper-heuristic algorithms

All hyper-heuristics have been tested using the set of 740 test instances introduced in Chapter VI.2.

The four proposed constructive hyper-heuristics (*CHH-1* to *CHH-4*) use as low-level heuristics six greedy heuristics (*Gr1* to *Gr6*). Two ways of fine-tuning the parameters of *CHH-2* and *CHH-3* have been proposed (considering the overall instances or considering the instances per class). Table 9 shows the average RTV values of the obtained solutions, where the method *BH* consists of running the six greedy heuristics and getting the best solution for each instance.

Table 9. Average RTV values for the constructive hyper-heuristics

		Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
Gr1		22,822.01	121.84	933.41	8,502.80	81,730.00
Gr2		23,736.83	147.19	1,077.88	9,106.04	84,616.22
Gr3		22,513.37	120.09	915.74	8,347.60	80,670.03
Gr4		22,478.08	125.06	914.70	8,295.41	80,577.15
Gr5		8,851.46	88.02	553.06	3,894.31	30,870.45
Gr6		46,086.95	405.39	2,583.30	17,450.83	163,908.26
BH		8,430.34	79.45	510.93	3,745.39	29,385.59
CHH-1		7,664.60	68.78	440.00	3,335.37	26,814.25
CHH-2	Overall prob.	7,782.61	78.89	477.38	3,377.66	27,196.49
	Per class prob.	7,556.80	83.19	522.98	3,297.68	26,323.33
CHH-3	Overall prob.	6,610.44	83.05	426.50	2,754.06	23,178.13
	Per class prob.	6,358.27	104.53	599.37	3,186.30	21,542.86
CHH-4		5,735.42	118.75	500.76	2,716.69	19,605.49

Table 9 shows that the best individual greedy heuristic is clearly *Gr5*, which is much better than the second best heuristic (*Gr4*). And *BH* is, obviously, better than *Gr5*. On the other hand, all hyper-heuristic methods outperform, on average, *BH*. The best of the hyper-heuristics is *CHH-4*, which use a random strategy for selecting the low-level heuristic. *CHH-4* obtains a RTV average 31.97% better than the average obtained by *BH*. Moreover, the computing times of the hyper-heuristics were very small: for *CHH-*

1, *CHH-2*, *CHH-3* and *CHH-4* it was 2.523, 0.590, 0.040 and 0.046 seconds, respectively.

Three improvement hyper-heuristic methods ($IHH\text{-}1\text{-}H_I^0$, $IHH\text{-}2\text{-}H_I^0$ and $IHH\text{-}3\text{-}H_I^0$, respectively) that use as low-level heuristics three local search procedures based on the neighbourhoods *N1*, *N2* and *N3*, respectively, are proposed. The hyper-heuristics are compared with a composite hill-climbing method (*CHC*) that applies iteratively the three local search procedures until a local optimum with respect the three neighbourhoods is obtained. The results obtained when the maximum computing time is set to 1,000 seconds are shown in Table 10.

Table 10. Average RTV values for the local search based hyper-heuristics

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
CHC	124.49	16.39	39.91	101.90	339.75
$IHH\text{-}1\text{-}H_I^0$	119.60	15.71	38.99	102.51	321.20
$IHH\text{-}2\text{-}H_I^0$	117.55	15.71	38.99	100.44	315.05
$IHH\text{-}3\text{-}H_I^0$	116.40	15.71	38.99	96.55	314.33

The hyper-heuristics are able to decide intelligently when to use each local search during the optimisation process, rather than systematically using them in a specific order. All three hyper-heuristics outperformed *CHC*. The hyper-heuristics $IHH\text{-}1\text{-}H_I^0$, $IHH\text{-}2\text{-}H_I^0$ and $IHH\text{-}3\text{-}H_I^0$ were, on average, 3.93%, 5.57% and 6.50% better overall than *CHC*, respectively.

Finally, three hyper-heuristics ($IHH\text{-}1\text{-}H_I^1$, $IHH\text{-}2\text{-}H_I^1$ and $IHH\text{-}3\text{-}H_I^1$, respectively) are proposed that use as low-level heuristics a TS algorithm, a VNS algorithm and *CHC*. Table 11 shows the results obtained with 1,000 computing seconds.

Table 11. Average RTV values for the metaheuristic based hyper-heuristics

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
CHC	124.49	16.39	39.91	101.90	339.75
TS	229.47	10.74	42.68	175.03	689.44
VNS	131.99	11.36	24.53	83.54	408.52
$IHH\text{-}1\text{-}H_I^1$	159.98	10.39	25.18	74.71	529.65
$IHH\text{-}2\text{-}H_I^1$	135.19	10.37	24.55	72.46	433.39
$IHH\text{-}3\text{-}H_I^1$	109.20	10.39	24.72	66.91	334.76

The best results shown in Table 11 are obtained, on average, by the hyper-heuristic $IHH\text{-}3\text{-}H_I^1$, which are better than the results found by any of the low-level heuristics when applied in isolation. The RTV average obtained by $IHH\text{-}3\text{-}H_I^1$ is 12.28%, 17.27% and 52.41% better than the averages obtained by CHC, the TS algorithm and the VNS algorithm, respectively.

The results obtained in the experimentation are encouraging by two reasons. First, better RTV values, on average, have been obtained within the hyper-heuristic scheme than applying the low-level (meta-)heuristics in an isolate way. And second, improvements are obtained although quite simple hyper-heuristic algorithms have been designed.

VI.3. Fine-tuning

During the development of this thesis, we have experienced the importance of setting the parameter values of almost all algorithms proposed to solve the RTVP. In fact, this stage is vital to decide if a proposed algorithm is good solving the RTVP since the algorithm may be very sensitive to the parameter values. For example, we have noticed when fine-tuning that an algorithm can perform even 10 times worse when wrong parameter values are used.

CALIBRA has been used to fine-tuning the parameters of the proposed B&B and metaheuristic methods. We think that not only is important using CALIBRA to obtain good parameter values but also the systematic fine-tuning process has allowed to make a fair comparison between the developed metaheuristic methods.

According to the fine-tuning procedure that we propose (that is, applying CALIBRA followed by N&M) when enough fine-tuning time is available, the results of a computational experiment show that our proposal is better than the proposal suggested by the CALIBRA authors (that is, applying two times CALIBRA, the second one in a narrow range around the obtained parameter values). We refer to our fine-tuning procedure as *CALIBRA+N&M* and we refer to the alternative procedure as *CALIBRA+CALIBRA*.

The two fine-tuning proposals were tested fine-tuning the parameters of three metaheuristics developed during this thesis: a PSO algorithm (*DPSOpoi-*c_p*dyn*), the EM algorithm and the PSC algorithm. Table 12 shows average RTV values obtained by the PSO, EM and PSC algorithms using the parameter values returned by CALIBRA, *CALIBRA+N&M* and *CALIBRA+CALIBRA*) for the 740 test instances when the algorithms are run 1,000 seconds.

Table 12. Average RTV values

	PSO	**EM**	**PSC**
CALIBRA	1,537.34	330.29	161.60
CALIBRA+N&M	794.93	295.31	160.72
CALIBRA+CALIBRA	1,115.72	426.58	208.49

We can see that the three metaheuristics perform better using the parameter values returned by our proposed fine-tuning procedure than using the parameter values returned by CALIBRA or by *CALIBRA+CALIBRA*. The RTV averages obtained by the PSO, EM and PSC algorithms when using the *CALIBRA+N&M* parameter values are 48.29%, 10.59% and 0.30% better than the RTV averages obtained using the CALIBRA parameter values, respectively. On the other hand, applying again CALIBRA may be detrimental. In the case of the EM and PSC algorithm, the RTV averages obtained when using the *CALIBRA+CALIBRA* parameter values are around 22% worse than the averages when using the CALIBRA parameters.

To sum up, CALIBRA is able to obtain in a reasonable time quite good parameter values. However, if more fine-tuning time is available, the fine-tuning procedure that we propose obtain parameter values that may help to the algorithm to perform still better. Moreover, the proposed procedure, as well as CALIBRA, requires little human intervention.

VII. Conclusions

The objectives of this thesis have been successfully achieved:

1a. The exact solution of the RTVP has been improved with the proposed B&B algorithm. The size of the instances that can be solved in a practical time has been increased from 40 copies to 55 copies to be sequenced.

1b. Larger instances can be solved quickly using a robust non-exact algorithm called $MS+SA_{N1}$. We have evidences that this method may obtain optimal or near-optimal solutions.

2. A contribution in the hyper-heuristic research has been done showing the viability of working with metaheuristics under the hyper-heuristic methodology.

3. A systematic, hands-off fine-tuning procedure has been proposed. Moreover, in the articles and communication derived from this thesis we have tried to make aware of the importance of the fine-tuning.

The RTVP defined as a cyclic, not distance-constrained problem oriented to minimise the RTV metric (Equation 1) has exhaustively studied and solved during the development of this thesis. We think that the work developed here is a good starting point to deal in a future other academic or real-life variants of this problem:

- The problem of scheduling the advertisements introduced in Bollapragada *et al.* (2004), which is a non cyclic variant of the RTVP and the objective function is to minimise the sum of the absolute value of the distance discrepancies instead of the square value of the distance discrepancies.

- The *minmax* RTVP; that is, to minimise the maximum of the discrepancies.

- The other problems introduced in León *et al.* (2003) based on combining the characteristics explained in Chapter II.1

- The commonly used measure between two successive symbols is one unit of distances, but this could be generalised to be dependent on the type of symbols and their relationships. For instance, in the Bollapragada *et al.* problem, each advertisement lasts a different time, so a more realistic approximation would be considering the allocation of the same commercials as evenly spaced in time instead of spaced in positions.

It is worth to point that a feasible solution of the RTVP is also a feasible solution in all aforementioned variants in which only the objective function changes. Thus, most of the proposed methods can be easily adapted to solve these variants and it seems that their efficiency will not be change perceptibly. For instance, the adaptation of the metaheuristic and hyper-heuristic proposed methods is immediate since only the fitness function has to be modified. It seems also that the developed heuristic proposed here (*ENH-H*, see chapter V.3.1) can be applied to solve the non-cyclic variant and the *minmax* RTVP (in fact, the *minmax* RTVP is solved by the proposed heuristic to provide the results as future benchmarking purposes). On the other hand, the B&B algorithm would need more adaptations according to the variant to be solved since it was designed taking the advantages of the special characteristics of the RTVP and these characteristics may be change in the problem variant to be solved.

With respect to the hyper-heuristic field, a future study could investigate how to enhance the different mechanisms of the proposed hyper-heuristics. For instance, the amount of time used in the learning and launching stages of our hyper-heuristics decreases/increases deterministically at each cycle of the search. However, it seems better than this amount of time changes dynamically according to many attributes including the running time, problem characteristics and individual performance of the metaheuristics, among others.

Another future interesting research to solve the RTVP is adding the SA-based and the VNS hybrid algorithms (which perform very well for solving the RTVP) as low-level heuristics in the proposed hyper-heuristics.

References

Adenso-Díaz, B. and Laguna, M. (2006) ‘Fine-tuning of algorithms using fractional experimental designs and local search’, *Operations Research,* Vol. 54, pp. 99-114.

Anily, S., Glass, C.A. and Hassin, R. (1998) ‘The scheduling of maintenance service’, *Discrete Applied Mathematics,* Vol. 82, pp. 27-42.

Balinski, M.L. and Young, H.P. (1982) *Fair Representation*, Yale University Press, New Haven.

Balinski, M. and Shahidi, N. (1998) ‘A simple approach to the product rate variation problem via axiomatics’, *Operation Research Letters*, Vol. 22, pp. 129-135.

Bar-Noy, A., Nisgav, A. and Patt-Shamir, B. (2002) ‘Nearly optimal perfectly-periodic schedules’, *Distributed Computing*, Vol. 15, pp. 207–220.

Bautista, J., Companys, R. and Corominas, A. (1995) ‘Seqüenciació d’unitats en context JIT’, *TOE,* Vol. 9, Edicions UPC, ISBN 84-7653-497-3.

Bautista, J., Companys, R. and Corominas, A. (1996) ‘A Note on the Relation between the Product Rate Variation (PRV) Problem and the Apportionment Problem’, *Journal of the Operational Research Society*, Vol. 47, pp. 1410-1414.

Bautista, J., Companys, R. and Corominas, A. (1997) ‘Modelling and solving the production rate variation problem (PRVP)’, *TOP*, Vol. 5, pp. 221-239.

Bollapragada, S., Bussieck, M.R. and Mallik, S. (2004) ‘Scheduling Commercial Videotapes in Broadcast Television’, *Operations Research*, Vol. 52, pp. 679-689.

Brusco, M.J. (2008) ‘Scheduling advertising slots for television’, *Journal of the Operational Research Society*, Vol. 59, pp. 1363-1372.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P. and Schulenburg, S. (2003) ‘Hyper-heuristics: An Emerging Direction in Modern Search Technology’, Chapter 16 in *Handbook of Metaheuristics*, Eds. Glover and Kochenberger, Kluwer Academic Publishers, pp. 457-474.

Burke, E. and Kendall, G., Eds. (2005), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer.

Chiarandini, M., Paquete, L., Preuss, M.. and Ridge, E. (2007), ‘Experiments on metaheuristics: Methodological overview and open issues’, *Technical Report DMF-2007-03-003*, The Danish Mathematical Society. Available at http://bib.mathematics.dk/preprint.php?lang=en&id=IMADA-PP-2007-04.

Corominas, A., Kubiak, W. and Moreno, N. (2007) ‘Response time variability’, *Journal of Scheduling*, Vol. 10, pp. 97-110.

Corominas, A., Kubiak, W. and Pastor, R. (2009) ‘Heuristics for the Response Time Variability problem’, *Technical report IOC-DT-P-2009-03*, Universitat Politècnica de Catalunya, Spain.

Corominas, A., Kubiak, W. and Pastor, R. (2010) ‘Mathematical programming modeling of the Response Time Variability Problem’, *European Journal of Operational Research*, Vol. 200, pp. 347-357.

Dong, L., Melhem, R. and Mosse, D. (1998) ‘Time slot allocation for real-time messages with negotiable distance constrains requirements’, *Fourth IEEE Real-Time Technology and Applications Symposium* (RTAS'98), Denver, CO. pp. 131-136.

Eiben, A.E., Hinterding, R. and Michalewicz, Z. (1999) ‘Parameter control in evolutionary algorithms’, *IEEE Transactions on evolutionary computation*, Vol. 3, pp. 124-141.

Glover, F. and Kochenberger, G., Eds. (2003), *Handbook of Metaheuristics*, Kluwer Academic Publishers.

Han, C.C., Lin, K.J. and Hou, C.J. (1996) ‘Distance-constrained scheduling and its applications in real-time systems’, *IEEE Transactions on Computers*, Vol. 45, pp. 814-826.

Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report*, University of Maryland, USA. Available at http://hdl.handle.net/1903/7082.

Herrmann, J.W. (2009) ‘Using aggregation to reduce response time variability in cyclic fair sequences’, *Journal of Scheduling*, doi 10.1007/s10951-009-0127-7.

Inman, R.R. and Bulfin, R.L. (1991) ‘Sequencing JIT mix-model assembly lines’, *Management Science*, Vol. 37, pp. 901-904.

Korkmazel, T. and Meral, S. (2001) ‘Bicriteria sequencing methods for the mixed-model assembly line in just-in-time production systems’, *European Journal of Operational Research*, Vol. 131, pp. 188-207.

Kubiak, W. (1993) ‘Minimizing variation of production rates in just-in-time systems: A survey’, *European Journal of Operational Research*, Vol. 66, pp. 259-271.

Kubiak, W. (2004) ‘Fair Sequences’, Chapter 19 in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapman and Hall.

Kubiak, W. (2009) ‘Proportional optimization and fairness’, Springer.

León, D., Corominas, A. and Lusa, A. (2003) ‘Resolución del problema PRV min-var’, *Technical report IOC-DT-I-2003-03*, Universitat Politècnica de Catalunya, Spain.

Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 35, pp. 192-207.

Monden, Y. (1983) ‘Toyota Production Systems’, *Industrial Engineering and Management Press*, Norcross, GA.

Moreno, N. (2002) ‘Solving the product rate variation problem (PRVP) of large dimensions as an assignment problem’, Doctoral Thesis, DOE, ETSEIB-UPC.

Nelder, J.A. and Mead, R. (1965) ‘A simplex method for function minimization’, *The Computer Journal*, Vol. 7, pp. 308-313.

Ross, P. (2005) ‘Hyper-heuristics’, Chapter 17 in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Eds. Burke and Kendall, Springer, pp. 529-556.

Waldspurger, C.A. and Weihl, W.E. (1994) ‘Lottery Scheduling: Flexible Proportional-Share Resource Management’, *First USENIX Symposium on Operating System Design and Implementation*, Monterey, California.

Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Scheduling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528*, Massachusetts Institute of Technology, MIT Laboratory for Computer Science. Available at https://eprints.kfupm.edu.sa/67117

Wei, W.D. and Liu, C.L. (1983) ‘On a periodic maintenance problem’, *Operations Research Letters*, Vol. 2, pp. 90-93.

References derived from the thesis

Corominas, A., García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics’ *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 184, pp. 128-137.

Corominas, A., García-Villoria, A., and Pastor, R. (2009a) ‘Using an Ant Colony System to solve the Response Time Variability Problem’, *Technical report IOC-DT-P-2009-06*, Universitat Politècnica de Catalunya, Spain.

Corominas, A., García-Villoria, A. and Pastor, R. (2009b) ‘Solving the Response Time Variable Problem by means of a Variable Neighbourhood Search Algorithm’, *13th IFAC Symposium of Information Control Problems in Manufacturing* (INCOM 2009), Moscow, Russia.

Corominas, A., García-Villoria, A. and Pastor, R. (2009c) ‘Using Tabu Search for the Response Time Variability Problem’, *3rd International Conference on Industrial Engineering and Industrial Management* (CIO 2009), Barcelona and Terrassa, Spain.

Corominas, A., García-Villoria, A. and Pastor, R. (2009d) ‘Resolución del response time variability problem mediante tabu search’, *VIII Evento Internacional de Matemática y Computación* (COMAT’2009), Universidad de Matanzas, Cuba.

Corominas, A., García-Villoria, A. and Pastor, R. (2009e) ‘Metaheuristic algorithms hybridized with variable neighbourhood search for solving the response time variability problem’, *Journal of Scheduling* (1st review in progress).

Corominas, A., García-Villoria, A. and Pastor, R. (2009f) ‘An enhanced metaheuristic for solving the response time variability problem’, *Technical report IOC-DT-P-2009-07*, Universitat Politècnica de Catalunya, Spain.

Corominas, A., García-Villoria, A. and Pastor, R. (2009g) ‘A systematic procedure based on CALIBRA and the Nelder & Mead algorithm for fine-tuning metaheuristics’, *Journal of the Operational Research Society* (2nd review in progress).

Corominas, A., García-Villoria, A. and Pastor, R. (2010) ‘A Parametric Multi-start Algorithm for Solving the Response Time Variability Problem’, *Lecture Notes in Computer Science*, Vol. 5910, pp. 315-322.

García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics’, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 146, pp. 187-194.

García-Villoria, A., Corominas, A., Delorme, X., Dolgui, A., Kubiak, W. and Pastor, R. (2009a) ‘A branch and bound approach for the response time variability problem’, *Technical report IOC-DT-P-2009-05*, Universitat Politècnica de Catalunya, Spain.

García-Villoria, A., Corominas, A. and Pastor, R. (2010) ‘Solving the Response Time Variability Problem by means of the Cross-Entropy Method’, *Special Issue on ‚Production Line Systems: Concepts, Methods and Applications of the International Journal of Manufacturing Technology and Management*, Vol. 20, pp. 316-330.

García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of a psychoclonal approach’, *Special Issue on Advances in Metaheuristics of the Journal of Heuristics*, doi:10.1007/s10732-008-9082-2.

García-Villoria, A. and Pastor, R. (2009a) ‘Introducing dynamic diversity in a discrete Particle Swarm Optimization’, *Computers & Operations Research*, Vol. 36, pp. 951-966.

García-Villoria, A. and Pastor, R. (2009b) ‘Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism’, *International Journal of Production Research*, doi: 10.1080/00207540902862545.

García-Villoria, A. and Pastor, R. (2010) ‘Solving the Response Time Variability Problem by means of a genetic algorithm’, *European Journal of Operational Research*, Vol. 202, pp. 320-327.

García-Villoria, A., Salhi, S., Corominas, A. and Pastor, R. (2009b) ‘Hyper-heuristic Approaches for the Response Time Variability Problem’, *European Journal of Operational Research* (2nd review in progress).

Salhi, S. and García-Villoria, A. (2009) ‘An adaptive-based heuristic for the Response Time Variability Problem’, *Operations Research* (1st review in progress).

Annex A1. Articles published or accepted in journals included in the JCR index

Introducing dynamic diversity in a discrete Particle Swarm Optimization

Article published as [Computers & Operations Research, Volume 36, Issue 3, March 2009, Pages 951-966] [DOI: http://dx.doi.org/10.1016/j.cor.2007.12.001] © [copyright Elsevier]

Introducing dynamic diversity into a discrete Particle Swarm Optimization

Alberto GARCÍA-VILLORIA and Rafael PASTOR[*]
Institute of Industrial and Control Engineering (IOC)
Technical University of Catalonia (UPC)
{alberto.garcia-villoria / rafael.pastor}@upc.edu

Abstract. Particle Swarm Optimization (PSO) is an evolutionary metaheuristic inspired by the flocking behaviour of birds which has successfully been used to solve several kinds of problems, although there are few studies aimed at solving discrete optimization problems. One disadvantage of PSO is the risk of a premature search convergence. To prevent this, we propose to introduce diversity into a discrete PSO by adding a *random velocity*. The degree of the introduced diversity is not static (i.e., preset before running PSO) but instead changes dynamically according to the heterogeneity of the population (i.e. if the search has converged or not).We solve the Response Time Variability Problem (RTVP) to test these two new ideas. The RTVP is an NP-hard combinatorial scheduling problem that has recently appeared in the literature. It occurs whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the instants at which they receive the necessary resources is minimized. The most efficient algorithm for solving non-small instances of the RTVP published to date is a classical PSO algorithm, referred to by the authors as *PSO-M1F*. In this paper, we propose ten discrete PSO algorithms for solving the RTVP: one based on the ideas described above (*PSO-$c_3$dyn*) and nine based on strategies proposed in the literature and adapted for solving a discrete optimization problem such as the RTVP. We compare all eleven PSO algorithms and the computational experiment shows that, on average, the best results obtained are due to our proposal of dynamic control mechanism for introducing diversity.

Keywords: particle swarm optimization, adaptive parameters, response time variability, scheduling

1. Introduction

Particle Swarm Optimization (PSO) is an evolutionary stochastic metaheuristic designed by Kennedy and Eberhart (1995) that has been studied by several researchers since its publication (Hu et al., 2004). PSO has been successfully applied to a variety of

[*] Corresponding author: Rafael Pastor, IOC – Institute of Industrial and Control Engineering, Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; Tel. + 34 93 401 17 01; Fax. + 34 93 401 66 05; e-mail: rafael.pastor@upc.edu

problems such as artificial neural network training (Chau, 2006; Geethanjali et al., 2007), combinatorial optimization problems (Andrés et al., 2004; Liao et al., 2007; Secrest, 2001; Tasgetiren et al., 2007) and multiobjective optimization problems (Hu and Eberhart, 2002; Yin et al., 2007). Most of the PSO applications published in the literature were designed to solve continuous optimization problems, but there are few PSO applications for discrete optimization problems. The applications of PSO on combinatorial optimization problems are still considered limited, but the advantages of PSO include a simple structure, immediately accessible for practical applications, easy of implementation, speed to acquire solutions and robustness (Pan et al., 2007). In this paper, we develop ten discrete PSO algorithms for solving an NP-hard scheduling problem.

The core of PSO is based on an analogy of the social behaviour of flocks of birds when they search for food. Since it is a population-based evolutionary metaheuristic, PSO has a population (known as swarm in the PSO ambit) of particles. Each particle has an associated point in the search space (which represents a solution) and an associated velocity (which indicates how the point of the particle is moved in the search space). The current velocity of a particle is typically a linear combination of three types of velocity: 1) the inertia velocity (i.e. its previous velocity); 2) the velocity to the best point found by the particle; and 3) the velocity to the best point found by the swarm. The PSO algorithm iteratively modifies the point and the velocity of each particle as it looks for the optimal solution.

The trade-off between the exploration (i.e. the global search) and the exploitation (i.e. the local search) of the search space is critical to the success of an evolutionary metaheuristic. Trelea (2003) demonstrated that PSO always converges at certain values of its parameters but if the convergence is premature then several regions of the search space will remain unexplored. Several strategies have been proposed in the literature for correcting the tendency to converge prematurely on a local optimum.

Clerc (2004) suggested a PSO variant in which the velocity to the best point found by the swarm is replaced by the velocity to the current best point of the swarm, although he does not test this variant. The idea was later implemented in a discrete PSO for solving the flowshop problem (Liao et al, 2007).

Several authors developed strategies based on dynamically modifying the value of the PSO parameter called *inertia weight*, which weights the inertia velocity. Larger inertia weight values facilitate a more global behaviour and smaller values facilitate a more local behaviour. Therefore, the inertia weight is changed to achieve better balance dynamics between the global and local search capabilities. Since exploration is more important at the beginning of the search process and exploitation is more important at the end, the usual strategy is to start with a large inertia weight value and then decrease it over the iterations of the algorithm (Poli et al., 2007), which is the method adopted by Eberhart and Shi (2001), He et al. (2004) and Shi and Eberhart (1998a).

Other authors introduce diversity into the swarm to escape from the current local optimum. There are different ways of introducing diversity and controlling the degree of diversity introduced. Clerc (2006) and Zhang et al. (2003) dynamically change the size of the swarm according to the performance of the algorithm. The size of the swarm is important because too few particles will cause the algorithm to converge prematurely to a local optimum, while too many particles will slow down the algorithm. Note that adding new particles when the algorithm converges introduces diversity into the swarm. He et al. (2004) developed a PSO variant, called PSO Passive Congregation (PSOPC), which was inspired by the work of Parrish and Hamner (1997) in which spatial structure of animal group organizations are modeled. PSOPC uses an analogy of passive congregation, which describes a situation in which an individual is attracted to other group members but there is no display of social behavior. Passive congregation is expressed in PSOPC by adding a new term to the velocity at which the particle tends to move toward the point of another particle in the swarm that is randomly selected at each iteration of the PSOPC algorithm. This new term is weighted by a factor which is initially set to a small value and then linearly increased over the iterations of the algorithm. Xie et al. (2002) propose two Dissipative PSO (DPSO) algorithms: in the first, for each particle some *components* of the velocity are chosen at random and weighted by a random weight that is uniformly distributed between 0 and 1; in the other, for each particle some *components* of the point are chosen at random and weighted by a random weight that is uniformly distributed between 0 and 1. Fieldsend and Singh (2002) add a new term to the velocity called *turbulence*, which represents a random velocity. This idea was initially proposed in the early development of PSO (Kennedy and Eberhart, 1995), which used a stochastic variable called *craziness*, but this was soon omitted from the classical PSO algorithms.

Various other solutions have been proposed for preventing premature convergence: objective functions which change over time (Hu and Eberhart, 2001); noisy evaluation of the function objective (Parsopoulos and Vrahatis, 2001); repulsion to keep particles away from the optimum (Parsopoulos and Vrahatis, 2004); dispersion between particles that are too close to one another (Loøvbjerg and Krink, 2002); reduction of the attraction of the swarm centre to prevent the particles clustering too tightly in one region of the search space (Blackwell and Bentley, 2002); hybrids with other metaheuristic such as genetic algorithms (Robinson et al., 2002; Angeline, 1998); or ant colony optimization (Hendtlass, 2001), etc. For an up-to-date overview of the particle swarm optimization, see Poli et al. (2007).

Finally, several researchers have experimented with the swarm topology. Mendes (2004) carried out an in-depth study of the effect of static topologies. Clerc (2006) and Liang and Suganthan (2005) conducted interesting research into dynamic topologies. In the present study we consider a static fully connected graph, which is a static topology (Poli et al., 2007).

The papers mentioned above are a sample of the wider research effort aimed at preventing premature convergence and enabling the PSO algorithm to escape from a local optimum. Although there are a lot of proposals for preventing premature convergence (e.g. by modifying the value of the inertia weight; by introducing diversity adding more particles, passive congregation or turbulence; hybridization, etc.), only three types of feedback of the current state of the search are taken, which are used to modify some of the parameter values accordingly: 1) no feedback, because all of the parameter values are constant over the execution of the algorithm; 2) the current iteration of the algorithm; and 3) the performance of the algorithm.

In the present study we propose a new method for introducing diversity into the swarm by adding a term of randomness to the particle velocity. This term of randomness is weighted by a coefficient that we call the *diversity coefficient*. We take feedback of the heterogeneity of the swarm to control the degree of diversity introduced (i.e. the value of the diversity coefficient). To our knowledge, no PSO algorithm uses a measure of the swarm heterogeneity to dynamically modify the values of its parameters. It seems reasonable to assume that the value of the diversity coefficient should not to be set before running the PSO algorithm. Instead, it is preferable to allow the value to change dynamically during the execution of the algorithm according to the convergence of the swarm (i.e. the convergence of the search): the more heterogeneous the population, the smaller the value of the diversity coefficient, and vice versa.

We use the Response Time Variability Problem (RTVP) to test our proposal. The RTVP is a scheduling problem that was recently defined in the literature (Corominas et al., 2007) and is very difficult to solve optimally (it is NP-hard). This problem has a wide range of real-life applications: it occurs whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the instants at which they receive the necessary resources is minimized. For example, it can be used to regularly sequence models in the automobile industry (Monden, 1983), to broadcast video and audio data frames of applications over asynchronous transfer mode networks as constantly as possible (Dong et al., 1998), in the stride scheduling technique (Waldspurger and Weihl, 1995) and in the periodic machine maintenance problem when the distances between consecutive services of the same machine are equal (Anily et al., 1998).

García et al. (2006) used the classical PSO metaheuristic to solve the RTVP and proposed four PSO variations. The best of these, called *PSO-M1F* by the authors, is the best heuristic method that has been published to date for solving non-small instances of the RTVP.

In order to compare our PSO algorithm (that we call *PSO-$c_3$dyn*), we have adapted a representative set of PSO algorithms for solving the RTVP: the PSO with a velocity to the best current point (Liao et al., 2007); the PSOPC (He et al., 2004); the two DPSO (Xie et al., 2002); the PSO with turbulence (Fieldsend and Singh, 2002); and two classical PSOs: the *PSO-M1F* adaptation (García et al., 2006) and a Constriction PSO (Clerc and Kennedy, 2002)). Moreover, we propose some variants of these algorithms that incorporate our proposed control mechanism of diversity or our proposed random velocity.

A computational experiment is carried out and it is shown that the best results are achieved when using the control mechanism of diversity proposed in this paper.

The remainder of the paper is organized as follows. Section 2 describes the scheme of the classical PSO and our newly proposal for introducing diversity according to the convergence of the population. Section 3 presents a formal definition of the Response Time Variability Problem. Section 4 explains our PSO algorithm and the adaptation of the mentioned PSO algorithms to solve the RTVP. Section 5 provides the results of the computational experiment. Finally, some conclusions and suggestions for future research are given in Section 6.

2. Particle Swarm Optimization

First, the classical PSO is briefly explained in Section 2.1. Next, the method that we propose for introducing diversity according to the convergence of the population is presented in Section 2.2. Finally, how to set the value of the coefficient that weights the diversity introduced into the population is discussed in Section 2.3.

2.1. Classical scheme of PSO

Particle Swarm Optimization (PSO) was first described by Kennedy and Eberhart (1995). It is an evolutionary metaheuristic based on the behaviour of flocks of birds when they look for food.

The population of PSO or swarm is composed of particles (birds), which have an associated multi-dimensional real point in the search space (which represents a solution) and an associated velocity (the movement of the point in the multi-dimensional real space). The velocity of a particle is typically a linear combination of three types of velocity: 1) the inertia velocity; 2) the velocity to the best point found by the particle; and 3) the velocity to the best point found by the swarm. The point and the velocity of each particle *i* of the population are iteratively modified by the algorithm as it looks for an optimal solution. In the original PSO developed by Kennedy and Eberhart (1995), the point and the velocity of the particles are modified according to the following two equations:

$$v_{t+1,i} = v_{t,i} + c_1 \cdot (P^*_{t,i} - X_{t,i}) + c_2 \cdot (SP^*_t - X_{t,i}) \tag{1a}$$

$$X_{t+1,i} = X_{t,i} + v_{t+1,i} \tag{2}$$

where $v_{t,i}$ is the inertia velocity of the particle *i* at iteration *t*, $X_{t,i}$ is the point of the particle *i* at iteration *t*, $P^*_{t,i}$ is the best point found by the particle *i* up to iteration *t*, SP^*_t is the best point found by the swarm up to iteration *t*, and c_1 and c_2 are the coefficients, called *acceleration coefficients*, that weight the relevance of the last two types of velocity.

In the original PSO it was necessary to dampen the particle dynamics and the solution proposed was to maintain the velocity within the range [-V_{max}, + V_{max}], where V_{max} was a parameter of the original PSO.

To find an appropriate value of V_{max} according to the problem to be solved was a hard task. Thus, Shi and Eberhart (1998b) proposed to modify the original first PSO equation (Eq. 1a) to the following expression:

$$v_{t+1,i} = \omega \cdot v_{t,i} + c_1 \cdot (P^*_{t,i} - X_{t,i}) + c_2 \cdot (SP^*_t - X_{t,i}) \tag{1b}$$

where ω is the coefficient (called *inertia weight*) that weights the inertia velocity. Equations (1b) and (2) are the core of the classical PSO.

At each iteration of the PSO algorithm, Equations (1b) and (2) reflect the compromise of each particle between following its own exploration, moving towards the best point it has found by itself and moving towards the best point found by the swarm. Figure 1 shows the pseudocode of the classical PSO algorithm.

```
1. Set t = 0
2. Randomly initialize positions of all particles
3. Initialize velocities of all particles with void velocities
4. While stopping criteria is not reached do:
5.     For each particle $i$ in the swarm:
6.         **Calculate fitness**: Set $f_i$ = fitness of $X_{t,i}$
7.         **Update** $P_{t,i}^*$: If $f_i$ is better than the fitness of $P_{t,i}^*$, then set $P_{t,i}^*$ to $X_{t,i}$
8.     End For
9.     **Update** $SP_t^*$: Set $Xbest_t$ = the best point of the current swarm
                       If $Xbest_t$ is better than $SP_t^*$, then set $SP_t^*$ to $Xbest_t$
10.    For each particle $i$ in the swarm:
11.        **Update velocity**: Calculate velocity $V_{t+1,i}$ using Equation (1b)
12.        **Update point**:    Calculate point $X_{t+1,i}$ using Equation (2)
13.    End For
14.    Set t = t + 1
15. End While
```

Figure 1. Pseudocode of the classical PSO algorithm

Clerc and Kennedy (2002) used a series of theoretical analyses to develop a strategy of *constriction coefficients*. They proposed the Constriction PSO (CPSO), in which Equation (1a) is modified to the following expression:

$$v_{t+1,i} = \chi \cdot \left(v_{t,i} + c_1 \cdot (P_{t,i}^* - X_{t,i}) + c_2 \cdot (SP_t^* - X_{t,i}) \right) \qquad (1c)$$

Although CPSO is algebraically equivalent to PSO with inertia weight, CPSO can generate higher-quality solutions than PSO with inertia weight for some of the problems studied in the literature (Eberhart and Shi, 2001).

2.2. Introducing a diversity factor into PSO

As mentioned above, Trelea (2003) demonstrated that the classical PSO metaheuristic always converges for some values of its parameters. Although the convergence of the metaheuristic is a desirable property, it can converge too fast and become trapped in a local optimum, particularly when PSO deals with integer variables (Hu et al., 2004).

In Section 1 several ideas for preventing a premature convergence of the PSO algorithm have been presented, including the introduction of diversity. In this paper we propose a new way to introduce diversity into the population by adding a new type of velocity (a random velocity) to the linear combination formulated in Equation (1b). The modified equation is as follows:

$$v_{t+1,i} = \omega \cdot v_{t,i} + c_1 \cdot (P_{t,i}^* - X_{t,i}) + c_2 \cdot (S\ {}_t^{*} - X_{t,i}) + c_3 \cdot (R_{t,i} - X_{t,i}) \qquad (1d)$$

where $R_{t,i}$ is a random point generated for the particle i at iteration t, and c_3 (called the *diversity coefficient*) is the coefficient that weights the relevance of the new type of velocity. Note that the random velocity introduces diversity into the movements of the particles.

The value of the diversity coefficient is important in preventing PSO from converging prematurely and leaving regions of the search space unexplored. A large value of the diversity coefficient enables the PSO algorithm to carry out a wide-ranging exploration of the search space and thereby look for new promising points in exchange for a small exploitation; and vice versa for a small value of the diversity coefficient.

2.3. Parameter tuning versus parameter control

One laborious aspect of metaheuristics is choosing the right parameter values. This is a difficult task which requires considerable effort. Eiben et al. (1999) draw a distinction between two principal ways of setting parameter values: parameter tuning and parameter control. Parameter tuning refers to finding and to setting the parameter values before running the algorithm, whereas parameter control refers to using parameter values that change during the execution of the algorithm.

Population-based metaheuristics such as PSO are intrinsically dynamic and the optimal parameter values might depend on which search state the algorithm is in. Therefore, the parameter values should be modified during the execution of the algorithm according to the search state. According to Eiben et al. (1999), there are two ways of doing this: by using adaptive parameter control or self-adaptive parameter control.

The parameter values in adaptive parameter control are explicitly adaptive: the changes in values are given by a heuristic rule which takes feedback from the current search state and modifies the parameter values accordingly. The input information for the current state is usually the number of iterations of the algorithm (i.e. the current iteration of the search), the performance of operators (i.e. the progress of the search) or the diversity of the population (for more details, see Eiben et al., 1999).

In self-adaptive parameter control, the parameters of the metaheuristic are incorporated into the representation of the solution. Thus, the parameter values are implicitly adaptive because they evolve together with the solutions of the population. Self-adaptive parameter control is more common in genetic algorithms in which the parameters can be incorporated into the chromosomes, which renders them subject to evolution (Angeline, 1996; Hinterding et al., 1996).

In Section 2.2, we proposed introducing diversity into the PSO population by adding a random velocity weighted by the diversity coefficient (c_3). The value of the diversity coefficient may be static (i.e. its value is obtained by parameter tuning) or dynamic (i.e. its value is obtained by adaptive parameter control). The main difficulty of using a static diversity coefficient is to find a value that is (if possible) high enough to facilitate good exploration but low enough to facilitate good exploitation. Alternatively, a dynamic diversity coefficient can be used. We propose to use a control that changes the value of the diversity coefficient according to the heterogeneity of the population: the more heterogeneous the current population, the smaller the value of the diversity coefficient, and the less heterogeneous the current population, the larger the value of the diversity coefficient. To our knowledge, no PSO algorithm presented in the literature uses an adaptive parameter control whose feedback is based on the heterogeneity of the swarm.

3. The Response Time Variability Problem (RTVP)

The Response Time Variability Problem (RTVP) occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources. This combinatorial optimization problem is easy to formulate, but Corominas et al. (2007) proved that it is NP-hard and, therefore, very difficult to solve optimally.

The RTVP was recently defined in the literature and first presented by Corominas et al. (2007). The RTVP is formulated as follows. Let *p* be the number of products, d_i the number of units of product *i* (*i* = 1,…, *p*) and *D* the total number of units ($D = \sum_{i=1}^{p} d_i$). Let *s* be a solution of an instance in the RTVP that consists of a circular sequence of units ($s = s_1 s_2 \ldots s_D$), where s_j is the unit sequenced in position *j* of sequence *s*. For all product *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions at which the units *k* + 1 and *k* of product *i* are found (i.e. the number of positions between the units, where the distance between two consecutive positions is considered equal to 1). As the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_i}^i$ is the distance between the first unit of product *i* in a cycle and the last unit of the same product in the preceding cycle. Let $\bar{t}_i$ be the average distance between two consecutive units of product *i* ($\bar{t}_i = D/d_i$). For all product *i* in which $d_i = 1$, t_1^i is equal to $\bar{t}_i$. The objective is to minimize the $RTV = \sum_{i=1}^{p} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2$.

For example, let $p = 3$, $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\bar{t}_C = 2$. Any sequence is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a solution, where $RTV = \left((5-4)^2 + (3-4)^2\right) + \left((2-4)^2 + (6-4)^2\right)$ $+ \left((2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2\right) = 2 + 8 + 2 = 1$.

There are only three published works that make reference to the RTVP. Corominas et al. (2007) presented the RTVP and proposed a mixed integer linear programming (MILP) model and five heuristic algorithms to solve it. Corominas et al. (2006) presented an improved MILP model with a practical limit for obtaining optimal solutions of around 40 units to be scheduled. García et al. (2006) presented six metaheuristic algorithms: a multi-start algorithm, a GRASP (Greedy Randomized Adaptive Search Procedure) algorithm and four PSO algorithm variations. In order to solve non-small instances of the RTVP, one of the PSO algorithm variations (referred to as *PSO-M1F*) is the method published to date that obtains, on average, the best results.

4. PSO algorithms for solving the RTVP

We propose eleven different PSO algorithms for solving the RTVP. The first approach, *PSO-M1F*, is taken from García et al. (2006) and is based on the classical PSO. The second algorithm, *CPSO*, is an adaptation of the Constriction PSO proposed by Clerc and Kennedy (2002). The third algorithm, *PSO-*c_3*dyn*, is our proposal in which the random velocity is introduced together with the dynamic control mechanism of the value of the diversity weight according to the heterogeneity of the swarm. The fourth algorithm, *PSOCB*, incorporates the idea of replacing the velocity to the best point found with the swarm by the velocity to the current best point of the swarm (Clerc, 2004). The fifth algorithm, *PSOPC*, is an adaptation of the PSO with passive congregation proposed by He et al. (2004). The sixth and seventh algorithms, *DPSOvel* and *DPSOpoi*, are adaptations of the two dissipative PSOs proposed by Xie et al. (2002) in which the velocity or the point are randomly modified. The eighth algorithm, *PSO-*c_3*stat*, is an adaptation of the PSO with turbulence proposed by Fieldsend and Singh (2002). Finally, the ninth, tenth and eleventh algorithm, *PSO-*c_3*dyn'*, *PSOPC'* and *DPSOpoi-*c_p*dyn*, are variations that we developed by merging ideas incorporated into the designs of the previous algorithms.

In the next eleven sections, each PSO algorithm is explained. In Section 4.12 we explain how the parameters of the eleven PSO algorithms were fine-tuned.

4.1. A classical PSO algorithm (PSO-M1F)

PSO was originally designed for working in multi-dimensional real spaces. The representation of a solution of the RTVP (and many other combinatorial optimization problems) consists of an ordered sequence of integer numbers. Therefore, the PSO metaheuristic has to be adapted to work with this type of solution representation (i.e. the point of a particle is now a sequence of integer numbers). This is done by redefining the elements (point and velocity) and the operations (external multiplication of a coefficient by a velocity, sum of velocities and sum of a velocity plus a point) of Equations (1b) and (2). Moreover, it is also necessary to determinate how the initial population is set and the stopping criteria

García et al. (2006) proposed four variations based on the classical PSO with inertia weight (Shi and Eberhart, 1998b) for solving the RTVP. The differences between these variations derive from the way in which the velocity is redefined. The PSO algorithm that produces the best results is *PSO-M1F*. In the following sections, the elements and the operations of *PSO-M1F* are briefly explained (for a more detailed explanation, see García et al., 2006).

The definitions of elements and operations are the same for all eleven PSO algorithms.

4.1.1. Point of the particle

A point consists of a solution represented by a *D*-length array that contains the sequence of *D* units to be scheduled. For example, a point could be the sequence (C, A, C, B, C, B, A, C).

4.1.2. Velocity of the particle

The expression (X_2 – X_1), where X_2 and X_1 are two points, represents the difference between two points and the velocity needed to go from X_1 to X_2. This velocity is an ordered list of transformations (called movements) that must be applied sequentially to the particle so that its current point, X_1, changes to the other one, X_2. A movement is a pair of values (α / j). For each position u in the sequence (point) X_1, the algorithm determines whether the unit that is in position u of sequence X_1 is the same unit that is in position u of sequence X_2. If the units are different, α is the unit in position u of X_2 and j is equal to position u. Thus, this movement denotes that to go from the sequence X_1 to the sequence X_2, the unit in position j must be exchanged for the unit α.

For example, let $X_2 = (A_1, C_1, B_2, C_2, A_2, C_4, B_1, C_3)$ and $X_1 = (A_1, B_1, C_2, C_1, B_2, C_4, A_2, C_3)$. The sub-indices of the units are fictitious identifiers used to distinguish between the units of the same product. Thus, the velocity (X_2 – X_1) is formed by the list of movements [(C_1/2), (B_2/3), (C_2/4), (A_2/5), (B_1/7)], which are the movements for moving X_1 to X_2:

$X_1 = (A_1, B_1, C_2, C_1, B_2, C_4, A_2, C_3)$
$(C_1/2) \rightarrow (A_1, \mathbf{C_1}, C_2, \mathbf{B_1}, B_2, C_4, A_2, C_3)$
$(B_2/3) \rightarrow (A_1, C_1, \mathbf{B_2}, B_1, \mathbf{C_2}, C_4, A_2, C_3)$
$(C_2/4) \rightarrow (A_1, C_1, B_2, \mathbf{C_2}, \mathbf{B_1}, C_4, A_2, C_3)$
$(A_2/5) \rightarrow (A_1, C_1, B_2, C_2, \mathbf{A_2}, C_4, \mathbf{B_1}, C_3)$
$(B_1/7) \rightarrow (A_1, C_1, B_2, C_2, A_2, C_4, \mathbf{B_1}, C_3) = X_2$

4.1.3. External multiplication of a coefficient by a velocity

The values of the coefficients ω, c_1 and c_2 in Equation (1b) are between 0 and 1. When a coefficient is multiplied by a velocity, it indicates the probability of each movement to be applied. For example, if we multiply the coefficient 0.6 by the velocity [(C_1/2), (B_2/3), (C_2/4), (A_2/5), (B_1/7)], five random numbers between 0 and 1 are generated for comparison with the value 0.6. If the random number is lower than 0.6, the movement is applied. Therefore, if the values of the random numbers are 0.8, 0.3, 0.7, 0.4 and 0.2, movements (B_2/3), (A_2/5) and (B_1/7) are applied, whereas movements (C_1/2) and (C_2/4) are not. The resulting velocity of the multiplication is therefore [(B_2/3), (A_2/5), (B_1/7)], which, as previously stated, represents a list of movements to be applied to a point.

4.1.4 Sum of velocities

The sum of two velocities is simply the concatenation of their own list of movements.

4.1.5. Sum of a velocity plus a point

The sum of a velocity plus a point gives the result of sequentially applying each movement of the velocity to the point.

4.1.6. Initial population

The initial population is generated by setting a void velocity and a random point for each particle. As has been previously mentioned, each point consists of a solution represented by a sequence of integer numbers. A random solution is generated as follows: for each position in the sequence, a product to be sequenced is chosen at random. The probability of each product is equal to the number of units of this product that remain to be sequenced divided by the total number of units that remain to be sequenced.

4.1.7. Stopping criteria

The PSO algorithm stops after it has run for a preset time.

4.2. A Constriction PSO algorithm (*CPSO*)

As explained in Section 2.1, Clerc and Kennedy (2002) proposed a PSO algorithm based on *constriction coefficients*, whose core is Equations (1c) and (2). We propose a Constriction PSO algorithm (called *CPSO*) for solving the RTVP in which the particles behave according to Equations (1c) and (2) and the parameter values (the size of the swarm and the coefficients χ, c_1, and c_2) are preset.

4.3. A PSO algorithm with a dynamic value of the diversity coefficient (PSO-$c_3$dyn)

As explained in Section 2.2, the diversity coefficient (parameter c_3 in Equation (1d)) weights the relevance of the random velocity. This random velocity is important in preventing PSO from converging prematurely and leaving regions of the search space unexplored. A large value of the diversity coefficient enables to PSO algorithm to carry out a wide-ranging exploration of the search space and, thereby look for new promising points in exchange for a small exploitation; and vice versa for a small value of the diversity coefficient. Therefore, we propose a new PSO algorithm (called *PSO-$c_3$dyn*) in which an adaptive parameter control dynamically changes the diversity coefficient.

A good heuristic rule for dynamically changing the value of the diversity coefficient should be one in which the more heterogeneous the current population, the smaller the value given to the diversity coefficient and vice versa. First, a measure of heterogeneity of the population at iteration *t* is needed. The following expression is used:

$$het(t) = \frac{\sum_{i \in P} \left| v_{t,i} \right|}{D \cdot \left| P \right|} \tag{3}$$

where *P* is the population, $\left| v_{t,i} \right|$ is the number of movements of the velocity of particle *i* at iteration *t*, *D* is the number of units to be sequenced, and $\left| P \right|$ is the size of the population *P*. As the diversity is introduced into the movements of the particles, the measure of heterogeneity *het(t)* records the movement of the particles.

Given a specific value of the population heterogeneity (*het(t)*) at iteration *t*, it is necessary to set the corresponding value of the diversity coefficient (c_3). There are infinite possible heuristic rules for obtaining the value of the diversity coefficient according to *het(t)*. In the present study we used a set of rules according to which the value of the diversity coefficient decreases exponentially with *het(t)* (let the set be called $\hat{h}_e$). Set $\hat{h}_e$ is formulated as $\hat{h}_e = \left\{ e^{-K \cdot het(t)} : K \geq 0 \right\}$. Figure 2 shows some heuristic rules of $\hat{h}_e$ in graph form.

According to the value of *K*, a heuristic rule from set $\hat{h}_e$ is used. Therefore, *K* is considered to be one of the parameters of *PSO-c₃dyn* in addition to the population size and the coefficients ω, *c₁*, and *c₂* that need to be fine-tuned.

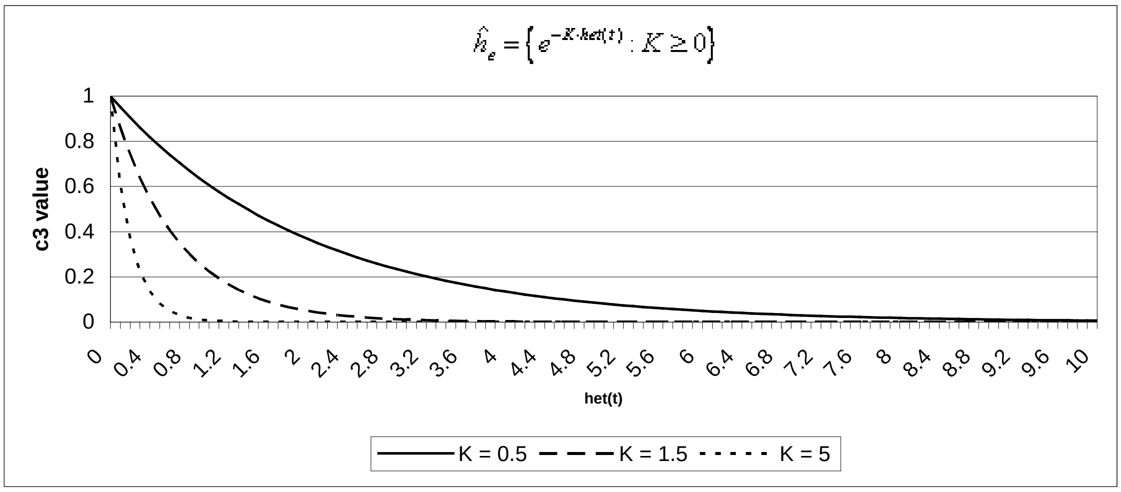


Figure 2. Examples of heuristic rules from $\hat{h}_e$ for obtaining the value of the diversity coefficient (c_3)

4.4. A PSO algorithm with a current best swarm point (PSOCB)

Clerc (2004) suggested a possible classical PSO variant in which the velocity to the best point found by the swarm is replaced by the velocity to the best current point. Thus, Equation (1b) is replaced by the following equation:

$$v_{t+1,i} = \omega \cdot v_{t,i} + c_1 \cdot (P^*_{t,i} - X_{t,i}) + c_2 \cdot (SCP^*_t - X_{t,i}) \tag{1e}$$

where SCP^*_t is the best point of the swarm at iteration *t*.

We propose a discrete version (that we call *PSOCB*) in which the parameter values (the size of the swarm and the coefficients ω, c_1, and c_2) are preset.

4.5. A PSO algorithm with passive congregation (PSOPC)

He et al. (2004) developed a PSO called PSOPC which models passive congregation behaviour. In PSOPC, Equation (1b) is changed to the following equation:

$$v_{t+1,i} = \omega \cdot v_{t,i} + c_1 \cdot (P^*_{t,i} - X_{t,i}) + c_2 \cdot (S^*_t - X_{t,i}) + c_3 \cdot (P^R_{t,i} - X_{t,i}) \tag{1f}$$

where $P^R_{t,i}$ is the point of a particle selected at random for the swarm for the particle *i* at iteration *t*. Note that the fourth terms of Equation (1d) (*PSO-c₃dyn*) and Equation (1f) (*PSOPC*) are different: Equation (1f) uses a point of an existing particle ($P^R_{t,i}$), whereas Equation (1d) uses a randomly generated point ($R_{t,i}$).

He et al. (2004) used an adaptive parameter control according to the current iteration of the algorithm to dynamically set the values of the coefficients ω and c_3: ω decreases linearly and c_3 increases linearly when the number of iterations increases. We adopt the same strategy in the *PSOPC* algorithm.

Thus, the parameter values that need to be preset are the size of the swarm, the coefficients c_1 and c_2, the starting values of ω and c_3 and their deceasing and increasing slopes.

4.6. A PSO algorithm with dissipative velocity (DPSOvel)

Xie et al. (2002) added the following equation to the classical PSO algorithm which is applied after Equations (1b) and (2) to introduce additional diversity into the velocity of the particles:

$$\text{IF } (rand() < c_v) \text{ THEN } v_{t,i}^{d} = rand() * v_{max,d} \tag{3a}$$

where c_v is a factor between 0 and 1, $v_{t,i}^{d}$ is the component *d* of the velocity $v_{t,i}$, *rand*() is a random value between 0 and 1 and $v_{max,d}$ is an upper bound of the component *d* of the velocity.

This PSO algorithm was developed to solve continuous optimization problems, so the way of introducing additional diversity has to be adapted to our discrete PSO algorithm (*DPSOvel*). Each movement of a velocity (which is a list of movements) has a probability c_v of been swapped with another, randomly selected movement.

The parameter values (the size of the swarm, the coefficients ω, c_1, and c_2 and the factor c_v) are preset.

4.7. A PSO algorithm with dissipative point (DPSOpoi)

Xie et al. (2002) added the following equation to the classical PSO algorithm which is applied after Equations (1b) and (2) to introduce additional diversity into the point of the particles:

$$\text{IF } (rand() < c_p) \text{ THEN } x_{t,i}^{d} = Random\left(l_d, u_d\right) \tag{3b}$$

where c_p is a factor between 0 and 1, $x_{t,i}^{d}$ is the component *d* of the point $X_{t,i}$, $Random\left(l_d, u_d\right)$ is a random value between l_d and u_d, l_d is a lower bound of the component *d* of the point and u_d is an upper bound of the component *d* of the point.

This PSO algorithm was developed to solve continuous optimization problems, so the way of introducing additional diversity has to be adapted to our discrete PSO algorithm (*DPSOpoi*). For each position of the point (which is a sequence), the position has a probability c_p of being swapped with another, randomly selected position.

The parameters values (the size of the swarm, the coefficients ω, c_1, and c_2 and the factor c_p) are preset.

4.8. A PSO algorithm with turbulence (PSO-$c_3$stat)

Fieldsend and Singh (2002) introduced a random velocity (that they call *turbulence*) into the classical PSO for solving continuous optimization problems. In this PSO algorithm, Equation (1b) is replaced by the following equation:

$$v_{t+1,i} = \omega \cdot v_{t,i} + c_1 \cdot (P^*_{t,i} - X_{t,i}) + c_2 \cdot (S_t^* - X_{t,i}) + c_3 \cdot RV_{t,i} \tag{1g}$$

where $RV_{t,i}$ is a random velocity generated for the particle *i* at iteration *t*.

In our adaptation we propose to generate the random velocity so that the particle moves to a randomly generated point, i.e. our proposal to introduce diversity into the swarm (see Section 2.2). The value of c_3 is static in the Fieldsend and Singh algorithm, so we call our adaptation *PSO-$c_3$stat*.

The parameters values of *PSO-$c_3$stat* (the size of the swarm, the coefficients ω, c_1, c_2, and c_3) are preset.

4.9. A PSO-$c_3$dyn variant (PSO-$c_3$dyn')

We propose to vary *PSO-$c_3$dyn* (see Section 4.3) by introducing diversity with passive congregation (i.e. as in *PSOPC* (see Section 4.5)) instead of random velocity. We call this variant *PSO-$c_3$dyn'*.

4.10. A PSOPC variant (PSOPC')

We propose to vary *PSOPC* (see Section 4.5) by introducing diversity with random velocity (i.e. as in *PSO-$c_3$dyn* (see Section 4.3)) instead of passive congregation. We call this variant *PSOPC'*.

4.11. A DPSOpoi variant (DPSOpoi-$c_p$dyn)

We propose to vary *DPSOpoi* (see Section 4.7) by using our adaptive control based on the population heterogeneity to dynamically change the value of the parameter c_p (i.e., the diversity control mechanism used in *PSO-$c_3$dyn* (see Section 4.3)).

4.12. Fine-tuning of the PSO parameters

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although the parameter values are extremely important because the results of the metaheuristic for each problem are very sensitive to them, the selection of parameter values is commonly justified in one of the following ways (Eiben et al., 1999; Adenso-Díaz and Laguna, 2006): 1) "by hand" on the basis of a small number of experiments that are not specifically referenced; 2) by using the general values recommended for a wide range of problems; 3) by using the values reported to be effective in other similar problems; or 4) by choosing values without any explanation.

Adenso-Díaz and Laguna (2006) proposed a new technique called CALIBRA for fine-tuning the parameters of heuristic and metaheuristic algorithms. CALIBRA is based on Taguchi's fractional factorial experimental designs coupled with a local search procedure. CALIBRA has been chosen for fine-tuning the PSO parameters using a set of 60 representative training instances (generated as explained in Section 5).

García et al. (2006) used CALIBRA to fine-tune *PSO-M1F*. The same technique is also used to fine-tune the other proposed PSO algorithms. The following parameter values are obtained:

1. *PSO-M1F*: size of the swarm = 21, $\omega = 0.87$, $c_1 = 0.87$ and $c_2 = 0.37$
2. *CPSO*: size of the swarm = 75, $\chi = 0.75$, $c_1 = 0.75$ and $c_2 = 0.75$
3. *PSO-$c_3$dyn*: size of the swarm = 13, $\omega = 0.87$, $c_1 = 0.87$, $c_2 = 0.87$ and $K = 3.80$
4. *PSOCB*: size of the swarm = 25, $\omega = 0.87$, $c_1 = 0.75$ and $c_2 = 0.37$
5. *PSOPC*: size of the swarm = 25, $\omega = 0.63$, $c_1 = 0.37$, $c_2 = 0.75$, $c_3 = 0.63$, ω slope = -0.00250 and c_3 slope = 0.00437
6. *DPSOvel*: size of the swarm = 25, $\omega = 0.37$, $c_1 = 0.87$, $c_2 = 0.25$ and $c_v = 0.087$
7. *DPSOpoi*: size of the swarm = 25, $\omega = 0.25$, $c_1 = 0.75$, $c_2 = 0.25$ and $c_p = 0.050$
8. *PSO-$c_3$stat*: size of the swarm = 8, $\omega = 0.13$, $c_1 = 0.75$, $c_2 = 0.87$ and $c_3 = 0.25$
9. *PSO-$c_3$dyn'*: size of the swarm = 13, $\omega = 0.87$, $c_1 = 0.25$, $c_2 = 0.87$ and $K = 6.30$
10. *PSOPC'*: size of the swarm = 87, $\omega = 0.87$, $c_1 = 0.87$, $c_2 = 0.87$, $c_3 = 0.25$, ω slope = -0.00625 and c_3 slope = 0.00125
11. *DPSOpoi-$c_p$dyn*: size of the swarm = 13, $\omega = 0.75$, $c_1 = 0.13$, $c_2 = 0.75$ and $K = 8.70$

Sine CALIBRA cannot fine-tune more than five parameters, *PSOPC* and *PSOPC'* (which have seven parameters) are fine-tuned in two steps. In the first step, the initial values of ω and c_3 are set to 0.9 and 0.4, respectively (as in He et al. (2004)) and the remaining parameters (the size of the swarm, c_1, c_2, the ω slope and the c_3 slope) are fine-tuned. In the second step the values of c_1 and c_2 are set at the values obtained in the first step and the remaining parameters (the size of the swarm, ω, c_3, the ω slope and the c_3 slope) are fine-tuned.

5. Computational experiment

The computational experiment for the eleven proposed PSO algorithms was carried out for the same instances and conditions used in García et al. (2006). That is, the algorithms ran 740 instances which were grouped into four classes (185 instances in each class) according to their size. The instances in the first class (*CAT1*) were generated using a random value of D (number of units) uniformly distributed between 25 and 50, and a random value of p (number of products) uniformly distributed between 3 and 15; for the second class (*CAT2*), D was between 50 and 100 and p between 3 and 30; for the third class (*CAT3*), D was between 100 and 200 and p between 3 and 65; and for the fourth class (*CAT4*), D was between 200 and 500 and p between 3 and 150. For all instances and for each type of product $i = 1,\ldots,p$, a random value of d_i (number of units of the product i) was between 1 and $\left|\frac{D-p+1}{2.5}\right|$ such that $\sum_{i=1}^{p} d_i = D$. All algorithms were coded in Java and the computational experiment was carried out using a 3.4 GHz Pentium IV with 512 Mb of RAM.

The eleven algorithms were run for 50 seconds for each instance. Table 1 shows the averages of the RTV values to be minimized for the global of 740 instances and for each class of instances (*CAT1* to *CAT4*) obtained with the PSO algorithms.

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
DPSOpoi-c_p*dyn*	**4,625.54**	16.42	51.34	610.34	17,824.04
PSO-c_3*dyn*	**6,986.05**	15.72	57.10	1,261.81	26,609.56
PSOCB	**8,316.51**	73.79	433.98	3,106.96	29,651.33
PSO-M1F	**8,502.83**	66.44	424.60	3,000.52	30,519.76
DPSOvel	**8,686.47**	19.28	179.60	2,287.05	32,259.96
CPSO	**8,774.06**	74.51	478.13	3,478.72	31,064.89
DPSOpoi	**8,792.70**	17.14	50.50	810.58	34,292.58
PSO-c_3*dyn'*	**11,133.09**	146.77	804.12	5,251.08	38,330.39
PSOPC	**14,579.82**	82.03	563.05	4,021.67	53,652.54
PSO-c_3*stat*	**18,707.12**	40.41	853.26	7,959.23	65,975.58
PSOPC'	**19,626.03**	145.26	1,178.29	9,086.24	68,094.33

Table 1. Averages of the RTV values for 50 seconds

Table 1 shows that for the global of all instances, the results of *PSO-M1F*, which was until now the best method proposed in the literature for solving the RTVP, are improved on by the results of three PSO algorithms: *DPSOpoi-*c_p*dyn* is 45.60% better, *PSO-*c_3*dyn* is 17.84% better and *PSOCB* is 2.19% better. There are three algorithms that obtain slightly poorer results than *PSO-M1F* (*DPSOvel*, *CPSO* and *DPSOpoi*) and the four remaining algorithms provide much poorer results (*PSO-*c_3*dyn'*, *PSOPC*, *PSO-*c_3*stat* and *PSOPC'*). If we consider the results according to class, the three best algorithms for *CAT1* instances are *PSO-*c_3*dyn*, *DPSOpoi-*c_p*dyn* and *DPSOpoi* (76.34%, 75.29% and 74.20% better than *PSO-M1F*, respectively); the three best algorithms for *CAT2* instances are *DPSOpoi*, *DPSOpoi-*c_p*dyn* and *PSO-*c_3*dyn* (88.11%, 87.91% and 86.55% better than *PSO-M1F*, respectively); the three best algorithms for *CAT3* instances are *DPSOpoi-*c_p*dyn*, *DPSOpoi* and *PSO-*c_3*dyn* (79.66%, 72.98% and 57.95% better than *PSO-M1F*, respectively); and the three best algorithms for *CAT4* instances are *DPSOpoi-*c_p*dyn*, *PSO-*c_3*dyn* and *PSOCB* (41.60%, 12.81% and 2.84% better than *PSO-M1F*, respectively).

The results in Table 1 show that our proposed mechanism for controlling the degree of diversity introduced into the swarm is very effective. The best and second best results are obtained with two PSO algorithms that apply our control mechanism of diversity (*DPSOpoi-*c_p*dyn* and *PSO-*c_3*dyn*, respectively). *DPSOpoi-*c_p*dyn* produces an improvement of 47.39% with respect to *DPSOpoi*, which is its static version, i.e. the degree of diversity introduced into the swarm is preset and is not changed by the proposed control mechanism. In addition, *PSO-*c_3*dyn* produces an improvement of 62.66% with respect to *PSO-*c_3*stat*, which is its static version. Our proposal of introducing diversity through a random velocity works well when the degree of diversity is dynamically controlled according to our proposal.

In Table 2 we compare the number of times that each PSO algorithm reaches the best RTV value obtained with all eleven algorithms. The results are shown for the 740 instances overall and for each class of instances.

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
DPSOpoi-$c_p$dyn	**468**	71	70	145	182
PSO-$c_3$dyn	**142**	88	52	1	1
PSOCB	**1**	0	0	0	1
PSO-M1F	**0**	0	0	0	0
DPSOvel	**60**	50	9	0	1
CPSO	**1**	1	0	0	0
DPSOpoi	**179**	66	74	39	0
PSO-$c_3$dyn'	**0**	0	0	0	0
PSOPC	**0**	0	0	0	0
PSO-$c_3$stat	**46**	46	0	0	0
PSOPC'	**0**	0	0	0	0

Table 2. Number of times that the best solution is reached

As we expect from the results in Table 1, Table 2 shows that *DPSOpoi-$c_p$dyn* reaches the best solution the greatest number of times (in 63.24% for the global of all instances). Surprisingly, although *DPSOpoi* is the seventh best algorithm according to the obtained RTV values, it is the algorithm that reaches the best solution the second-highest number of times (in 24.19% for the global of all instances) followed by *PSO-$c_3$dyn* (in 19.19% for the global of all instances). In addition, *DPSOvel* and *PSO-$c_3$stat* reach the best solution the fourth- and the fifth-highest number of times, respectively.

To complete the analysis of the results, their dispersion is observed. We use the following expression to define a measure of the dispersion (let it be called σ) of the RTV values obtained by each algorithm, *alg*, for a given instance, *ins*:

$$\sigma(alg, ins) = \left(\frac{\mathrm{RTV}_{ins}^{(alg)} - \mathrm{RTV}_{ins}^{(best)}}{\mathrm{RTV}_{ins}^{(best)}} \right)^2 \qquad (4)$$

where $\mathrm{RTV}_{ins}^{(alg)}$ is the RTV value of the solution obtained with the algorithm *alg* for the instance *ins*, and $\mathrm{RTV}_{ins}^{(best)}$ is the best RTV value of the solutions obtained with the eleven algorithms for the instance *ins*. Table 3 shows the average σ dispersion for the 740 overall instances and for each class of instances.

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*	*
DPSOpoi-$c_p$dyn	**0.11**	0.29	0.16	0.01	0.00	(1)
PSO-$c_3$dyn	**0.64**	0.19	0.23	1.62	0.51	(3)
PSOCB	**72.20**	44.44	159.40	83.77	1.21	(6)
PSO-M1F	**65.63**	35.68	150.51	75.15	1.17	(5)
DPSOvel	**10.87**	0.97	18.44	22.96	1.10	(4)
CPSO	**81.87**	42.07	190.13	93.45	1.83	(7)
DPSOpoi	**0.41**	0.41	0.10	0.18	0.95	(2)
PSO-$c_3$dyn'	**313.65**	261.54	692.52	295.26	5.30	(10)
PSOPC	**123.10**	52.95	292.21	141.44	5.83	(8)
PSO-$c_3$stat	**302.44**	8.73	613.62	566.83	20.58	(9)
PSOPC'	**622.75**	199.92	1,394.34	874.23	22.52	(11)

* Order according to the average σ dispersion for the global of all instances

Table 3. Average σ dispersion with respect to the best solution found

We can see from Table 3 that *DPSOpoi-c$_p$dyn*, *DPSOpoi* and *PSO-c$_3$dyn* have the lowest average σ dispersion for the global of all instances. Therefore, the two algorithms that apply the proposed mechanism for controlling the diversity introduced (*DPSOpoi-c$_p$dyn* and *PSO-c$_3$dy*) not only produce the best RTV values but also exhibit very stable behaviour, i.e. when they do not obtain the best result for a given instance, they obtain a value that is very close to it. On the other hand, although *DPSOpoi* and *DPSOvel* produce slightly worse RTV values than *PSOCB* and *PSO-M1F*, they exhibit much more stable behaviour.

The results in Table 1 and 2 show that the main differences between the seven best PSO algorithms (*DPSOpoi-c$_p$dyn*, *PSO-c$_3$dyn*, *PSOCB*, *PSO-M1F*, *DPSOvel*, *CPSO* and *DPSOpoi*) are in the average values of the *CAT1*, *CAT2* and *CAT3* instances, whereas the average values of the *CAT4* instances are more similar (except for those obtained by *DPSOpoi-c$_p$dyn*). This may occur because, on average, some PSO algorithms may converge to a local minimum on 50 seconds for small and medium instances (*CAT1*, *CAT2* and *CAT3* instances), whereas other algorithms continue to explore the search space. However, 50 seconds might not be enough time for the PSO algorithms to converge for large instances (*CAT4* instances). Figure 3 shows how the averages of the RTV values for the global of all instances decrease over the computing time for the seven best algorithms.

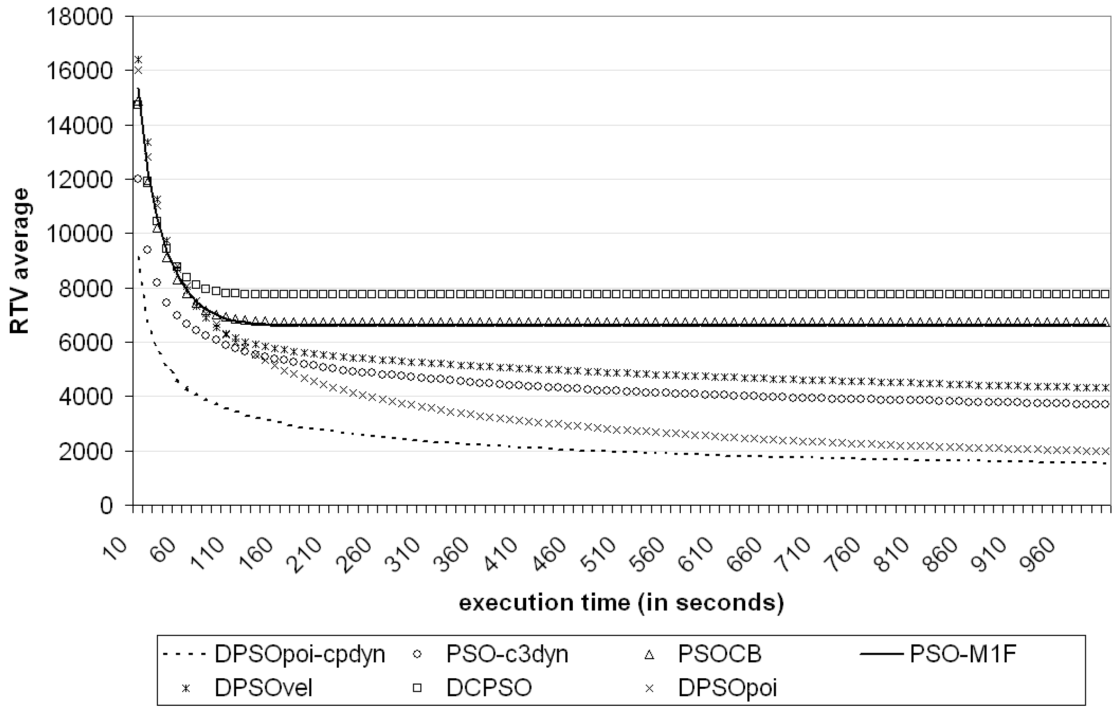


Figure 3. Average of the RTV values obtained over the computing time

Figure 3 shows that the RTV values of the seven algorithms decrease exponentially during the first 110 seconds of computing time. At this point, the three algorithms into which no diversity is introduced (*PSOCB*, *PSO-M1F* and *CPSO*) have converged for almost all the instances and remain trapped in a local optimum during the remaining computing time. In contrast, the other four algorithms (*DPSOpoi-c$_p$dyn*, *PSO-c$_3$dyn*, *DPSOvel* and *DPSOpoi*) introduce diversity and are able to move away from the local optimum. Therefore, these four algorithms continue to explore new regions of the search space and find better local optima during the remaining computing time.

Table 4 shows the averages of the RTV values for the global of all instances and for each class of instances (*CAT1* to *CAT4*) obtained with the eleven PSO algorithms when they are run for 1,000 seconds.

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
DPSOpoi-$c_p$dyn	**1,537.34**	14.35	46.55	143.95	5,944.51
DPSOpoi	**1,980.20**	14.63	46.13	142.58	7,717.47
PSO-$c_3$dyn	**3,696.44**	13.83	42.18	391.54	14,338.20
DPSOvel	**4,312.30**	17.75	84.17	1,036.87	16,110.42
PSO-M1F	**6,619.34**	66.44	424.54	3,000.51	22,985.85
PSOCB	**6,731.24**	73.79	433.98	3,106.96	23,310.24
CPSO	**7,746.85**	74.51	478.13	3,478.72	26,956.02
PSOPC	**8,838.70**	82.03	563.05	4,021.67	30,688.03
PSO-$c_3$dyn’	**11,133.09**	146.77	804.12	5,251.08	38,330.39
PSO-$c_3$stat	**16,212.08**	16.75	592.64	6,520.72	57,718.22
PSOPC’	**18,495.01**	138.76	1,056.59	8,414.15	64,370.53

Table 4. Averages of the RTV obtained values for 1,000 seconds

Table 4 shows that *DPSOpoi-$c_p$dyn* also produces the best average for the global of all instances when the algorithms are run for 1,000 seconds. However, the second best algorithm is now DPSO*poi*, followed by *PSO-$c_3$dyn*. We can see from Figure 3 that the results obtained by *PSO-$c_3$dyn* are better than those obtained by DPSO*poi* up to an execution time of 130 seconds. Thus, it is recommended to use *PSO-$c_3$dyn* instead of DPSO*poi* for execution times of less than two minutes. Anyway, Figure 3 shows that it is always advisable to use *DPSOpoi-$c_p$dyn*, in which the way of introducing diversity into the swarm is based on one proposal by Xie et al (2002) and the degree of diversity is regulated by the control mechanism that we propose in this paper.

There are four algorithms that improve on the results obtained by *PSO-M1F* when the total computation time equals to 1,000 seconds: *DPSOpoi-$c_p$dyn* (76.78% better), *DPSOpoi* (70.08% better), *PSO-$c_3$dyn* (44.16% better) and *DPSOvel* (34.85% better). On the other hand, the poor results shown in Tables 1 and 3 for *PSOPC* and *PSO-$c_3$dyn’* show that proposal of using passive congregation to introduce diversity into the swarm for a discrete optimization problem such as the RTVP is unsuccessful.

6. Conclusions and future research

PSO is an evolutionary metaheuristic proposed by Kennedy and Eberhart (1995) which has achieved good results for several types of problems. The metaheuristic uses an analogy of the flocking behaviour of birds to look for the optimal solution. PSO has the desirable property of convergence, but this convergence can be premature, in which case certain regions of the search space remain unexplored. To prevent this, several strategies have been proposed in the literature, some of which are based on introducing diversity into the swarm. Most published PSO applications are designed to solve continuous optimization problems, but there are few PSO applications for solving discrete optimization problems.

We propose a PSO algorithm (which we call *PSO-$c_3$dyn*) for solving a discrete optimization problem. To develop *PSO-$c_3$dyn*, two novel ideas are introduced: 1) a new way of introducing diversity into the swarm, based on introducing a random velocity; and 2) a mechanism for controlling the degree of diversity introduced into the swarm that takes feedback from the heterogeneity of the swarm.

The algorithm is designed to solve the RTVP, which is a scheduling problem that occurs whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the instants at which they receive the necessary resources is minimized. The *PSO-M1F* algorithm proposed by García et al. (2006) is currently the best published algorithm for solving non-small instances of the RTVP.

We propose nine more PSO algorithms (in addition to *PSO-M1F*) for comparison with *PSO-$c_3$dyn* which are used to test the way of introducing diversity and the control mechanism of the diversity that we propose. Eight of these nine algorithms are based on PSO algorithms that have been tested in continuous optimization problems but not in discrete optimization problems. Therefore, this paper tests the validity of these algorithms when applied to discrete optimization problems such as the RTVP. In addition, we demonstrate that CALIBRA is very useful for fine-tuning the parameters of the PSO algorithms.

The results of the computational experiment show that the proposed mechanism for controlling the degree of diversity introduced into the swarm works very well. The best results obtained, on average, are obtained under this diversity control. Thus, this paper presents a control mechanism that is very effective and contributes to improve the performance of PSO. Moreover, in this paper is improved the previous results published in the literature of the RTVP.

The two novel ideas proposed in our paper (random velocity and diversity control) are very easy to adapt to a continuous space. Therefore, our future research will focus on testing the proposed diversity control in PSO algorithms for solving continuous optimization problems and determining whether it improves the overall results. Since the proposed way of introducing diversity by using a random velocity works correctly, we will also test it in continuous optimization problems.

ACKNOWLEDGEMENTS

The authors are very grateful to Professor Albert Corominas (Technical University of Catalonia) and to the anonymous reviews for his valuable comments which have helped to enhance this paper. The authors gratefully acknowledge the support of grant DPI2007-61905 (Ministerio de Educación y Ciencia, Spain, and FEDER).

REFERENCES

Adenso-Díaz, B. and Laguna, M. (2006) 'Fine-tuning of algorithms using fractional experimental designs and local search', *Operations Research,* Vol. 54, pp. 99-114.

Andrés, C., Pastor, R. and Framiñán, J.M. (2004) ‘Optimización mediante cúmulo de partículas del problema de secuenciación CONWIP’, *Eighteenth Conference on Statistics and Operations Research SEIO’04*, Cádiz, Spain.

Angeline, P.J. (1996) ‘Two self-adaptive crossover operators for genetic programming’, *Advances in Genetic Programming*, Vol. 2, pp. 89-109.

Angeline, P.J. (1998) ‘Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences’, In *Proceeding s of evolutionary programming VII*, pp. 601-610.

Anily, S., Glass, C.A. and Hassin, R. (1998) ‘The scheduling of maintenance service’, *Discrete Applied Mathematics*, Vol. 82, pp. 27-42.

Blackwell, T. and Bentley, P.J. (2002) ‘Don’t push me! Collision-avoiding swarms’, In *Proceedings of the IEEE congress on evolutionary computation*, pp. 1691-1696.

Chau, K.W. (2006) ‘Particle swarm optimization training algorithm for ANNs in stage prediction of Shing Mun River’, *Journal of Hydrology*, Vol. 329, pp. 363-367.

Clerc, M. (2004) ‘Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem’, *New Optimization Techniques in Engineering,* Springer, pp. 219-239.

Clerc, M. (2006) ‘Particle Swarm Optimization’, *ISTE*.

Clerc, M. and Kennedy, J. (2002) ‘The particle swarm: explosion, stability, and convergence in a multidimensional complex space’, *IEEE Transactions on Evolutionary Computation*, Vol. 6 (1), pp. 58-73.

Corominas, A., Kubiak, W. and Moreno, N. (2007) ‘Response time variability’, *Journal of Scheduling*, Vol. 10, pp. 97-110.

Corominas, A., Kubiak, W. and Pastor, R. (2006) ‘Solving the Response Time Variability Problem (RTVP) by means of mathematical programming’, *Working paper IOC-DT*, Universistat Politècnica de Catalunya, Spain.

Dong, L., Melhem, R. and Mossel, D. (1998) ‘Time slot allocation for real-time messages with negotiable distance constraint requirements’, *Real-time Technology and Application Symposium,* RTAS, Denver.

Eberhart, R.C. and Shi, Y. (2001) ‘Comparing inertia weights and constriction factors in particle swarm optimization’, In *IEEE International Conference on Evolutionary Computation,* pp. 81-86.

Eiben, A.E., Hinterding, R. and Michalewicz, Z. (1999) ‘Parameter control in evolutionay algorithms’, *IEEE Transactions on evolutionary computation*, Vol. 3, pp. 124-141.

Fieldsend, J.E. and Singh, S. (2002) ‘A Multi-Objective Algorithm based upon Particle Swarm Optimisation, an Efficient Data Structure and Turbulence’, In *Proceedings of U.K. Workshop on Computational Intelligence (UKCI’02),* United Kingdom, pp. 37-44.

García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics’, *Artificial Intelligence Research and Development*, Vol. 146, pp. 187-194.

Geethanjali, M., Mary Raja Slochanal, S. and Bhavani, R. (2007) ‘PSO trained ANN-based differential protection scheme for power transformers’, *Neurocomputing,* In Press, Corrected Proof.

He, S., Wu, Q.H., Wen J.Y., Saunders, J.R. and Paton, R.C. (2004) ‘A particle swarm optimizer with passive congregation’, *Biosystems*, Vol. 78, pp. 135-147.

Hendtlass, T. (2001) ‘A combined swarm differential evolution algorithm for optimization problems’, *Lecture Notes in Computer Science*, Vol. 2070, pp. 11-18.

Hinterding, R., Michalewicz, Z. and Peachey, T.C. (1996) ‘Self-adaptive genetic algorithm for numeric functions’, *Proc. 4th Conf. Parallel Problem Solving from Nature,* Lecture Notes in Computer Science, Vol. 1141, pp. 420-429.

Hu, X. and Eberhart, R.C. (2001) ‘Tracking dynamic systems with PSO: where’s the cheese?’, In *Proceedings of the workshop on particle swarm optimization*, Indianapolis, USA.

Hu, X. and Eberhart, R.C. (2002) ‘Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization’, In *IEEE Congress on Evolutionay Computation*, Honolulu, USA.

Hu, X., Eberhart, R.C. and Shi, Y. (2004) ‘Recent advances in particle swarm’, *IEEE Congress on Evolutionary Computation*, Portland, Oregon, USA.

Kennedy, J. and Eberhart, R.C. (1995) ‘Particle swarm optimization’, In *IEEE International Conference on Neural Networks,* Australia, pp. 1942-1948.

Liang, J.J., and Suganthan, P.N. (2005) ‘Dynamic multiswarm particle swarm optimizer (DMS-PSO)’, *Proceedings SIS 2005 IEEE swarm intelligence*, pp. 124-129.

Liao, C.J., Tseng, C.T. and Luarn, P. (2007) ‘A discrete version of particle swarm optimization for flowshop scheduling problems’, *Computer & Operations Research,* Vol. 34, pp. 3099-3111.

Loøvbjerg, M. and Krink, T. (2002) ‘Extending particle swarms with self-.organized criticality’, In *Proceedings of the IEEE congress on evolutionary computation*, pp. 1588-1593.

Mendes, R. (2004) ‘Population topologies and their influence in particle swarm performance’, *PhD thesis,* Departamento de Informatica, Escola de Engenharia, Universidade do Minho.

Monden, Y. (1983) ‘Toyota Production Systems’, *Industrial Engineering and Management Press,* Norcross, GA.

Pan, Q-K., Tasgetiren, M.F. and Liang, Y-C. (2007) ‘A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem’, *Computers & Operations Research,* In press, Corrected Proof.

Parrish, J.K. and Hamner, W.M. (1997) ‘Animal Groups in Three Dimensions’, In *Cambridge University Press,* Cambridge, United Kingdom.

Parsopoulos, K.E. and Vrahatis, M.N. (2001) ‘Particle swarm optimizer in noisy and continuously changing environments’, *Artificial intelligence and soft computing,* Ed. M.H. Hamza, pp. 289-294.

Parsopoulos, K.E. and Vrahatis, M.N. (2004) ‘On the computation of all blobal minimizers thorugh particle swarm optimization’, In *IEEE Transactions on Evolutionary Computation,* Vol. 8, pp. 211-224.

Poli, R., Kennedy, J, and Blackwell, T. (2007) ‘Particle swarm optimization. An overview’, *Swarm Intelligence,* Springer, Chapter 5.

Robinson, J., Sinton, S. and Rahmat-Samii, Y. (2002) ‘Particle swarm, genetic algorithm, and their hybrids: optimization of a profiled corrugated horn antenna’, In *IEEE swarm intelligence symposium,* pp. 314-317.

Secrest, B. (2001) ‘Travelling salesman problem for surveillance mission using PSO’, *PhD thesis*, Air Force Institute of Technology, Ohio, USA.

Shi, Y. and Eberhart, R.C. (1998a) ‘Parameter selection in particle swarm optimization’, In *Proceedings 7th Annual Conference on Evolutionary Programming,* pp. 591-600.

Shi, Y. and Eberhart, R.C. (1998b) ‘A modified particle swarm optimizer’, In *IEEE international conference of Evolutionary Computation,* pp. 69-73.

Tasgetiren, M.F., Liang, Y-C., Sevkli, M. and Gencyilmaz, G. (2007) ‘A particle swarm optimization algorithm for makespan and total flowtime minmization in permutation flowshop sequencing problem', *European Journal of Operational Research*, Vol. 177, pp. 1930-1947.

Trelea, I.C. (2003) 'The particle swarm optimization algorithm: convergence analysis and parameter selection', *Information Processing Letters*, Vol. 85, pp. 317-325.

Waldspurger, C.A. and Weilh, W.E. (1995) 'Stride Scheduling: Deterministic Proportional-Share Resource Management', *Technical Memorandum MIT/LCS/TM-528*, MIT, Laboratory for Computer Science, Cambridge.

Xie, X.F., Zhang, W.J. and Yang, Z.L. (2002) 'A Dissipative Particle Swarm Optmization', In *IEEE Congress on Evolutionary Computation (CEC'02)*, Hawai, USA.

Yin, P.-Y., Yu, S.-S., Wang, P.-P. and Wang, Y.-T. (2007) 'Multi-objective task allocation in distributed computing systems by hybrid particle swarm optimization', *Applied Mathematics and Computation*, Vol. 184, pp. 407-420.

Zhang, W., Liu, Y. and Clerc, M. (2003) 'An adaptive PSO algorithm for reactive power optimization', 6th International Conference on Advances in Power Control, Operation and Management, Hong Kong.

Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism

Article published as [International Journal of Production Research, First published on 11 December 2009] [DOI: http://dx.doi.org/10.1080/00207540902862545] © [copyright Taylor & Francis Group]

Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism

Alberto García-Villoria and Rafael Pastor

Institute of Industrial and Control Engineering (IOC), Department of Management, Universitat Politècnica de Catalunya (UPC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain

The Response Time Variability Problem (RTVP) is an NP-hard combinatorial scheduling problem that has been recently formalised in the literature. The RTVP has a wide range of real-life applications such as in the automobile industry, when models to be produced on a mixed-model assembly line have to be sequenced under a just-in-time production. The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources. In two previous studies, three metaheuristic algorithms (a multi-start, a GRASP and a PSO algorithm) were proposed to solve the RTVP. We propose solving the RTVP by means of the electromagnetism-like mechanism (EM) metaheuristic algorithm. The EM algorithm is based on an analogy with the attraction-repulsion mechanism of the electromagnetism theory, where solutions are moved according to their associated charges. In this paper we compare the proposed EM metaheuristic procedure with the three metaheuristic algorithms aforementioned and it is shown that, on average, the EM procedure improves strongly on the obtained results.

Keywords: response time variability; fair sequences; scheduling; just-in-time; metaheuristics; electromagnetism-like mechanism

1. Introduction

The Response Time Variability Problem (RTVP) is a scheduling problem that has recently been defined in the literature (Corominas *et al*. 2007). The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources. Although this combinatorial optimisation problem is easy to formulate, it is very difficult to solve (it is NP-hard, Corominas *et al*. 2007).

The RTVP has a broad range of real-life applications. For example, it can be used to sequence regularly models in the automobile industry (Ding and He 2007), to resource allocation in computer multi-threaded systems and network servers (Waldspurger and Weihl 1995), to broadcast video and sound data frames of applications over asynchronous transfer mode networks (Dong *et al*. 1998), in the periodic machine maintenance problem when the distances between consecutive services of the same machine are equal (Anily *et al*. 1998) and in the collection of waste (Herrmann 2007).

One of the first problems in which has appeared the importance of sequencing *regularly* is at the sequencing on the mixed-model assembly production lines at Toyota Motor Corporation under the just-in-time (JIT) production system. One of the most important JIT objectives is to get rid of all kinds of waste and inefficiency and, according to Toyota, the main waste is due to the stocks. To reduce the stock, JIT production systems require to producing only the necessary models in the necessary quantities at the necessary time (Aigbedo 2004). To achieve this, one main goal, as Monden (1983) says, is scheduling the units to be produced to keep a constant consumption rates of the components involved in the production process. Miltenburg (1989) deals with this scheduling problem considering only the demand rates of the models. He proposed four metrics to measure the regularity of a sequence based on the discrepancies, for each model, between the real production rate and the ideal one (i.e., the one that would correspond to a constant rate of production). This problem is known as the Product Rate Variation (PRV) problem (Kubiak 1993). The PRV problem has been reformulated by Kubiak and Sethi (1994) as an assignment problem and, therefore, it can be solved efficiently.

Although the sequencing on the mixed-model assembly production lines is usually considered in the literature as a PRV problem (Miltenburg 1989, Kubiak 1993, Steiner and Yeomans 1993), in our experience with practitioners of manufacturing industries we noticed that they usually refer to a good mixed-model sequence not in terms of ideal production, but in terms of having distances between the units for the same model as regular as possible. Therefore, the metric used in the RTVP reflects the way in which practitioners refer to a desirable regular sequence.

In this paper, the electromagnetism-like mechanism (EM) metaheuristic is proposed to solve the RTVP. EM is a recent population-based metaheuristic that was first proposed by Birbil and Fang (2003). It is based on an analogy with the attraction-repulsion mechanism of electromagnetism theory. Each solution is considered as a point with an electrical charge that is measured by the objective function. This charge determines the magnitude of attraction or repulsion of the other points for applying the electromagnetism equations and EM iteratively calculates the movement of the points.

The EM metaheuristic has yielded good results when it has been used to solve several combinatorial optimisation problems (Debels and Vanhoucke 2006, Debels et al. 2006, Yuan *et al*. 2006, Maenhout and Vanhoucke 2007, Chang et al. 2009). The EM algorithm proposed to solve the RTVP is compared with efficient procedures for solving non-small instances: the multi-start and the GRASP algorithms proposed in García *et al*. (2006) and the PSO algorithm called *DPSOpoi-*c_p*dyn* proposed in García-Villoria and Pastor (2009).

The remainder of this paper is organized as follows. Section 2 presents a formal definition of the RTVP. Section 3 briefly exposes three metaheuristic algorithms presented in García *et al*. (2006) and García-Villoria and Pastor (2009). Section 4 describes the basic scheme of the EM. Section 5 proposes a procedure an EM algorithm for solving the RTVP. Section 6 provides the computational experiments and the comparison with the other metaheuristic algorithms. Finally, some conclusions and suggestions for future research are given in Section 7.

2. The Response Time Variability Problem (RTVP)

The aim of the Response Time Variability Problem (RTVP) is to minimise variability in the distances between any two consecutive units of the same model to be scheduled.

The RTVP is formulated as follows. Let *n* be the number of models, d_i the number of units to be scheduled of model *i* (*i* = 1,…,*n*) and *D* the total number of units $\left(D = \sum_{i=1..n} d_i\right)$. Let *s* be a solution of a RTVP instance that consists of a circular sequence of units ($s = s_1 s_2 \ldots s_D$), where s_j is the unit sequenced in position *j* of sequence *s*. For all unit *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the units *k* + 1 and *k* of the model *i* are found (i.e. the number of positions between them, where the distance between two consecutive positions is considered equal to 1). Since the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_i}^i$ is the distance between the first unit of the model *i* in a cycle and the last unit of the same model in the preceding cycle. Let $\bar{t}_i$ be the average distance between two consecutive units of the model *i* ($\bar{t}_i = D/d_i$). For all symbol *i* in which $d_i = 1$, t_1^i is equal to $\bar{t}_i$. The objective is to minimize the metric Response Time Variability (RTV) which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2 \tag{1}$$

For example, let $n = 3$, $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\bar{t}_C = 2$. Any sequence is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a solution, where

$$RTV = \left((5-4)^2 + (3-4)^2\right) + \left((2-4)^2 + (6-4)^2\right) + \left((2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2\right) = 1$$

3. Three metaheuristic algorithms for the RTVP

Corominas *et al*. (2007) proposed a mixed integer lineal programming (MILP) model to solve the RTVP. Corominas *et al*. (2006) improved the previous MILP model but the practical limit to obtain optimal solutions is 40 units to be scheduled. Thus, the use of heuristic or metaheuristic methods for solving real-life instances of the RTVP is justified. Corominas *et al*. (2007) proposed five heuristic algorithms. García *et al*. (2006) proposed six metaheuristic algorithms: a multi-start, a GRASP (Greedy Randomized Adaptive Search Procedure) and four PSO (Particle Swarm Optimisation) algorithms. Eleven variants of the PSO metaheuristic were also used to solve the RTVP in García-Villoria and Pastor (2009).

The three most effective aforementioned procedures are the multi-start and the GRASP algorithm proposed in García *et al*. (2006) and the PSO algorithm called *DPSOpoi-c_pdyn* proposed in García-Villoria and Pastor (2009). Next, the algorithms are briefly explained (for more details of the three algorithm procedures, see García *et al*. 2006 and García-Villoria and Pastor 2009).

3.1. The Multi-start algorithm

The multi-start method is based on generating initial random solutions and on improving each of them to find a local optimum, which is usually done by means of a local search procedure (Martí 2003). Random solutions are generated as follows. For each position, a model to be sequenced is randomly chosen. The probability of each model is equal to the number of units of this model that remain to be sequenced divided by the total number of units that remain to be sequenced. The local search procedure used is applied as follows. A local search is performed iteratively in a neighbourhood that is generated by interchanging each pair of two consecutive units of the sequence that represents the current solution; the best solution in the neighbourhood is chosen; the optimisation ends when no neighbouring solution is better than the current solution.

3.2. The GRASP algorithm

GRASP, designed by Feo and Resende (1989), can be considered to be a variant of the multi-start method in which the initial solutions are obtained using directed randomness. The solutions are generated by means of a greedy strategy in which random steps are added and the choice of elements to be included in the solution is adaptive. The random step in the GRASP proposed by García *et al.* (2006) consists of selecting the next model to be added to the solution; the probability of each candidate model is proportional to the value of its Webster index, which is based on the parametric method of apportionment with parameter $\delta = 1/2$ (Balinski and Young 1982). The Webster index for model i (i = 1,…,n) is evaluated as $\frac{d_i}{\left(x_{it} + \delta\right)}$, where x_{it} is the number of units of model i in the sequence of length t = 0,…,D (assume x_{i0} = 0). The local search procedure applied to the initial solutions is the same local search as in the multi-start method.

3.3. The PSO algorithm

PSO is a population-based metaheuristic designed by Kennedy and Eberhart (1995), which is based on an analogy of the social behaviour of flocks of birds when they search for food. The population or swarm is composed of particles (birds), which have an m-dimensional real point (which represents a feasible solution) and a velocity (the movement of the point in a m-dimensional real space). The velocity of a particle is typically a combination of three kinds of velocities: 1) inertia velocity; 2) velocity to the best point found by the particle; and 3) velocity to the best point found by the swarm. These components of the particles are iteratively modified by the PSO algorithm as it looks for an optimal solution.

In the *DPSOpoi-c*$_p$*dyn* algorithm (García-Villoria and Pastor 2009), random modifications to the points of the particles are introduced. The frequency of the modifications changes dynamically according to the homogeneity of the swarm. The aim is to prevent premature convergence and to enable the PSO algorithm to escape a local optimum. Although the PSO metaheuristic was originally designed for working in a m-dimensional real space, *DPSOpoi-c*$_p$*dyn* is adapted to work with a sequence that represents the solution. In this adaptation, a point is now the sequence of units of the models that represents a solution, and the velocity is an ordered list of transformations that must be applied to the particle so that it changes from its current point to another point; each transformation consists of a pair of positions of the point (sequence) that are to be swapped. The velocity to the best point found by the particle is the list of transformations needed to obtain the best particle point from the current position; the same applies for the velocity to the best point found by the swarm.

4. The EM metaheuristic

The electromagnetism-like mechanism (EM) metaheuristic is a new population-based metaheuristic created by Birbil and Fang (2003). The EM metaheuristic has been applied successfully to the following problems: the project scheduling problem (Debels and Vanhoucke 2006; Debels et al. 2006), neural network training (Wu *et al*. 2004), the permutation flowshop scheduling problem (Yuan *et al*. 2006), the nurse scheduling problem (Maenhout and Vanhoucke 2007), the single machine scheduling problem (Chang et al. 2009) and multi-objective optimisation problems (Tsou and Kao 2008). On the other hand, in the Birbil's PhD thesis (Birbil 2002), the EM metaheuristic is compared with other methods and shown to have substantial performance.

The EM metaheuristic operates basically as follows. EM starts with an initial population of solutions that will be attracted to the deep valleys and repulsed from the steep hills (if we wish to minimise the value of the solutions). Each solution can be thought of as a particle charged according to its objective function value. Then, an analogy of the attraction-repulsion mechanism of the electromagnetism theory can be applied. Moreover, some solutions are improved by a local search.

Next, we present the framework of the EM metaheuristic; for further details, see Birbil and Fang (2003). This algorithm works with a special class of optimisation problems with bounded variables in the following form:

$$\begin{array}{rl} \text{min (max)} & f(x) \\ \text{subject to } & x \in \Re^m \mid l_j \le x_j \le u_j,\ j = 1, \ldots, m \end{array}$$

where f is the function that evaluates a point (which represents a solution), m is the dimension of the problem (in the case of the RTVP, m would be equal to D, which is the total number of units) and x_j is the coordinate of the jth dimension, which is lower bounded by l_j and upper bounded by u_j.

The EM metaheuristic is divided into four phases (which are explained in Subsections 4.1 to 4.4): 1) the initialization of the population of the points; 2) the application of the local search; 3) the calculation of the total force vector; and 4) the movement according to the total force. The pseudocode of the metaheuristic is shown in Figure 1.

Figure 1. Pseudocode of the EM metaheuristic.

```
1: P = initial population
2: while the stopping criteria is not reached do
4:        $x^{best}$ = best point of P
3:        Local search
4:        for each point $x$ do: $F_x$ = total force vector($x$, P)
5:        for each point $x$ do: Move(x, $F_x$)
6: end while
```

4.1. ***Initial population***

The metaheuristic starts generating randomly the initial population, which consists of *p* points of the feasible domain. Each coordinate of each point is uniformly distributed between their upper and lower bounds.

4.2. ***Local search***

The local search procedure provides the EM algorithm with a good balance between the exploration and exploitation of the feasible region. Birbil and Fang (2003) propose two approaches according to the points to which the local search can be applied: local search applied to all points and local search applied only to the current best point.

Local search applied to all points promotes a more meticulous examination of the region around the points. However local search applied only to the best point usually gives as good results and less time is spent on the local search.

In both cases, a simple local search is recommended rather than a powerful one because it is enough for a good convergence (Birbil and Fang 2003). The local search is not applied until a local optimal point is reached; the local search stops when a number of iterations (let it be called *lsiter*) is executed.

4.3. ***Calculation of the total force vector***

The charge of each point *x* belonging to the population *P* (let it be called q_x), which determines the intensity of attraction or repulsion of the point, changes at each iteration of the EM metaheuristic. The charge is first evaluated as follows:

$$q_x = \exp\left(-m \frac{f(x) - f(x^{best})}{\sum_{y \in P} \left(f(y) - f(x^{best}) \right)} \right) \tag{2}$$

Note that, unlike electrical charges, no signs are associated with the charges. The direction of a particular force between two points is determined when their objective values have been compared. The total force for each point belonging to the population *P* (let it be called F_x) is evaluated as follows:

$$F_x = \sum_{y \in P | y \neq x} \left\{ \begin{array}{llll} (y-x)\dfrac{q_x q_y}{\|y-x\|^2} & if & f(y) < f(x) & \text{(Attraction)} \\ (x-y)\dfrac{q_x q_y}{\|y-x\|^2} & if & f(y) \geq f(x) & \text{(Repulsion)} \end{array} \right\} \tag{3}$$

where $\|y-x\|$ is the euclidean distance between the two points.

4.4. *Movement according to the total force*

Each point *x* belonging to the population *P* is moved according to the next equation:

$$x = x + \lambda \frac{F_x}{\|F_x\|} \tag{4}$$

where λ denotes a random number uniformly distributed between 0 and 1 and $\|F_x\|$ is the norm of the force vector. The parameter λ is used to ensure that the points have a nonzero probability of moving to the unvisited regions in this direction. Furthermore, the force applied to each point is normalized, so the feasibility is maintained (i.e., each coordinate of each point will be between l_j and u_j).

5. The EM algorithm for the RTVP

The objective function and the equations of the EM metaheuristic work with points of a region of the *m*-dimensional real space. Others procedures such as PSO algorithms or other optimisation algorithms of real variables are also designed for working in an *m*-dimensional real space. However, a solution of many combinatorial optimisation problems is usually represented as an ordered sequence of integer numbers (as in the RTVP), so these metaheuristics (EM, PSO and others) are incompatible with this representation of the solution as an ordered sequence of integer numbers. There are two ways of applying algorithms of this kind to the RTVP: to adapt the algorithm to work with a sequence of integer numbers or to adapt the representation of the solution as an *m*-dimensional real point.

To adapt the PSO algorithm to a sequence of integer numbers for the RTVP is done in García-Villoria and Pastor (2009), as explained in Section 2. As would happen in the EM algorithm, this way involves redefining several mathematical operators used by the algorithm. For example, the difference between two points ($(y-x)$ and $(x-y)$ in Equation 3) would now be the difference between two sequences of integer numbers and this new different operator should be defined.

On the other hand, a sequence of integer numbers can be represented by an *m*-dimensional real point using random key representation (RK) (Bean 1994). The main advantage of using RK is that each solution corresponds to a real point, so that geometric operations (for example, the evaluation of a point charge (Equation 2)) can be performed on its components. Since geometric operations are the cornerstone of several metaheuristics (such as EM), RK allows a straightforward application of this type of metaheuristics to solve combinatorial optimisation problems. Although there is empirical evidence that Genetic Algorithms that applies RK may obtain slight worst results that Genetic Algorithms adapted to the combinatorial problem (Bean 1994), in other metaheuristics there is no conclusion about which approach is better. In the case of EM algorithms, to the best of our knowledge, most of the papers in which an EM algorithm is proposed to solve a combinatorial optimisation problem RK is used (Debels and Vanhoucke 2006; Debels et al. 2006; Yuan et al. 2006; Chang et al. 2009) and only one paper adapts EM (Maenhout and Banhoucke, 2007). Because RK have been effectively applied to the EM metaheuristic to solve several combinatorial problems, in this paper RK is also used in the EM algorithm for solving the RTVP.

Random key representation for the RTVP is explained in Subsection 4.1. How the initial population is generated is described in Subsection 4.2. The local search used in our EM algorithm is explained in Subsection 4.3. The calculations of the total force vectors and the movements according to the total force are directly implemented according to Equations (2), (3) and (4). Finally, Subsection 4.4 explains the fine-tuning of the parameter values of the EM algorithm: the size of the initial population (p) and the maximum number of iterations of the local search procedure ($lsiter$).

5.1. *Random key representation*

Random key representation (Bean 1994) consists of an m-length sequence of different real numbers called *keys*. Let the key sequence be $r = r_1, \ldots, r_m$, where r_j is the key of the position j. In the context of the proposed EM algorithm, the key sequence has D (number of units to be sequenced with d_i units of model i) keys. As the EM metaheuristic works with bounded variables, the values of the keys are bounded between 0 and 1.

Given a key sequence r, the solution s (sequence of models) that is represented by r is as follows. First, for each position $j = 1,...,D$ of r, key r_j is associated with a model. The association is done in a way that, for each model i, there are d_i consecutive keys associated with model i. For each key sequence, the association for key r_j will be always done with the same model, i.e., if, for example, key r_1 is associated with model A in every key sequence r, r_1 will be associated with this model. Next, a new key sequence, r', is obtained by putting r (and therefore their associated models) in descending order according to the values of the keys. Then, for each position $j = 1,...,D$, model s_j is the model associated with key r'_j, i.e., the model sequenced in position j is the model associated with key r_j that is in the position j in the key sequence r'.

For example, let a RTVP instance be $n = 3$, $d_A = 2$, $d_B = 2$ and $d_C = 4$. Given the key sequence $r = (0.12, 0.26, 0.67, 0.08, 0.14, 0.45, 0.87, 0.62)$, each key r_j ($j = 1,\ldots,8$) is associated with a model as follows:

models	A	A	B	B	C	C	C	C
	↑	↑	↑	↑	↑	↑	↑	↑
keys	0.12	0.26	0.67	0.08	0.14	0.45	0.87	0.62

So, the descending ordered key sequence is $r' = (0.87, 0.67, 0.62, 0.45, 0.26, 0.14, 0.12, 0.08)$ and, therefore, the solution represented is (C, B, C, C, A, C, A, B).

5.2. *Initial population*

The initial population of points consists of p solutions generated randomly. As has been introduced previously, each solution is represented by a key sequence where each key value is bounded between 0 and 1. To get a solution, we generate a random value uniformly distributed in [0,1] for each key.

5.3. *Local search*

The local search procedure used in the EM algorithm is as follows. A local search is performed iteratively in a neighbourhood that is generated by interchanging two units of different consecutive and non-consecutive models; the first solution found in the neighbourhood that is better than the current solution is selected; the optimisation ends when the maximum number of iterations is reached or no neighbouring solution is better than the current solution.

Local search applied to all points and local search applied only to the best point were tested by an initial experiment. To apply the local search only to the best point provided much better solutions for the RTVP, so the local search applied only to the best point is used in the EM algorithm.

5.4. *Fine-tuning of the EM parameters*

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although parameter values are extremely important because the results of the metaheuristic for each problem are highly sensitive to them, the selection of parameter values is commonly justified in one of the following ways (Eiben *et al*. 1999, Adenso-Díaz and Laguna 2006): 1) "by hand", on the basis of a small number of experiments that are not specifically referenced; 2) using the general values recommended for a wide range of problems; 3) using the values reported to be effective in other similar problems; or 4) choosing values without any explanation.

Adenso-Díaz and Laguna (2006) proposed a new technique called CALIBRA for fine-tuning the parameters of heuristic and metaheuristic algorithms. CALIBRA is based on Taguchi's fractional factorial experimental designs coupled with a local search procedure. The local search is applied to promising regions of the parameter values and the promising regions are found by means of Taguchi's experimental designs. One assumption of Taguchi's experimental designs is the linear interdependence between the parameters, whereas the interdependence is usually non-linear (Adenso-Díaz and Laguna 2006). CALIBRA uses the analysis of the factorial experiment results only as a guideline to narrow the search and to initiate the next round of experiments. Because the search focuses on narrower ranges for each parameter value, the linear assumption becomes less restrictive and the predicted optimal values approach to the true optimal values.

CALIBRA was chosen for fine-tuning the EM algorithm parameters. A set of 60 representative training instances, which were generated as explained in Section 5, was used. The values obtained for the size of the population (p) and the maximum number of iterations of the local search (*lsiter*) are shown in Table 1.

The GRASP and PSO algorithms (the multi-start algorithm does not have parameters) were also fine-tuned using CALIBRA and the same 60 training instances. GRASP only has one parameter, which is the size of the candidate list (*CL*). *DPSOpoi-*c_p*dyn* has five parameters: the size of the population (p), the coefficient that weights the inertia velocity (ω), the coefficient that weights the velocity to the best particle point (c_1), the coefficient that weights the velocity to the best swarm point (c_2) and the factor of the degree of the random modifications introduced (K). The parameter values obtained are shown in Table 1.

Table 1. Parameter values obtained with CALIBRA

EM	GRASP	*DPSOpoi-$c_p$dyn*
p = 25	*CL* size = 3	p = 13
lsiter = 5		ω = 0.75
		c_1 = 0.13
		c_2 = 0.75
		K = 8.70

6. Computational experiment

The computational experiment for the EM algorithm is carried out for the same instances and conditions used in García *et al.* (2006) and in García-Villoria and Pastor (2009). That is, the algorithms ran 740 instances, which were grouped into four classes (from *CAT1* to CAT4 with 185 instances in each class) according to their size. The instances were generated using the random values of *D* (number of units) and *n* (number of models) shown in Table 2. For all instances and for each model *i* = 1,…,*n*, a random value of d_i (number of units of model *i*) is between 1 and $\left|\frac{D-n+1}{2.5}\right|$ such that $\sum_{i=1..n} d_i = D$. All algorithms were coded in Java and the computational experiment was carried out using a 3.4 GHz Pentium IV with 512 Mb of RAM.

Table 2. Uniform distribution for the *D* and *n* values of the test instances

	CAT1	CAT2	CAT3	CAT4
D	~U(25, 50)	~U(50, 100)	~U(100, 200)	~U(200, 500)
n	~U(3, 15)	~U(3, 30)	~U(3, 65)	~U(3, 150)

For each instance, the four algorithms were run for 50 seconds. Table 3 shows the averages of the RTV values to be minimized for the global of 740 instances and for each class of instances (*CAT1* to *CAT4*).

Table 3. Averages of the RTV values for 50 seconds

	EM	Multi-start	GRASP	*DPSOpoi-$c_p$dyn*
Global	**3,747.05**	**21,390.40**	**14,168.83**	**4,625.54**
CAT1	19.14	12.08	15.47	16.42
CAT2	54.54	44.36	88.48	51.34
CAT3	260.79	226.90	510.44	610.34
CAT4	14,653.72	85,278.25	56,060.92	17,824.04

For the global of all instances, the EM algorithm is 18.99% better than *DPSOpoi-$c_p$dyn*, 73.55% better than the GRASP algorithm and 82.48% better than the multi-start algorithm. Observing the results in Table 3 by class, we can see that a simple algorithm such as the multi-start algorithm obtains the best averages for small and medium instances (*CAT1*, *CAT2* and *CAT3*) but a very poor average for large instances (*CAT4*). On the other hand, *DPSOpoi-$c_p$dyn* works well for small and large instances (*CAT1*, *CAT2* and *CAT4*) but obtains bad results for medium instances (*CAT3*). Finally, the EM algorithm works fine for small and medium instances; and for large instances, which are the most difficult to solve, it obtains the best results.

To complete the analysis of the results, their dispersion is observed. A measure of the dispersion (let it be called σ) of the RTV values obtained by each algorithm *mh* =

{EM, multi-start, GRASP, *DPSOpoi-c$_p$dyn* } for a given instance, *ins*, is defined as follows:

$$\sigma(mh, ins) = \left(\frac{\mathrm{RTV}_{ins}^{(mh)} - \mathrm{RTV}_{ins}^{(best)}}{\mathrm{RTV}_{ins}^{(best)}} \right)^2 \tag{5}$$

where $\mathrm{RTV}_{ins}^{(mh)}$ is the RTV value of the solution obtained with the algorithm *mh* for the instance *ins*, and $\mathrm{RTV}_{ins}^{(best)}$ is, for the instance *ins*, the best RTV value of the solutions obtained with the four algorithms. Table 4 shows the average *σ* dispersion for the global of 740 instances and for each class of instances.

Table 4. Average *σ* dispersion regarding the best solution found for 50 seconds

	EM	Multi-start	GRASP	*DPSOpoi-c$_p$dyn*
Global	**10.45**	**50.56**	**81.88**	**4.79**
CAT1	1.62	0.04	0.65	0.76
CAT2	0.49	0.09	4.38	0.34
CAT3	0.24	0.13	7.28	5.29
CAT4	39.45	201.96	315.20	12.78

For the global of all instances, the EM procedure has the second least average *σ* dispersion: 87.24% better than the GRASP algorithm and 79.33% better than the multi-start algorithm, but the *DPSOpoi-c$_p$dyn* *σ* dispersion is better than the EM procedure *σ* dispersion. Observing the results in Table 4 by class, we see that the EM, multi-start and PSO algorithms have a very stable performance for small instances (*CAT1* and *CAT2*). For medium instances (*CAT3*), only the EM and multi-start algorithms shows a very stable performance. Finally, no algorithm has a very stable performance for the largest instances (*CAT4*). This may occur because 50 computing seconds are not be enough time for the algorithms to converge for the *CAT4* instances. Table 5 and Table 6 show the averages of the RTV values and the *σ* dispersion, respectively, for all instances and for each class of instance (*CAT1* to *CAT4*) obtained with the four algorithms when they are run for 1,000 seconds.

Table 5. Averages of the RTV values for 1,000 seconds

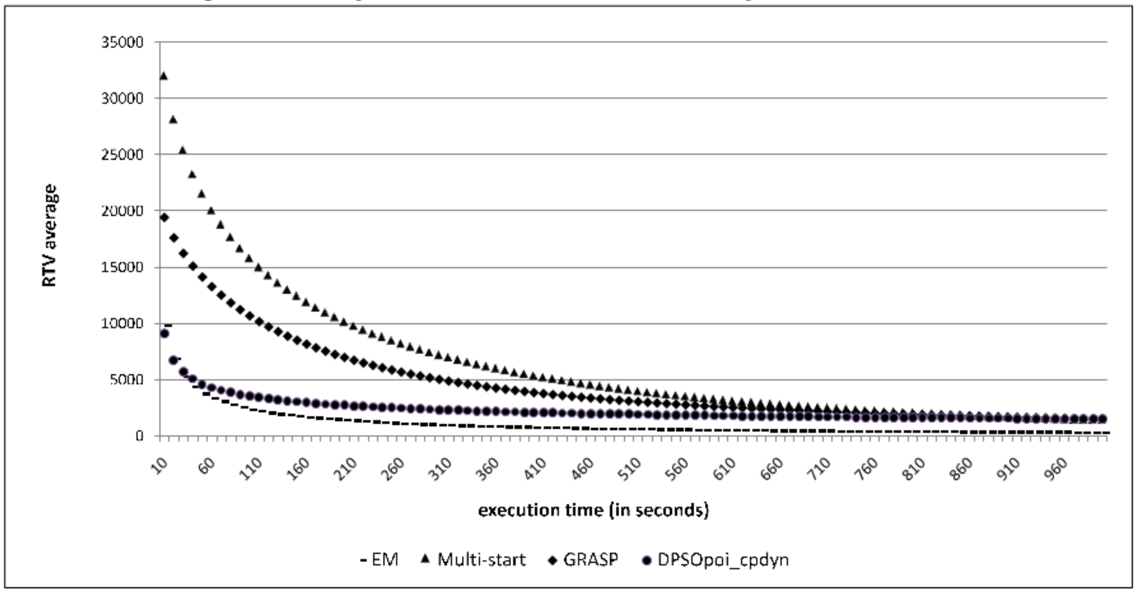
	EM	Multi-start	GRASP	*DPSOpoi-c$_p$dyn*
Global	**330.29**	**1,378.58**	**1,495.12**	**1,537.34**
CAT1	18.64	10.93	13.59	14.35
CAT2	52.97	35.48	75.08	46.55
CAT3	157.20	160.67	428.86	143.96
CAT4	1,092.36	5,307.25	5,462.95	5,944.51

Table 6. Average *σ* dispersion regarding the best solution found for 1,000 seconds

	EM	Multi-start	GRASP	*DPSOpoi-c$_p$dyn*
Global	**0.79**	**4.80**	**14.47**	**7.61**
CAT1	1.84	0.04	0.48	0.52
CAT2	1.03	0.04	5.29	0.53
CAT3	0.21	0.12	10.98	0.07
CAT4	0.08	18.98	41.06	29.27

When a computing time of 1,000 seconds is used—which seems to be long enough for all algorithms to converge (see Figure 2)—the EM algorithm is clearly the best algorithm: it is 76.04%, 77.91% and 78.52% better than the multi-start, GRASP and PSO algorithm, respectively. Moreover, when the EM algorithm has converged, it has a very stable performance for all type of instances (CAT1 to CAT4). That is, when the best solution is not obtained with the EM algorithm, the EM solution is always very close to the best solution.

Figure 2. Average of the RTV values obtained during the execution time



7. Conclusions and future research

The EM metaheuristic is a population-based metaheuristic for optimisation recently proposed by Birbil and Fang (2003). The method uses an attraction-repulsion mechanism to move the points of the population towards the optimality. In this paper, an EM algorithm is presented for solving the Response Time Variability Problem (RTVP), which has been recently appeared in the literature.

This scheduling problem arises in a variety of real-life environments including mixed-model assembly lines, multi-threaded computer systems, periodic machine maintenance, and waste collection. The aim of the RTVP is to minimize the variability in the distances between any two consecutive units of the same model. Since the RTVP is an NP-hard problem, heuristic and metaheuristic methods are needed to solve real-life instances. García *et al*. (2006) and García-Villoria and Pastor (2009) have proposed a multi-start, a GRASP and several PSO algorithms for solving the RTVP. A computational experiment was done and the results obtained with the EM algorithm are better than the results of the aforementioned algorithms. Moreover, the EM algorithm has a very stable performance when it has converged.

There are two approaches for applying a metaheuristic that works in a real space for solving combinatorial optimisation problems: to adapt the algorithm for working with a sequence of integer numbers or to adapt the representation of the solution as a real point with a random key representation. One of the best referenced procedures is a PSO algorithm (*DPSOpoi-$c_p$dyn*) that follows the first approach; on the other hand, the proposed EM algorithm follows the second approach. We propose as a future research to develop a version of *DPSOpoi-$c_p$dyn* following the second approach, and to develop a version of the EM algorithm following the first approach. The objective is to obtain better results for the RTVP.

Acknowledgements

This paper was supported by the Spanish Ministry of Education and Science under project DPI2007-61905 and co-funded by the European Regional Development Fund (ERDF). The authors wish to express their gratitude to the anonymous reviewers for their valuable comments and suggestions, which have improved the quality of this paper.

References

Adenso-Díaz, B. and Laguna, M., 2006. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54, 99-114.

Aigbedo, Henry, 2004. Analysis of parts requirements variance for a JIT supply chain. *International Journal of Production Research*, 42, 417-430.

Anily, S., Glass, C.A. and Hassin, R., 1998. The scheduling of maintenance service, *Discrete Applied Mathematics*, 82, 27-42.

Balinski, M.L. and Young, H.P., 1982. Fair Representation: meeting the ideal of one man, one vote. *Yale University Press*. New Haven CT.

Bean, J.C., 1994. Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing*, 6, 154-160.

Birbil, S.I., 2002. Stochastic global optimization techniques. *PhD Thesis*, North Carolina, State University, Raleigh, NC.

Birbil, S.I. and Fang, S.C., 2003. An Electromagnetism-like Mechanism for Global Optimization. *Journal of Global Optimization*, 25, 263-282.

Chang, P.-C., Chen, S.-H and Fan, C.-Y, 2009. A hybrid electromagnetism-like algorithm for single machine scheduling problem. *Expert Systems with Applications*, 36, 1259-1267.

Corominas, A., Kubiak, W. and Pastor, R., 2006. *Mathematical programming modelling of the response time variability problem* [online]. IOC, Universistat Politècnica de Catalunya, Spain. Available from: http://hdl.handle.net/2117/408.

Corominas, A., Kubiak, W. and Moreno, N., 2007. Response time variability. *Journal of Scheduling*, 10, 97-110.

Debels, D. and Vanhoucke, M., 2006. The Electromagnetism Meta-heuristic Applied to the Resource-Constrained Project Scheduling Problem. *Lecture Notes in Computer Science*, 3871, 259-270.

Debels, D., De Reyck, B., Leus, R. and Vanhoucke, M., 2006. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling Problem. *European Journal of Operational Research*, 169, 638-653.

Ding, F.-Y. and He, J., 2007. A heuristic procedure for the automobile assembly-line sequencing problem considering multiple product options. *International Journal of Production Research*, doi: 10.1080/00207540701381291. Available online, 13 July 2007.

Dong, L., Melhem, R. and Mossel, D., 1998. Time slot allocation for real-time messages with negotiable distance constraint requirements. *Real-time Technology and Application Symposium*. RTAS, Denver.

Eiben, A.E., Hinterding, R. and Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, 3, 124-141.

Feo, T.A. and Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8, 67-81.

García, A., Pastor, R. and Corominas, A., 2006. Solving the Response Time Variability Problem by means of metaheuristics. *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, 146, 187-194.

García-Villoria, A. and Pastor, R., 2009. Introducing dynamic diversity into a discrete particle swarm optimization. *Computers & Operations Research*, 36, 951-966.

Herrmann, J.W., 2007. *Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling* [online]. University of Maryland, USA. Available from: http://hdl.handle.net/1903/7082

Kennedy, J. and Eberhart, R.C., 1995. Particle swarm optimization. *IEEE International Conference on Neural Networks,*1942-1948.

Kubiak, W., 1993. Minimizing variation of production rates in just-in-time systems: A survey. *European Journal of Operational Research*, 66, 259-271.

Kubiak, W. and Sethi, S.P., 1994. Optimal just-in-time schedules for flexible transfer lines. T*he International Journal of Flexible Manufacturing Systems*, 6, 137-154.

Maenhout, B. and Vanhoucke, M., 2007. An electromagnetic meta-heuristic for the nurse scheduling problem. *Journal of Heuristics*, 13, 359-385.

Martí, R., 2003. Multi-start methods. *In:* Glover and Kochenberger, eds. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 355-368.

Miltenburg, J., 1989. Level schedules for mixed-model assembly lines in just-in-time production systems. *Management Science*, 35, 192-207.

Monden, Y., 1983. Toyota Production Systems. *Industrial Engineering and Management Press*. Norcross, GA.

Steiner, G. and Yeomans S., 1993. Level Schedules for Mixed-Model, Just-in-Time Processes. *Management Science*, 39, 728-735.

Tsou, C.-S. and Kao, C.-H., 2008. Multi-objective inventory control using electromagnetism-like meta-heuristic. *International Journal of Production Research*, 46, 3859-3874.

Waldspurger, C.A. and Weihl, W.E., 1995. *Stride Schedulling: Deterministic Proportional-Share Resource Management* [online]. Institute of Technology, MIT Laboratory for Computer Science. Available from: http://ebbets.poly.edu/wein/cs6243/stride.pdf.

Wu, P., Yang, K.-J. and Hung, Y.-Y., 2004. A study of electromagnetism-like mechanism for training fuzzy neural network. *In Proc. 1st Annu. Conf. OR Society at Taiwan*, 471-480.

Yuan, K., Hennequin, S., Wang, X. and Gao, L., 2006. A new heuristic-EM for permutation flowshop scheduling. *12th IFAC Symposium on Information Control Problems in Manufacturing*.

Solving the Response Time Variability Problem by means of a psychoclonal approach

Article published as [Journal of Heuristics, Published online on 15 July 2008] [DOI: http://dx.doi.org/ 10.1007/s10732-008-9082-2] © [copyright Springer]

Solving the Response Time Variability Problem by means of a psychoclonal approach[*]

Alberto GARCÍA-VILLORIA[*] and Rafael PASTOR
Institute of Industrial and Control Engineering
Technical University of Catalonia (UPC)
{alberto.garcia-villoria / rafael.pastor}@upc.edu

Abstract. The Response Time Variability Problem (RTVP) is a combinatorial scheduling problem which has recently appeared in the literature. This problem has a wide range of real-life applications in, for example, manufacturing, hard real-time systems, operating systems and network environment. Originally, the RTVP occurs whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the instants at which they receive the necessary resources is minimized. Since RTVP is hard to solve, heuristic techniques are needed for solving it. In two previous studies, three metaheuristic algorithms (a multi-start, a GRASP and a PSO algorithm) were proposed to solve the RTVP. These three metaheuristic algorithms have been the most efficient to date in solving non-small instances of the RTVP. We propose solving the RTVP by means of a psychoclonal algorithm based approach. The psychoclonal algorithm inherits its attributes from the need hierarchy theory proposed by Maslow and the artificial immune system (AIS) approach, specifically the clonal selection principle. In this paper we compare the proposed psychoclonal algorithm with the other three metaheuristic algorithms previously mentioned and show that, on average, the psychoclonal algorithm strongly improves the obtained results.

Keywords: response time variability, fair sequences, scheduling, psychoclonal algorithm, clonal selection, metaheuristics

1. Introduction

The concept of *fair sequence* has emerged independently from scheduling problems of diverse environments, principally from manufacturing, hard real-time systems, operating systems and networks environments. The common framework for these scheduling problems is defined by Kubiak (2004) as to build a fair sequence using n symbols, where symbol $i = 1,...,n$ is to occur given number d_i of times in the sequence. The fair sequence will be that one that allocates a fair share of positions in any prefix of the sequence to each symbol i. This *fair* or *ideal* share of positions allocated to symbol i in a sequence prefix of length k is proportional to a relative importance (d_i) of symbol i with respect to the total copies of competing symbols (equal to $\sum_{i=1}^{n} d_i$). There is not a

[*] Sponsored by the Spanish Ministry of Education and Science's project DPI2007-61905; co-funded by the FEDER.
[*] Corresponding author: Alberto García-Villoria, IOC – Institute of Industrial and Control Engineering, Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; e-mail: alberto.garcia-villoria@upc.edu

universally definition of fairness because several reasonable metrics of fairness can be defined according to the specific problem.

The first problem in which seems to have appeared the idea of fair sequence is the sequencing on the mixed-model assembly production lines at Toyota Motor Corporation under the just-in-time (JIT) production system. One of the most important JIT objectives is to get rid of all kinds of waste and inefficiency and, according to Toyota, the main waste is due to the stocks. To reduce the stock, JIT production systems require producing only the necessary models in the necessary quantities at the necessary time. To achieve this, one main goal, as Monden (1983) says, is scheduling the units to be produced to keep a constant consumption rates of the components involved in the production process. Miltenburg (1989) deals with this scheduling problem and he assumes that models require approximately the same number and mix of parts. Thus, he considers only the demand rates for the models. Miltenburg proposes four objective functions based on the fairness idea of scheduling the models so that the proportion of model *i* produced, over each time period, to the total production is as close to its ideal production as possible. That is, if the number of models is *n* ($i = 1,...,n$) and the units of model *i* to be produced is d_i, then the total number of units to be produced (D) is equal to $\sum_{i=1}^{n} d_i$ and the ideal production of model *i* at the period time *k* ($k = 1,...,D$) is $\frac{d_i}{D}k$. This problem is known as Product Rate Variation (PRV) problem (Kubiak 1993). Kubiak and Sethi (1991) reformulated the PRV problem as an assignment problem and, therefore, it can be solved with an algorithm whose complexity is polynomial in *D*.

Independently of assembly lines, the fair sequencing idea has appeared in computer multithreaded systems when Waldspurger and Weihl (1995) proposed the *stride scheduling* to resource allocation in these systems. Multithreaded systems (operating systems, network servers, media-based applications, etc.) need to manage the scarce resources in order to service requests of *n* clients. Resources are allocated in discrete time slices (authors refer to the duration of a standard time slice as a *quantum*). Resource rights are represented by *tickets* and each client *i* has a given number d_i of tickets. A fair scheduling is obtained when the resources that a client has received (i.e. the number of quanta in which has been assigned) during the first k allocations, k = 1,...,*D* (where $D = \sum_{i=1}^{n} d_i$), are directly proportional to its ticket allocations. Thus, a client with twice as many tickets as another will receive twice as much of a resource in a given time interval. Waldspurger and Weihl suggest two metrics to evaluate the fairness of a sequence: the throughput error and the response time variability. The throughput error measures the maximum absolute deviation, for each client *i* and allocation *k*, between the resources received and the ideal resources $\left(\frac{d_i}{D}k\right)$. The problem of minimizing the throughput error can be efficiently solved using the Earliest Due Date rule defining fictitious earliest and latest due dates (see Kubiak 2004).

The problem of minimizing the response time variability is known as Response Time Variability Problem (RTVP). Waldspurger and Weihl define the response time as the elapsed time from a client's completion of one quantum up to including its completion of next. Since the quantum duration is fixed, this definition is equivalent to the number of quanta between a client’s two consecutive quantum allocations plus one. Thus, the response time variability for a client is the variance of its response times.

This metric is not exclusively useful on computer system environments. For example, in our experience with practitioners of manufacturing industries, we noticed that practitioners usually refer to a good mixed-model sequence not in terms of ideal production as it is usual in the literature (Miltenburg 1989), but in terms of having distances between the units for the same model as regular as possible (i.e. there should not be variance in the response times of the models). Herrmann (2007) found the RTVP while was working with a healthcare facility that needed to schedule the collection of waste from waste collection rooms throughout the building. Given data about how often a trash handler needs to visit each room, the facilities manager wanted these visits to occur as regularly as possible so that excessive waste would not collect in any room. For instance, if a room needs four visits per eight-hour shift, then, ideally, it would be visited every two hours. The problem is difficult because different rooms require a different number of visits per shift. Herrmann proposed a heuristic algorithm based on the stride scheduling for solving it.

The RTVP is, unfortunately, NP-hard (Corominas et al. 2007). To solve the RTVP, Waldspurger and Weihl (1995) proposed the stride scheduling, which is a greedy heuristic algorithm. Corominas et al. (2007) proposed a mixed integer linear programming (MILP) model and five greedy heuristic algorithms. In Corominas et al. (2006), an improved MILP model is proposed (the practical limit to obtain optimal solutions is 40 units to be scheduled). García et al. (2006) proposed seven metaheuristic algorithms: a multi-start, a GRASP (Greedy Randomized Adaptive Search Procedure) and four variants of a discrete PSO (Particle Swarm Optimization) algorithm. Finally, eleven variants of a discrete PSO algorithm were used in García-Villoria and Pastor (2007).

In this paper, a psychoclonal algorithm based approach is proposed to solve the RTVP. Psychoclonal is a very new population-based metaheuristic that was first proposed by Tiwari et al. (2005). This metaheuristic inherits its attributes from the need hierarchy theory of Maslow (1954) and the artificial immune system (AIS) approach, specifically the clonal selection principle (Gaspar and Collard 2000; de Castro and von Zuben 2002). There are five levels of needs arranged in the Maslow’s hierarchy, named physiological needs, safety needs, social needs, growth needs and self-actualization needs. Clonal selection explains the response of immune systems to non-self antigens. The cells (lymphocytes) that produce antibodies that can recognize the intruding antigens are selected to proliferate by cloning and further are undergone to an affinity maturation process that consists in hypermutations in order to obtain cells that produce antibodies that can improve their affinities to the non-self antigens. The worst cells are undergone receptor editing: cells are deleted and replaced by new ones. The whole process continues until the self-actualization level is reached.

The psychoclonal metaheuristic has yielded very good results when it has been used to solve several scheduling and combinatorial optimization problems (Prakash and Tiwari 2005; Tiwari et al. 2005; Kumar et al. 2006a, 2006b; Singh et al. 2006). The proposed psychoclonal algorithm for solving the RTVP is compared with the most efficient procedures for solving non-small instances published to date: the multi-start and the GRASP algorithms proposed in García et al. (2006) and the PSO algorithm called

DPSOpoi-c$_p$dyn (García-Villoria and Pastor 2007). On average, the psychoclonal algorithm improves strongly on previous results.

The rest of the paper is organized as follows. Section 2 presents a formal definition of the RTVP and briefly exposes the three metaheuristic procedures presented by García et al. (2006) and García-Villoria and Pastor (2007) for its solution. Section 3 describes the basic scheme of the psychoclonal metaheuristic. Section 4 proposes a psychoclonal algorithm based approach for solving the RTVP. Section 5 provides the computational experiment and the comparison with the other metaheuristics. Finally, some conclusions are given in Section 6.

2. The Response Time Variability Problem (RTVP)

The aim of the Response Time Variability Problem (RTVP) is to minimize variability in the distances between any two consecutive copies of the same symbol.

The RTVP is formulated as follows. Let *n* be the number of symbols, d_i the number of copies to be scheduled of the symbol *i* (*i* = 1,…,*n*) and *D* the total number of copies ($D = \sum_{i=1}^{n} d_i$). Let *s* be a solution of an instance in the RTVP that consists of a circular sequence of copies ($s = s_1 s_2 \ldots s_D$), where s_j is the copy sequenced in position *j* of sequence *s*. For all symbol *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the copies *k* + 1 and *k* of the symbol *i* are found (where the distance between two consecutive positions is considered equal to 1). Since the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_i}^i$ is the distance between the first copy of the symbol *i* in a cycle and the last copy of the same symbol in the preceding cycle. Let $\bar{t}_i$ be the average distance between two consecutive copies of the symbol *i* ($\bar{t}_i = D/d_i$). For all symbol *i* in which $d_i = 1$, t_1^i is equal to $\bar{t}_i$. The objective is to minimize the metric Response Time Variability (RTV) which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2 \tag{1}$$

For example, let $n = 3$, $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\bar{t}_C = 2$. Any sequence is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a solution, where $RTV = \left((5-4)^2 + (3-4)^2\right) + \left((2-4)^2 + (6-4)^2\right) + \left((2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2\right) = 2 + 8 + 2 = 1$.

As has been introduced in Section 1, the best type of procedures to date for solving the RTVP are three metaheuristics. Therefore, next the three best algorithms based on these metaheuristics are briefly explained (for more details of the three algorithms, see García et al. 2006 and García-Villoria and Pastor 2007).

The multi-start method is based on generating initial random solutions and on improving each of them to find a local optimum, which is usually done by means of a local search procedure (Martí 2003). The multi-start proposed in García et al. (2006) is as follows. Random solutions are generated as follows. For each position, a symbol to be sequenced is chosen at random. The probability of each symbol is equal to the number of copies of this symbol that remain to be sequenced divided by the total number of copies that remain to be sequenced. The local search procedure used is applied as follows. A local search is performed iteratively in a neighborhood that is generated by interchanging each pair of two consecutive symbols of the sequence that represents the current solution; the best solution in the neighborhood is chosen; the optimization ends when no neighboring solution is better than the current solution.

GRASP, designed by Feo and Resende (1989), can be considered as a variant of the multi-start method in which the initial solutions are obtained using directed randomness. The solutions are generated by means of a greedy strategy in which random steps are added and the choice of elements to be included in the solution is adaptive. The random step in the GRASP proposed by García et al. (2006) consists of selecting the next symbol to be sequenced from a set called candidate list; the probability of each candidate symbol is proportional to the value of its Webster index, which is based on the parametric method of apportionment with parameter $\delta = 1/2$ (Balinski and Young 1982). The Webster index for the symbol *i* (*i* = 1,…,*n*) is evaluated as $\frac{d_i}{\left(x_{ik} + \delta\right)}$, where x_{ik} is the number of copies of the symbol *i* in the sequence of length *k* = 0,…,*D* (assume x_{i0} = 0). The candidate list is composed by the symbols with greater value of their Webster index. The local search procedure applied to the initial solutions is the same local search that is applied by the multi-start method.

PSO is a population-based metaheuristic algorithm designed by Kennedy and Eberhart (1995), which is based on an analogy of the social behaviour of flocks of birds when they search for food. The population or swarm is composed of particles (birds), which have a multi dimensional real point (which represents a feasible solution) and a velocity (the movement of the point in the *n*-dimensional real space). The velocity of a particle is typically a linear combination of three types of velocity: 1) the inertia velocity; 2) the velocity to the best point found by the particle; and 3) the velocity to the best point found by the swarm. The PSO algorithm iteratively modifies the point and the velocity of each particle as it looks for the optimal solution. In the *DPSOpoi-*c_p*dyn* algorithm (García-Villoria and Pastor 2007), random modifications to the points of the particles are introduced. The frequency of the modifications changes dynamically according to the homogeneity of the swarm. The aim is preventing a premature convergence and enabling the PSO algorithm to escape from a local optimum. Although the PSO algorithm was originally designed for working in an *n*-dimensional real space, *DPSOpoi-*c_p*dyn* is adapted to work with a sequence that represents the solution. In this adaptation of the PSO algorithm, a point is now the sequence of copies of the symbols that represents a solution and the velocity is an ordered list of transformations that must be applied to the particle so it changes from its current point to another point; each transformation consists of a pair of positions of the point (sequence) to be swapped. In the case of the velocity to the best point found by the particle, this velocity is a list of transformations needed to obtain the best particle point from the current position; the case is the same for the velocity to the best point found by the swarm.

3. The psychoclonal metaheuristic

The psychoclonal metaheuristic has been recently proposed by Tiwari et al. (2005) which has been successfully used for solving the following scheduling and combinatorial problems: the disassembly line balancing problem (Prakash and Tiwari 2005), the assembly configuration problem (Tiwari et al. 2005), the flow shop problem (Kumar et al. 2006a), the make-to-stock inventory deployment problem (Kumar et al. 2006b) and the product mix decision problem (Singh et al. 2006).

This metaheuristic is inspired by the need hierarchy theory of Maslow (1954) and the clonal selection principle developed by Gaspar and Collard (2000). First, the salient concepts of these theories are briefly explained in Section 3.1. Next, the psychoclonal metaheuristic scheme is described in Section 3.2.

3.1. Background theories of the psychoclonal metaheuristic

Psychologists have investigated the motivations of people behavior during their lifetime. Maslow proposed a theory, known as need hierarchy theory, which hypothesizes that the people behavior is motivated for satisfying their needs. These needs are grouped into five sets that are hierarchically arranged according to the degree of necessity. These levels are the following (Tiwari et al. 2005): A) physiological needs (in optimization, this corresponds to the generation of possible sequences based upon the problem environment), B) safety needs (evaluation of a particular entity or candidate solution), C) social needs (selection and interaction between candidate solutions), D) growth needs (candidate solutions diversify to extend the search space) and E) self-actualization needs (a stop condition is required to decide the near-optimal solution). In order to satisfy the upper levels, first the lowest levels have to be satisfied.

Artificial Immune Systems (AIS) are an emerging kind of computational intelligence paradigm inspired by the biological immune system of vertebrate animals. Their applications include optimization, anomaly detection, fault diagnosis and patter recognition (de Castro and Timmis 2002). Wang et al. (2004) classify the methods based in AIS in three main categories: clonal selection principle-based, Genetic Algorithms (GA)-aided and immune networks-based approaches.

The clonal selection explains the response of the immune system of the vertebrate animals when they are attacked by foreign antigens. Immune system has lymphocytes or white cells that secrete antibodies that neutralize the foreign antigens (since a given lymphocyte only produces a single type of antibodies, in AIS there is no distinction between a lymphocyte and its antibodies). The effectiveness of the immune response depends on the affinity that has the antibodies with the antigens. The first time that the body is exposed to a given antigen, immune system has low affinity antibodies, each one with different affinity. But immune system is able to learn how to produce high affinity antibodies (it is known as reinforcement learning). The antibodies are selected to proliferate according to their affinities (clonal selection). Next, antibody clones are diversified by two mechanisms: hypermutation and receptor editing. This phenomenon is referred as maturation of the immune response. The hypermutation introduces random changes into an antibody inversely proportional to its affinity with the antigen. A large proportion of the hypermuted clones becomes more dysfunctional but, however, occasionally an effective hypermutation improves their affinity. The receptor editing deletes the lowest affinity antibodies and develops new ones through genetic recombination; in AIS, the genetic recombination is modeled creating a new antibody at random. Figure 1 illustrates hypermutation guides the affinity to a local optimum whereas receptor editing escapes from the local optimum.

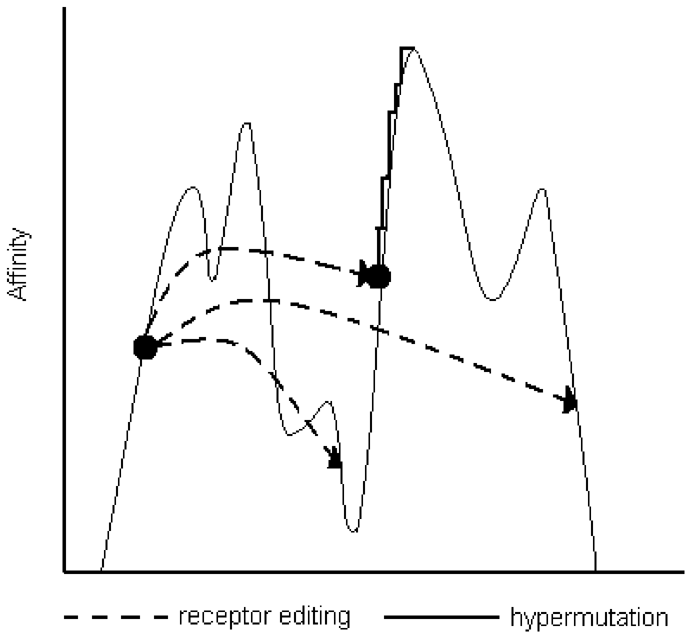


Figure 1. Representation of hypermutation and receptor editing

The reinforcement learning strategy makes that immune system continuously improving its efficiency to block foreign antigens. Since the body would be expected a given antigen more times during its lifespan, immune system keeps clones of the highest affinity antibodies (immune memory). Therefore, immune system ensures speed and accuracy in its responses across the time.

3.2. The psychoclonal metaheuristic scheme

The aim of Tiwari et al. (2005) when they developed the psychoclonal metaheuristic was to obtain a generic scheme based on the need of explotation (i.e. the local search) and the exploration (i.e. the global search) of the search space. The need hierarchy theory of Maslow and the clonal selection principle were used by Tiwari et al. (2005) to develop the psychoclonal metaheuristic, which scheme is as follows:

Need level A (physiological needs). Each antibody represents a solution. Thus, an affinity function has to be defined based on the objective function. It is also required an initial population of antibodies generated at random depending upon the environment of the problem.

Need level B (safety needs). The antibodies are exposed to the antigen, i.e. the value of their affinity function is calculated.

Need level C (social needs). An interaction is carried out between the antibodies to identify the best antibodies of the population. The best antibodies are selected and cloned proportionally to their affinity function value.

Need level D (growth needs). The generated clones are submitted for hypermutation with a rate inversely proportional to their affinity function value. After satisfaction of need level D, it is necessary to check the needs of level B (i.e. to calculate the affinity function values of the clones after hypermutation).

Need level E (self-actualization needs). The best clones are selected to be part of the new population generation. In addition, new antibodies generated at random are added to the new generation (receptor editing). The process repeats until the self-actualization is reached (e.g. a maximum number of generation or a maximum computing time).

4. The psychoclonal algorithm based approach for solving the RTVP

In Section 4.1 we design an algorithm based on the Pychoclonal metaheuristic for solving the RTVP. The algorithm has several parameters that influence in its efficiency. The selection of their values is discussed in Section 4.2.

4.1. Design of the psychoclonal algorithm

In this paper we propose an algorithm for solving the RTVP based on the psychoclonal metaheuristic.

The first consideration is the choice of the antibody representation for a solution. For the RTVP, the more intuitive representation consists in a *D*-length sequence of the symbols (where $D = \sum_{i=1}^{n} d_i$). The design of algorithm is explained below:

A1. The affinity function *f* for the antibody *ab* is defined as $f(ab) = 1 / \left(RTV(ab) + \varepsilon\right)$ where RTV(*ab*) is the RTV value of the solution represented by *ab* and *ε* is a small value to avoid a division by zero.

A2. The initial population is set by antibodies that are generated as in the multi-start algorithm. That is, for each position of the sequence (antibody), a symbol to be sequenced is chosen at random. The probability of each symbol is equal to the number of copies of this symbol that remain to be sequenced divided by the total number of copies that remain to be sequenced. The total number of antibodies that form the population is *N*.

B. For each antibody *ab* of the current population, *f(ab)* is evaluated.

C. The best *n* antibodies according to their affinity value are selected to be cloned. The number of clones (*NC*) that are generated for each selected antibody is calculated with the following expression:

$$NC(ab_i) = \text{round}\left(\frac{\beta \cdot N}{i}\right), \; i = 1, \ldots, n \qquad (2)$$

where ab_i is the *i*th best antibody of the current population, *round* is an operator that rounds its argument toward the closest integer and *β* is a multiplying factor.

D1. The clones are submitted for hypermutation with a rate inversely proportional to their affinity value. The hypermutation rate (σ) for a clone *ab* is calculated with the following expression:

$$\sigma(ab) = \max\left(1, \text{round}\left(D \cdot e^{-K\frac{f(ab)}{f^*}}\right)\right) \tag{3}$$

where K is the control factor of decay and f^* is the affinity value of the best antibody of the current population. The hypermutation rate indicates how many simple mutations are applied to the cloned antibodies. A simple mutation consists in choosing randomly two positions of the sequence that represents the antibody and swapping them. In order to maintain the best antibodies, we keep one original (parent) antibody unhypermutated.

D2. For each cloned antibody *ab, f(ab)* is evaluated.

E1. The current population is set with the (N - d)th best cloned antibodies.

E2. The current population is completed adding d new antibodies generated at ramdon as explained in step *A2*.

E3. Until the computing time of the algorithm does not reach a preset time go to step *B*.

The way that the number of clones (Equation 2) and the hypermutation rate (Equation 3) are evaluated is based on the CLONALG algorithm (de Castro and von Zuben 2002), which is one of the most widely applied method based on AIS (Wang et al. 2004).

The algorithm that we propose has 5 parameters: N (size of the population), n (number of the best antibodies to be cloned), β (multiplying factor to calculate the number of clones of a given antibody), K (control factor of decay of the hypermutation rate) and d (the number of new generated antibodies to be added into the population). Their suitable values are discussed in the next section.

4.2. Fine-tuning of the psychoclonal algorithm parameters

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although the parameter values are extremely important because the results of the metaheuristic for each problem are very sensitive to them, the selection of parameter values is commonly justified in one of the following ways (Eiben et al. 1999; Adenso-Díaz and Laguna 2006): 1) "by hand" on the basis of a small number of experiments that are not specifically referenced; 2) by using the general values recommended for a wide range of problems; 3) by using the values reported to be effective in other similar problems; or 4) by choosing values without any explanation.

Adenso-Díaz and Laguna (2006) proposed a new technique called CALIBRA for fine-tuning the parameters of heuristic and metaheuristic algorithms. CALIBRA is based on Taguchi's fractional factorial experimental designs coupled with a local search procedure.

CALIBRA has been chosen for fine-tuning the psychoclonal algorithm parameters using a set of 60 representative training instances (generated as explained in Section 5). The following parameter values are obtained: $N = 25$, $n = 3$, $\beta = 1.3$, $K = 7.6$ and $d = 3$.

The GRASP and PSO algorithms (the multi-start algorithm has not parameters) are also fine-tuned with CALIBRA and the same 60 training instances. GRASP has only one parameter, which is the size of the candidate list; the obtained value is 3. *DPSOpoi-$c_p$dyn* has five parameters and the following values are obtained: size of the population = 13, coefficient that weights the inertia velocity (ω) = 0.75, coefficient that weights the velocity to the best particle point (c_1) = 0.13, coefficient that weights the velocity to the best swarm point (c_2) = 0.75 and factor of the degree of the random modifications introduced (K) = 8.70.

5. Computational experiment

The multi-start, GRASP and PSO algorithms explained in Section 2 are the most efficient algorithms published to date for solving non-small RTVP instances. Therefore, we compare our proposed psychoclonal algorithm with them.

The computational experiment for the four metaheuristic algorithms was carried out for the same instances and conditions used in García et al. (2006). That is, the algorithms ran 740 instances which were grouped into four classes (185 instances in each class) according to their size. The instances in the first class (*CAT1*) were generated using a random value of D (number of copies) uniformly distributed between 25 and 50, and a random value of n (number of symbols) uniformly distributed between 3 and 15; for the second class (*CAT2*), D was between 50 and 100 and n between 3 and 30; for the third class (*CAT3*), D was between 100 and 200 and n between 3 and 65; and for the fourth class (*CAT4*), D was between 200 and 500 and n between 3 and 150. For all instances and for each type of symbol $i = 1,\dots,n$, a random value of d_i (number of copies of the symbol i) was between 1 and $\left|\frac{D-n+1}{2.5}\right|$ such that $\sum_{i=1}^{n} d_i = D$. All algorithms were coded in Java and the computational experiment was carried out using a 3.4 GHz Pentium IV with 512 MB of RAM.

For each instance, the four metaheuristics were run for 50 seconds. Table 1 shows the averages of the RTV values to be minimized for the global of 740 instances and for each class of instances (*CAT1* to *CAT4*).

	Psychoclonal	**Multi-start**	**GRASP**	***DPSOpoi-$c_p$dyn***
Global	**235.68**	**21,390.39**	**14,168.83**	**4,625.54**
CAT1	14.92	12.08	15.47	16.42
CAT2	44.25	44.36	88.48	51.34
CAT3	137.07	226.90	510.44	610.34
CAT4	746.48	85,278.25	56,060.92	17,824.04

Table 1. Averages of the RTV values for 50 seconds

For the global of all instances, the psychoclonal algorithm is 94.90% better than *DPSOpoi-$c_p$dyn*, 98.34% better than the GRASP algorithm and 98.90% better than the multi-start algorithm. Observing the results in Table 1 by class, we can see that a simple algorithm such as the multi-start algorithm obtains good averages for small instances (*CAT1* and *CAT2*) but a very poor average for large instances (*CAT4*). On the other hand, *DPSOpoi-$c_p$dyn* produces quite good results for small instances (*CAT1* and *CAT2*), worse results for medium instances (*CAT3*) and better results for large ones. Finally, the psychoclonal algorithm works very well for all class of instances. For the smallest instances (*CAT1*) it is the second best shortly overcome by the multi-start algorithm. For the remaining classes, the psychoclonal algorithm obtains the best RTV averages. For *CAT2* instances, it is 0.25%, 49.99% and 13.81% better than the multi-start, GRASP and PSO algorithms, respectively. For *CAT3* instances, it is 39.59%, 73.15% and 77.54% better than the multi-start, GRASP and PSO algorithms, respectively. Finally, for *CAT4* instances, it is 99.12%, 98.67% and 95.81% better than the multi-start, GRASP and PSO algorithms, respectively.

In Table 2 we compare the number of times that each algorithm reaches the best RTV value obtained with all four algorithms. The results are shown for the 740 instances overall and for each class of instances.

	Psychoclonal	**Multi-start**	**GRASP**	***DPSOpoi-$c_p$dyn***
Global	**447**	**231**	**190**	**66**
CAT1	61	147	102	37
CAT2	72	70	45	27
CAT3	148	14	24	2
CAT4	166	0	19	0

Table 2. Number of times that the best solution is reached

As we expect from the results in Table 1, Table 2 shows that the psychoclonal algorithm reaches the best solution the greatest number of times (in 60.40% for the global of all instances). Observing the results by class, we can see that for *CAT3* instances, although the multi-start algorithm obtains a better RTV average than the GRASP, the GRASP algorithm reaches mores times the best solution. And for *CAT4* instances the GRASP algorithm reaches more times the best solution than *DPSOpoi-$c_p$dyn*, although *DPSOpoi-$c_p$dyn* is better according to the RTV value average.

To complete the analysis of the results, their dispersion is observed. A measure of the dispersion (let it be called σ) of the RTV values obtained by each algorithm *alg* = {psychoclonal, multi-start, GRASP, *DPSOpoi-$c_p$dyn*} for a given instance, *ins*, is defined as follows:

$$\sigma(alg,ins)=\left(\frac{\mathrm{RTV}_{ins}^{(alg)}-\mathrm{RTV}_{ins}^{(best)}}{\mathrm{RTV}_{ins}^{(best)}}\right)^2 \quad (5)$$

where $\mathrm{RTV}_{ins}^{(alg)}$ is the RTV value of the solution obtained with the algorithm *alg* for the instance *ins*, and $\mathrm{RTV}_{ins}^{(best)}$ is, for the instance *ins*, the best RTV value of the solutions obtained with the four metaheuristics. Table 3 shows the average σ dispersion for the global of 740 instances and for each class of instances.

	Psychoclonal	Multi-start	GRASP	*DPSOpoi-$c_p$dyn*
Global	**0.23**	**12,562.54**	**27,928.94**	**292.60**
CAT1	0.79	0.05	0.71	0.78
CAT2	0.11	0.10	5.70	0.45
CAT3	0.02	0.75	17.74	21.13
CAT4	0.01	50,249.23	111,691.60	1,148.03

Table 3. Average σ dispersion regarding the best solution found

For the global of all instances, the psychoclonal algorithm has the least average σ dispersion very far from the dispersion of the other algorithms. Observing the results in Table 3 by class, we see that the behavior of the dispersions is almost analogous to the behavior of the RTV values. For the smallest instances (*CAT1* and *CAT2*), the multi-start algorithm gives the smallest average dispersion and it is near followed by the psychoclonal algorithm. For the medium and big instances (*CAT3* and *CAT4*), the psychconal algorithm shows clearly the least dispersion, followed by the multi-start algorithm for the medium instances. Although the multi-start algorithm gives the worst RTV solutions for the *CAT4* instances, it has less dispersion than the GRASP algorithm; this indicates that multi-start algorithm is more stable than the GRASP algorithm in this case. To summarize, the results in Tables 1-3 show that the psychoclonal algorithms has a very good performance in terms of the RTV values and also has a very stable behavior for all classes of instances.

The bad results for the larger instances (*CAT4*) obtained by the multi-start, GRASP and PSO algorithms may occur because 50 seconds might not be enough time for them to converge. Table 4 shows the averages of the RTV values for the global of all instances and for each class of instances (*CAT1* to *CAT4*) obtained with the four algorithms when they are run for 1,000 seconds.

	Psychoclonal	Multi-start	GRASP	*DPSOpoi-$c_p$dyn*
Global	**161.60**	**1,378.58**	**1,495.12**	**1,537.34**
CAT1	14.90	10.93	13.59	14.35
CAT2	39.90	35.48	75.08	46.55
CAT3	122.38	160.67	428.86	143.96
CAT4	469.23	5,307.25	5,462.95	5,944.51

Table 4. Averages of the RTV values for 1000 seconds

With 1000 seconds of execution time, which seems time enough for the convergence of the four algorithms (see Figure 2), the psychoclonal algorithm is for the global of all instances 88.28%, 88.19% and 89.49% better than the multi-start, GRASP and PSO algorithms, respectively. Although the multi-start, GRASP and PSO algorithms improve a lot their average results, the psychoclonal algorithm is clearly better. Indead, the results obtained with the psychoclonal algorithm for 50 computing seconds are much better than the results obtained with the other algorithms for 1000 computing seconds.

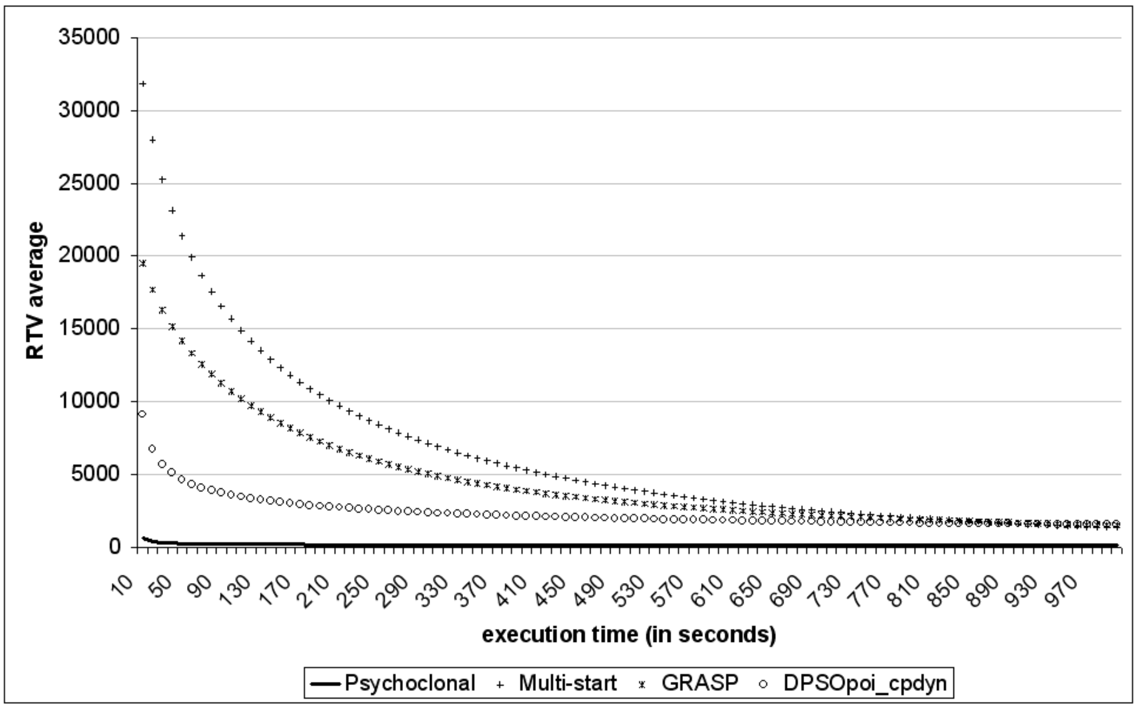


Figure 2. Average of the RTV values obtained over the computing time

Finally, to show the strength of the psychoclonal algorithm, the best heuristics proposed in Corominas et al. (2007) to solve the RTVP (the bottleneck, Webster, Jefferson and insertion heuristics) are compared with the proposed psychoclonal algorithm. The execution time of the heuristics for each instance was always close to 0.1 seconds. We show in Table 5 the averages of the RTV values for the global of all instances and for each class of instances obtained with the four heuristics and the psychoclonal for different running times. With only 0.5 seconds of computing time, the average of the RTV values obtained with the psychoclonal algorithm is 51.03% better than the average of the RTV values obtained with the best heuristic (bottleneck heuristic); and with 5 seconds, it is 93.52% better..

					Psychoclonal		
	Bottleneck	**Webster**	**Jefferson**	**Insertion**	(0.5 sec.)	(1 sec.)	(5 sec.)
Global	**9,849.99**	**22,821.94**	**23,736.83**	**25,811.24**	**4,823.48**	**2,498.85**	**638.49**
CAT1	107.09	121.84	147.19	172.69	16.35	15.61	15.24
CAT2	693.38	933.11	1,077.88	1,254.29	50.97	48.08	45.83
CAT3	4,369.44	8,502.80	9,106.04	10,248.21	593.01	316.82	145.90
CAT4	34,230.05	81,730.00	84,616.22	91,569.77	18,633.57	9,614.87	2,347.94

Table 5. Averages of the RTV values

6. Conclusions

In this paper, the Response Time Variability Problem (RTVP) is solved. This problem is an NP-hard scheduling problem that proposes a new metric to measure the *fairness* of a solution according to the relative importance of the different symbols to be sequenced. In the RTVP, the aim is to minimize variability in the distances between any two consecutive copies of the same symbol, i.e. to distribute the symbols the more *regular* as possible.

The RTVP occurs in diverse environments as manufacturing, hard real-time systems, operating systems and networks environments. Since it is a NP-hard problem, metaheuristic methods are needed for solving non-small instances. García et al. (2006) have proposed a multi-start algorithm and a GRASP algorithm and García-Villoria and Pastor (2007) have proposed a PSO algorithm called *DPSOpoi-*c_p*dyn*, which are the most efficient algorithms published to date for solving the RTVP. In order to improve the published results, a psychoclonal algorithm based approach is proposed.

The psychoclonal metaheuristic inherits its attributes from the need hierarchy theory of Maslow (1954) and the artificial immune system (AIS) approach, specifically the clonal selection principle (Gaspar and Collard 2000). The main features of this metaheuristic are various levels of needs, affinity maturation to guide the solution to a local optimum and receptor editing to escape from local optima and to explore new regions of the solution search space.

A computation experiment was carried out and its results show that the psychoclonal algorithm that we propose improves strongly the previous results obtained with the other three algorithms on average. In addition, the psychoclonal algorithm has a very stable behavior for small, medium and large instances. The theoretical properties of the psychoclonal metaheuristic have not been mathematically studied in the literature. This is a future research topic.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to the anonymous reviewers for their valuable comments and suggestions, which have improved the quality of this paper. The authors are also grateful to Professor Albert Corominas (Technical University of Catalonia) for his helpful assistance.

REFERENCES

Adenso-Díaz, B. and Laguna, M. (2006) 'Fine-tuning of algorithms using fractional experimental designs and local search', *Operations Research*, Vol. 54, pp. 99-114.

Balinski, M.L. and Young, H.P. (1982) 'Fair Representation: meeting the ideal of one man, one vote', *Yale University Press*, New Haven CT.

Corominas, A., Kubiak, W. and Pastor, R. (2006) 'Solving the Response Time Variability Problem (RTVP) by means of mathematical programming', *Working paper IOC-DT*, Universistat Politècnica de Catalunya, Spain.

Corominas, A., Kubiak, W. and Moreno, N. (2007) 'Response time variability', *Journal of Scheduling*, Vol. 10, pp. 97-110.

De Castro, L.N. and Timmis, J. (2002) 'Artificial Immune Systems: A New Computational Intelligence Approach', *Springer-Verlag*, London, United Kingdom.

De Castro, L.N. and von Zuben, F.J. (2002) 'Learning and optimization using the clonal selection principle', *IEEE Transactions on Evolutionary Computation* , Vol. 6, No. 3, pp. 239-251.

Eiben, A.E., Hinterding, R. and Michalewicz, Z. (1999) 'Parameter control in evolutionay algorithms', *IEEE Transactions on evolutionary computation*, Vol. 3, pp. 124-141.

Feo, T.A. and Resende, M.G.C (1989) ‘A probabilistic heuristic for a computationally difficult set covering problem’, *Operations Research Letters*, Vol. 8, pp. 67-81.

García-Villoria, A. and Pastor, R. (2007) ‘Introducing dynamic diversity into a discrete particle swarm optimization’, *Computers & Operations Research*, In Press, Corrected Proof, Avalaible online 7 December 2007, doi:10.1016/j.cor.2007.12.001.

García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 146, pp.187-194.

Gaspar, A. and Collard, P. (2000) ‘Two models of immunization for time dependent optimization’, in *Proceeding of the IEEE International Conference on Systems Manufacturing and Cybernetics*, pp. 113-118.

Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report*, University of Maryland, USA.

Kennedy, J. and Eberhart, R.C. (1995) ‘Particle swarm optimization’, *IEEE International Conference on Neural Networks*, Australia.

Kubiak, W. (1993) ‘Minimizing variation of production rates in just-in-time systems: A survey’, *European Journal of Operational Research*, Vol. 66, No. 3, pp. 259-271.

Kubiak, W. (2004) ‘Fair Sequences’, In *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapman and Hall.

Kubiak, W. and Sethi, S.P. (1991) ‘A Note on Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 37, pp. 121-122.

Kumar, A., Prakash, A., Shankar, R. and Tiwari, M.K. (2006a) ‘Psycho-Clonal algorithm based approach to solve continuous flow shop scheduling problem’, *Expert Systems with Applications*, Vol. 31, No. 3, pp. 504-514.

Kumar, A., Prakash, A., Tiwari, M.K. and Chan, F.T.S. (2006b) ‘Stochastic make-to-stock inventory deployment problem: an endosymbiotic psychoclonal algorithm based approach’, *International Journal of Production Research*, Vol. 44, No. 11, pp. 2245-2263.

Martí, R. (2003) ‘Multi-start methods’, *Handbook of Metaheuristics*, Glover and Kochenberger (eds.), Kluwer Academic Publishers, pp. 355-368.

Maslow, A.H. (1954) *Motivation and personality*, New York: Harper & Bros.

Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 35, No. 2, pp. 192-207.

Monden, Y. (1983) ‘Toyota Production Systems’, *Industrial Engineering and Management Press*, Norcross, GA.

Prakash, A. and Tiwari, M.K. (2005) ‘Solving a dissassembly line balancing problem with task failure using a psycho-clonal algorithm’, *International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, California, USA.

Singh, R.K., Prakash, Kumar, S. and Tiwari, M.K. (2006) ‘Psycho-clonal based approach to solve a TOC product mix decision problem’, *International Journal of Advanced Manufacturing Technology*, Vol. 29, pp. 1194-1202.

Tiwari, M.K., Prakash, A., Kumar, A. and Mileham, A.R. (2005) ‘Determination of an optimal sequence using the psychoclonal algorithm’, *ImechE, Part B: Journal of Engineering Manufacture*, Vol. 219, pp. 137-149.

Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Schedulling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528*, Massechusetts Institute of Technology, MIT Laboratory for Computer Science.

Wang, X., Gao, X.Z. and Ovaska, S.J. (2004) ‘Artificial Immune Optimization Methods and Applications – A Survey’, *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4, pp. 3415-3420.

Solving the Response Time Variability Problem by means of a genetic algorithm

Article published as [European Journal of Operational Research, Volume 202, Issue 2, 16 April 2010, Pages 320-327] [DOI: http://dx.doi.org/10.1016/j.ejor.2009.05.024] © [copyright Elsevier]

Solving the response time variability problem by means of a genetic algorithm[†]

Alberto GARCÍA-VILLORIA[*]and Rafael PASTOR
Institute of Industrial and Control Engineering (IOC)
Universitat Politècnica de Catalunya (UPC)
{alberto.garcia-villoria / rafael.pastor}@upc.edu

Abstract. The response time variability problem (RTVP) is a hard scheduling problem that has recently been defined in the literature and has a wide range of real-world applications in mixed-model assembly lines, multithreaded computer systems, network environments and others. The RTVP arises whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the points at which they receive the necessary resources is minimized. Since the RTVP is a complex problem, heuristic and metaheuristic techniques are needed to solve it. The best results in the literature for the RTVP have been obtained with a psychoclonal algorithm. We propose a genetic algorithm (GA) that is adapted to solve the RTVP. A computational experiment is carried out and it is shown that, on average, the GA produces better results than the psychoclonal algorithm.

Keywords: response time variability, fair sequences, scheduling, genetic algorithm, evolution program, metaheuristics

1. Introduction

The concept of *fair sequence* has emerged independently from scheduling problems in diverse environments, principally from manufacturing, hard real-time systems, operating systems and network environments. The common aim of these scheduling problems, as defined in Kubiak (2004), is to build a fair sequence using n symbols, where symbol i (i = 1,...,n) must occur d_i times in the sequence. The fair sequence is the one which allocates a fair share of positions to each symbol i in any subsequence. This *fair* or *ideal* share of positions allocated to symbol i in a subsequence of length k is proportional to the relative importance (d_i) of symbol i with respect to the total copies of competing symbols (equal to $\sum_{i=1..n} d_i$). There is not a universal definition of fairness, as several reasonable metrics can be defined according to the specific problem considered.

[†] Supported by the Spanish Ministry of Education and Science under project DPI2007-61905, co-funded by the ERDF.
[*] Corresponding author: Alberto García-Villoria, Institute of Industrial and Control Engineering (IOC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; Tel.: +34 93 4054010; E-mail: alberto.garcia-villoria@upc.edu

Among the different definitions of fairness, the concept of *Response Time Variability* (RTV) has emerged. In RTV, the ideal distance for symbol i between any two consecutive copies of this symbol is equal to D/d_i, where D is the length of the sequence $\left(D = \sum_{i=1..n} d_i\right)$. The RTV metric is the sum, for all symbols i, of the squares of the differences between the ideal and the real distances corresponding to all pairs of consecutive copies of symbol i. Thus, the RTV metric measures the non-fairness of a sequence. The Response Time Variability Problem (RTVP) lies in finding the optimal sequence according to the RTV metric, that is, the sequence that minimizes the response time variability. Thus, the distance between any two consecutive copies of the same symbol should be as regular as possible (ideally constant).

This problem has a broad range of real-world applications. One of the first situations in which the idea of the fair sequence appeared was the sequencing of mixed-model assembly lines at Toyota Motor Corporation under the just-in-time (JIT) production system. One of the main aims of JIT is to eliminate sources of waste and inefficiency. In the case of Toyota, the main source of waste was the production of excessive volumes of stock. To solve this problem, JIT systems produce only the specific models required and in the quantities needed at any given time. According to Monden (1983), in this type of system the units should be scheduled in such a way that the consumption rates of the components in the production process remain constant. Miltenburg (1989) also studied this scheduling problem and considered only the demand rates for the models, thus defining the product rate variation (PRV) problem (Miltenburg, 1989; Kubiak, 1993). The PRV problem is intended to minimize variations in the production rate of different models. However, feedback from the manufacturing industry suggests that a good mixed-model sequence is one in which the distances between units of the same model are as regular as possible.

Apart from assembly lines, the fair sequencing idea has appeared in computer multithreaded systems (Waldspurger and Weihl, 1995; Dong et al., 1998). Multithreaded systems (operating systems, network servers, media-based applications, etc.) do different tasks to attend to the requests of client programs that take place concurrently. These systems need to manage scarce resources in order to service the requests of n clients. For example, multimedia systems must not display video frames too early or too late, as this would produce jagged motion perceptions (Corominas et al., 2007). Waldspurger and Weihl considered that resource rights could be represented by *tickets* and that each client i had a given number d_i of tickets. They suggested the Response Time Variability (RTV) metric to evaluate the fairness of a sequence of resource rights.

Other contexts in which the RTVP appears are the periodic machine maintenance problem (Anily et al., 1998), the schedule of commercial videotapes for television (Bollapragada et al., 2004; Brusco, 2008) and the schedule of waste collection (Herrmann, 2007). A study by Bollapragada et al. (2004) was motivated by a problem faced by the National Broadcasting Company (BNC), which is one of the main American firms in the television industry. Major advertisers buy hundreds of slots from the BNC to air commercials. The advertisers request that the airings of their commercials are as evenly spaced as possible over the broadcast season. Hermann (2007) came up with the RTVP while working with a healthcare facility that needed to schedule the collection of waste from waste collection rooms throughout the building. Based on data about how often a waste collector had to visit each room and in view of the fact that different rooms require a different number of visits per shift, the facility manager wanted these visits to occur as regularly as possible, so that excessive waste would not collect in any room. For instance, if a room needed four visits per eight-hour shift, it ideally had to be visited every two hours.

The RTVP is NP-hard (Corominas et al., 2007). To solve the RTVP, Waldspurger and Weihl (1995) used the Jefferson method of apportionment (Balinski and Young, 1982), a greedy heuristic algorithm which they renamed as the stride scheduling technique. Herrmann (2007) solved the RTVP by applying a heuristic algorithm based on the stride scheduling technique. Corominas et al. (2007) proposed four other greedy heuristic algorithms and a mixed-integer linear programming (MILP) model. Corominas et al. (2009) proposed an improved MILP model and increased the practical limit for obtaining optimal solutions from 25 to 40 copies to be scheduled. García et al. (2006) proposed six metaheuristic algorithms: a multi-start, a greedy randomized adaptive search procedure (GRASP) and four variants of a discrete particle swarm optimisation (PSO) algorithm. Another ten discrete PSO algorithms were proposed in García-Villoria and Pastor (2009a). A cross-entropy approach was used in García-Villoria et al. (2007). The electromagnetism-like mechanism was proposed to solve the RTVP in García-Villoria and Pastor (2009b). Finally, the best results recorded to date were obtained with a psychoclonal algorithm (García-Villoria and Pastor, 2008).

To improve the results obtained in prior studies, we propose using a genetic algorithm (GA)-based approach to solve the RTVP. GA is a well known metaheuristic that was proposed in the 1970s (Holland, 1975) and has proved to be very effective in solving hard optimisation problems. We adapt the GA by defining a suitable representation of the solutions and genetic operators for the RTVP. The proposed GA algorithm is compared with the most efficient procedure for solving non-small instances published in the literature, which is a psychoclonal algorithm proposed in García-Villoria and Pastor (2008). On average, the proposed GA algorithm improves the best previous results reported in the literature by more than 20%.

The remainder of the paper is organized as follows: Section 2 presents a formal definition of the RTVP and briefly describes the psychoclonal algorithm for solving the problem. Section 3 contains an introduction to GAs. Section 4 proposes a GA procedure for solving the RTVP. Section 5 presents the computational experiment and the comparison between our algorithm and the psychoclonal algorithm. Finally, some conclusions are given in Section 6.

2. The Response Time Variability Problem (RTVP)

The RTVP is designed to minimize variability in the distances between any two consecutive copies of the same symbol and is formulated as follows. Let n be the number of symbols, d_i the number of copies of the symbol i to be scheduled (i = 1,…,n), and D the total number of copies ($D = \sum_{i=1..n} d_i$). Let s be a solution of an instance in the RTVP. This consists of a circular sequence of copies ($s = s_1 s_2 \dots s_D$), where s_j is the copy sequenced in position j of sequence s. For all symbols i such that $d_i \geq 2$, let t_k^i be the distance between the positions in which copies k + 1 and k of symbol i are found (i.e. the number of positions between them, where the distance between two consecutive positions is considered equal to 1). Since the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_i}^i$ is the distance between the first copy of symbol *i* in a cycle and the last copy of the same symbol in the preceding cycle. Let $\bar{t}_i$ be the desired average distance between two consecutive copies of symbol *i* ($\bar{t}_i = D/d_i$). For all symbols *i* such that $d_i = 1$, t_1^i is equal to $\bar{t}_i$. The aim is to minimize the metric RTV, which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2 \tag{1}$$

For example, let $n = 3$, $d_A = 3$, $d_B = 2$ and $d_C = 2$; thus, $D = 7$, $\bar{t}_A = 7/3$, $\bar{t}_B = 7/2$ and $\bar{t}_C = 7/2$. Any sequence that contains exactly d_i times the symbol i $(\forall i)$ is a feasible solution. For example, the sequence (A, B, A, C, B, A, C) is a feasible solution, where:

$$RTV = \left(\left(2 - 7/3\right)^2 + \left(3 - 7/3\right)^2 + \left(2 - 7/3\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) = 5/3$$

As introduced in Section 1, the psychoclonal algorithm proposed in García-Villoria and Pastor (2008) is the best procedure to date for solving the RTVP. This algorithm is an evolutionary metaheuristic (as is the GA) that was first proposed in Tiwari et al. (2005). According to the authors, this metaheuristic inherits its characteristics from the need hierarchy theory of Maslow (1954) and the clonal selection principle (Gaspar and Collard, 2000). The basic scheme of the psychoclonal metaheuristic is the following: 1) an initial population of solutions is generated and a function is given to evaluate the fitness of a solution; 2) the best solutions are selected and cloned in a number proportional to their fitness; 3) the generated clones are hypermutated (hypermutation is similar to the mutation operator of genetic algorithms, but the difference lies in the fact that the modification rate of the hypermutation is inversely proportional to the fitness of the solution); 4) the new generation is formed by the best clones and by new solutions generated at random; 5) steps 2-4 are repeated until a stop condition is reached. This metaheuristic was adapted to solve the RTVP (for a more detailed explanation, see García-Villoria and Pastor, 2008).

3. Genetic Algorithms

Genetic algorithms (GAs) are metaheuristic procedures based on the principles of natural selection and sexual reproduction. The first GA was proposed in Holland (1975). Prior to that, in the 1960s, some optimisation techniques were proposed that have in common the concepts of *selection* and *mutation*, which can be considered as straightforward developments of hill-climbing methods. The new concept introduced by Holland was the idea of *recombination* of solutions.

GAs are now commonly used to solve optimisation problems (Reeves, 2003), although Holland's original work did not emphasize the use of GAs for optimising objective functions. GAs have been applied successfully to all kind of optimisation problems (Michalewicz, 1996) and the number of GA applications that have been reported in the literature to solve combinatorial optimisation problems (such as the RTVP) has grown exponentially (Reeves, 2003).

The classical scheme of a GA is shown in Figure 1. First an initial population of *chromosomes* is generated, each of which represents a solution of the problem. A chromosome is composed of simple elements called *genes*. A fitness function is used to evaluate the fitness of the chromosomes. Then, a new population that evolves towards better chromosomes is iteratively generated from the current one until a stop condition is reached (convergence of the population, maximum computing time, etc.). The key to produce better chromosomes consists of two chromosome operators called *crossover* and *mutation*. Crossover combines parent chromosomes to generate offspring chromosomes that share some features taken from each parent. The selection of the parents depends on their fitness. The aim of the crossover is to form a new population with a higher proportion of the characteristics of the good chromosomes of the previous population (Beasley et al., 1993a). Mutation is applied to the offspring chromosomes and consists of modifications to the values of several genes selected at random. Mutation diversifies the current population, and thus prevents premature convergence (Bean, 1994; Carter and Ragsdale, 2006).

```
1. Current population = Generate the initial population of chromosomes
2. Evaluate the fitness of each chromosome
3. While stopping condition is not reached do:
4.         New population = Ø
5.         While new population is not full do:
6.              Select two parent chromosomes according to their fitness from the
                current population
7.              Apply crossover to the parents to obtain two offspring chromosomes
8.              Apply mutation to the obtained offspring chromosomes
9.              Add the generated offspring to the new population
10.        End While
11.        Current population = new population
12. End While
```

Figure 1. Scheme of a classical GA

In the early stages of GAs, they were designed as generic tools for solving complex problems. To achieve this, the data structures of the chromosomes were fixed-length binary strings (sequences of 0s and 1s) and standard genetic operators were used. The advantages of this approach are problem domain independence, which allows applications to be developed easily, and the existence of a theoretical basis (Michalewicz, 1996). However, most researchers have used modified GAs with more powerful data structures that are adapted for real problems (Koza, 1990). To use a special representation, suitable genetic operators (crossover and mutation) must be defined that are adapted to work with this structure. The adaptation is usually performed by analogy with classical crossover and mutation. Although there is a poor theoretical basis for modified GAs, the advantage of incorporating problem-specific knowledge into the chromosomes representation and the genetic operators is that, in practice, modified GAs outperform classical GAs when they are used in real-world problems (Beasley et al., 1993b; Michalewicz, 1996).

Thus, a modified GA is proposed in this paper to solve the RTVP (see Section 4). When the designed algorithm uses a structure and operators that are different from the classical ones, some authors call it an *evolution program* (EP) (Michalewicz, 1996). We will consider our algorithm as a GA, only because the term GA is more popular (for a broad explanation of GAs and EPs, see Michalewicz (1996)).

4. Using a GA to solve the RTVP

In this paper we propose a modified GA for solving the RTVP. Seven elements have to be designed: 1) the representation of solutions (the data structure of the chromosomes); 2) the generation of the initial population of solutions; 3) the fitness function; 4) crossover; 5) mutation; 6) the generation of the offspring population; and 7) the stop condition. The proposed designs of these 7 elements are explained in the following subsections. Moreover, the fine-tuning of the parameter values of the GA algorithm is explained in Subsection 4.8.

4.1. Representation of solutions

A straightforward representation of a solution consists of the positions of the copies of each symbol to be sequenced. Thus, each copy has a gene associated with it and the value of each gene indicates the position of its associated copy. The building block hypothesis (Goldberg, 1989) is considered to code the chromosomes. This hypothesis states that a successful coding scheme is one that encourages the formation of *building blocks*. A building block is a list of consecutive genes that work well together.

To code the chromosomes of our GA, we decided to have a building block for each symbol *i* formed by the genes that indicate the positions in the sequence of the copies of symbol *i*. We selected this method as the quality of a solution depends on the response time variability for each symbol *i* (see Equation 1), which depends on the relative distances between the units of symbol *i*. Therefore, the positions of the units of symbol *i* have to be considered together.

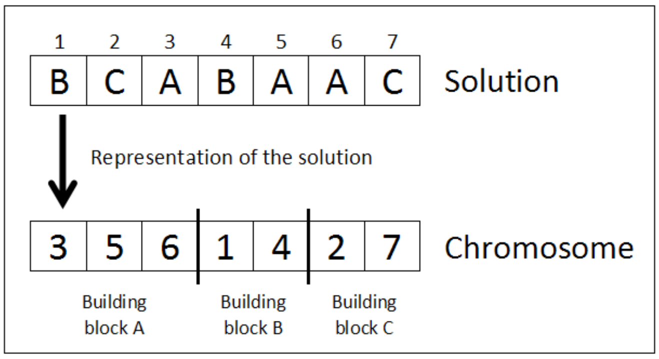


Figure 2. Representation of a RTVP solution

We will explain the representation of a solution with the following example: $n = 3$, $d_A = 3$, $d_B = 2$ and $d_C = 2$. A feasible solution is (B, C, A, B, A, A, C). The first three genes of a chromosome form the building block of symbol A, and their values are the positions of the first, second and third copies of symbol A (see Figure 2). The fourth and fifth genes form the building block of symbol B, and their values are the positions of the first and second copies of symbol B (see Figure 2). The sixth and seventh genes form the building block of symbol C, and their values are the positions of the first and second copies of symbol C (see Figure 2).

Note that the solution space using the proposed representation is not all the space of permutations. For instance, using the same example, the chromosome (5,3,6 | 1,4 | 2,7) is unfeasible, because it indicates that the position of the first copy of symbol A (which is 5) is greater than the position of the second copy of symbol A (which is 3). That is, the second copy of symbol A is sequenced before the first copy of symbol A, and this is incoherent with the definition of first copy and second copy.

4.2. Generation of the initial population

The initial population is made up of the chromosomes that represent solutions generated at random. Each solution is generated as follows. For each position of the sequence, a symbol to be sequenced is chosen at random. The probability of each symbol is equal to the number of copies of this symbol that remain to be sequenced, divided by the total number of copies that remain to be sequenced. The total number of chromosomes that make up the population is *N* (which is a parameter of the GA).

4.3. Fitness function

The fitness of a chromosome is only used in our GA to rank the chromosomes (see Section 4.6). Thus, an easy implementation of the fitness function is the inverse of the RTV value of the solution represented by the chromosome.

4.4. Crossover operator

The offspring obtained by applying the classical crossover operator in a permutation search space is usually unfeasible. To solve this difficulty, the partially matched crossover (PMX) has been successfully applied (e.g., Wu et al., 2007) since it was first proposed in Goldberg and Lingle (1985).

PMX cross two parent chromosomes as follows. First, two cut points are chosen at random along the chromosomes. Next, the section between these points defines an interchange mapping. Figure 3 shows an example of the application of the standard PMX.

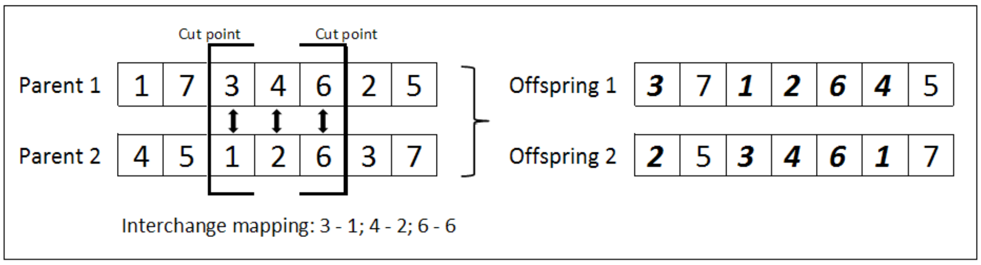


Figure 3. Application of the standard PMX

In this paper, a variation of the standard PMX is proposed that is adapted to the representation of the solutions. The first difference is that the proposed PMX selects a complete building block at random, instead of two random cut points, with the aim of ensuring the preservation of a good building block. The second difference is that a feasibility post-process is needed, as unfeasible offspring may be produced. As seen in the example in Section 4.1, the chromosome (5,3,6 | 1,4 | 2,7) is unfeasible because the second copy of symbol A is sequenced before the first copy of symbol A. To repair the chromosome, the genes of each building block are arranged in increasing order. In this example, the repaired chromosome is (3,5,6 | 1,4 | 2,7). Figure 4 shows a complete example of the application of the proposed PMX variation.

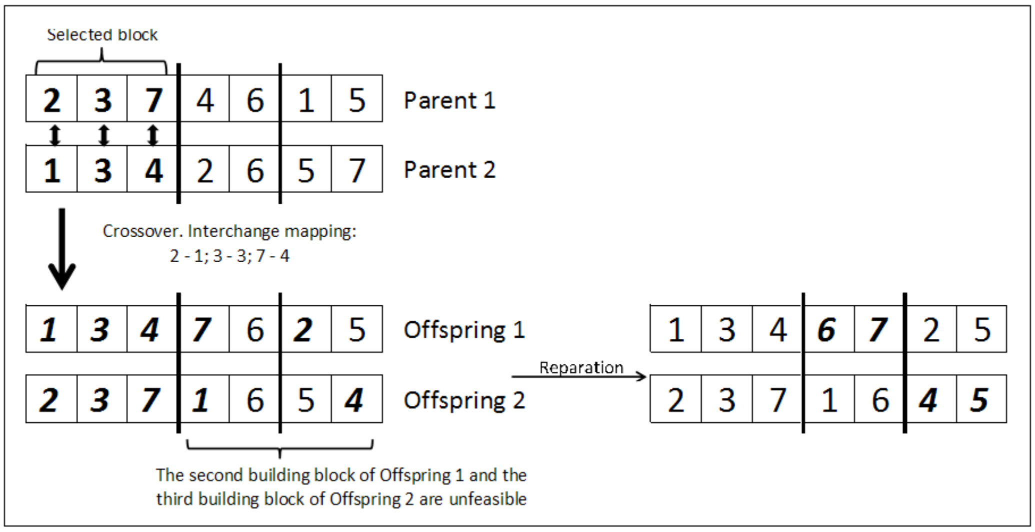


Figure 4. Application of the adapted PMX

4.5. Mutation operator

The mutation operator designed for our GA is analogous to the classical GA mutation. The proposed mutation operates as follows. Each gene has a probability *p* (which is a parameter of the GA) to mutate. If a gene is mutated, then there is a probability of 0.5 to increase its value by one (if the value is the length of the chromosome, then its value is changed to 1) and there is a probability of 0.5 to decrease its value by one (if the value is 1, then its value is changed to the length of the chromosome). Let *v* and *v*' be the original and the new value of the mutated gene, respectively. The value of the other gene whose value is *v*' is changed to *v*. After the mutations of the chosen genes, the repairing post-process used in the proposed crossover operator (see Section 4.4) is also needed. Figure 5 shows an example of the proposed mutation operator.

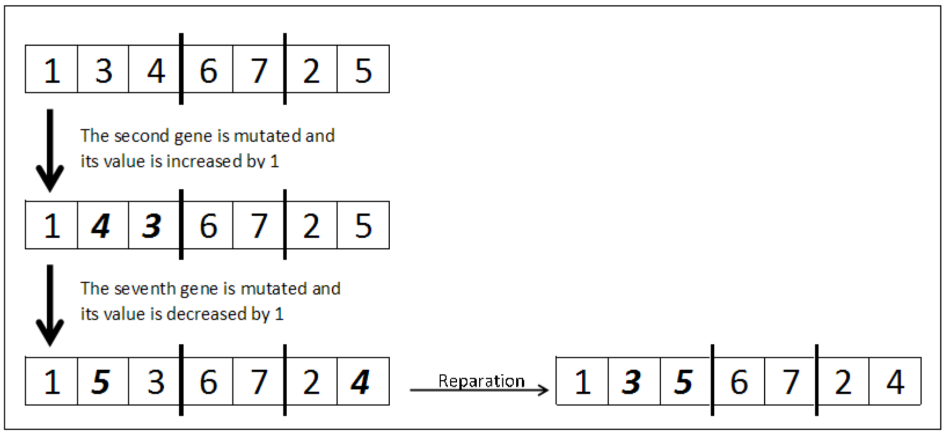

1 3 4 6 7 2 5
The second gene is mutated and its value is increased by 1
1 4 3 6 7 2 5
The seventh gene is mutated and its value is decreased by 1
1 5 3 6 7 2 4
Reparation
1 3 5 6 7 2 4

Figure 5. Application of the mutation operator

4.6. Generation of the offspring population

The classical generation of the offspring population is shown in Steps 4-9 of Figure 1. In this case, all chromosomes of the new population are obtained from the parent population by crossover and mutation. Several new ideas have been introduced in the literature to improve this traditional reproduction.

The *elitist strategy* proposed in Goldberg (1989) involves copying the best chromosomes from the parent population to the offspring population. The advantage of the elitist strategy is that the best solution monotonically improves from one generation to the next (Bean, 1994). However, this strategy has the disadvantage of premature convergence of the population. To avoid this, Bean (1994) employs the idea of *immigration*. This consists of including several new chromosomes that are generated at random in the offspring population.

The elitist strategy and the idea of immigration are used in our proposed GA. The proportion of best parent chromosomes (*B)* and the proportion of new chromosomes (*R*) introduced into the offspring population are parameters of our GA. Figure 6 shows a scheme of the generation of the offspring.

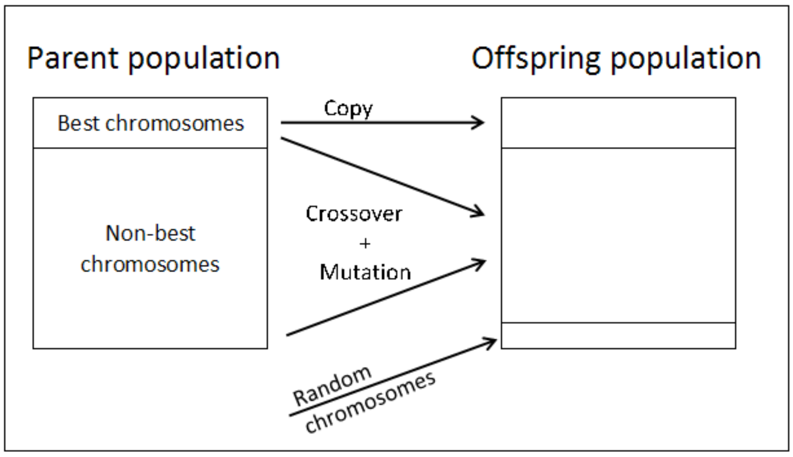


Figure 6. Generation of the offspring population

Parents are selected for the crossover as follows. One parent is chosen at random with a uniform probability from the best chromosomes population and the other parent is chosen at random with a uniform probability from the non-best population. The fitness function defined in Section 4.3 has been used to rank the chromosomes of the current population.

4.7. Stop condition

The GA algorithm stops once it has run for a preset time.

4.8. Fine-tuning the algorithm parameters

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although the parameter values may have a very strong effect on the results of the metaheuristic for each problem, they are often selected using one of the following methods, which are not sufficiently thorough (Eiben et al., 1999; Adenso-Díaz and Laguna, 2006): 1) "by hand", based on a small number of experiments that are not referenced; 2) using the general values recommended for a wide range of problems; 3) using the values reported to be effective in other similar problems; or 4) with no apparent explanation.

Adenso-Díaz and Laguna (2006) proposed a new technique called CALIBRA for fine-tuning the parameters of heuristic and metaheuristic algorithms. CALIBRA is based on using conjointly Taguchi's fractional factorial experimental designs and a local search procedure.

García-Villoria and Pastor (2008) used CALIBRA to fine-tune their psychoclonal algorithm, and we used the same technique to fine-tune our GA algorithm. The following parameter values were obtained: N (size of the population) = 13, p (mutation probability) = 0.013, B (proportion of best chromosomes) = 0.18 and R (proportion of new chromosomes) = 0.12.

5. Computational experiment

The psychoclonal algorithm proposed in García-Villoria and Pastor (2008) is the most efficient algorithm in the literature for solving non-small RTVP instances. Therefore, we compared the performance of our proposed GA algorithm with that psychoclonal algorithm. In the rest of this section, we refer to our GA algorithm as *GA-RTVP* and the psychoclonal algorithm as *Psycho-RTVP*.

The computational experiment was carried out for the same instances and conditions that were used in García-Villoria and Pastor (2008). That is, the algorithms were run for 740 instances, which were grouped into four classes (185 instances in each class) according to size. The instances in the first class (*CAT1*) were generated using a random value of D (number of copies) distributed uniformly between 25 and 50, and a random value of n (number of symbols) distributed uniformly between 3 and 15; for the second class (*CAT2*), D was between 50 and 100 and n between 3 and 30; for the third class (*CAT3*), D was between 100 and 200 and n between 3 and 65; and for the fourth class (*CAT4*), D was between 200 and 500 and n between 3 and 150. For all instances and for each symbol $i = 1,\ldots,n$, a random value of d_i (number of copies of symbol i) was between 1 and $\left|(D-n+1)/2.5\right|$ so that $\sum_{i=1..n} d_i = D$. The two algorithms were coded in Java and the computational experiment was carried out using a 3.4 GHz Pentium IV with 1.5 GB of RAM.

The algorithms were run for 50 seconds for each instance. Table 1 shows the average RTV values to be minimized for the global of 740 instances and for each class of instances (*CAT1* to *CAT4*) obtained with the two algorithms.

Table 1. Average RTV values for a computing time of 50 seconds

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
GA-RTVP	**186.94**	11.65	29.41	84.54	622.16
Psycho-RTVP	**235.68**	14.92	44.25	137.07	746.50

Table 1 shows that the global average results of the GA algorithm for all the instances considered are 20.68% better than the results obtained using the best method proposed in the literature. If we consider the results by class, the *GA-RTVP* also obtains better results than *Psycho-RTVP*: the results obtained with the GA algorithm are 21.92%, 33.54%, 38.32% and 16.66% better for *CAT1* instances, *CAT2* instances, *CAT3* instances and *CAT4* instances, respectively. Considerable improvements are observed in all classes. To analyse whether the differences are statistically significant we carried out the two sample test using the statistical software package Minitab® 15.1.0.0. With a confidence level above 99%, *GA-RTVP* is better than *Psycho-RTVP*, if we consider the overall results and the results by class.

Table 2 shows the number of times that each algorithm reaches the best RTV value obtained by either one. The results are shown for the total number of 740 instances and for each class.

Table 2. Number of times that the best solution is reached

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
GA-RTVP	**663**	171	175	172	145
Psycho-RTVP	**148**	72	23	13	40

As expected from the results in Table 1, Table 2 shows that *GA-RTVP* reaches the best solution on more occasions than *Psycho-RTVP*. For the total number of instances, the GA algorithm obtains the best solution in 89.59% of cases and the psychoclonal algorithm in 20.00%.

To complete the analysis of the results, we examined the relative discrepancies between the RTV values obtained for each algorithm and the best values obtained by both algorithms. A measure of the discrepancies (let it be called σ) of the RTV values obtained by each algorithm *alg* = {*GA-RTVP*, *Psycho-RTVP*} was defined for a given instance, *ins*, according to the following expression:

$$\sigma(alg,ins) = \left(\frac{\mathrm{RTV}_{ins}^{(alg)} - \mathrm{RTV}_{ins}^{(best)}}{\mathrm{RTV}_{ins}^{(best)}} \right)^2 \qquad (5)$$

where $\mathrm{RTV}_{ins}^{(alg)}$ is the RTV value of the solution obtained with the algorithm *alg* for the instance *ins*, and $\mathrm{RTV}_{ins}^{(best)}$ is the best RTV value of the solutions obtained with the two algorithms for the instance *ins*. Table 3 shows the average σ values for the total number of instances and for each class. Table 3 shows that both algorithms produce low average σ values for the total number of cases and for each instance class. That is, when an algorithm does not obtain the best RTV value for a given instance, it obtains a value that is very close to it. Although the behaviour of Psycho-RTVP is very stable, GA-RTVP improves it and has even more stable behaviour.

Table 3. Average σ values for the best solution found

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
GA-RTVP	**0.007**	0.009	0.006	0.004	0.010
Psycho-RTVP	**0.859**	0.750	0.697	1.323	0.664

A computing time of 50 seconds may not be long enough for the algorithms to converge for the largest instances. Table 4 shows the average RTV values for the total number of instances and for each class of instances (*CAT1* to *CAT4*) when the algorithms are run for 200, 400, 800 and 1,000 seconds.

Table 4. Average RTV values for a computing time of 200, 400, 800 and 1,000 seconds

		Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
200 s.	***GA-RTVP***	**131.81**	11.34	28.26	77.81	409.84
	Psycho-RTVP	**172.07**	14.92	41.54	131.67	500.16
400 s.	***GA-RTVP***	**117.93**	11.10	27.72	76.21	356.69
	Psycho-RTVP	**164.96**	14.92	40.61	128.03	476.28
600 s.	***GA-RTVP***	**112.28**	10.95	27.56	75.53	335.06
	Psycho-RTVP	**163.19**	14.92	40.22	125.85	471.80
800 s.	***GA-RTVP***	**109.00**	10.95	27.18	75.17	322.70
	Psycho-RTVP	**162.28**	14.92	40.01	123.89	470.29
1,000 s.	***GA-RTVP***	**106.68**	10.92	27.00	74.86	313.92
	Psycho-RTVP	**161.60**	14.90	39.90	122.38	469.23

When a computing time of 1,000 seconds is used—which seems to be long enough for both algorithms to converge (see Figure 7)—the *GA-RTVP* algorithm is 33.99% better than *Psycho-RTVP* for the total number of instances. If we consider the results by class, *GA-RTVP* is 26.71%, 32.33%, 38.83% and 33.10% better than the *Psycho-RTVP* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. Again, *GA-RTVP* obtains better results than *Psycho-RTVP*, considering the overall results and the results by class with a confidence level above 99%.

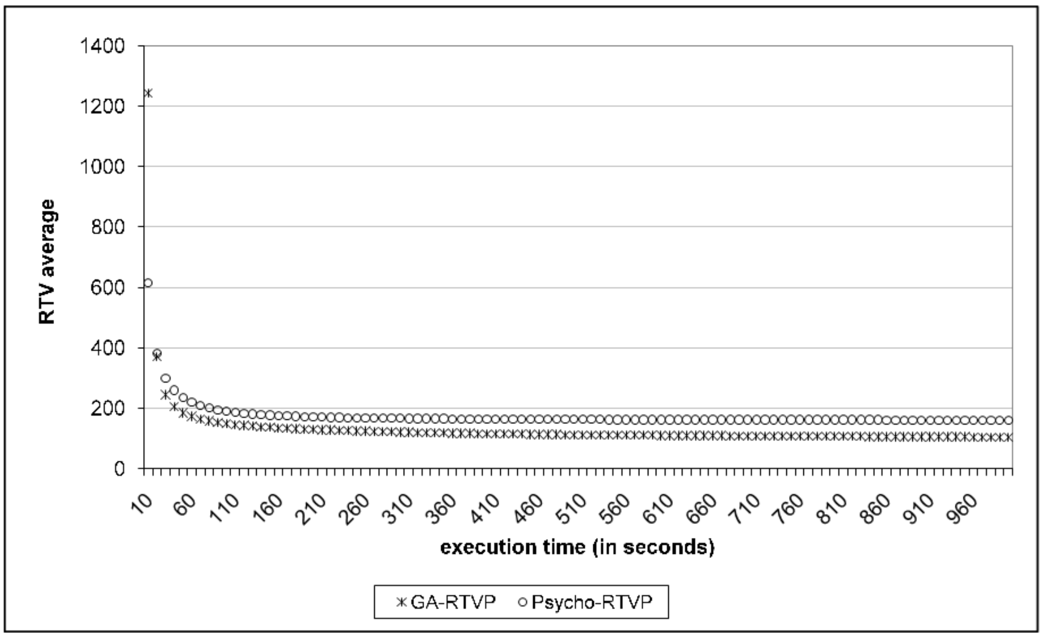


Figure 7. Average RTV values over the computing time

Finally, the real-life waste collection example presented by Herrmann (2007) was solved using *GA-RTVP* and *Psycho-RTVP*. This example has the following characteristics: $n = 14$, $d = (2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5)$ and, therefore, $D = 46$. This example was solved optimally with the MILP presented in Corominas et al. (2006). Both algorithms were run ten times and they always found an optimal solution. The minimum, average and maximum computing times needed by *GA-RTVP* to find the optimum were 0.375, 0.994 and 3.234 seconds, respectively. The minimum, average and maximum computing times needed by *Psycho-RTVP* to find the optimum were 0.687, 4.139 and 9.688 seconds, respectively.

6. Conclusions

In this paper, the response time variability problem (RTVP) is solved. This scheduling problem arises in a variety of real-world environments, including mixed-model assembly lines, multithreaded systems, network servers, periodic machine maintenance, and waste collection. The aim of the RTVP is to minimize the variability in the distances between any two consecutive copies of the same symbol.

Since the RTVP is an NP-hard problem, heuristic and metaheuristic methods are needed to solve real-world instances. Several metaheuristic algorithms have been developed to solve this hard combinatorial optimisation problem. The most efficient algorithm to date for solving the RTVP was a psychoclonal algorithm (García-Villoria and Pastor, 2008). A GA adapted to solve the RTVP was used to improve the published results. The computational experiment showed that the proposed GA improves on the best results obtained in the literature.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to Professor Albert Corominas (Universitat Politècnica de Catalunya) and to the anonymous reviewers for their valuable comments, which have helped to improve the quality of this paper.

REFERENCES

Adenso-Díaz, B. and Laguna, M. (2006) ‘Fine-tuning of algorithms using fractional experimental designs and local search’, *Operations Research*, Vol. 54, pp. 99-114.

Anily, S., Glass, C.A. and Hassin, R. (1998) ‘The scheduling of maintenance service’, *Discrete Applied Mathematics*, Vol. 82, pp. 27-42.

Balinski, M.L. and Young, H.P. (1982) *Fair Representation*, Yale University Press, New Haven.

Bean, J.C. (1994) ‘Genetic Algorithms and Random Keys for Sequencing and Optimization’, *ORSA Journal on Computing*, Vol. 6, pp. 154-160.

Beasley, D., Bull, D.R. and Martin, R.R. (1993a) ‘An Overview of Genetic Algorithms: Part 1, Fundamentals’, *University Computing*, Vol. 15, pp. 58-69.

Beasley, D., Bull, D.R. and Martin, R.R. (1993b) ‘An Overview of Genetic Algorithms: Part 2, Research Topics’, *University Computing*, Vol. 15, pp. 170-181.

Bollapragada, S., Bussieck, M.R. and Mallik, S. (2004) ‘Scheduling Commercial Videotapes in Broadcast Television’, *Operations Research*, Vol. 52, pp. 679-689.

Brusco, M.J. (2008) ‘Scheduling advertising slots for television’, *Journal of the Operational Research Society*, Vol. 59, pp. 1363-1372.

Carter, A.E. and Ragsdale, C.T. (2006) ‘A new approach to solving the multiple travelling salesperson problem using genetic algorithms’, *European Journal of Operational Research*, Vol. 175, pp. 246-257.

Corominas, A., Kubiak, W. and Moreno, N. (2007) ‘Response time variability’, *Journal of Scheduling*, Vol. 10, pp. 97-110.

Corominas, A., Kubiak, W. and Pastor, R. (2009) ‘Mathematical Programming Modeling of the Response Time Variability Problem’, *European Journal of Operational Research*, doi: 10.1016/j.ejor.2009.01.014.

Dong, L., Melhem, R. and Mosse, D. (1998) ‘Time slot allocation for real-time messages with negotiable distance constrains requirements’, *Fourth IEEE Real-Time Technology and Applications Symposium* (RTAS'98), Denver, CO. pp. 131-136.

Eiben, A.E., Hinterding, R. and Michalewicz, Z. (1999) ‘Parameter control in evolutionary algorithms’, *IEEE Transactions on evolutionary computation*, Vol. 3, pp. 124-141.

García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics’, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 146, pp. 187-194.

García-Villoria, A., Pastor, R. and Corominas, A. (2007) ‘Solving the Response Time Variability Problem by means of the Cross-Entropy Method’, *International Journal of Manufacturing Technology and Management* (to be published).

García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of a psychoclonal approach’, *Journal of Heuristics*, in press, corrected proof, available online, 16 July 2008, doi:10.1007/s10732-008-9082-2.

Gaspar, A. and Collard, P. (2000) ‘Two models of immunization for time dependent optimization’, in *Proceeding of the IEEE International Conference on Systems Manufacturing and Cybernetics*, pp. 113-118.

García-Villoria, A. and Pastor, R. (2009a) ‘Introducing dynamic diversity into a discrete particle swarm optimization’, *Computers & Operations Research*, Vol. 36, pp. 951-966.

García-Villoria, A. and Pastor, R. (2009b) ‘Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism’, *International Journal of Production Research*, doi: 10.1080/00207540902862545.

Goldberg, D.E. (1989) *Genetic Algorithms in search, optimization and machine learning*, Addison-Wesley.

Goldberg, D.E. and Lingle, R. (1985) *Alleles, Loci, and the Traveling Salesman Problem*, Proceeding of the First International Conference on Genetic Algorithms, pp. 154-159.

Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report*, University of Maryland, USA.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, MIT Press.

Koza, J.R. (1990) ‘Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems’, *Report STAN-CS-90-1314*, Stanford University.

Kubiak, W. (1993) ‘Minimizing variation of production rates in just-in-time systems: A survey’, *European Journal of Operational Research*, Vol. 66, pp. 259-271.

Kubiak, W. (2004) ‘Fair Sequences’, Chapter 19 in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapman and Hall.

Maslow, A.H. (1954) *Motivation and personality*, New York: Harper & Bros.

Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag. 3rd edition.

Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 35, pp. 192-207.

Monden, Y. (1983) ‘Toyota Production Systems’, *Industrial Engineering and Management Press*, Norcross, GA.

Reeves, C. (2003) ‘Genetic Algorithms’, Chapter 3 in *Handbook of Metaheuristics*, Eds. Glover and Kochenberger, Kluwer Academic Publishers, pp. 56-82.

Tiwari, M.K., Prakash, A., Kumar, A. and Mileham, A.R. (2005) ‘Determination of an optimal sequence using the psychoclonal algorithm’, *ImechE, Part B: Journal of Engineering Manufacture*, Vol. 219, pp. 137-149.

Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Scheduling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528*, Massachusetts Institute of Technology, MIT Laboratory for Computer Science.

Wu, X., Chu, C.-H., Wang, Y. and Yan, W. (2007) ‘A genetic algorithm for cellular manufacturing design and layout’, *European Journal of Operational Research*, Vol. 181, pp. 156-167.

Annex A2. Other works

A2.1. Articles submitted to journals included in the JCR index which are in process of review

Hyper-heuristic Approaches for the Response Time Variability Problem

Article submitted to European Journal of Operational Research (2nd review in progress)

Hyper-heuristic Approaches for the Response Time Variability Problem[†]

Alberto GARCÍA-VILLORIA[a*], Said SALHI[b], Albert COROMINAS[a], and Rafael PASTOR[a]
[a] Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
[b] The Centre for Logistics & Heuristic Optimisation (CLHO), Kent Business School, University of Kent at Canterbury, Canterbury CT2 7PE, UK
s.salhi@kent.ac.uk, {alberto.garcia-villoria / albert.corominas / rafael.pastor}@upc.edu

Abstract. We propose two classes for the implementation of hyper-heuristic algorithms. The first is based on constructive heuristics, whereas the second uses improvement methods. Within the latter class, a general framework is designed for the use of local search procedures and metaheuristics as low-level heuristics. A dynamic scheme to guide the use of these approaches is also devised. These ideas are tested on a hard scheduling problem known as the Response Time Variability Problem (RTVP). An intensive computational experiment shows the effectiveness of the proposed hyper-heuristics and their ability to select the right low-level heuristic at a given iteration during the search.

Keywords: hyper-heuristics, metaheuristics, response time variability, scheduling, fair sequences

1. Introduction

One of the most commonly used approaches to NP-hard optimisation problems is heuristics-based methods. Since each heuristic may have particular weaknesses and strengths depending on the characteristics of the instance or scenario in which it is applied (Bai *et al*., 2008), it seems reasonable to choose the *right* heuristic at a given iteration during the search. This is the key idea of hyper-heuristic methods.

[†] Supported by the Spanish Ministry of Education and Science under project DPI2007-61905 and co-funded by the ERDF. In addition, this study was supported by the Department of Innovation, Universities and Enterprise of the Generalitat de Catalunya, under grant BE-DGR-2008.
[*] Corresponding author: Alberto García-Villoria, Institute of Industrial and Control Engineering (IOC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; Tel.: (+34) 93 4054010; alberto.garcia-villoria@upc.edu. The research was conducted while visiting CLHO at Kent University.

Hyper-heuristics are an emerging methodology in search and optimisation (Burke *et al.*, 2003a). However, references to them can be found in the literature from the 1960s onwards (Ross, 2005). A short definition of hyper-heuristic methods is “heuristics to choose heuristics”. Hyper-heuristics apply the right heuristic during the problem solving process, according to the current state of the solution. Thus, an intelligent application of different heuristics at different times in the search could lead to better performance than the application of individual heuristics (Burke *et al.*, 2006). Some hyper-heuristics work on rules that are often extracted from offline learning and knowledge acquisition. This type of hyper-heuristics could be classified as an expert system.

Hyper-heuristics operate indirectly on the solutions by choosing the (meta)heuristic to be applied. They thus operate at a higher level than classical heuristics and metaheuristics. In fact, hyper-heuristics only have access to a set of low-level (meta)heuristics that are applied to the current solution. The main advantage of this is that hyper-heuristics can be designed independently of the problem domain. Thus, given a problem, a set of (meta)heuristics and a suitable fitness function, the same hyper-heuristic can be applied (Burke *et al.*, 2003a). In other words, an existing hyper-heuristic method can be applied to a new problem quickly and cheaply. For an overview of hyper-heuristics, see Burke *et al.* (2003a), Ross (2005) and Özcan *et al.* (2008), and for information on heuristics in general, see Salhi (2006).

Hyper-heuristics can be divided into two categories: *constructive* hyper-heuristics and *improvement* hyper-heuristics. Constructive hyper-heuristics use a set of constructive heuristics as the low-level heuristics, in order to construct a full solution. In contrast, improvement hyper-heuristics start from a complete initial solution and then improve on it, using either simple refinement heuristics or even more sophisticated, but time-consuming, metaheuristics. An extended classification can be found in Burke *et al.* (2009).

This paper proposes several constructive and improvement hyper-heuristics that can be applied to a variety of hard combinatorial optimisation problems. We use the Response Time Variability Problem (RTVP) as a platform to test this methodology. The RTVP is an NP-hard scheduling problem that was first reported in Waldspurger and Weihl (1994) and formally formulated by Corominas *et al.* (2007). This problem has a wide range of real-world applications: it occurs whenever products, clients or jobs need to be sequenced in such a way that there is minimal variability in the time between the instants at which the necessary resources are received. Applications include sequencing of mixed-model assembly lines under JIT (Kubiak, 1993; Miltenburg, 1989), resource allocation in computer multi-threaded systems such as operating systems, network servers and media-based applications (Dong *et al.*, 1998; Waldspurger and Weihl, 1994, 1995), the periodic machine maintenance problem when the times between consecutive services of the same machine are equal (Anily *et al.*, 1998; Wei and Liu, 1983), waste collection (Herrmann, 2007) and scheduling commercial videotapes for television (Bollapragada *et al.*, 2004; Brusco, 2008).

The set of low-level heuristics used in the proposed constructive hyper-heuristics consists of several greedy heuristics put forward by Bollapragada *et al.* (2004) and Corominas *et al.* (2007). For the improvement hyper-heuristic, we introduce local search procedures that are commonly applied, as well as metaheuristics. To the best of our knowledge, there are no studies on the use of metaheuristics as low-level heuristics. We understand the reason for this lack of interest, as metaheuristics are known to consume a large amount of cpu time. However, since hyper-heuristics can select the right heuristic during the solution generation process, it seems likely that they are also able to select the most suitable metaheuristic, according to the current state of the solution. In this paper, two sets of low-level heuristics are proposed for the improvement hyper-heuristics. The first consists of the three local search methods used in the classical improvement hyper-heuristics, whereas the second contains a composite hill-climbing method, which is based on iteratively applying the three aforementioned local search methods, one variant of tabu search (TS) and one variant of variable neighbourhood search (VNS).

The remainder of the paper is organized as follows: Section 2 presents a formulation of the RTVP and a brief state of the art in this area. Section 3 explains the heuristics and metaheuristics that will be used in the proposed hyper-heuristics. Section 4 describes our methodology for constructive hyper-heuristics. Section 5 deals with improvement hyper-heuristics. Section 6 presents our computational experiments. Finally, some conclusions and suggestions are given in Section 7.

2. The Response Time Variability Problem (RTVP)

The RTVP is formulated as follows. Let n be the number of symbols to be sequenced (representing products, clients, jobs, etc.), where symbol i (i = 1,...,n) is to be copied d_i times in the sequence (the number of times that symbol i has to receive the resource) and D is the total number of copies ($\sum_{i=1..n} d_i$). Let s be a solution of an instance in the RTVP that consists of a circular sequence of copies ($s = s_1 s_2 \dots s_D$), where s_j is the copy sequenced in position j of sequence s. For each symbol i in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the copies k + 1 and k of symbol i are found. We consider that the distance between two consecutive positions is equal to 1. Since the sequence is circular, position 1 comes immediately after the last position D. Therefore, $t_{d_i}^i$ is the distance between the first copy of symbol i in a cycle and the last copy of the same symbol in the preceding cycle. Let $\bar{t}_i$ be the desired average distance between two consecutive copies of symbol i ($\bar{t}_i = D/d_i$). The objective is to minimise the metric called the response time variability (RTV), which is defined by the sum of the square errors with respect to the $\bar{t}_i$ distances. Since the symbols i such that $d_i = 1$ do not intervene in the computation of RTV, we assume that for each of these symbols t_1^i is equal to $\bar{t}_i$. Thus, RTV is given by the following expression as $RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2$.

As an illustration, consider the following example. Let $n = 3$ with symbols A, B and C. Also consider $d_A = 2$, $d_B = 2$ and $d_C = 4$. Thus, $D = 8$, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\bar{t}_C = 2$. Any sequence that contains symbol i $(\forall i)$ exactly d_i times is a feasible solution. For example, the sequence (*C*, *A*, *C*, *B*, *C*, *B*, *A*, *C*) is a feasible solution, and has an $RTV = \left(\left(5-4\right)^2 + \left(3-4\right)^2\right) + \left(\left(2-4\right)^2 + \left(6-4\right)^2\right) + \left(\left(2-2\right)^2 + \left(2-2\right)^2 + \left(3-2\right)^2 + \left(1-2\right)^2\right) = 1$

Corominas *et al*. (2007) proposed a mixed integer lineal programming (MILP) model to solve the RTVP. Corominas *et al*. (2009a) then improved the previous MILP model to obtain optimal sequences up to 40 copies. Bollapragada *et al*. (2004) proposed a simple branch and bound algorithm and four heuristics. Five heuristics were presented in Corominas *et al*. (2007), of which three were greedy. García *et al*. (2006) developed six metaheuristic algorithms which include: a multi-start, a greedy randomized adaptive search procedure (GRASP) and four variants of a discrete particle swarm optimization (PSO) algorithm. Other discrete PSO algorithms were proposed in García-Villoria and Pastor (2009a). A multi-start algorithm and a GRASP algorithm were given by Corominas *et al*. (2008). Cross-entropy, psychoclonal and electromagnetism-like mechanism approaches were used in García-Villoria *et al*. (2007), García-Villoria and Pastor (2008) and García-Villoria and Pastor (2009b), respectively. The best results have recently been achieved with three methods based on the TS metaheuristic (Corominas *et al*., 2009b), the VNS metaheuristic (Corominas *et al*., 2009c) and the genetic algorithm (GA) metaheuristic (García-Villoria and Pastor, 2009c).

3. Low-level heuristics for the RTVP

The heuristics and metaheuristics that will be used in our study to solve the RTVP are briefly reviewed here. Section 3.1 presents six greedy heuristics for generating a solution for the RTVP, whereas Section 3.2 describes three local search methods, one composite hill-climbing method and two metaheuristic algorithms. These are our low-level heuristics, which will be embedded into our hyper-heuristic approaches.

3.1. Greedy heuristics for the RTVP

3.1.1. Parametric methods of apportionment

The parametric method of apportionment is defined as follows (Balinski and Young, 1982). Let x_{il} be the number of copies of symbol *i* that have already been sequenced in the sequence of length *l*, *l* = 0, 1, … (assume $x_{i0} = 0$); the symbol to be sequenced in position *l* + 1 is $i^* = \arg\max_i \left\{ \frac{d_i}{x_{il} + \delta} \right\}$, where $\delta \in (0,1]$. If there is a tie, then use lexicographical order.

The Webster method (*Gr1*) consists of applying the above method with $\delta = 0.5$, whereas the Jefferson method (*Gr2*) uses $\delta = 1$. We also propose an application of the parametric method, with $\delta = 0.25$ and $\delta = 0.75$ (*Gr3* and *Gr4*, respectively). Corominas *et al.* (2007) used the Webster and Jefferson methods to solve the RTVP.

3.1.2. A priority-based rule heuristic (Gr5)

Let x_{il} be the number of copies of symbol *i* that have already been sequenced in the sequence of length *l*, *l* = 0, 1, … (assume $x_{i0} = 0$); the symbol to be sequenced in position $l + 1$ is $i^* = \arg\max_i \left\{ \frac{(l+1)\cdot d_i}{D} - x_{il} \right\}$. If there is a tie, then the symbol i with the lowest d_i is sequenced. If this leads to another tie, then use lexicographical order. This idea is taken from the priority rule used in the GRASP algorithm proposed in Corominas *et al*. (2008).

3.1.3. A contribution-based rule (Gr6)

The symbol to be sequenced in position l (l = 1, 2, …) is the one that contributes least to the objective function (the RTV value). If there is a tie, then use lexicographical order. This method was proposed in Bollapragada *et al*. (2004).

3.2. Local search methods, a composite hill-climbing method and metaheuristics for the RTVP

3.2.1. Local search methods

Three local search methods (*LS-1*, *LS-2* and *LS-3*) are proposed. The *LS-1* neighbourhood is generated by swapping each pair of two consecutive positions of the sequence in the current solution. The *LS-2* neighbourhood is a generalisation of *LS-1*, in which the move is not restricted to consecutive positions. The *LS-3* neighbourhood is generated by removing each member of the sequence from its current position and inserting it in all other possible positions in the sequence. These neighbourhoods are proposed in Corominas *et al*. (2009c). All local search methods are performed iteratively in their neighbourhood. The best solution in the neighbourhood is chosen at each iteration and the optimisation ends when no neighbouring solution is better than the current solution.

3.2.2. A composite hill-climbing method

The composite hill-climbing method (*CHC*) consists of applying iteratively the three local search procedures (*LS-1*, *LS-2* and *LS-3*) until there is no improvement in the solution.

The order in which the three local search procedures make up the composite heuristic *CHC* could be important in terms of solution quality and cpu time. The procedures are ordered according to increasing neighbourhood complexity (that is, *LS-1*, *LS-2* and *LS-3*), as the solution is improved rapidly through *LS-1* at the first iteration of *CHC*. Thus, there will be generally fewer iterations in *LS-2* and specifically in *LS-3* (which consumes much more cpu time than *LS-1*), given the good solution quality found by *LS-1* and *LS-2*, respectively. Consequently, the heuristic *CHC* can run using a relatively larger number of iterations.

3.2.3. Tabu search

The disadvantage of local search methods is that they get trapped in a local optimum. Tabu Search (TS) is one of the metaheuristics that has the power to overcome such a limitation (Glover, 1986). TS is a deterministic and aggressive approach that is based on applying a local search in which non-improving movements are also allowed. To avoid cycling back to visited solutions, the most recent history of the search is recorded in a list of tabu (forbidden) solutions. A tabu solution can be overridden if a suitable *aspiration criterion* is met (see Salhi (2002) for more details on this issue).

TS has been successfully applied to solve the RTVP (Corominas *et al*., 2009b). The neighbourhood used is that of the *LS-2* local search method. The aspiration criterion relates to the move which produces a better solution than the best one found so far. A forbidden move on the tabu list involves two pairs consisting of (position, symbol). For instance, the move [(3, *A*), (5, *B*)] means that all solutions with the symbol *A* sequenced in position 3 and symbol *B* sequenced in position 5 are considered tabu.

The TS algorithm has one parameter that is the size of its tabu list. The choice of the parameter value may have a crucial effect on the results of the metaheuristic. Thus, it is important to find a suitable value of the parameter. We used a new technique called CALIBRA (Adenso-Díaz and Laguna, 2006), which is specifically designed for fine-tuning the parameters of heuristic and metaheuristic algorithms. CALIBRA operates in the same set of training instances as those used by the training stage of *CHH-2* and *CHH-3* (see Subsection 4.2). The tabu list size found by CALIBRA was 75.

3.2.4. Variable neighbourhood search

The variable neighbourhood search (VNS) metaheuristic is based on applying a systematic change of neighbourhood within a local search method (Mladenović and Hansen, 1997).

The VNS metaheuristic has been shown to be very effective to solve the RTVP (Corominas *et al*., 2009c). We propose a straightforward application of the VNS to solve the RTVP. The neighbourhoods used, N_1, N_2 and N_3, are those of *LS-1, LS-2* and *LS-3*, respectively. The local search method applied is the same as in the tabu search proposed in Corominas *et al*. (2009b), that is, *LS-2*. The acceptance criterion is that the RTV value of $S' \in N_k(S)$ is equal to or lower than the RTV value of *S*.

Note that the proposed VNS algorithm is different from the hill-climbing method (*CHC*), although both algorithms employ the same neighbourhood operators. The following aspects of the VNS algorithm differs from *CHC*: 1) a random neighbour is obtained from the current one, according to the current neighbourhood structure, 2) *LS-2* is applied to this random neighbour, 3) if the local optimum obtained after applying *LS-2* is not worse than the current solution, the current neighbourhood is not changed; otherwise, the current neighbourhood is changed to the next one.

4. Constructive hyper-heuristic procedures

We propose four constructive hyper-heuristics, based on the general hyper-heuristic framework presented by Burke *et al*. (2003a), see Figure 1.

1. Start with a set of heuristics, each of which is applicable to a problem state and transforms it to a new problem state.
2. Let the initial problem state be S_0.
3. If the problem state is S_i then find the heuristic that is in some sense the most suitable for transforming that state. Apply it, to obtain a new state of the problem S_{i+1}.
4. If the problem is solved, stop; otherwise go to Step 3.

Figure 1. A general hyper-heuristic scheme

4.1. A crude constructive hyper-heuristic (CHH-1)

Our first hyper-heuristic procedure (*CHH-1*) is a straightforward design based on Figure 1. The set H_C of low-level heuristics consists of the six greedy heuristics described in Section 3.1 (*Gr1* to *Gr6*). As it is a constructive hyper-heuristic, the initial problem states that S_0 is a void sequence. The aim of the algorithm is to decide which heuristic is worth using at each position l (l = 1,...,D) of the sequence that represents the current solution. The application of the selected heuristic at position l gives the symbol sequenced at this position. To decide which heuristic to use at step l, the following criterion is adopted. Based on the current partial solution (that is, the symbols selected at positions 1,...,l-1), all heuristics are applied to complete their corresponding full solutions. The heuristic that generates the best solution (according to the RTV value) yields the symbol to be chosen at position l. The entire process is then repeated until the position before last (i.e., D-1) is filled, as the last position is obviously known. The pseudocode and a graphic illustration of the proposed hyper-heuristic are shown in Figure 2 and Figure 3, respectively.

```
0.  Let $H_C$ = {Gr1, Gr2, Gr3, Gr4, Gr5, Gr6};
    Let S be the solution sequence, initially void;
1.  $l := 1$;
2.  While $l \le D-1$ do:
3.      For all $h \in H_C$, apply h to obtain a full solution with positions
        between [1...l-1] of the current S, which is considered fixed. Let
        $h_l^*$ be the heuristic that obtains the best solution at step l;
4.      Choose the symbol at position l identified by $h_l^*$;
5.      $l := l + 1$;
6.  End while;
7.  Add the remaining symbol at position D and return to S
```

Figure 2. Pseudocode of the hyper-heuristic *CHH-1*

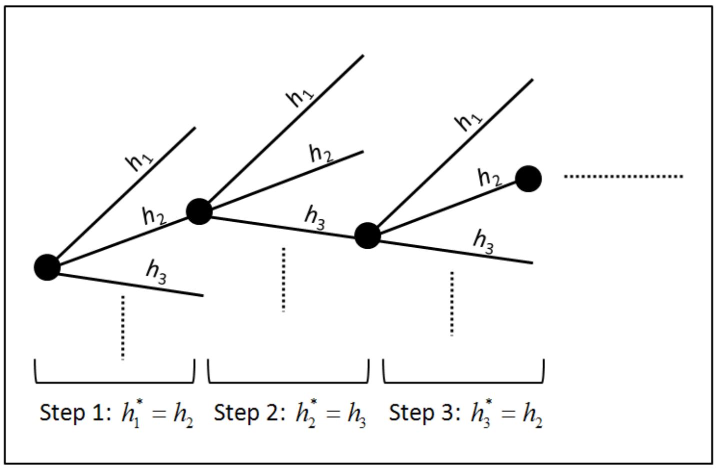


Figure 3. An illustration of the hyper-heuristic *CHH-1*

4.2. A learning-based constructive hyper-heuristic (CHH-2)

The second constructive hyper-heuristic (*CHH-2*) is similar to *CHH-1*, except that the set H_C^l of low-level heuristics is slightly restricted, as it keeps changing dynamically at each step *l* (*l* = 1,...,*D*-1) during the sequencing process. At step *l*, the heuristic *h* is included in set H_C^l if the *probability* (α_h^l) of obtaining the best full solution with fixed positions between [1...*l*-1] of the current subsequence is greater than or equal to a threshold probability (α_{TH}); i.e., $H_C^l = \{h \in H_C \mid \alpha_h^l \geq \alpha_{TH}\}$. We have considered two ways of calculating these probabilities α_h^l. A training stage is carried out from which the α_h^l values are derived. The idea of using a dynamic set H_C^l is to consider the most promising heuristics based on the training stage only. To decide which heuristic of H_C^l is selected at step *l*, the same criterion as in *CHH-1* is adopted.

Training stage
The training stage consists of solving a training set of RTVP instances with the hyper-heuristic *CHH-1*. The idea is to record the number of times each heuristic is selected at each step *l* (*l* = 1,...,*D*-1) of *CHH-1*, to see whether some heuristics are more promising than others when applied at the beginning, in the middle or at the end of the sequencing process. Since each instance may have different *D* values (and, therefore, a different *D*-length of its solution sequence), we decided to split the sequences into 20 slots, whose size depends on the *D* value of the instance. By splitting the position into slots, we do not look at the absolute position where the heuristics are used, but at the relative position in the sequence (that is, at the beginning, middle or end of the sequence). The training set has 60 instances, which were grouped into four classes (from *CAT1* to *CAT4*), according to their size. More details on these instances are given in the computation results section (Section 6). Figure 4 shows the average percentage of times that each heuristic has been selected at each slot for all training instances and for each class of instances (*CAT1* to *CAT4*). On average, *Gr5* is the most frequently selected heuristic during the first half of the sequencing process, *Gr3* in the second half, and *Gr1* at the end. This tendency is even more apparent for the largest instances (*CAT4*), whereas for the smallest instances (*CAT1*) there is no clear tendency.

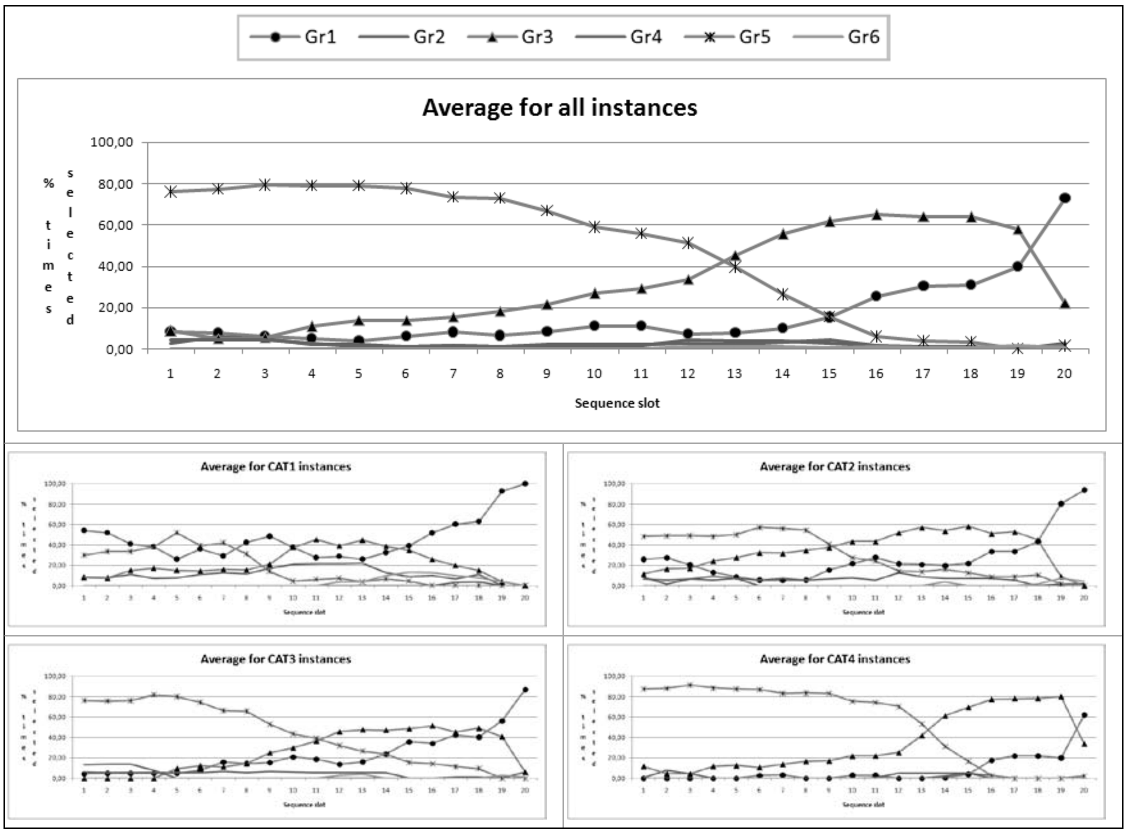


Figure 4. Average percentage of times that a heuristic is selected

The two ways of setting the probabilities α_h^l are given below:

(i) The probability α_h^l of heuristic *h* being selected at position *l* is the overall average percentage of times over all training instances that *h* has been selected in slot *s* $\left(s = \lfloor 20(l-1)/D \rfloor + 1\right)$.

(ii) Analogous to (i): the probability α_h^l of heuristic *h* being selected at position *l* is the average percentage of times that *h* has been selected in slot *s* over the training instances of the class in which the instance to be solved belongs to (*CAT1*, *CAT2*, *CAT3* or *CAT4*). An instance is classified into a class according to its size (see Table 1 in Section 6).

Another interesting way of calculating these probabilities is proposed in Qu *et al*. (2009). This is based on generating a set of solutions by randomly selecting the heuristic to be applied at each step *l* (*l* = 1,...,*D*-1) and counting the number of times that each heuristic has been applied in each slot *s* in the good solutions (see Qu *et al*. (2009) for more details).

4.3. An elitist-based constructive hyper-heuristic (CHH-3)

The proposed elitist-based constructive hyper-heuristic (*CHH-3*) can be considered ea special case of *CHH-2* where H_C^l is made up of one chosen heuristic only, which is the best heuristic at step *l*. In other words, at step *l* (*l* = 1,...,*D*), the set H_C^l is formed only by the *most probable* heuristic. That is, the heuristic applied is $h = \max_{h \in H_c} \arg\left(\alpha_h^l\right)$. The two settings for generating the probabilities α_h^l that are described in Subsection 4.2 are also used here.

4.4. A random-based constructive hyper-heuristic (CHH-4)

The forth constructive hyper-heuristic (*CHH-4*) is similar to *CHH-1*, but the low-level heuristic selection criterion used is a random one. At each position in the sequence, the heuristic to be applied at each step is selected at random using equal probabilities (1/6) for each heuristic.

5. Improvement hyper-heuristics

The performance of a hyper-heuristic may vary according to the available set of low-level heuristics. In the hyper-heuristic literature, it is common to use local search procedures as low-level heuristics in improvement hyper-heuristics (for example, Burke *et al*., 2003b; Dowsland *et al*., 2007; Pisinger and Ropke, 2007).

Metaheuristics have been used in the hyper-heuristic literature as the main framework, including TS (Burke *et al*., 2003b; Burke *et al*., 2007; Bai *et al*., 2008; Qu and Burke, 2009), VNS (Qu and Burke, 2009), genetic algorithm (Ho *et al*., 2007), ant algorithm (Burke *et al*., 2005) and simulated annealing (Dowsland *et al*., 2007; Bai *et al*., 2008). However, to the best of our knowledge, the use of metaheuristics in hyper-heuristics as low-level heuristics has not been reported in the literature to date.

In this study, we propose to use two different sets of low-level heuristics. One set, H_I^0, consists of the local search procedures described in Section 3.2.1 (*LS-1*, *LS-2* and *LS-3*). Moreover, we also propose using the set H_I^1, which consists of the more sophisticated heuristics given in Subsections 3.2.2 to 3.2.4 (*HC*, *TS* and *VNS*).

5.1. The general improvement-based hyper-heuristic framework

We propose a general improvement hyper-heuristic framework, based on that shown in Figure 1. The improvement hyper-heuristics starts from a full initial solution which is improved by iteratively performing several cycles, each of which consists of two stages: a *learning* stage and a *launching* stage (see Figure 5). The hyper-heuristic stops when it has run for a maximum preset available time (*T*). The pseudocode of the general improvement-based hyper-heuristic is shown in Figure 6.

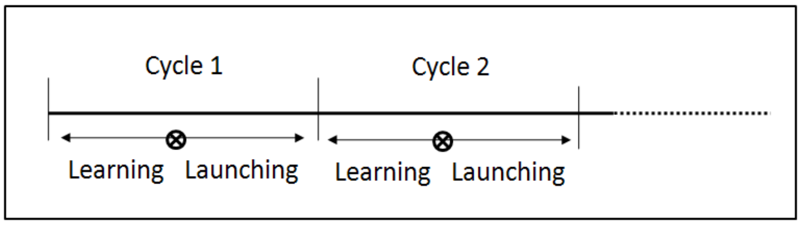


Figure 5. A graphical representation of the improvement hyper-heuristic scheme

Learning stage
The learning stage consists of running all of the low-level heuristics from the current solution. The objective is to see how many times each heuristic improves the current solution. The time spent on each heuristic in this stage (τ_0^{it}) depends on the current iteration of the hyper-heuristic (*it*) and is defined as follows: $\tau_0^{it} = \theta_0 \tau_0^{it-1}$, where $\tau_0^1 = \alpha T$, α ($0 < \alpha \ll 1$) is a coefficient that weights the time spent in the learning stage, and θ_0 ($0 < \theta_0 \leq 1$) is a non-increasing learning coefficient. Note that if $\theta_0 = 1$, the time spent in the learning stage remains equal at each iteration of the hyper-heuristic, whereas if $\theta_0 < 1$, this time will decrease exponentially.

Launching stage
A low-level heuristic is selected to be used in the launching stage, according to the improvement in the current solution obtained in the learning stage. The launching stage consists of running the selected heuristic from the best solution obtained in the learning stage. The time spent in this stage (τ_1^{it}) also depends on the current iteration of the hyper-heuristic, and is defined as follows: $\tau_1^{it} = \theta_1 \tau_1^{it-1}$, where $\tau_1^1 = \beta T$, β ($0 < \beta \ll 1$) is a coefficient that weights the time spent in the launching stage, and θ_1 ($\theta_1 \geq 1$) is a non-decreasing launching coefficient. Analogous to θ_0, if $\theta_1 = 1$, the time spent in the launching stage remains equal at each iteration of the hyper-heuristic, whereas if $\theta_1 > 1$, this time will increase exponentially.

```
0.  Let $H_I$ be the set of low-level (meta)heuristics;
    Let $S$ be an initial solution;
1.  $it := 1$;
2.  While execution time < $T$ do:
        Learning stage:
3.      For all $h \in H_I$, apply $h$ to $S$ during the time $\tau_0^{it}$ to obtain an
        improved solution;
4.      Let $S'$ be the best improved solution;
        Launching stage:
5.      Select a heuristic, $h^*$, of $H_I$ according to the selection mechanism;
6.      Apply $h^*$ to $S'$ during the time $\tau_1^{it}$;
7.      $it := it + 1$
8.      $S := S'$;
9.  End while;
10. Return $S$
```

Figure 6. Pseudocode of the general improvement-based hyper-heuristic

5.2. The selection mechanisms

Based on the framework, three criteria are proposed for selecting the low-level heuristic to be applied in the launching stage of each cycle for the improvement hyper-heuristics. All three criteria are based on the performance obtained during the learning stage.

(i) *Best performance-based hyper-heuristic*. The heuristic selected is the one that most improves the current solution during the learning stage.

(ii) *Probability-based hyper-heuristic*. The heuristic is selected in a pseudo-random way, according to the RTV value obtained in the previous learning stage. The probability of heuristic *h* being selected at iteration *it*, $p(h,it)$, is:

$$p(h,it) = \frac{1/RTV(h,it)}{\sum_{g \in H_I} 1/RTV(g,it)},$$

where $RTV(g,it)$ is the RTV value obtained with heuristic *g* in the learning stage at iteration *it*.

(iii) *Threshold probability-based hyper-heuristic*. At iteration *it*, only the heuristics that have obtained an improvement close to the best improvement in the learning stage are considered to be selected pseudo-randomly. That is,

$$p(h,it) = \begin{cases} \gamma_{h,it} & \text{if } RTV_T(h,it) \le \beta_{TH} \cdot RTV_T^*(it) \\ 0 & otherwise \end{cases}$$

where $\gamma_{h,it} = \frac{1/RTV_T(h,it)}{\sum_{g \in S_{it}} 1/RTV_T(g,it)}$ and $S_{it} = \left\{ h \in H_I | RTV_T(h,it) \le \beta_{TH} \cdot RTV_T^*(it) \right\}$.

$RTV_T^*(it)$ is the best RTV value obtained in the learning stage at iteration *it*, β_{TH} ($\beta_{TH} \ge 1$) is the threshold used in the selection and *T* is the maximum preset available time.

5.3. Local search based hyper-heuristic algorithms

We propose three hyper-heuristic algorithms that use a set of local search procedures as the set of low-level heuristics (H_I^0). We refer to them as $IHH\text{-}1\text{-}H_I^0$, $IHH\text{-}2\text{-}H_I^0$ and $IHH\text{-}3\text{-}H_I^0$ on the basis of the selection criteria (i), (ii) and (iii), respectively.

Although the scheme of the hyper-heuristic is independent of the set of low-level heuristics used, these three variants may finish their execution time before the maximum preset available time (*T*). This can happen if a local optimum using *LS-1*, *LS-2* and *LS-3* is found before the end of *T*.

5.4. Metaheuristic based hyper-heuristic algorithms

Three hyper-heuristic algorithms that use the set of the composite hill-climbing method and metaheuristic algorithms as the set of low-level heuristics (H_I^1) are presented, which differ in the choice of the selection criteria defined earlier. We refer to these three variants as $IHH\text{-}1\text{-}H_I^1$, $IHH\text{-}2\text{-}H_I^1$ and $IHH\text{-}3\text{-}H_I^1$, which use the selection criteria (i), (ii) and (iii), respectively.

6. Computational results

The proposed hyper-heuristics were coded in Java and executed on a PC 3.4 GHz Intel Pentium IV with 1.5 GB of RAM. The 60 training instances and 740 test instances used in Corominas *et al*. (2008, 2009b), García *et al*. (2006) and García-Villoria and Pastor (2008, 2009a, 2009b) were also used in this paper (all instances can be found at http://www.ioc.upc.edu/EOLI/research/). These instances were grouped into four classes (from *CAT1* to *CAT4* with 15 training instances and 185 test instances in each class) according to their size. The instances were generated using the random values of *D* (number of units) and *n* (number of models) shown in Table 1. For all instances and for each model $i = 1,\dots,n$, a random value of d_i (number of units of model i) is between 1 and $\left|(D-n+1)/2.5\right|$, such that $\sum_{i=1..n} d_i = D$.

Table 1. Uniform distributions for generating the *D* and *n* values

	CAT1	***CAT2***	***CAT3***	***CAT4***
D	U(25, 50)	U(50, 100)	U(100, 200)	U(200, 500)
n	U(3, 15)	U(3, 30)	U(3, 65)	U(3, 150)

We report the results of the three classes of our proposed hyper-heuristics in the next three sections.

6.1. Results of the constructive hyper-heuristics

The 740 test instances were solved with the six greedy heuristics (*Gr1*, *Gr2*, *Gr3*, *Gr4*, *Gr5* and *Gr6*) and the constructive hyper-heuristics *CHH-1*, *CHH-2*, *CHH-3* and *CHH-4*. The two probabilities settings presented in Section 4.2 were used. To run *CHH-2*, a preliminary experiment was carried out to set the value of the threshold probability α_{TH} to 0.2. The computing times were very small: for all greedy heuristics, the maximum time for solving an instance was 0.070 cpu seconds, and for *CHH-1*, *CHH-2*, *CHH-3* and *CHH-4* it was 2.523, 0.590, 0.040 and 0.046 cpu seconds, respectively. Since the computing times of the greedy heuristics is negligible, a better, fast heuristic (let it be called *BH*) can be easily constructed by merely running the six heuristics and getting the best solution. Table 2 shows the average RTV values to be minimised for each class of instances (*CAT1* to *CAT4*) obtained with all these greedy algorithms. For *CHH-2* and *CHH-3*, we show the results obtained by setting the α probabilities using the first way (*Overall prob.*) or the second way (*Per class prob.*), as explained in the training stage in Subsection 4.2.

Table 2 shows that the best individual greedy heuristic is clearly *Gr5*, which is much better than the second best heuristic (*Gr4*). *BH* is, obviously, better than Gr5. If we consider the results by class, *BH* is 9.74%, 7.62%, 3.82% and 4.81% better than *Gr5* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. On the other hand, *CHH-1* is 21.86%, 20.44%, 14.35% and 13.14% better than *Gr5* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. In addition, *CHH-1* is 13.43%, 13.88%, 10.95% and 8.75% better than *BH* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. This hyper-heuristic is the best for *CAT1* instances and the second best for *CAT2* instances, as well as being very close to the best results for the *CAT2* instances.

Table 2. Average RTV values for the constructive hyper-heuristics

		CAT1	***CAT2***	***CAT3***	***CAT4***
Gr1		121.84	933.41	8,502.80	81,730.00
Gr2		147.19	1,077.88	9,106.04	84,616.22
Gr3		120.09	915.74	8,347.60	80,670.03
Gr4		125.06	914.70	8,295.41	80,577.15
Gr5		88.02	553.06	3,894.31	30,870.45
Gr6		405.39	2,583.30	17,450.83	163,908.26
BH		79.45	510.93	3,745.39	29,385.59
CHH-1		**68.78**	440.00	3,335.37	26,814.25
CHH-2	Overall prob.	78.89	477.38	3,377.66	27,196.49
	Per class prob.	83.19	522.98	3,297.68	26,323.33
CHH-3	Overall prob.	83.05	**426.50**	2,754.06	23,178.13
	Per class prob.	104.53	599.37	3,186.30	21,542.86
CHH-4		118.75	500.76	**2,716.69**	**19,605.49**

Note that although *Gr1*, *Gr2*, *Gr3*, *Gr4* and *Gr6* are poor quality heuristics by themselves, the hyper-heuristic algorithms find their strengths and use them in the right moment in the solution generation process. In fact, Figure 4 shows that *CHH-1* found, on average for the training instances, that *Gr5* more suitable in the first half of the sequence, *Gr3* for almost all of the second half, and *Gr1* for the last positions of the sequence. *Gr2*, *Gr4* and *Gr6* were found to be dominated and inferior, except in the case of *CAT1* instances. This is an exciting and interesting observation of the behaviour of these heuristics and deserves theoretical support in the future.

The results obtained by the two variants of *CHH-2* are, on average, similar to those found by *CHH-1*, except that *CHH-2* is much faster than *CHH-1*. These results are expected since, at each step, the heuristics that perform badly are not considered.

When a training stage was used to decide a priori which heuristic generated the best full solution from a fixed current partial solution (i.e., the most likely heuristic to be selected per position), better solutions than *CHH-1* and *CHH-2* are obtained. On average, the best *CHH-3* hyper-heuristic variant is the one whose probabilities are calculated per instance class, which was found to be 14.94% and 26.69% better than *BH* for the medium and large instances (*CAT3* and *CAT4* instances), respectively. However, for the small instances (*CAT1* and *CAT2* instances), *BH* is 23.99% and 14.76% better, respectively. The best results for the *CAT2* instances are obtained, surprisingly, by the *CHH-3* variant whose probabilities are calculated based on all instances.

Finally, Table 2 shows that the random selection strategy (*CHH-4*) is the best for solving medium and, particularly, large instances (*CAT3* and *CAT4* instances, respectively). *CHH-4* is 27.47% and 33.28% better than *BH* for *CAT3* and *CAT4* instances, respectively. These results may be surprising. However, the hyper-heuristic literature shows that random selection strategies frequently yield good performances (e.g., Burke *et al.*, 2005; Bilgin *et al.*, 2007).

The results of *CHH-1* are not necessary better than those from all variants of the other *CHH-2* and *CHH-3*. This could be considered surprising, as *CHH-1* has all the low-level heuristics available for use at each iteration, whereas *CHH-2* and *CHH-3* rely on a subset only. This could be due to the lack of correlation between the qualities of the solutions at different iterations.

6.2. Results of the local search based hyper-heuristics

All test instances were solved with the composite hill-climbing method (*CHC*) and the three local search based hyper-heuristic algorithms ($IHH\text{-}1\text{-}H_I^0$, $IHH\text{-}2\text{-}H_I^0$ and $IHH\text{-}3\text{-}H_I^0$). The initial solution used for all algorithms was obtained with the best constructive hyper-heuristic. This is found by applying *CHH-3* using the probabilities calculated per instance class. The maximum time available to run all algorithms (T) was set to 1,000 seconds. A preliminary experiment was conducted to set the values of the following parameters, which were found to be $\alpha = 0.01$, $\theta_0 = 0.95$, $\beta = 0.02$, $\theta_1 = 1.2$ and $\beta_{TH} = 1.1$.

Table 3 shows the average RTV values for each class of instances (*CAT1* to *CAT4*) obtained with *CHC* and with the three improvement hyper-heuristics that use the local search methods as low-level heuristics (that is, $IHH\text{-}1\text{-}H_I^0$, $IHH\text{-}2\text{-}H_I^0$ and $IHH\text{-}3\text{-}H_I^0$).

Table 3. Average RTV values for the local search based hyper-heuristics

	CAT1	***CAT2***	***CAT3***	***CAT4***
CHC	16.39	39.91	101.90	339.75
$IHH\text{-}1\text{-}H_I^0$	**15.71**	**38.99**	102.51	321.20
$IHH\text{-}2\text{-}H_I^0$	**15.71**	**38.99**	100.44	315.05
$IHH\text{-}3\text{-}H_I^0$	**15.71**	**38.99**	**96.55**	**314.33**

A simple composite method such as *CHC*, which iteratively applies the three local search methods *LS-1*, *LS-2* and *LS-3*, is able to obtain very good solutions. In fact, as shown in Table 4, *CHC* obtained better results than *TS*, which was previously the best at solving the RTVP. However, hyper-heuristics decide intelligently when to use each local search during the optimisation process, rather than systematically using them in a specific order. All three hyper-heuristics outperformed *CHC*. The hyper-heuristics $IHH\text{-}1\text{-}H_I^0$, $IHH\text{-}2\text{-}H_I^0$ and $IHH\text{-}3\text{-}H_I^0$ were, on average, 3.93%, 5.57% and 6.50% better overall than *CHC*, respectively. Considering the best by class results, $IHH\text{-}3\text{-}H_I^0$ is 4.15%, 2.31%, 5.25% and 7.48% better than *CHC* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively.

6.3. Results of the metaheuristic based hyper-heuristics

All test instances were solved with the three metaheuristic based hyper-heuristic algorithms ($IHH\text{-}1\text{-}H_I^1$, $IHH\text{-}2\text{-}H_I^1$ and $IHH\text{-}3\text{-}H_I^1$), the composite hill-climbing method (*CHC*) and *TS* and *VNS*, which are based on two of the metaheuristics that perform best at solving the RTVP (Corominas *et al.*, 2009b, 2009c). The initial solution used for all algorithms was obtained with the best constructive hyper-heuristic (that is, *CHH-3* using the probabilities calculated per instance class). The same parameter values as in the previous subsection are also used here. The time available to run all algorithms was set to 1,000 seconds.

Table 4 shows the average RTV values for each class of instances (*CAT1* to *CAT4*) obtained with $IHH\text{-}1\text{-}H_I^1$, $IHH\text{-}2\text{-}H_I^1$ and $IHH\text{-}3\text{-}H_I^1$. The results obtained when using *TS* and *VNS* directly are also included here for comparison. Since the initial solutions used in Corominas et al. (2009b, 2009c) are different from the initial solutions used in this paper, we solved the instances again with *TS* and *VNS*.

Table 4. Average RTV values for the metaheuristic based hyper-heuristics

	CAT1	***CAT2***	***CAT3***	***CAT4***
CHC	16.39	39.91	101.90	339.75
TS	10.74	42.68	175.03	689.44
VNS	11.36	**24.53**	83.54	408.52
$IHH\text{-}1\text{-}H_I^1$	10.39	25.18	74.71	529.65
$IHH\text{-}2\text{-}H_I^1$	**10.37**	24.55	72.46	433.39
$IHH\text{-}3\text{-}H_I^1$	10.39	24.72	**66.91**	**334.76**

The best results shown in Table 4 are obtained, on average, by the hyper-heuristic algorithm $IHH\text{-}3\text{-}H_I^1$ and are 12.28% better than the results found by the best low-level heuristic (*CHC*) when applied in isolation. If we consider the results by class, we can see that *TS* performs better than *CHC* and *VNS* for the smallest instances (*CAT1*), *VNS* outperforms *CHC* and *TS* for the next two larger sets of instances (*CAT2* and *CAT3*) and *CHC* was the best for the largest set of instances (*CAT4*). However, $IHH\text{-}3\text{-}H_I^1$ was equally good or better than *CHC, TS* and *VNS* for all types of instances. In particular, $IHH\text{-}3\text{-}H_I^1$ is 3.26% better than *TS* for *CAT1* instances, 0.77% worse and 19.91% better than *VNS* for *CAT2* and *CAT3* instances, respectively, and 1.47% better than *CHC* for *CAT4* instances. Thus, we can confirm that low-level metaheuristic algorithms can yield promising and excellent results when they are combined intelligently during the improvement process, within a hyper-heuristic framework.

6.4. General discussion of the results

Although the goal of this study is not to improve the best results obtained in the literature to solve the RTVP, several of the hyper-heuristics that we developed are very competitive. We compile the averages of the RTV values together with the variance within each set of instances to facilitate observations of the different approaches. Moreover, the results obtained with the GA proposed in García-Villoria and Pastor (2009c) have been added for comparison purposes (the GA was run in the same conditions as the other algorithms; that is, it was executed on a PC 3.4 GHz Intel Pentium IV with 1.5 GB of RAM and stopped after 1,000 cpu seconds per instance).

Table 5. Averages of the RTV values (standard deviation of the RTV values)

	CAT1	*CAT2*	*CAT3*	*CAT4*
CHC	16.39 (8.28)	39.91 (17.47)	101.90 (38.15)	339.75 (257.47)
TS	10.74 (5.69)	42.68 (24.60)	175.03 (90.74)	689.44 (533.29)
VNS	11.36 (5.80)	24.53 (11.40)	83.54 (48.49)	408.52 (490.32)
GA	10.92 (5.83)	27.00 (14.28)	74.86 (43.92)	313.92 (233.03)
$IHH\text{-}1\text{-}H_I^0$	15.71 (8.18)	38.99 (16.39)	102.51 (43.23)	321.20 (156.19)
$IHH\text{-}2\text{-}H_I^0$	15.71 (8.18)	38.99 (16.39)	100.44 (40.47)	315.05 (158.86)
$IHH\text{-}3\text{-}H_I^0$	15.71 (8.18)	38.99 (16.39)	96.55 (41.13)	314.33 (141.08)
$IHH\text{-}1\text{-}H_I^1$	10.39 (5.54)	25.18 (11.34)	74.71 (35.16)	529.65 (585.64)
$IHH\text{-}2\text{-}H_I^1$	10.37 (5.48)	24.55 (11.33)	72.46 (43.02)	433.39 (396.44)
$IHH\text{-}3\text{-}H_I^1$	10.39 (5.47)	24.72 (11.47)	66.91 (33.58)	334.76 (359.41)

The number of times that each algorithm reaches the best RTV value obtained using all algorithms are shown in Table 6. The results obtained with the constructive hyper-heuristic algorithms have not been included in Table 5 or Table 6, because these algorithms are only used to obtain an initial solution and the cpu times are negligible.

Table 6. Averages of the number of times that the best solution is reached

	CAT1	*CAT2*	*CAT3*	*CAT4*
CHC	0.23	0.07	0.01	0.05
TS	0.88	0.28	0.03	0.01
VNS	0.72	0.53	0.22	0.16
GA	0.80	0.39	0.36	0.45
$IHH\text{-}1\text{-}H_I^0$	0.26	0.10	0.02	0.06
$IHH\text{-}2\text{-}H_I^0$	0.26	0.10	0.02	0.07
$IHH\text{-}3\text{-}H_I^0$	0.26	0.10	0.02	0.09
$IHH\text{-}1\text{-}H_I^1$	0.97	0.55	0.18	0.05
$IHH\text{-}2\text{-}H_I^1$	0.97	0.61	0.31	0.09
$IHH\text{-}3\text{-}H_I^1$	0.95	0.62	0.34	0.15

Hyper-heuristics that use metaheuristic methods as low-level heuristics obtain the best RTV average when solving the smallest instances (*CAT1* instances), with no significant differences between them (confidence level of 95%). Moreover, they obtain the best solution for 95% ($IHH\text{-}3\text{-}H_I^1$) or 97% ($IHH\text{-}1\text{-}H_I^1$ and $IHH\text{-}2\text{-}H_I^1$) of the *CAT1* instances. For the *CAT2* instances, the use of low-level metaheuristics also gives the best RTV average with no significant difference between $IHH\text{-}2\text{-}H_I^1$, $IHH\text{-}3\text{-}H_I^1$ and *VNS*, although the two hyper-heuristics obtain the best solution more frequently (61% and 62% of the instances, respectively) than the *VNS* (53% of the instances). In the case of *CAT3* instances, the RTV average obtained by $IHH\text{-}3\text{-}H_I^1$ is significantly the best, although the *GA* algorithm is able to obtain the best solution slightly more times (36% of times) than $IHH\text{-}3\text{-}H_I^1$ (34% of times). Finally, for the biggest instances (*CAT4* instances), the GA algorithm obtains the best solutions the highest number of times (45% of times) and also obtains the best RTV average, although there is no significant difference between the RTV averages obtained by *CHC*, $IHH\text{-}1\text{-}H_I^0$, $IHH\text{-}2\text{-}H_I^0$ and $IHH\text{-}3\text{-}H_I^0$.

7. Conclusions

The overall goal of this paper is to propose schemes for implementing hyper-heuristics within a general scheme that is general enough to be easily adapted to solve a variety of hard combinatorial optimisation problems. In this paper, we have chosen a sequencing NP-hard problem known as the Response Time Variability Problem (RTVP) to test the efficiency of the proposed hyper-heuristic schemes. The hyper-heuristic algorithms proposed in this study operate at a higher level of abstraction and have no knowledge of the problem domain. The problem domain is in the low-level heuristics used by the hyper-heuristic and in the evaluation function used to evaluate the goodness of a solution. In other words, the same hyper-heuristic algorithms can solve other combinatorial problems without too much extra effort by replacing the low level heuristics and the evaluation function only.

Our first attempt is to design a constructive-based hyper-heuristic approach which is based on the reuse of a set of existing greedy simple heuristics known in the literature. Our goal was not to compete with the best results reported in the literature, but to check whether reusing the six greedy simple heuristics reported in the literature produced better solutions than those obtained when the heuristics were used in an isolated way. Four constructive hyper-heuristic algorithms were designed by a straightforward application of the general hyper-heuristic scheme shown in Figure 1. The difference between the four algorithms is the low-level selection criterion. An extensive experiment showed the ability of the hyper-heuristics to use each low-level heuristic at the appropriate moment, according to the current state of the partial solution. On average, the best proposed constructive hyper-heuristic is able to obtain approximately 35% improvement in the best low-level heuristic, if used in an isolated way. This success is particularly remarkable, as the results show that the other five heuristics obtained very bad solutions by themselves.

To investigate the impact of a relatively higher complexity to the low level heuristics, local search methods were introduced instead of simple greedy ones. We have proposed three local search methods based on three neighbourhoods for the RTVP that exist in the literature. The improvement hyper-heuristic approach is found to be useful in deciding the most suitable local search to be applied at a given time based on the current state of the search. The computational experiment shows the success of this implementation of hyper-heuristics that combines the use of the local search methods

Our final exploration is to go one step further in terms of time complexity of the low-level heuristics by introducing metaheuristics instead. Such an investigation, to the authors knowledge, has not been reported in the literature. We believe this is mainly because of the excessive computational time that these may require. Though we appreciate this drawback our aim is to find a way of overcoming this problem. We have therefore put forward a mechanism on how to deal with this issue. We presented appropriate schemes on how to select low-level metaheuristics based on regular use of learning and launching stages at each cycle of the search. The goal is to control the computational burden while guiding the search toward good solutions. Encouraging results show the usefulness of integrating more sophisticated low level algorithms such as metaheuristics within the hyper-heuristic methodology. We believe this is a challenging issue that merits further study. Future studies could investigate the design of a mechanism to decide the amount of time used in the learning and launching stages in each cycle of the search. This obviously will depend on many attributes including the running time, problem characteristics and individual performance of the local searches or the metaheuristics, among other factors. The design of robust mechanisms for selecting the most appropriate low-level (meta)heuristic at a given iteration is an open question that may not have an optimal answer in practice, but any insightful attempt would, in our view and without doubt, be a step in the right direction.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to the anonymous reviewers for their detailed and valuable comments, which have helped to improve the quality of this paper.

REFERENCES

Adenso-Díaz, B. and Laguna, M. (2006) 'Fine-tuning of algorithms using fractional experimental designs and local search', *Operations Research*, Vol. 54, pp. 99-114.

Anily, S., Glass, C.A. and Hassin, R. (1998) 'The scheduling of maintenance service', *Discrete Applied Mathematics*, Vol. 82, pp. 27-42.

Bai, R., Burke, E.K. and Kendall, G. (2008) 'Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation', *Journal of the Operational Research Society*, Vol. 59, pp. 1387-1397.

Balinski, M.L. and Young, H.P. (1982) 'Fair Representation: meeting the ideal of one man, one vote', Yale University Press, New Haven CT.

Bilgin, B., Özcan, E. and Korkmaz, E.E. (2007) 'An Experiment Study on Hyper-heuristics and Exam Timetabling', *Lecture Notes on Computer Science*, PATAT2006 selected papers, Springer-Verlag.

Bollapragada, S., Bussieck, M.R. and Mallik, S. (2004) 'Scheduling Commercial Videotapes in Broadcast Television', *Operations Research*, Vol. 52, pp. 679-689.

Brusco, M.J. (2008) 'Scheduling advertising slots for television', *Journal of the Operational Research Society*, Vol. 59, pp. 1363-1372.

Burke, E.K., Kendall, G., Newall, J., Hart, E., Ross, P. and Schulenburg, S. (2003a) 'Hyper-heuristics: An Emerging Direction in Modern Search Technology', Chapter

16 in *Handbook of Metaheuristics*, Eds. Glover and Kochenberger, Kluwer Academic Publishers, pp. 457-474.

Burke, E.K., Kendall, G. and Souibeiga, E. (2003b) 'A Tabu-Search Hyperheuristic for Timetabling and Rostering', *Journal of Heuristics*, Vol. 9, pp. 451-470.

Burke, E.K., Kendall, G., Silva, D.L. and O'Brien, R. (2005) 'An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem', *2005 IEEE Congress on Evolutionary Computation*, pp. 2263-2270.

Burke, E.K., Petrovic, S. and Qu, R. (2006) 'Case-based heuristic selection for timetabling problems', *Journal of Scheduling*, Vol. 9, pp. 115-132.

Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R. (2007) 'A graph-based hyper-heuristic for educational timetabling problems', *European Journal of Operational Research*, Vol. 176, pp. 177-192.

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. and Woodward, J. (2009). 'A Classification of Hyper-heuristics Approaches', *Technical Report No. NOTTCS-TR-SUB-0907061259-5808* (to appear in Handbook of Meta-heuristics), School of Computer Science and Information Technology, University of Nottingham.

Corominas, A., Kubiak, W. and Moreno, N. (2007) 'Response time variability', *Journal of Scheduling*, Vol. 10, pp. 97-110.

Corominas, A., García-Villoria, A. and Pastor, R. (2008) 'Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics', *Frontiers in Artificial Intelligence and Applications*, Vol. 184, pp. 128-137.

Corominas, A., Kubiak, W. and Pastor, R. (2009a) 'Mathematical Programming Modeling of the Response Time Variability Problem', *European Journal of Operational Research*, doi: 10.1016/j.ejor.2009.01.014.

Corominas, A., García-Villoria, A., Pastor, R. (2009b) 'Using Tabu Search for the Response Time Variability Problem', *3rd International Conference on Industrial Engineering and Industrial Management* (CIO 2009), Barcelona and Terrassa, Spain.

Corominas, A., García-Villoria, A., Pastor, R. (2009c) 'Solving the Response Time Variable Problem by means of a Variable Neighbourhood Search Algorithm'; *13th IFAC Symposium of Information Control Problems in Manufacturing* (INCOM 2009); Moscow, Russia.

Dong, L., Melhem, R. and Mosse, D. (1998) 'Time slot allocation for real-time messages with negotiable distance constrains requirements', *Fourth IEEE Real-Time Technology and Applications Symposium* (RTAS'98), Denver, CO. pp. 131-136.

Dowsland, K.A., Soubeiga, E. and Burke, E.K. (2007) 'A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation', *European Journal of Operational Research*, Vol. 179, pp. 759-774.

García-Villoria, A. and Pastor, R. (2008) 'Solving the Response Time Variability Problem by means of a psychoclonal approach', *Journal of Heuristics*, doi:10.1007/s10732-008-9082-2.

García-Villoria, A. and Pastor, R. (2009a) 'Introducing dynamic diversity into a discrete particle swarm optimization', *Computers & Operations Research*, Vol. 36, pp. 951-966.

García-Villoria, A. and Pastor, R. (2009b) 'Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism', *International Journal of Production Research*, doi: 10.1080/00207540902862545.

García-Villoria, A. and Pastor, R. (2009c) ‘Solving the response time variability problem by means of a genetic algorithm’, *European Journal of Operational Research*, doi:10.1016/j.ejor.2009.05.024.

García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics, *Frontiers in Artificial Intelligence and Applications*, Vol. 146, pp.187-194.

García-Villoria, A., Pastor, R. and Corominas, A. (2007) ‘Solving the Response Time Variability Problem by means of the Cross-Entropy Method’, *International Journal of Manufacturing Technology and Management*, forthcoming.

Glover, F., (1986) ‘Future paths for Integer Programming and Links to Artificial Intelligence’, *Computers and Operations Research*, Vol. 5, pp. 533-549.

Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report TR 2007-12*, University of Maryland, USA. Available at http://hdl.handle.net/1903/7082.

Ho, N.B., Tay, J.C. and Lai, E.M.-K (2007) ‘An effective architecture for learning and evolving flexible job-shop schedules’, *European Journal of Operational Research*, Vol. 179, pp. 316-333.

Kubiak, W. (1993) ‘Minimizing variation of production rates in just-in-time systems: A survey’, *European Journal of Operational Research*, Vol. 66, pp. 259-271.

Mladenović, N. and Hansen, P. (1997) ‘Variable neighbourhood search’, *Computers & Operations Research*, Vol. 24, pp. 1097-1100.

Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 35, pp. 192-207.

Özcan, E., Bilgin, B. and Korkmaz, E.E. (2008) ‘A comprehensive analysis of hyper-heuristics’, *Intelligent Data Analysis*, Vol. 12, pp. 3-23.

Pisinger, D. and Ropke, S. (2007) ‘A general heuristic for vehicle routing problems’, *Computers and Operations Research*, Vol. 34, pp. 2403-2435.

Qu, R. and Burke, E.K. (2009) ‘Hybridisations within a Graph Based Hyper-heuristic Framework for University Timetabling Problems’, *Journal of Operational Research Society*, doi: 10.1057/jors.2008.102.

Qu, R., Burke, E.K. and McCollum, B. (2009) ‘Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems’, *European Journal of Operational Research*, Vol. 198, pp 392-404.

Ross, P. (2005) ‘Hyper-heuristics’, Chapter 17 in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Eds. Burke and Kendall, Springer, pp. 529-556.

Salhi, S. (2002) ‘Defining tabu list size and aspiration criterion within tabu search methods’, *Computers and Operations Research*, Vol. 29, pp. 67-86.

Salhi, S. (2006) ‘Heuristic Search in Action: the Science of Tomorrow’, In: Salhi S. (Ed.) OR48 Keynote Papers, GORS, London, pp. 39-58.

Waldspurger, C.A. and Weihl, W.E. (1994) ‘Lottery Scheduling: Flexible Proportional-Share Resource Management’, *First USENIX Symposium on Operating System Design and Implementation*.

Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Scheduling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528*, Massachusetts Institute of Technology, MIT Laboratory for Computer Science. Available at https://eprints.kfupm.edu.sa/67117.

Wei, W.D. and Liu, C.L. (1983) ‘On a periodic maintenance problem’, *Operations Research Letters*, Vol. 2, pp. 90-93.

A systematic procedure based on CALIBRA and the Nelder & Mead algorithm for fine-tuning metaheuristics

Article submitted to Journal of the Operational Research Society (2nd review in progress)

A systematic procedure based on CALIBRA and the Nelder & Mead algorithm for fine-tuning metaheuristics

Albert COROMINAS, Alberto GARCÍA-VILLORIA* and Rafael PASTOR
Institute of Industrial and Control Engineering (IOC)
Technical University of Catalonia (UPC)
{albert.corominas / alberto.garcia-villoria / rafael.pastor}@upc.edu

Abstract. The problem of setting the parameter values of a metaheuristic algorithm that optimise its performance is complex and time-consuming. Although the performance of a metaheuristic can be very sensitive to the parameter values, it is usual in the literature that the selection of the value parameters is not enough justified. There are in the literature two procedures that facilitate the task of fine-tuning: CALIBRA and the Nelder & Mead algorithm. We propose a hands-off systematic procedure for fine-tuning metaheuristics that takes the advantages of CALIBRA and the Nelder & Mead algorithm.

Keywords: fine-tuning, parameter setting, metaheuristics, CALIBRA, Nelder and Mead.

1. Introduction

Over the last decades, metaheuristics have shown to be very useful to solve effectively many types of complex problems. Most of the algorithms proposed in the literature have a set of parameters whose values have to be set before their execution, although the choice of their parameter values is not trivial (see, for instance, Altinel and Öncan, 2005; Battarra *et al*, 2008). The selection of the parameter values is an important task because it may has a great influence in the performance of the algorithm. Much research effort can be spent in fine-tuning an algorithm (Barr *et al*, 1995). The task of fine-tuning

* Corresponding author: Alberto García-Villoria, Institute of Industrial and Control Engineering (IOC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; e-mail: alberto.garcia-villoria@upc.edu

is usually hard due to the following three reasons (Adenso-Díaz and Laguna, 2006): 1) the algorithm may be very sensitive to the parameter values, 2) a non-linear interdependence can be involved between the parameters, and 3) in stochastic algorithms (as metaheuristic algorithms usually are) each execution may provide a different solution.

Despite the relevance of the selection of parameter values, this selection is commonly justified in one of the following ways (Adenso-Díaz and Laguna, 2006, Eiben *et al*, 1999): 1) "by hand" on the basis of a small number of experiments that are not specifically referenced, 2) by using the general values of the method recommended for a wide range of problems, 3) by using the values of the method reported to be effective in other similar problems, and 4) without any explanation.

A right and well detailed practice for fine-tuning heuristics is described in Coy *et al* (2001), but it has the disadvantage of being tedious and needing a lot of human time for the experimental designs. The new systematic procedure that we propose in this paper is a hands-off tool for fine-tuning metaheuristics based on CALIBRA (Adenso-Díaz and Laguna, 2006) and the Nelder & Mead algorithm (N&M) (Nelder and Mead, 1965). It should be taken into account that the evaluation of a calibration is usually very time expensive in the case of metaheuristics. But the procedure proposed in this paper needs few evaluations to find a good calibration.

The remainder of the paper is organized as follows: Section 2 and 3 introduce CALIBRA and N&M, respectively. Section 4 explains the systematic procedure that we propose for fine-tuning metaheuristics. We apply the proposed procedure for fine-tuning three different metaheuristics in Section 5. Finally, some conclusions are given in Section 6.

2. CALIBRA

CALIBRA is a tool proposed in Adenso-Díaz and Laguna (2006) specifically designed for fine-tuning algorithms within a specified range of parameter values. CALIBRA is based on using conjointly Taguchi’s fractional factorial experimental designs (Taguchi,

1987) and a local search procedure. The maximum number of parameters supported by the current version of CALIBRA is 5. If the algorithm has more than 5 parameters, its authors suggest determining the 5 most significant parameters and fixing the others (Adenso-Díaz and Laguna, 2006).

Notice that one assumption of Taguchi's experimental designs is the linear interdependence between the parameters but, as it has been mentioned in Section 1, the interdependence is usually non-linear. CALIBRA uses the analysis of the experiment results only as a guideline to narrow the search and to initiate the next round of experiments. Because the search focuses on narrower ranges for each parameter value, the linear assumption becomes less restrictive and the predicted optimal values can be approached to the true optimal values.

The user has to provide CALIBRA with the number of iterations (whose minimal value is $2^k + 9$, where k is the number of parameters to be fine-tuned) and with a representative training set of instances. For a more detailed explanation of CALIBRA see Adenso-Díaz and Laguna (2006).

3. The Nelder & Mead algorithm

The problem of fine-tuning k parameters of an algorithm can be approached as an optimisation problem, in which the problem consists of finding the k parameter values that optimise the algorithm performance. Let φ be a function whose variables are the algorithm parameters and whose image is the performance of the algorithm. Therefore, the fine-tuning optimisation problem is equivalent to the problem of optimising φ. Since the set of instances of a problem is infinite, we must use a representative training set to calculate the φ image.

Since φ is not expected to have any special or recognizable property, a direct optimisation algorithm (i.e., it only uses the values of the function) is needed to solve the fine-tuning optimisation problem. The Nelder & Mead algorithm (N&M) (Nelder and Mead, 1965), also known as the flexible polyhedron algorithm, is a direct optimisation algorithm that has offered and still offers good results in the literature

(Chelouah and Siarry, 2005). N&M is based on k+1 points that are the vertex of a hypertetrahedron (preferably regular) in a k-dimensional space. The coordinates of a point represents the parameter values, and each point is evaluated with φ. Next, the points are iteratively moved over the space according to their evaluations until a local optimal point is reached. N&M is able to approach to the global optimal point whether φ is unimodal; if φ is multimodal, then N&M approaches to one local optimum depending on the initial hypertetrahedron. N&M was originally designed for working with only real coordinates; however, it can be easily adapted to admit integer coordinates. For a more detailed explanation of N&M, see Nelder and Mead (1965).

4. A systematic procedure for fine-tuning metaheuristics

The fine-tuning procedure that we propose takes the advantages of CALIBRA and N&M. In a situation in which there is little knowledge about the right value of a parameter, a wide range of values can be used in CALIBRA. For example, to fine-tune the size of the population of a genetic algorithm, the range [1..200] can be specified in CALIBRA. In this case, CALIBRA is able to return a good approximation of the right calibration of the population size. On the other hand, N&M needs an initial point (remind that, in the context of fine-tuning, the coordinates of a point represents the parameter values of the algorithm to be fine-tuned) to build the initial hypertetrahedron. It is advisable that the initial point used by N&M to build the initial hypertetrahedron is a good point. The reason is that, for multimodal functions, N&M tends to approach to one local optimal near from the initial point.

The fine-tuning procedure that we propose has two steps. First we can obtain a quite good fine-tuning applying CALIBRA with wider or narrower ranges of the parameter values according to the knowledge that we have. The narrower the ranges the better the fine-tuning obtained by CALIBRA. Next, we can obtain a more precise calibration by N&M using as the start point the values returned by CALIBRA.

In the conclusions given in Adenso-Díaz and Laguna (2006), its authors suggest that CALIBRA can be used to search in a narrow range around parameters values that have been tested. Thus, at the second step of our proposed procedure, CALIBRA could be used again instead of N&M. This alternative is compared with respect to our procedure and with respect to using only one time CALIBRA in Section 5.

5. Computational experiment

In this section, we analyse the proposed fine-tuning procedure and the alternative explained in Section 4. That is, first we obtain initial parameter values with CALIBRA. Next, we can apply N&M using the initial values obtained at the first step or we can apply again CALIBRA using narrower ranges around the values obtained at the first step. In the remainder of this section, we refer to our fine-tuning procedure as *CALIBRA+N&M* and we refer to the alternative procedure as *CALIBRA+CALIBRA*.

The set of the parameter values will be done in three metaheuristic algorithms proposed in the literature for solving a new scheduling problem known as Response Time Variability Problem (RTVP). The RTVP arises whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the points at which they receive the necessary resources is minimized. The variability is measured with the RTV metric, which is a weighted variance. The objective of the RTVP is obtaining a sequence with the minimal RTV value. This problem is NP-hard (for more details, see Corominas *et al* (2007). The following metaheuristic algorithms have been chosen to analyse the fine-tuning procedure: a particle swarm optimisation (PSO) algorithm proposed in García-Villoria and Pastor (2009) called *DPSOpoi-*c_p*dyn* by the authors, the psychoclonal algorithm proposed in García-Villoria and Pastor (2008a) and the electromagnetism-like mechanism (EM) algorithm proposed in García-Villoria and Pastor (2008b).

CALIBRA and N&M evaluate the performance of the parameter values as follows. A training set of 60 representative instances is solved with a computing time limit of 50 seconds for each instance. The performance is the average RTV values of the solutions (sequences) obtained. The used training set is the training set used in García-Villoria and Pastor (2008a, 2008b, 2009) for fine-tuning their algorithms. Since the evaluation of each iteration of the calibration takes 50 minutes, we use CALIBRA in *CALIBRA+N&M* and in *CALIBRA+CALIBRA* with the minimal number of iterations allowed by the tool ($2^k + 9$, where k is the number of parameters to be fine-tuned). Thus, in order to make a fair comparison, N&M is stopped after ($2^k + 9$)*50 minutes although it had not converged to a local optimum.

In order to test the obtained calibration, the algorithm run 740 test instances used in García-Villoria and Pastor (2008a, 2008b, 2009) using the parameter values of the calibration.

5.1. Fine-tuning of the PSO algorithm

Parameters		N	ω	c_1	c_2	c_p
CALIBRA (Step 1)						
Ranges		[1..100]	[0..1]	[0..1]	[0..1]	[0..10]
Precision		0	2	2	2	1
Values		13	0.75	0.13	0.75	8.7
$\overline{RTV}$	50 s.	4,625.54				
	1,000 s.	1,537.34				
CALIBRA + *N&M* (Step 2)						
Start point		(13, 0.75, 0.13, 0.75, 8.7)				
Values		13	0.853	0.188	0.810	8.584
$\overline{RTV}$	50 s.	3,992.28				
	1,000 s.	794.93				
CALIBRA+*CALIBRA* (Step 2)						
Ranges		[1..25]	[0.6..0.9]	[0..0.25]	[0.6..0.9]	[6..10]
Precision		0	2	2	2	1
Values		3	0.67	0.16	0.86	7.0
$\overline{RTV}$	50 s.	4,063.69				
	1,000 s.	1,115.72				

Table 1. Fine-tuning of the PSO algorithm

The *DPSOpoi-$c_p$dyn* algorithm has 5 parameters: size of the population (N), ω, c_1, c_2 and c_p. By definition, ω, c_1, and c_2 values are between 0 and 1. Table 1 shows the ranges and their precision (in number of decimals) used in CALIBRA (*Ranges* and *Precision*, respectively), the start point used in N&M (*Start point*), the returned values of the parameters (*Values*) and the average RTV values obtained by the PSO algorithm (using the parameter values returned) for the 740 test instances when the algorithm is run 50 and 1,000 seconds ($\overline{RTV}$).

It can be observed that the calibration obtained using two times CALIBRA (*CALIBRA+CALIBRA*) improves the solutions 12.15% and 27.43% with respect to using one time CALIBRA for 50 and 1,000 computing seconds, respectively. But the improvement obtained using the proposed fine-tuning procedure, *CALIBRA+N&M*, is still better: 13.69% and 48.29%.

5.2. Fine-tuning of the psychoclonal algorithm

Parameters		*N*	*n*	*β*	*d*	*K*
CALIBRA (Step 1)						
Ranges		[1..200]	[1..200]	[0..10]	[1..200]	[3..10]
Precision		0	0	1	0	1
Values		25	3	1.3	3	7.6
$\overline{RTV}$	50 s.	235.68				
	1,000 s.	161.60				
CALIBRA + *N&M* (Step 2)						
Start point		(25, 3, 1.3, 3, 7.6)				
Values		25	8	1.538	0	7.581
$\overline{RTV}$	50 s.	188.86				
	1,000 s.	160.72				
CALIBRA+CALIBRA (Step 2)						
Ranges		[1..40]	[0..10]	[0..2.5]	[0..10]	[6..9]
Precision		0	0	1	0	1
Values		5	1	1.9	0	8.3
$\overline{RTV}$	50 s.	208.49				
	1,000 s.	169.58				

Table 2. Fine-tuning of the psychoclonal algorithm

The psychoclonal algorithm has 5 parameters: size of the population (*N*), *n*, *β*, *d*, and *K*. Table 2 shows the fine-tuning process of the psychoclonal algorithm.

The calibration obtained using *CALIBRA+CALIBRA* improves the solutions 11.54% with respect to using one time CALIBRA for 50 executing seconds, whereas *CALIBRA+N&M* improves 19.87%. On the other hand, for 1,000 computing seconds, *CALIBRA+CALIBRA* get 4.94% worse with respect to using one time CALIBRA, whereas *CALIBRA+N&M* improves slightly (0.54%).

5.3. Fine-tuning of the EM algorithm

The EM algorithm has 2 parameters: size of the population (*N*) and *lsiter*, both integers. Table 3 shows the fine-tuning process of the EM algorithm.

Parameters		*N*	*lsiter*
CALIBRA (Step 1)			
Ranges		[1..100]	[0..20]
Precision		0	0
Values		25	5
$\overline{RTV}$	50 s.	3,747.05	
	1,000 s.	330.29	
CALIBRA + N&M (Step 2)			
Start point		(25, 5)	
Values		26	6
$\overline{RTV}$	50 s.	3,683.46	
	1,000 s.	295.31	
CALIBRA+CALIBRA (Step 2)			
Ranges		[10..40]	[0..15]
Precision		0	0
Values		17	3
$\overline{RTV}$	50 s.	3,930.48	
	1,000 s.	426.58	

Table 3. Fine-tuning of the EM algorithm

The results obtained using *CALIBRA+CALIBRA* are worse than the results obtained using one time CALIBRA (4.90% and 29.15% worse for 50 and 1,000 computing seconds, respectively). On the other hand, *CALIBRA+N&M* always improves the results with respect to CALIBRA (1.70% and 10.59% better for 50 and 1,000 computing seconds, respectively).

6. Conclusions

In this paper we propose a systematic procedure for fine-tuning metaheuristics. Even though the performance of the metaheuristic algorithms proposed in the literature may be very sensitive to the parameter values, the selection of the values is usually not enough justified. The proposed procedure is able to find good parameter values for all kind of instance of the problem spending little computing time.

Other fine-tuning designs have been proposed in the literature, but they have the disadvantage of being very laborious and human-time consuming. Instead, the procedure that we propose needs of little human intervention. Thus, we believe that the proposed fine-tuning procedure can be very useful for researchers and practitioners.

CALIBRA is a valuable tool for a first and quick approximation to good parameter values. In the case that more fine-tuning time is available, the CALIBRA authors suggest applying again CALIBRA in a narrow range around the obtained parameter values. The computational experience shows that the proposal introduced in this paper is better. That is, applying the Nelder & Mead algorithm to fine-tuning the parameter values is better than applying again CALIBRA.

Acknowledgements

The authors gratefully acknowledge the support of grant DPI2007-61905 (Ministry of Education and Science, Spain, and FEDER).

REFERENCES

Altinel IK and Öncan T (2005). A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem. *Journal of the Operational Research Society* **56**: 954-961.

Adenso-Díaz B and Laguna M (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research* **54**: 99-114.

Barr RS, Golden BL, Kelly JP, Resende MGC and Stewart WR (1995). Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics* **1**: 9-32.

Battarra M, Golden B and Vigo D (2008). Tuning a parametric Clarke-Wright heuristic via a genetic algorithm. *Journal of the Operational Research Society* **59**: 1568-1572.

Chelouah R and Siarry P (2005). A hybrid method combining continuous tabu search and Nelder-Mead simplex algorithms for the global minimization of multiminima functions. *European Journal of Operational Research* **161**: 636-654.

Corominas A, Kubiak W and Moreno N (2007). Response time variability. *Journal of Scheduling* **10**: 97-110.

Coy SP, Golden BL, Runger GC and Wasil EA (2001). Using Experimental Design to Find Effective Parameter Settings for Heuristics. *Journal of Heuristics* **7**: 77-97.

Eiben AE, Hinterding R and Michalewicz Z (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation* **3**: 124-141.

García-Villoria A and Pastor R (2009). Introducing dynamic diversity into a discrete particle swarm optimization. *Computers & Operations Research* **36**: 951-966.

García-Villoria A and Pastor R (2008a). Solving the Response Time Variability Problem by means of a psychoclonal approach. *Journal of Heuristics*, In Press, Corrected Proof, available online, 16 July 2008, doi:10.1007/s10732-008-9082-2.

García-Villoria A and Pastor R (2008b). Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism. *Technical Report IOC-DT-P-2008-03*, Technical University of Catalonia (UPC), Spain. Available at http://hdl.handle.net/2117/2013.

Nelder JA and Mead R (1965). A simplex method for function minimization. *The Computer Journal* **7**: 308-313.

Taguchi G (1987). *System of Experimental Design: Engineering Methods to Optimize Quality and Minimize Costs*. Vols. 1 & 2, UNIPUB/Kraus International Publications: New York.

Article submitted to Operations Research (1st review in progress)

An adaptive-based heuristic for the Response Time Variability Problem[†]

Said SALHI[a] and Alberto GARCÍA-VILLORIA[b*]
[a] The Centre for Logistics & Heuristic Optimisation (CLHO), Kent Business School, University of Kent at Canterbury, Canterbury CT2 7PE, UK
[b] Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
s.salhi@kent.ac.uk, alberto.garcia-villoria@upc.edu

Abstract. The Response Time Variability Problem (RTVP) is an NP-hard combinatorial scheduling problem which has recently been reported and formalised in the literature. This problem has a wide range of real-world applications in mixed-model assembly lines, multi-threaded computer systems, broadcast of commercial videotapes and others. The RTVP arises whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the points at which they receive the necessary resources is minimised. We propose a greedy but adaptive heuristic that avoids being trapped into a poor solution by incorporating a look ahead strategy suitable for this particular scheduling problem. The proposed heuristic outperforms the best existing methods, while being much faster and easier to understand and to implement.

Keywords: response time variability, heuristics, adaptive search, scheduling, fair sequences

1. Introduction

The concept of a *fair sequence* has emerged independently from scheduling problems of diverse environments. The common aim of these scheduling problems, as defined in Kubiak (2004), is to build a fair sequence using n symbols, where symbol s ($s = 1,...,n$) must occur d_s times in the sequence. The fair sequence is the one which allocates a fair share of positions to each symbol s in any subsequence. This fair or ideal share of positions allocated to symbol s in a subsequence of length k is proportional to the relative importance (d_s) of symbol s with respect to the total copies of competing symbols (equal to $\sum_{s=1}^{n} d_s$). There is no universal definition of fairness because several reasonable metrics can be defined according to the specific problem considered.

Among the different definitions of fairness, several fair sequencing problems have emerged, among them the Response Time Variability Problem (RTVP). This problem

† Supported by the Spanish Ministry of Education and Science under project DPI2007-61905; co-funded by the ERDF. And supported by the Department of Innovationm, Universities and Enterprise of Generalitat de Catalunya under grant BE-DGR-2008.
* Corresponding author: Alberto García-Villoria, Institute of Industrial and Control Engineering (IOC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; tel.: +34 93 4054010; e-mail: alberto.garcia-villoria@upc.edu. The research was conducted while visiting CLHO at Kent.

has been reported for the first time by Waldspurger and Weihl (1994) but formalised several years later by Corominas *et al*. (2007). In the RTVP, the fair sequence is the one which minimises the sum of the variability in the distances between any two consecutive copies of the same symbol. In other words, the distance between any two consecutive copies of the same symbol should be as regular as possible (i.e., ideally constant).

In practice, the RTVP arises whenever products, clients or jobs need to be sequenced so as to minimise the variability in the time between the instants at which they receive the necessary resources (Corominas *et al*., 2007). This problem has a broad range of real-world applications. These include, for instance, the sequencing of mixed-model assembly lines under JIT (Kubiak, 1993; Miltenburg, 1989), the resource allocation in computer multi-threaded systems such as operating systems, network servers and media-based applications (Dong *et al*., 1998; Waldspurger and Weihl, 1994, 1995), the periodic machine maintenance problem when the times between consecutive services of the same machine are equal (Anily *et al*., 1998; Wei and Liu, 1983), the collection of waste (Herrmann, 2007), the schedule of commercial videotapes for television (Bollapragada *et al*., 2004; Brusco, 2008) and the design of sales catalogues (Bollapragada *et al*., 2004).

Corominas *et al*. (2007) studied the computational complexity of the RTVP and proved that it is NP-hard. Since this problem is a difficult combinatorial optimisation problem, several heuristic and metaheuristic algorithms have been proposed for its solution. Waldspurger and Weihl (1994) propose an algorithm that generates a solution randomly. The same authors (Waldspurger and Weihl, 1995) improve their previous results using the Jefferson method of apportionment (Balinski and Young, 1982), a greedy heuristic algorithm which they renamed as the stride scheduling technique. Herrmann (2007) solved the RTVP by applying a heuristic algorithm based on the stride scheduling technique. Corominas *et al*. (2007) proposed the Jefferson method together with other four constructive type heuristic algorithms. Seven new heuristics are also given by Corominas *et al*. (2009). Metaheuristics for the RTVP were recently proposed in García-Villoria and Pastor (2008, 2009, 2010) and these include a psychoclonal algorithm, an electromagnetism-like mechanism (EM) algorithm, and a genetic algorithm (GA) respectively.

The best five classical heuristics are described by (Corominas *et al*., 2009) and known as *Oc*, *AWe/dg*, *We/dg*, *Je/dg* and *In*. On the other hand, the best results recorded to date using relatively a larger computing time have been obtained with a GA (García-Villoria and Pastor, 2010).

In this paper, a simple constructive greedy heuristic using an adaptive search based on a look ahead strategy is proposed. The reasoning behind this approach and a couple of theorems to support it are put forward. An extensive computational experiment is carried out to assess the superiority of this heuristic over the aforementioned classical heuristics for both solution quality and computational effort. Moreover, the solutions obtained with the proposed heuristic are also found competitive when compared to the GA while requiring a fraction of its cpu time.

In this study, we also introduce a new but related scheduling problem for the first time that we call *minmax* RTV problem. In this problem, the objective is to minimise the maximum absolute discrepancy in the distances between any two consecutive copies of the same symbol. Although the heuristic introduced in this paper has been specifically designed to solve the RTVP, the way the look ahead strategy is defined led itself to solve the *minmax* RTVP as well. The obtained results are reported here to provide a platform for benchmarking purposes in the future.

The remainder of the paper is organised as follows: First, Section 2 presents a formal definition of the RTVP. The next section represents the main body of the research and it covers the new heuristic algorithm, the supporting theorems and the proposed enhancements. The results of our computational experiment are presented in Section 4. A new but related problem, the *minmax* RTVP, is briefly described and its results summarised in Section 5. Finally, some conclusions and suggestions for future research are provided in the last section.

2. The Response Time Variability Problem (RTVP)

The RTVP is formulated as follows. Let *n* be the number of symbols, d_s the number of copies to be sequenced of symbol *s* (*s* = 1,…,*n*) and *D* the total number of copies ($\sum_{s=1..n} d_s$). Let *seq* be a solution of an instance in the RTVP that consists of a circular sequence of copies ($seq = s_1 s_2 \ldots s_D$), where s_j is the copy sequenced in position *j* of sequence *seq*. For each symbol *s* in which $d_s \geq 2$, let t_k^s be the distance between the positions in which the copies *k* + 1 and *k* of symbol *s* are found. We consider the distance between two consecutive positions to be equal to 1. Since the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_s}^s$ is the distance between the first copy of symbol *s* in a cycle and the last copy of the same symbol in the preceding cycle. Let $\overline{t_s}$ be the ideal average distance between two consecutive copies of symbol *s* ($\overline{t_s} = D/d_s$). Note that for each symbol *s* in which $d_s = 1$, t_1^s is equal to $\overline{t_s}$. The objective is to minimise the metric called response time variability (RTV), which is defined by the sum of the square errors with respect to the $\overline{t_s}$ distances. This is defined as $RTV = \sum_{s=1}^{n} \sum_{k=1}^{d_s} (t_k^s - \overline{t_s})^2$.

For an illustration, consider the following example. Let $n = 3$ with symbols *A*, *B* and *C*. Also consider $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\overline{t_C} = 2$. Any sequence such that contains symbol *s* $(\forall s)$ exactly d_s times is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a feasible solution, and has an RTV value = $\left((5-4)^2+(3-4)^2\right)+\left((2-4)^2+(6-4)^2\right)+\left((2-2)^2+(2-2)^2+(3-2)^2+(1-2)^2\right)$ = 12.

3. An adaptive heuristic for the RTVP

In this section we propose a constructive greedy heuristic to solve the RTVP which uses a look ahead strategy. The heuristic has D ($\sum_{s=1..n} d_s$) steps and at each step p (p = 1,…,D) it is decided which symbol is sequenced at position p of the sequence. In fact, it could be considered that the heuristic has $D-1$ steps since the symbol to be sequenced at the last step will be automatically determined. The reasoning behind the strategy to select the symbol to be sequenced at each step is discussed in subsection 3.1 which also contains two theorems to support our selection process. The initial implementation is explained in subsection 3.2 and several enhancements are then proposed in subection 3.3.

3.1. The basic idea of the heuristic

Let first introduce some additional nomenclature:

seq_p:	The partial sequence obtained at step p; $p = 0,\ldots,D-1$. Initially seq_0 is a void sequence
$\hat{d}(s,p)$:	The number of times left for symbol s to be sequenced in seq_p; $s = 1,\ldots,n$, $p = 0,\ldots,D-1$
$SS(p)$:	The set of symbols that have been sequenced in seq_p at least once; $p = 0,\ldots,D-1$
$lsp(s, p)$:	The last position in which symbol s has been sequenced in seq_p; $s \in SS(p)$, $p = 0,\ldots,D-1$
$t(s, p)$:	$p - lsp(s, p-1)$; $s \in SS(p-1)$, $p = 1,\ldots,D$
$S^+(p)$:	The set of symbols $\left\{ s \in S\,S(p-1) \mid t(s,p) \geq \overline{t}_s \wedge \hat{d}(s,p-1) \geq 1 \right\}$; $p = 1,\ldots,D$
$S^-(p)$:	The set of symbols $\left\{ s \in S\,S(p-1) \mid t(s,p) < \overline{t}_s \wedge \hat{d}(s,p-1) \geq 1 \right\}$; $p = 1,\ldots,D$

Given a partial solution sequence seq_{p-1}, the aim is to decide which symbol to be sequenced at position p (p = 1,…,D). The symbols that still have copies to be sequenced at step p (that is, all symbol s (s = 1,…,n) such as $\hat{d}(s,p-1) \geq 1$) can be grouped into either the set $S^+(p)$ or the set $S^-(p)$. Given a symbol $s \in S^+(p)$ and a symbol $s' \in S^-(p)$, if one of them has to be sequenced at step p, then the decision that gives the lowest increment to the RTV value of the partial solution for the symbols s and s' is to sequence the symbol s in position p and to sequence the symbol s' in a later position. The validity of this claim is shown in Theorem 1. The reasoning behind this argument is that we try to avoid accumulating an excessive future increase in the distance between the next copy to be sequenced of symbol s and its last sequenced copy. This is important as the square error between ideal distances and real distances is used and this can be amplified very quickly. On the other hand, we allow that the distance between the next copy to be sequenced of symbol s' and its last sequenced copy increases. Note that its discrepancy between this distance and $\overline{t}_{s'}$ will be reduced as shown in Figure 1.

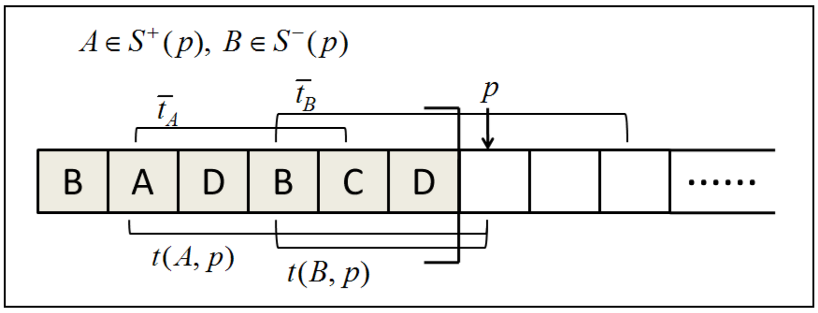


Figure 1. A graphical illustration of the sequencing distance concept

Theorem 1 Let *seq*$_{p\text{-}1}$ be a partial sequence solution obtained at step *p*-1 (*p* = 1,…,*D*). Given a symbol $s \in S^+(p)$ and a symbol $s' \in S^-(p)$, if one of them has to be sequenced at step *p*, then the best decision is to sequence the symbol *s* in position *p* and the symbol s' in a later position *p'* (*p'* > *p*).

Proof. By definition of the sets $S^+(p)$ and $S^-(p)$, we have that $t(s,p)-\bar{t}_s > t(s'p)-\bar{t}_{s'}$ or, equivalently, $t(s,p)-\bar{t}_s = t(s'p)-\bar{t}_{s'}+u$ where $u>0$. Analogously, $p'=p+q$ where $q \ge 1$. Consider the two possible options for sequencing the two symbols.

Option 1: The symbols *s* and *s'* are sequenced in the positions *p* and *p'*, respectively. The increment of the RTV value (Δ^1_{RTV}) is the following:
$\Delta^1_{RTV} = \left(t(s,p)-\bar{t}_s\right)^2 + \left(t(s',p')-\bar{t}_{s'}\right)^2 = \left(t(s',p)-\bar{t}_{s'}+u\right)^2 + \left(t(s',p+q)-\bar{t}_{s'}\right)^2 =$
$\left(t(s',p)-\bar{t}_{s'}+u\right)^2 + \left(t(s',p)+q-\bar{t}_{s'}\right)^2.$

Option 2: The symbols *s* and *s'* are sequenced in the positions *p'* and *p*, respectively. The increment of the RTV value (Δ^2_{RTV}) is the following:
$\Delta^2_{RTV} = \left(t(s,p')-\bar{t}_s\right)^2 + \left(t(s',p)-\bar{t}_{s'}\right)^2 = \left(t(s,p+q)-\bar{t}_s\right)^2 + \left(t(s',p)-\bar{t}_{s'}\right)^2 =$
$\left(t(s,p)+q-\bar{t}_s\right)^2 + \left(t(s',p)-\bar{t}_{s'}\right)^2 = \left(t(s',p)+q-\bar{t}_{s'}+u\right)^2 + \left(t(s',p)-\bar{t}_{s'}\right)^2.$

Let $\theta = t(s'p)-\bar{t}_{s'}$. Thus, $\Delta^1_{RTV} = (\theta+u)^2 + (\theta+q)^2 = 2\theta^2 + 2\theta u + 2\theta q + u^2 + q^2$ and
$\Delta^2_{RTV} = \left(\theta+(q+u)\right)^2 + \theta^2 = 2\theta^2 + 2\theta u + 2\theta q + u^2 + q^2 + 2q\ \ .u$
Therefore, $\Delta^2_{RTV} = \Delta^1_{RTV} + 2qu$. Since $q \ge 1$ and $u > 0 \Rightarrow \Delta^1_{RTV} < \Delta^2_{RTV}$. ■

We can generalize Theorem 1 by extending it for any pair of symbols *s* and *s'* without considering if they are included in the set $S^+(p)$ or in the set $S^-(p)$.

Theorem 2 Let *seq*$_{p\text{-}1}$ be a partial sequence solution obtained at step *p*-1 (*p* = 1,…,*D*). Given the symbols $s, s' \in SS(p)$, when one of them has to be sequenced at step *p*, then the best decision is to sequence the symbol $s^* = \arg\max_{i \in \{s,s'\}} \left(t(i,p)-\bar{t}_i\right)$ in position *p* and the other symbol s$^\#$ $\left(s^\# = \{s,s'\} - \{s^*\}\right)$ in a later position *p'* (*p'* > *p*).

Proof. By hypothesis, we have that $t(s^*,p)-\overline{t}_{s^*} \geq t(s^\#,p)-\overline{t}_{s^\#}$. If $t(s^*,p)-\overline{t}_{s^*} > t(s^\#,p)-\overline{t}_{s^\#}$ then we can apply Theorem 1. In the other hand, if $t(s^*,p)-\overline{t}_{s^*} = t(s^\#,p)-\overline{t}_{s^\#}$ then it is indifferent which of the two symbols is sequenced first. ■

Lemma. When all symbols have been sequenced at least once, the symbol $s^* = \underset{s\in SS(p)|\hat{d}(s,p)\geq 1}{\arg\max}\left\{t(s,p)-\overline{t}_s\right\}$ is sequenced at step *p*.

The above lemma constitutes the cornerstone idea in which the proposed heuristic will be based upon.

3.2. An initial implementation

We propose an initial heuristic based on Theorem 2 and the above lemma. At each step *p* (*p* = 1,…,*D*) of the heuristic, the symbols that still have copies to be sequenced are classified into the following three sets:

$S_1(p)$: The set of symbols $\left\{s\in\{1,\ldots,n\} \mid (d_s=1)\wedge\left(\hat{d}(s,p-1)=1\right)\right\};\ p=1,\ldots,D$

$S_2(p)$: The set of symbols $\left\{s\in\{1,\ldots,n\} \mid (d_s\geq 2)\wedge\left(\hat{d}(s,p-1)=d_s\right)\right\};\ p=1,\ldots,D$

$S_3(p)$: The set of symbols $\left\{s\in\{1,\ldots,n\} \mid (d_s\geq 2)\wedge\left(0<\hat{d}(s,p-1)<d_s\right)\right\};\ p=1,\ldots,D$

Note that the symbols with only one copy to be sequenced have the following interesting property. All symbol *s* of $S_1(p)$ (and, therefore, $t_1^s=\overline{t}_s$), will never increase the RTV value of the solution (this is explained in Section 2). The heuristic will sequence these symbols (i.e., those in which $d_s=1$) whenever it is *not suitable* to sequence any other symbol *s* from $S_2(p)$ or $S_3(p)$.

Let the function $\Delta(s,p)$ $\forall s\in S_1(p)\cup S_3(p)$ and $\forall p$ (*p* = 1,…,*D*) be defined as follows: $\Delta(s,p)=\begin{cases} t(s,p)-\overline{t}_s & \text{,if } d_s\geq 2 \\ 0 & \text{,if } d_s=1\end{cases}$

Note that, by definition, the symbols of the sets $S^+(p)$ have $\Delta\geq 0$, whereas those symbols of the sets $S^-(p)$ have $\Delta<0$. Ideally, the remaining copies of the symbols that have been sequenced at least once should be next sequenced at step *p* in which their Δ value is 0. In general, however, this is not always possible, so the idea is to sequence the symbols with the highest Δ value according to Theorem 2.

The pseudo-code of the proposed heuristic is shown in Figure 2. The algorithm has two phases. Let *R* be the number of steps used by the algorithm to sequence all symbols *s* in which $d_s \geq 2$ at least once. That is, *R* is the step in which $S_2(R+1)=\varnothing$ and $S_2(R)\neq\varnothing$. The first phase applies during the first *R* steps (lines 2 to 4 of the pseudocode) and the second phase uses the remaining $D-R$ steps (lines 5 and 6 of the pseudo-code).

```
8.  Let $seq_0$ be a void sequence
9.  For $p=1$ to $D$ do:
10.     If $S_2(p) \neq \varnothing$ then:
11.         If $\exists s: s \in S_3(p) \mid \Delta(s,p) \geq 0$ then $s_p^*$ is the symbol $s \in S_3(p)$ with
                the highest $\Delta(s,p)$ value. In case of tie, use the tie breaker of
                Figure 3.
12.         Otherwise $s_p^*$ is the symbol $s \in S_2(p)$ with the highest $d_s$ value. If
                there is a tie, use lexicographical order.
13.     Otherwise ($S_2(p) = \varnothing$):
14.         $s_p^*$ is the symbol $s \in S_1(p) \cup S_3(p)$ with highest $\Delta(s,p)$ value. In
                case of tie, use the tie breaker of Figure 3.
15.     $seq_p$ is obtained by sequencing $s_p^*$ in $seq_{p-1}$
16. Next $p$
17. Return $seq_D$
```

Figure 2. The pseudocode of the initial heuristic

- If there is a tie, select the symbol with the highest $\hat{d}(s,p)$ value.
- If there is again a tie, select the symbol with the highest d_s value.
- Finally, if there is a tie, use lexicographical order.

Figure 3. The tie breaker

Phase I. In this phase, all symbols s in which $d_s \geq 2$ are sequenced at least once. At each step p ($p = 1,\ldots,R$), only symbols of $S_2(p)$ or $S_3(p)$ are considered to be sequenced. The symbols of $S_1(p)$ are not considered in this phase because they are kept for the second phase to fill the positions which are not suitable for any other symbols. All symbols s in which $d_s = 1$ can be used as a *wild card*. The main objective of this phase is to sequence at least the first copy of all symbols s in which $d_s \geq 2$. However, if there is one or more symbols of $S_3(p)$ that have $\Delta \geq 0$, then the symbol with the highest value is selected.

Phase II. In this phase, all symbols s in which $d_s \geq 2$ have been sequenced at least once. Thus, according to Theorem 2, at each step p ($p = R+1,\ldots,D$), the symbol which has the highest Δ value is chosen. Note that if all symbols of $S_3(p)$ have a negative Δ value, then a symbol of $S_1(p)$ is sequenced (if $S_1(p)$ is not void), since its Δ value is 0. This scheme is introduced to stop the Δ values of the symbols of $S_3(p)$ to be increased at the next steps.

3.3. Enhancements

Three modifications to improve the performance of the initial heuristic are proposed in the following three subsections (3.3.1 to 3.3.3). The pseudo-code of the enhanced approach is given in the last subsection (3.3.4).

3.3.1. Effect of the distances between the first and last copies of the symbols

When the last copy of symbol *s* remains to be sequenced, only the distance between this copy and its second to the last copy (i.e., $t^s_{d_s-1}$) is taken into account. However, the distance between its last copy and its first copy in the preceding cycle (i.e., $t^s_{d_s}$) should also be taken into consideration. The function $\Delta(s,p)$ is therefore redefined to overcome this discrepancy:

$$\Delta(s,p)=\begin{cases} t(s,p)-\bar{t}_s & \text{if } \left(d_s\geq 2\right)\wedge\left(\hat{d}(s,p-1)\geq 2\right) \\ \left[t(s,p)-\bar{t}_s\right]+\left[\bar{t}_s-\left(D+fsp(s)\right)-p\right] & \text{if } \left(d_s\geq 2\right)\wedge\left(\hat{d}(s,p-1)=1\right) \\ 0 & \text{if } d_s=1 \end{cases}$$

where *fsp*(*s*) returns the first position in which symbol *s* has been sequenced.

3.3.2. Effect of the competition for the same position

The initial heuristic sequences, at each step *p*, a symbol of $S_2(p)$ (during the first phase) or a symbol of $S_1(p)$ (during the second phase) when all symbols of $S_3(p)$ have negative Δ values. However, there are situations in which it is better to sequence a symbol of $S_3(p)$ though its Δ value is negative.

A counter-example
Let $n=5$ with symbols *A*, *B*, *C*, *D* and *E* in which $d_A=1$, $\bar{t}_B=5.7$, $\bar{t}_C=3.9$, $\bar{t}_D=2.6$ and $\bar{t}_E=2.8$, and let suppose that at step *p* the sequence *seq_p* shown in Figure 4a has been generated.

The initial proposed heuristic will produce the partial sequence shown in Figure 4b as follows:

- At step p, $\Delta(A,p)=0$, $\Delta(B,p)=-1.7$, $\Delta(C,p)=-0.9$, $\Delta(D,p)=-0.6$ and $\Delta(E,p)=-1.8$; thus, the symbol *A* is sequenced since it has the highest Δ value.
- At step $p+1$, $\Delta(B,p+1)=-0.7$, $\Delta(C,p+1)=0.1$, $\Delta(D,p+1)=0.4$ and $\Delta(E,p+1)=-0.8$, so symbol *D* is sequenced.
- At step $p+2$, $\Delta(B,p+2)=0.3$, $\Delta(C,p+2)=1.1$, $\Delta(D,p+2)=-1.6$ and $\Delta(E,p+2)=0.2$, so symbol *C* is sequenced.
- At step $p+3$, $\Delta(B,p+3)=1.3$, $\Delta(C,p+3)=-2.9$, $\Delta(D,p+3)=-0.6$ and $\Delta(E,p+3)=1.2$, so symbol *B* is sequenced.

- At step $p+4$, $\Delta(B, p+4) = -4.7$, $\Delta(C, p+4) = -1.9$, $\Delta(D, p+4) = 0.4$ and $\Delta(E, p+4) = 2.2$, so symbol *E* is sequenced.

The increment of the RTV value obtained from the copies of the symbols *B*, *C*, *D* and *E* sequenced from step $p+1$ to step $p+4$ is $(7-5.7)^2 + (5-3.9)^2 + (3-2.6)^2 + (5-2.8)^2 = 7.9$.

On the other hand, a lower RTV increment could be obtained with the sequence shown in Figure 4c, which is $(6-5.7)^2 + (4-3.9)^2 + (2-2.6)^2 + (4-2.8)^2 = 1.9$. In this case, the symbol *D* has been sequenced at step *p* although $\Delta(D, p) = -0.6$.

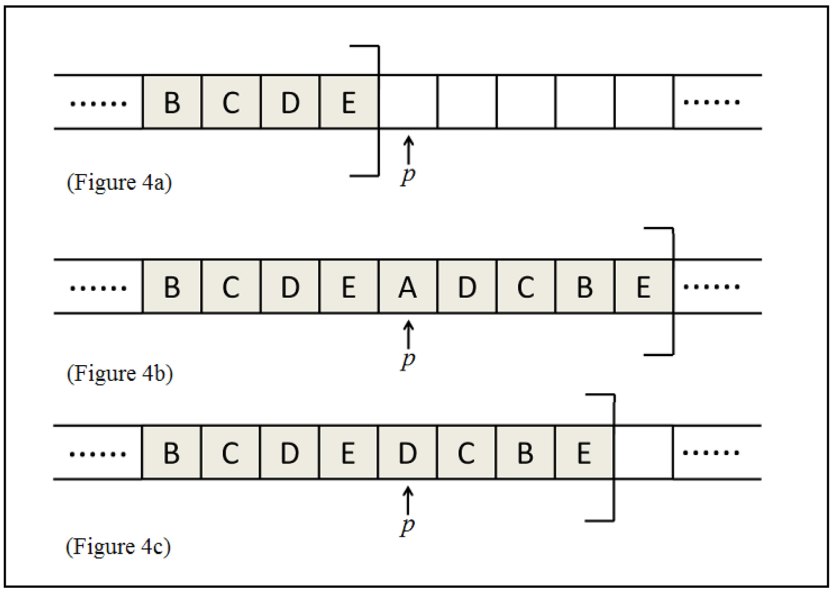


Figure 4. Different ways of sequencing

The proposed condition for sequencing at step *p* a symbol of $S_3(p)$ though all its symbols have a negative Δ value is that there could be *too many* symbols that would be sequenced during the next immediate positions of *p*. To overcome this shortcoming, the following condition is introduced:

$$\exists q \in \{p+1 \ldots D\} : \left|\tilde{S}_3(p,q)\right| \geq (q-p+M),$$

where M ($M \geq 1$) is a parameter that quantifies the effect of the cardinality of the set $\tilde{S}_3(p,q) = \{s \in S_3(p) : \Delta(s,p) + (q-p) \geq 0\}$. The value of M that obtains the best performance was found empirically to be 2.

3.3.3. Effect of dynamic ideal distances

In the initial heuristic, the ideal distance between two copies of symbol *s* is considered to be equal to $\overline{t_s}$ in all steps of the construction of the solution. On the other hand, it seems better to adjust dynamically the ideal distance of symbol *s* according to the current partial solution. This aims to sequence the remaining copies of *s* more regularly among the remaining positions. The adjusted ideal distances $\hat{t}(s,p)$ are then defined for all $s \in S_3(p)$ and for all step *p* (*p* = 1,…,*D*) as follows:

$$\hat{t}(s,p) = \frac{D - lsp(s,p-1) + fsp(s)}{\hat{d}(s,p-1)+1}$$

3.3.4. The enhanced heuristic

The pseudo-code of our enhanced heuristic is shown in Figure 5, with the summary of the modifications as explained in the last three subsections:

- $M = 2$
- $\Delta(s,p) = \begin{cases} t(s,p) - \hat{t}(s,p) & \text{, if } \left(d_s \geq 2\right) \wedge \left(\hat{d}(s,p) \geq 2\right) \\ \left[t(s,p) - \hat{t}(s,p)\right] + \left[\hat{t}(s,p) - \left(D + fsp(s)\right) - p\right] & \text{, if } \left(d_s \geq 2\right) \wedge \left(\hat{d}(s,p) = 1\right) \\ 0 & \text{,if } d_s = 1 \end{cases}$

  $p = 1,\ldots,D$, $\forall s \in S_1(p) \cup S_3(p)$
- $\tilde{S}_3(p,q) = \left\{ s \in S_3(p) : \Delta(s,p) + (q-p) \geq 0 \right\}$; $p = 1,\ldots,D$, $q = p+1,\ldots,D$

```
0.  Let $seq_0$ be a void sequence
1.  For $p = 1$ to $D$ do:
2.      If $S_2(p) \neq \varnothing$ then:
3.          If $\left(\exists s : s \in S_3(p) \mid \Delta(s,p) \geq 0\right) \vee$
            $\left(\exists q \in \{p+1 \ldots D\} : \left|\tilde{S}_3(p,q)\right| \geq (q-p+M)\right)$ then $s_p^*$ is the symbol
                $s \in S_3(p)$ with the highest $\Delta(s,p)$ value. In case of tie, use the
                tie break procedure of Figure 3.
4.          Otherwise $s_p^*$ is the symbol $s \in S_2(p)$ with the highest $d_s$ value. If
                there is a tie, use lexicographical order.
5.      Otherwise ($S_2(p) = \varnothing$):
6.          If $\left(\exists q \in \{p+1 \ldots D\} : \left|\tilde{S}_3(p,q)\right| \geq (q-p+M)\right)$ then $S' = S_3(p)$;
                otherwise, $S' = S_1(p) \cup S_3(p)$
7.          $s_p^*$ is the symbol $s \in S'$ with highest $\Delta(s,p)$ value. In case of tie, use
                the tie break procedure of Figure 3.
8.      $seq_p$ is obtained by sequencing $s_p^*$ in $seq_{p-1}$
9.  Next $p$
10. Return $seq_D$
```

Figure 5. The pseudocode of the enhanced heuristic

4. Computational results for the RTVP

To assess the performance of our proposed heuristic we conduct a large experiment of around 800 instances and compare our results against the best from the classical heuristics as well as the metaheuristics. All algorithms are coded in Java and executed on a 3.4 GHz Pentium IV with 1.5 GB of RAM.

4.1. Comparison vs. the best classical heuristics

The proposed heuristic is compared with the five best existing classical heuristics proposed (Corominas *et al*., 2009). Those are known as *Oc*, *AWe/dg*, *We/dg*, *Je/dg* and *In*. In their study, 600 test instances were used, which were grouped into three classes according to size (classes *CAT1* to *CAT3,* with 200 instances in each class). In this study, we also add 200 other larger test instances under class *CAT4*. All instances were generated using the random values of *D* (total number of copies) and *n* (number of symbols) shown in Table 1. For all instances and for each symbol $s = 1,\ldots,n$, a random number of copies to be sequenced of model *s* (d_s) is randomly generated between 1 and $\left|(D-n+1)/2.5\right|$ such that $\sum_{s=1..n} d_s = D$. The 800 instances are available at http://www.ioc.upc.edu/EOLI/research.

Table 1. Uniform distribution for the *D* and *n* values of the test instances

	CAT1	*CAT2*	*CAT3*	*CAT4*
D	U(25, 50)	U(50, 100)	U(100, 200)	U(200, 500)
n	U(3, 15)	U(3, 30)	U(3, 65)	U(3, 150)

The results are analysed by considering all the sets of instances as well as in each class of instances (*CAT1* to *CAT4*). We show the results of the proposed initial heuristic (let it be called *IN-H*) and those of the enhanced heuristic (let it be called *ENH-H*). The average RTV values of the solutions obtained with all heuristics are given in Table 2.

Table 2. Average RTV values obtained by the classical heuristics

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
ENH-H	**144.30**	**26.96**	**60.85**	**135.45**	**353.92**
IN-H	652.16	43.44	141.20	481.83	1,942.15
Oc	215.61	28.96	74.20	198.61	560.68
Awe/dg	405.88	47.03	120.32	349.13	1,107.03
We/dg	434.56	50.93	129.62	376.27	1,181.43
Je/dg	594.51	57.52	164.19	499.72	1,656.61
In	778.51	121.16	308.45	658.21	2,026.21

We can see in Table 2 that *Oc* was the best existing heuristic in the literature. This observation is valid for the overall RTV averages as well as in each class of instances (*CAT1* to *CAT4*). Our initial heuristic (*IN-H*) performs well but not as competitive. On the other hand, the enhanced heuristic (*ENH-H*) obtains, on average, better solutions than *Oc*. If we consider the results by class, *ENH-H* is 6.91%, 17.99%, 31.80% and 36.88% better than *Oc* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. Thus, the results point that the larger the instance, the more competitive is our heuristic. Moreover, *ENH-H* is much faster than *Oc* as it is shown in Table 3. On average, *ENH-H* requires only 1.82 milliseconds to solve an instance, whereas *Oc* needs 1,479.99 milliseconds (i.e., nearly 810 times slower).

Table 3. Average computing time (in milliseconds) used by the classical heuristics

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
ENH-H	0.72	0.12	0.22	0.43	2.12
IN-H	0.63	0.11	0.20	0.38	1.82
Oc	1,479.99	13.38	83.32	511.66	5,311.62
Awe/dg	4.56	0.86	1.45	3.91	12.01
We/dg	4.42	0.65	1.35	4.27	11.41
Je/dg	3.47	0.15	0.55	4.06	9.12
In	0.48	0.30	0.30	0.40	0.90

Robustness of the solutions
The dispersion with respect to the best RTV value obtained is also recorded. A measure of the dispersion (let it be *σ*) of the RTV values obtained by each algorithm, say *alg*, for a given instance, say *ins*, is defined as $\sigma(alg, ins) = \left(\left(\mathrm{RTV}_{ins}^{(alg)} - \mathrm{RTV}_{ins}^{(best)}\right) \big/ \mathrm{RTV}_{ins}^{(best)}\right)^2$, where $\mathrm{RTV}_{ins}^{(alg)}$ is the RTV value of the solution obtained with the algorithm *alg* for the instance *ins*, and $\mathrm{RTV}_{ins}^{(best)}$ is, for the instance *ins*, the best RTV value of the solutions obtained with all heuristics. Table 4 shows the average *σ* dispersion values.

Table 4. Average *σ* dispersion values regarding the best solution found by the classical heuristics

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
ENH-H	**0.11**	0.26	**0.07**	**0.03**	**0.07**
IN-H	19.00	3.69	5.66	12.64	54.01
Oc	0.27	**0.21**	0.23	0.35	0.28
Awe/dg	7.15	4.01	3.50	4.89	16.21
We/dg	9.09	4.44	4.11	6.45	21.36
Je/dg	22.18	8.29	8.57	15.66	56.22
In	48.27	54.52	85.67	21.82	31.06

ENH-H and *Oc* both obtain low averages of the *σ* dispersion values. This indicates that both algorithms are very stable especially our enhanced heuristic which besides outperforming all the other heuristics, it is found to be extremely robust and consistent in generating excellent results.

4.2. Comparison vs metaheuristics

We also compare the results of our heuristic with the best results obtained by the GA of García-Villoria and Pastor (2010). In this scenario, a set of 740 test instances is used instead. This is a subset of the 800 test instances (the other 60 instances were used to calibrate the parameters of the GA in their study). As in the previous subsection, these

740 instances are also grouped into four classes according to size (classes *CAT1'* to *CAT4'*, with 185 instances in each class). Table 5 shows the averages of the RTV values obtained by our proposed heuristic and the GA with 10, 50, 200, 500 and 1,000 seconds of computing time.

Table 5. Average RTV values for a computing time of 10, 50, 200, 500 and 1,000 seconds

		Global	***CAT1'***	***CAT2'***	***CAT3'***	***CAT4'***
ENH-H		159.50	27.56	62.76	151.91	395.77
	10 s.	1,245.10	12.13	31.85	111.47	4,824.94
	50 s.	186.94	11.65	29.41	84.54	622.16
GA	**200 s.**	131.81	11.34	28.26	77.81	409.84
	500 s.	114.39	11.00	27.63	75.59	343.33
	1,000 s.	106.68	10.92	27.00	74.86	313.92

On average, the GA is able to improve *ENH-H*. Observing the results by class, the metaheuristic algorithm obtains, on average, better solutions for all type of instances (*CAT1'* to *CAT4'*), though these results are not directly comparable due to the large difference in the computing times. For instance, the GA needs more than 200 seconds to obtains better results for the largest instances (*CAT4'*) while our heuristic requires tiny fraction of a second (0.72 milliseconds) only. As our heuristic is so fast and generates reasonably good solutions, it could be an invaluable tool to be incorporated within other powerful meta-heuristics for the generation of the initial solution, or be part of some exact methods for providing tighter upper bounds.

5. The *minmax* RTVP

As our approach is flexible enough to cater for other type objective functions, in this paper we introduced a related RTVP which we refer to as the *minmax* RTVP. Here, the objective is to minimise the metric that we call the maximum response time variability (*max*RTV). This is defined by the maximum of the absolute errors with respect to the $\bar{t}_s$ distances, $maxRTV = \max_{s=1}^{n} \max_{k=1}^{d_s} \left(\left| t_k^s - \bar{t}_s \right| \right)$.

For an illustration, consider the same example introduced in Section 2. That is, let $n = 3$ with symbols *A*, *B* and *C*. Also consider $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\bar{t}_C = 2$. Any sequence such that contains symbol *s* $(\forall s)$ exactly d_s times is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a feasible solution. The *maxRTV* value of the illustrative example is, therefore, $\max\left(\max\left(|5-4|,|3-4|\right),\max\left(|2-4|,|6-4|\right),\max\left(|2-2|,|2-2|,|3-2|,|1-2|\right)\right)=1\ 2.$

The *minmax* RTVP is solved for all 800 test instances using the original implementation (*IN-H*) and its enhanced version *ENH-H* algorithm. Since this is the first time in the literature this related problem is presented, there is obviously no comparison with other existing results. In Table 6, we provide our results which can be used for future benchmarking purposes which hopefully will entice other researchers to investigate this or related scheduling problems.

Table 6. Smallest, average and largest *max*RTV value obtained with *ENH-H* and *IN-H*

	***max*RTV**	**Global**	***CAT1***	***CAT2***	***CAT3***	***CAT4***
ENH-H	**Smallest**	0.58	0.91	0.58	1.38	1.58
	Average	3.17	2.19	2.72	3.43	4.33
	Largest	10.18	4.00	5.14	7.55	10.18
IN-H	**Smallest**	0.91	0.91	1.69	2.04	3.66
	Average	7.85	3.17	4.99	7.77	15.48
	Largest	80.40	8.07	14.11	22.27	80.40

6. Conclusions and future research

This paper proposes a new constructive greedy heuristic based on an adaptive search to solve the Response Time Variability Problem (RTVP). The RTVP is an NP-hard scheduling problem that appears in a broad range of real-life applications. Several heuristics and metaheuristic algorithms have been proposed in previous studies to solve the RTVP. The best solutions have been achieved by means of metaheuristics, but they need a lot of computing time (1,000 seconds). On the other hand, classical heuristics only require a fraction of that amount, but the solutions were usually found to be inferior.

The heuristic that we propose improves upon the performance of the best existing classical heuristics in terms of solution quality and computing time. Moreover, the solutions obtained are also competitive with the best solutions found by the existing metaheuristics while requiring a fraction of their computing time especially for the largest tested instances. In addition, we adopted this heuristic to tackle a related but a new scheduling problem namely the *minmax* RTVP with computational results for benchmarking purposes.

A promising line of research is to develop additional properties to make the enhanced heuristic even more powerful. Another simple way is to incorporate post optimisation. For instance partial enumeration can easily be implemented a few positions before the end, local search procedures as well as metaheuristics such as tabu search or simulated annealing can also be introduced. From a practical view point other metrics to define the fairness could also be attempted for this exciting scheduling problem. The commonly used measure between two successive symbols is one unit of distances, this could be generalised to be dependent on the type of symbols and their relationships. This additional feature will obviously make the problem more complex but practically interesting and academically challenging.

REFERENCES

Anily, S., Glass, C.A. and Hassin, R. (1998) ‘The scheduling of maintenance service’, *Discrete Applied Mathematics*, **82**, 27-42.

Balinski, M.L. and Young, H.P. (1982) *Fair Representation*, Yale University Press, New Haven.

Bollapragada, S., Bussieck, M.R. and Mallik, S. (2004) ‘Scheduling Commercial Videotapes in Broadcast Television’, *Operations Research*, **52**, 679-689.

Brusco, M.J. (2008) ‘Scheduling advertising slots for television’, *Journal of the Operational Research Society,* **59**, 1363-1372.

Corominas, A., Kubiak, W. and Moreno, N. (2007) ‘Response time variability’, *Journal of Scheduling*, **10**, 97-110.

Corominas, A., Kubiak, W. and Pastor, R. (2009) ‘Heuristic algorithms for solving the Response Time Variability problem’, *Technical report IOC-DT-P-2009-03*, Universitat Politècnica de Catalunya, Spain.

Dong, L., Melhem, R. and Mosse, D. (1998) ‘Time slot allocation for real-time messages with negotiable distance constrains requirements’, *Fourth IEEE Real-Time Technology and Applications Symposium* (RTAS'98), Denver, CO., pp. 131-136.

García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of a psychoclonal approach’, *Journal of Heuristics*, doi:10.1007/s10732-008-9082-2.

García-Villoria, A. and Pastor, R. (2009) ‘Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism’, *International Journal of Production Research,* doi: 10.1080/00207540902862545.

García-Villoria, A. and Pastor, R. (2010) ‘Solving the response time variability problem by means of a genetic algorithm’, *European Journal of Operational Research,* **202**, 320-327.

Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report TR 2007-12*, University of Maryland, USA. Available at http://hdl.handle.net/1903/7082.

Kubiak, W. (1993) ‘Minimizing variation of production rates in just-in-time systems: A survey’, *European Journal of Operational Research*, **66**, 259-271.

Kubiak, W. (2004) ‘Fair Sequences’, Chapter 19 in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapman and Hall.

Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, **35,** 192-207.

Waldspurger, C.A. and Weihl, W.E. (1994) ‘Lottery Scheduling: Flexible Proportional-Share Resource Management’, *First USENIX Symposium on Operating System Design and Implementation*, Monterey, California.

Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Scheduling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528,* Massachusetts Institute of Technology, MIT Laboratory for Computer Science. Available at https://eprints.kfupm.edu.sa/67117.

Wei, W.D. and Liu, C.L. (1983) ‘On a periodic maintenance problem’, *Operations Research Letters*, **2,** 90-93.

Article submitted to Journal of Scheduling (1st review in progress)

Metaheuristic algorithms hybridized with variable neighbourhood search for solving the response time variability problem[†]

Albert COROMINAS, Alberto GARCÍA-VILLORIA[*]and Rafael PASTOR
Institute of Industrial and Control Engineering (IOC)
Universitat Politècnica de Catalunya (UPC)
{albert.corominas / alberto.garcia-villoria / rafael.pastor}@upc.edu

Abstract. The response time variability problem (RTVP) is a scheduling problem with a wide range of real-world applications: mixed-model assembly lines, multi-threaded computer systems, network environments, broadcast of commercial videotapes and machine maintenance, among others. The RTVP arises whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the points at which they receive the necessary resources is minimised. Since the RTVP is NP-hard, several heuristic and metaheuristic techniques are needed to solve non-small instances. The best procedure in the literature for the RTVP is an algorithm based on a variant of the variable neighbourhood search (NVS), called Reduced VNS (RVNS). We propose hybridizing with RVNS three existing algorithms based on tabu search, multi-start and particle swarm optimisation. The aim is to combine the strengths of the metaheuristics. A computational experiment is carried out and it is shown that, on average, all proposed hybrid methods are able to improve the best published solutions.

Keywords: response time variability, fair sequences, scheduling, just-in-time, variable neighbourhood search, hybrid metaheuristics

1. Introduction

The concept of *fair sequence* has emerged independently from scheduling problems in diverse environments. The common aim of these scheduling problems, as defined in Kubiak (2004), is to build a fair sequence using n symbols, where symbol i ($i = 1,...,n$) must occur d_i times in the sequence. The fair sequence is the one which allocates a fair share of positions to each symbol i in any subsequence. This *fair* or *ideal* share of positions allocated to symbol i in a subsequence of length k is proportional to the relative importance (d_i) of symbol i with respect to the total copies of competing symbols (equal to $\sum_{i=1..n} d_i$). There is not a universal definition of fairness, as several reasonable metrics can be defined according to the specific problem considered. For a detailed introduction to fair sequences, see Kubiak (2009).

[†] Supported by the Spanish Ministry of Education and Science under project DPI2007-61905; co-funded by the ERDF.
[*] Corresponding author: Alberto García-Villoria, Institute of Industrial and Control Engineering (IOC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; tel.: +34 93 4054010; e-mail: alberto.garcia-villoria@upc.edu

Among the different definitions of fairness, the concept of *response time variability* (RTV) has emerged. In RTV, the ideal distance for symbol i between any two consecutive copies of this symbol is equal to D/d_i, where D is the length of the sequence $\left(D = \sum_{i=1..n} d_i\right)$. The RTV metric is the sum, for all symbols i, of the squares of the differences between the ideal and the real distances corresponding to all pairs of consecutive copies of symbol i. Thus, the RTV metric measures the non-fairness of a sequence. The response time variability problem (RTVP) lies in finding the optimal sequence according to the RTV metric, that is, the sequence that minimises the RTV. Thus, the distance between any two consecutive copies of the same symbol should be as regular as possible (ideally constant).

This problem has a broad range of real-world applications. One of the first situations in which the idea of the regular sequence appeared was the sequencing of mixed-model assembly lines at Toyota Motor Corporation under the just-in-time (JIT) production system. Since Toyota popularized the just-in-time (JIT) production systems, the problem of sequencing on mixed-model assembly lines has acquired high relevance. One of the main aims of JIT is to eliminate sources of waste and inefficiency. In the case of Toyota, the main source of waste was the production of excessive volumes of stock. To solve this problem, JIT systems produce only the specific models required and in the quantities needed at any given time. According to Monden (1983), in this type of system the units should be scheduled in such a way that the consumption rates of the components in the production process remain constant. Miltenburg (1989) also studied this scheduling problem and considered only the demand rates for the models (Miltenburg, 1989; Kubiak, 1993). The problem proposed by Miltenburg intended to minimise variations in production rate in different models. However, feedback received from the manufacturing industry suggests that a good mixed-model sequence is one in which the distances between units of the same model are as regular as possible. One drawback of the Miltenburg problem is that, on the contrary of the RTVP, it takes the positions of the models with only one unit to be produced into account although the positions of these models are irrelevant for the regularity of the consumption rates.

The RTVP also appears in computer multithreaded systems (Waldspurger and Weihl, 1994 and 1995; Dong *et al*., 1998; Bar-Noy *et al*., 2002). Multithreaded systems (operating systems, network servers, media-based applications, etc.) do different tasks to attend to the requests of client programs that take place concurrently. These systems need to manage the scarce resources in order to service the requests of n clients. For example, multimedia systems must not display video frames too early or too late, because this would produce jagged motion perceptions (Kubiak, 2009). Waldspurger and Weihl, considering that resource rights could be represented by *tickets* and that each client i had a given number d_i of tickets, suggested the RTV metric to evaluate the sequence of resource rights.

Other contexts in which the RTVP can be applied are the design of sales catalogues (problem introduced in Bollapragada *et al*., 2004), the periodic machine maintenance problem (Anily *et al*., 1998; Wei and Liu, 1983) as well as other distance-constrained problems (e.g., see Han *et al*., 1996).

Two real-life cases of RTVP applications were reported in the literature. In Bollapragada *et al*. (2004), the study is motivated by the problem faced by the National Broadcasting Company (BNC) of U.S., one of the main firms in the television industry.

Major advertisers buy to BNC hundreds of time slots to air commercials. The advertisers ask to BNC that the airings of their commercials are evenly spaced as much as possible over the broadcast season. The problem solved finally is not the RTVP, but a non-cycling variant. This study is continued in Brusco (2008). In Herrmann (2007), the author came up with the RTVP while working with a healthcare facility that needed to schedule the collection of waste from waste collection rooms throughout the building. Based on data about how often a waste collector had to visit each room and in view of the fact that different rooms require a different number of visits per shift, the facility manager wanted these visits to occur as regular as possible so that excessive waste would not collect in any room. For instance, if a room needed four visits per eight-hour shift, it should be ideally visited every two hours.

Although the RTVP is in general NP-hard (Corominas *et al*., 2007), the two-symbol case can be optimally solved with a polynomial algorithm proposed in Corominas *et al*. (2007). For the other cases, Corominas *et al*. (2007) proposed a mixed-integer linear programming (MILP) model whose practical limit to obtain optimal solutions is 25 copies to be sequenced. Corominas *et al*. (2010) proposed an improved MILP model and increased the practical limit for obtaining optimal solutions from 25 to 40 copies to be sequenced.

For solving largest instances, heuristic methods have been proposed. This problem has been first time solved in Waldspurger and Weihl (1994) using a method that authors called *lottery scheduling*, which consists on generating a solution at random. Later, Waldspurger and Weihl (1995) used the Jefferson method of apportionment (Balinski and Young, 1982), a greedy heuristic algorithm which they renamed as the stride scheduling technique. Herrmann (2007) solved the RTVP by applying a heuristic algorithm based on the stride scheduling technique. An aggregation approach was used in Herrmann (2009). Corominas *et al*. (2007) proposed also the Jefferson method together with other four greedy heuristic algorithms and a local search method. García *et al*. (2006) proposed six metaheuristic algorithms: a multi-start (MS), a greedy randomized adaptive search procedure (GRASP) and four variants of a discrete particle swarm optimisation (PSO) algorithm. An enhanced multi-start algorithm and an enhanced GRASP algorithm were proposed in Corominas *et al*. (2008), and other ten discrete PSO algorithms were proposed in García-Villoria and Pastor (2009a). A cross-entropy method (CE) algorithm, a psychoclonal algorithm, an electromagnetism-like mechanism (EM) algorithm, and a genetic algorithm (GA) were used in García-Villoria *et al*. (2007) and García-Villoria and Pastor (2008, 2009b, 2010), respectively. Finally, two tabu search (TS) algorithms and a variable neighbourhood search (VNS) algorithm were proposed in Corominas *et al*. (2009a, 2009b, 2009c), respectively.

The best results when solving the RTVP has been achieved using the VNS metaheuristic (Corominas *et al*., 2009c). In order to improve the solution of the RTVP, we propose three hybrid solution approaches: an hybridization of TS with VNS (*TS+VNS*), an hybridization of MS with VNS (*MS+VNS*) and an hybridization of PSO with VNS (*PSO+VNS*). Hybrid frameworks can combine the strengths of different metaheuristics to obtain a more efficient method. In the proposed *TS+VNS*, TS is used as the main framework and the VNS principle of alternating dynamically between neighbourhoods is incorporated. On the other hand, VNS is used as an intensification phase in the proposed *MS+VNS* and *PSO+VNS*. A computational experiment shows the benefits of hybridizing with VNS since the obtained results are, on average, 11.71% better with respect to the best results published in the literature.

The remainder of the paper is organized as follows. Section 2 presents a formal definition of the RTVP. Section 3 justifies the hybrid algorithms proposed in this work and gives a detailed explanation of their design. Section 4 presents the results of a computational experiment. Finally, some conclusions are given in Section 5.

2. The Response Time Variability Problem (RTVP)

The RTVP is formulated as follows. Let *n* be the number of symbols to be sequenced (that represent products, clients, jobs, …), where symbol *i* (*i* = 1,...,*n*) is to be copied d_i times in the sequence (that represent the number of times that symbol *i* has to receive the resource) and *D* is the total number of copies ($\sum_{i=1..n} d_i$). Let *s* be a solution of an instance in the RTVP that consists of a circular sequence of copies ($s = s_1 s_2 \dots s_D$), where s_j is the copy sequenced in position *j* of sequence *s*. For each symbol *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the copies *k* + 1 and *k* of symbol *i* are found. We consider the distance between two consecutive positions to be equal to 1. Since the sequence is circular, position 1 comes immediately after the last position *D*; therefore, $t_{d_i}^i$ is the distance between the first copy of symbol *i* in a cycle and the last copy of the same symbol in the preceding cycle. Let $\overline{t_i}$ be the desired average distance between two consecutive copies of symbol *i* ($\overline{t_i} = D/d_i$). The objective is to minimise the metric called response time variability (RTV), which is defined by the sum of the square errors with respect to the $\overline{t_i}$ distances. Since the symbols *i* such that $d_i = 1$ do not intervene in the computation of RTV, we assume that for each of these symbols t_1^i is equal to $\overline{t_i}$. The aim is to minimise the metric RTV, which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \overline{t_i})^2 \qquad (1)$$

For example, let $n = 3$, $d_A = 3$, $d_B = 2$ and $d_C = 2$; thus, $D = 7$, $\overline{t_A} = 7/3$, $\overline{t_B} = 7/2$ and $\overline{t_C} = 7/2$. Any sequence that contains exactly d_i times the symbol *i* $(\forall i)$ is a feasible solution. For example, the sequence (A, B, A, C, B, A, C) is a feasible solution, where:

$$RTV = \left(\left(2 - 7/3\right)^2 + \left(3 - 7/3\right)^2 + \left(2 - 7/3\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) = 5/3$$

3. Hybrid algorithms for the RTVP

The best method to solve the RTVP is an algorithm proposed in Corominas *et al.* (2009c) which is based on a variant of the VNS metaheuristic called Reduced VNS

(RVNS). We propose three hybrid methods based on hybridizing RVNS with TS (*TS+VNS*), MS (*MS+VNS*) and PSO (*PSO+VNS*).

The TS algorithm proposed in Corominas *et al.* (2009b) is the second best method to solve the RTVP. On average, the TS algorithm slightly outperforms the VNS algorithm when solving instances up to 200 copies to be sequenced whereas VNS clearly outperforms TS when solving larger instances (Corominas *et al.*, 2009c). Thus, it seems natural to try combining the advantages of these two best metaheuristics to solve the RTVP. We propose an hybrid method based on applying the main TS framework but incorporating a mechanism that dynamically alters neighbourhood. This idea has been successfully applied to solve other scheduling problems (Xu *et al.*, 2006; Ekşioğlu *et al.*, 2008).

The RVNS algorithm has the following handicap. After certain computing time the search will be trapped in a local optimum with respect all neighbourhoods which may be not a global optimum. Thus, the multi-start technique (MS) could be applied in order to escape from the local optimum. MS consists on iteratively generating a random solution and then applying an intensification search. Two MS algorithms have been proposed to solve the RTVP (García *et al.*, 2006; Corominas *et al.*, 2008). In both, a local search is used in the intensification search. Although both MS algorithms have the potential to find good solutions, the applied local search needs a lot of time to converge to a local optimum when the initial random solution has a very bad quality. We propose to use the RVNS in the intensification phase instead.

On the other hand, in the classical PSO the process of diversification is more taken into account than the process of intensification (Tchomté and Gourgand, 2009). Maybe this is the reason of the bad performance (compared to other metaheuristic algorithms) of the PSO algorithms proposed in the literature to solve the RTVP (García *et al.*, 2006; García-Villoria and Pastor, 2009a). In order to overcome this shortcoming, Tasgetiren *et al.* (2007) and Anghinolfi and Paolucci (2009) propose to improve the best solution of the population at each iteration by means of a local search method and a stochastic local search method to solve the permutation flowshop problem and a single-machine tardiness problem, respectively. We propose to use the RVNS algorithm as the improvement mechanism.

We first explain in Subsection 3.1 the RVNS algorithm in which is based the three proposed hybrid algorithms. Then the TS, MS and PSO algorithms hybridized with VNS (*TS+VNS*, *MS+VNS* and *PSO+VNS*, respectively) are explained in Subsections 3.2, 3.3 and 3.4, respectively. Finally, the selection of the parameter values of all hybrid algorithms is explained in Subsection 3.5

3.1. A RVNS algorithm

Variable Neighbourhood Search (VNS) is a metaheuristic proposed in Mladenovic and Hansen (1997) for combinatorial optimization. The basic idea of VNS is applying a systematic change of neighbourhood within a local search method (Mladenovic and Hansen, 1997). VNS is based on the following three simple facts (Hansen and Mladenovic, 2003): 1) a local optimum with respect to one neighbourhood structure is not necessarily so with another, 2) a global optimum is a local optimum with respect to all possible neighbourhood structures, and 3) it has been observed empirically that for many problems local optima with respect to one or several neighbourhood structures are relatively close to each other.

In the basic VNS proposed in (Mladenovic and Hansen, 1997) there is a local search step, which can be costly in terms of cpu time for large instances of some problems (Hansen and Mladenovic, 2003). In Hansen and Mladenovic (1999) is proposed the Reduced VNS (RVNS), in which the local search step is removed. The general scheme of RVNS is shown in Figure 1.

```
1.  Select the set of neighbourhood structures $N_k$ ($k$=1..$k_{max}$), where $k_{max}$ is the
    number of neighbourhoods
2.  Let $S$ an initial solution
3.  While stopping condition is not reached do:
4.      Set $k$ := 1
5.      While $k \leq k_{max}$ do:
6.          Select a solution $S'$ at random from $N_k(S)$
7.          If the acceptance criterion is satisfied, then $S$ := $S'$ and $k$ := 1;
            otherwise $k$ := $k$ + 1
8.      End While
9.  End While
10. Return $S$
```

Figure 1. General scheme of RVNS

Corominas *et al.* (2009c) proposed a RVNS-based algorithm for solving the RTVP because it is shown that the local search step for large RTVP instances is very costly in terms of computing time. The following three neighbourhood structures are used: 1) interchanging each pair of two consecutive units of the sequence that represents the current solution (N_1), 2) interchanging each pair of consecutive or no-consecutive units of the sequence (N_2), and 3) inserting each unit in each position of the sequence (N_3). The acceptance criteria is that the neighbour solution S' is better than or equal to the current solution S.

3.2. TS+VNS

Local search methods have the great disadvantage that the local optimum found is often a fairly mediocre solution (Gendreau, 2003). To overcome this limitation, the Tabu Search metaheuristic (TS) has been proposed by Glover (1986). TS is based on applying a local search in which non-improving movements are allowed. To avoid cycling back to visited solutions, the most recent history of the search is recorded in a tabu list of tabu (forbidden) solutions. The complete tabu solutions could be recorded in the tabu list, but this may require a lot of memory, make it expensive to check whether a solution is tabu or not and, above all, does not diversify sufficiently the search. Thus, it is common to record only the last moves (transformations) performed on the current solution and forbidding reverse transformations (Gendreau, 2003). The tabu lists are usually implemented as a list of fixed length with a FIFO (First In, First Out) policy. A tabu solution can be overridden if a suitable aspiration criterion is met.

Two straightforward applications of the TS classical scheme shown in Figure 2 has been proposed in the literature to solve the RTVP (Corominas *et al.*, 2009a and 2009b). The only difference between both TS algorithms is the definition of the neighbourhood. In

Corominas *et al*. (2009a), the neighbourhood N_2 is used (see Subsection 3.1), whereas in Corominas *et al*. (2009b), the neighbourhood N_3 is used (see Subsection 3.1).

In the classical TS, the current solution is moved in an only neighbourhood structure. In our paper, we propose a TS algorithm based on changing dynamically between the neighbourhoods N_1, N_2 and N_3. Note that N_1 is included in N_2 or N_3, so it seems unnecessary. However, the benefit of using N_1 is that it helps to the algorithm to converge very fast without detrimental of its performance (Corominas *et al*., 2009c).

Some of the benefits of altering neighbourhood can be the following (Xu *et al*., 2006):

- Different neighbourhood moves bring in various degrees of changes for the new solution, so it carries diversification effects.
- Some neighbourhoods are harder to evaluate than others, but they are more effective in locating better solutions. Dynamic neighbourhood moves can better address the balance between the efficiency and effectiveness of the TS algorithm.

The mechanism of neighbourhood changing that we propose is the following. First, the TS algorithm starts using N_1. Each move that does not improve the best solution (S^*) is counted. When a maximum number of non-improvement moves is reached (max_nim_1), the next neighbourhood N_2 is used. Similarly, when max_nim_2 non-improvement moves is reached using N_2, the current neighbourhood is change to N_3. Again, when max_nim_3 non-improvement moves is reached using N_3, the process continues iteratively using N_1 again.

The scheme of *TS+VNS* is shown in Figure 2.

```
1.  Select the set of neighbourhood structures $N_k$ ($k$ = 1..3)
2.  Set the values of the parameters $max\_nim_k$ ($k$ = 1..3).
3.  Let $S$ an initial solution and $S^* := S$
4.  $k := 1$
5.  $ni := 0$
6.  While stopping condition is not reached do:
7.        Let $S'$ the best solution from $N_k(S)$ which is non-tabu or allowed by aspiration
8.        Add the current move in the tabu list (removing its last move if the list is full)
9.        If $S'$ is better than $S^*$, then $S^* := S'$
10.       Otherwise:    $ni := ni + 1$
                        If $ni = max\_nim_k$, then $k := (k \bmod 3) + 1$ and $ni := 0$
11.       $S := S'$
12. End While
13. Return $S^*$
```

Figure 2. Scheme of *TS+VNS*

The elements of the proposed algorithm for solving the RTVP are defined as follows:

- *Initial solution*. The initial solution is obtained from the best solution returned by the five heuristics proposed in Corominas *et al*. (2007).

- *Neighbourhoods*. The neighbourhoods N_1, N_2 and N_3 are the same neighbourhoods explained in Subsection 3.1.
- *Tabu moves*. A forbidden move of the tabu list consists of two pairs of position/symbol. For instance, the move [(3, A), (5, B)] means that all solutions with the symbol A sequenced in position 3 and the symbol B sequenced in position 5 are considered tabu. In the case of the neighbourhood N_3, if the symbol A is inserted into position 3, then the move [(3, A), (3, A)] is recorded in the tabu list.
- *Aspiration criterion*. The aspiration criterion is that the move produces a solution better than the best solution found in the past.
- *Stopping condition*. The algorithm stops once it has run for a preset time.

3.3. MS+VNS

The multi-start metaheuristic is a general scheme that consists of two phases. The first phase obtains an initial solution and the second phase improves the obtained initial solution. These two phases are applied iteratively until a stop condition is reached. This scheme has been first used at the beginning of 80's (Boender *et al.*, 1982). The generation of the initial solution, how to improve them and the stop condition can be very simple or very sophisticated. The combination of these elements gives a wide variety of multi-start methods. For a good review of multi-start methods, see Martí (2003) and Hoos and Stützle (2005).

We propose the following multi-start VNS algorithm. At each iteration, a random initial solution is obtained and then is improved by means of the RVNS algorithm explained in Subsection 3.1. The random solutions are generated as it is done in García *et al*. (2006) and Corominas *et al*. (2008). That is, for each position of the sequence, the symbol to be sequenced is chosen at random. The probability of each symbol of being chosen is equal to the number of its copies that remain to be sequenced divided by the total number of copies that remain to be sequenced. The stopping condition of the RVNS algorithm consists in reaching a maximum number of iterations without improving the current solution. The maximum number of iterations is $\lfloor D \cdot \alpha_{ms+vns} \rfloor$, where D is the total number of copies to be sequenced and α_{ms+vns} is the parameter of the algorithm. The multi-start VNS algorithm stops after it has run for a preset time. Figure 3 shows the scheme of the proposed multi-start VNS algorithm.

```
1. Let $S^*$ be a random solution
2. While stopping condition is not reached do:
3.        Let $S$ be a random solution
4.        Apply the RVNS algorithm to $S$ and get $S'$
5.        If $S'$ is better than $S^*$, then $S^* := S'$
6. End While
7. Return $S^*$
```

Figure 3. Scheme of *MS+VNS*

3.4. PSO+VNS

PSO is a population metaheuristic introduced by Kennedy and Eberhart (1995) which is based on the social behaviour of flocks of birds when they search for food. The population or swarm is composed of particles (birds), whose attributes are an *m*-dimensional real point (which represents a feasible solution) and a velocity (the movement of the point in the *m*-dimensional real space). The velocity of a particle is typically a combination of three types of velocities: 1) the inertia velocity (i.e., the previous velocity of the particle); 2) the velocity to the best point found by the particle; and 3) the velocity to the best point found by the population. These components of the particles are modified iteratively by the algorithm as it searches for an optimal solution. These modifications are formalized with the following two equations:

$$v_i^{t+1} = \omega \cdot v_i^t + c_1 r_1 \cdot \left(p_i^t - x_i^t \right) + c_2 r_2 \cdot \left(gbest - x_i^t \right), \tag{2}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{3}$$

where *t* is the current iteration, v_i^t is the current velocity of particle *i* at iteration *t*, *ω* is the *inertia* parameter that weights the previous velocity of particle *i*, c_1 and c_2 are two parameters multiplied by two random numbers, r_1 and r_2 respectively, uniformly distributed in the range [0, 1], x_i^t is the current point of particle *i* at iteration *t*, $\left(p_i^t - x_i^t \right)$ is the velocity towards the best point found so far by the particle, and $\left(gbest - x_i^t \right)$ is the velocity towards the best point found so far by the whole swarm.

One of the first attempts to solve the RTVP through metaheuristics was by means of PSO-based algorithms (García *et al.*, 2006). Later, more sophisticated PSO algorithms were proposed to solve the RTVP; the best of the PSO algorithms to solve the RTVP is called *DPSOpoi-c_pdyn* by its authors (García-Villoria and Pastor, 2009a). Although the PSO metaheuristic was originally designed for *m*-dimensional real spaces, *DPSOpoi-c_pdyn* is adapted to work with the sequence that represents a solution (for details, see García-Villoria and Pastor (2009a)). Moreover, *DPSOpoi-c_pdyn* introduces random modifications to the points of the particles after being applied Equations 2 and 3 with a frequency that changes dynamically as follows. For each position of the point (which is a sequence that represents a solution), the position has a probability c_p ($0 \le c_p \le 1$) of being swapped with another, randomly selected position. The parameter c_p changes dynamically according to the heterogeneity of the swarm at iteration *t* according to Equation 4:

$$c_p = e^{-K \cdot het(t)}, \tag{4}$$

where *K* is a parameter to be set, *het*(*t*) is a measure of the heterogeneity of the population defined as $het(t) = \sum_{i \in P} \left| v_i^t \right| \Big/ D \cdot |P|$, $\left| v_i^t \right|$ is the module of the velocity of particle *i* at iteration *t*, *D* is the total number of copies to be sequenced, and $|P|$ is the size of the population *P*.

We propose the following hybridization of *DPSOpoi-$c_p$dyn* with the RVNS algorithm. At each iteration, the best point (solution) found by the swarm, *gbest*, is improved by applying the RVNS algorithm explained in Subsection 3.1. The stopping condition of the RVNS algorithm consists on reaching a maximum number of iterations without improving the current solution. The maximum number of iterations is $\lfloor D \cdot \alpha_{pso-vns} \rfloor$, where D is the total number of copies to be sequenced and $\alpha_{pso+vns}$ is a parameter of the algorithm. The scheme of *PSO+VNS* is shown in Figure 4.

```
1. Initialize population
2. While stopping condition is not reached do:
3.     For each particle $i$ do:
4.         Update velocity of $i$ according to Equation 2
5.         Update point of $i$ according to Equation 3
6.         For each position of the point i, swap it with another position selected at
           random with a probability $c_p$ (Equation 4)
7.         Update best point of particle $i$
8.     End For
5.     Update best point of the population
6. End While
7. Return best point of the population
```

Figure 4. Scheme of *PSO+VNS*

3.5. Fine-tuning the algorithm parameters

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although the parameter values may have a very strong effect on the results of the metaheuristic for each problem, they are often selected using one of the following methods, which are not sufficiently thorough (Eiben *et al*., 1999; Adenso-Díaz and Laguna, 2006): 1) “by hand”, based on a small number of experiments that are not referenced; 2) using the general values recommended for a wide range of problems; 3) using the values reported to be effective in other similar problems; or 4) with no apparent explanation.

Adenso-Díaz and Laguna (2006) proposed a new technique called CALIBRA for fine-tuning the parameters of algorithms. CALIBRA is based on using conjointly Taguchi’s fractional factorial experimental designs and a local search procedure. We propose to use CALIBRA for setting the parameter values of our hybrid algorithms. CALIBRA was applied to a representative training set of 60 instances which were generated as explained in the Section 4. The following parameter values were obtained:

- *TS+VNS*: size of the tabu list = 127, max_nim_1 = 751, max_nim_2 = 8 and max_nim_3 = 26.

- *MS+VNS*: α_{ms+vns} = 37.5.

- *PSO+VNS*: size of the population = 6, ω = 0.87, c_1 = 0.75, c_2 = 0.87, K = 27.5 and $\alpha_{pso+vns}$ = 9.4.

Since CALIBRA cannot fine-tune more than five parameters, *PSO+VNS* (which have six parameters) are fine-tuned in two steps. In the first step, the initial value of the size of the population is set to 13 (which is the value used for *DPSOpoi-c$_p$dyn* in García-Villoria and Pastor, (2009)) and the remaining parameters (ω, c_1, c_2, K and $\alpha_{pso+vns}$) are fine-tuned. In the second step the value of ω is set at the value obtained in the first step and the remaining parameters (size of the population, c_1, c_2, K and $\alpha_{pso+vns}$) are fine-tuned.

4. Computational experiment

In the computational experiment the three proposed hybrid algorithms (*TS+VNS*, *MS+VNS* and *PSO+VNS*) and the four original metaheuristic algorithms –the RVNS algorithm proposed in Corominas *et al.* (2009c) (let it be called *RVNS$_{RTVP}$*), the TS proposed in Corominas *et al.* (2009b) (let it be called *TS$_{RTVP}$*), the MS proposed in García et al. (2006) (let it be called *MS$_{RTVP}$*) and the best PSO proposed in García-Villoria and Pastor (2009a) (called *DPSOpoi-c$_p$dyn*)– were run.

All algorithms are coded in Java and executed on a PC 3.4 GHz Intel Pentium IV with 1.5 GB of RAM. The same 60 training instances and 740 test instances used in García *et al.* (2006), Corominas *et al.* (2009b, 2009c) and García-Villoria and Pastor (2009a) are also used in this paper (all instances can be found at http://www.ioc.upc.edu/EOLI/research/). These instances were grouped into four classes (from *CAT1* to *CAT4* with 15 training instances and 185 test instances in each class) according to their size. The instances were generated using the random values of *D* (number of copies) and *n* (number of symbols) shown in Table 1. For all instances and for each model $i = 1,\ldots,n$, a random value of d_i (number of copies of symbol i) is between 1 and $\left|(D-n+1)/2.5\right|$ such that $\sum_{i=1..n} d_i = D$.

Table 1. Uniform distributions for generating the *D* and *n* values

	CAT1	***CAT2***	***CAT3***	***CAT4***
D	U(25, 50)	U(50, 100)	U(100, 200)	U(200, 500)
n	U(3, 15)	U(3, 30)	U(3, 65)	U(3, 150)

The stop condition of all algorithms is to be run for a preset time. We run the algorithms for 10, 50, 200, 500 and 1,000 seconds. The results obtained are shown and explained in the following two subsections. In Subsection 4.1, the original metaheuristic algorithms are compared with their hybrid version in order to show the benefits of hybridizing with *RVNS$_{RTVP}$*. In Subsection 4.2, the hybrid algorithms are compared with the two best methods up to now (*RVNS$_{RTVP}$* and *TS+VNS*) to solve the RTVP.

4.1. Original metaheuristics versus hybrid metaheuristics

Table 2 shows the results obtained with *TS$_{RTVP}$* and its hybrid version, *TS+VNS*. The addition of variable neighbourhood to the TS algorithm helps it to improve its performance for the largest instances (*CAT4* instances) and the hybrid version is able to obtain solutions 40.48% better, on average, for these instances after 1,000 computing seconds. With respect to the results obtained for small and medium instances (*CAT1* to *CAT3* instances), for which *TS$_{RTVP}$* was the best method after 1,000 computing seconds, no significant differences (according to the t-Test paired two sample for means with a confidence level of 95%) in the quality of the solutions with respect to *TS+VNS* are observed. Moreover, the hybrid version presents a faster convergence without or very little losing quality at the solutions. This tendency to faster convergence is especially observable in the medium and largest instances.

Table 2. Average RTV values for TS_{RTVP} and *TS+VNS*

		Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
10 s.	$\boldsymbol{TS_{RTVP}}$	**339.59**	10.42	25.32	128.29	1,194.31
	TS+VNS	**87.18**	10.63	25.43	60.67	251.99
50 s.	$\boldsymbol{TS_{RTVP}}$	**210.47**	10.26	22.56	73.26	735.78
	TS+VNS	**71.57**	10.38	24.00	53.99	193.83
200 s.	$\boldsymbol{TS_{RTVP}}$	**123.98**	10.25	21.67	55.53	408.47
	TS+VNS	**63.60**	10.27	23.08	50.57	170.49
500 s.	$\boldsymbol{TS_{RTVP}}$	**90.74**	10.24	21.29	50.19	279.37
	TS+VNS	**58.35**	10.24	22.74	49.04	151.38
1,000 s.	$\boldsymbol{TS_{RTVP}}$	**78.62**	10.24	21.16	48.12	234.96
	TS+VNS	**55.05**	10.24	22.48	47.66	139.84

Tables 3 shows the results obtained with MS_{RTVP} and *MS+VNS*. The advantages of incorporating $RVNS_{RTVP}$ in the MS algorithm are that much better solutions are obtained with a very fast convergence. After 1,000 computing seconds, the results obtained with *MS+VNS* are 6.31%, 40.98%, 73.08% and 97.26% better than the results of MS_{RTVP} for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. There is a clear tendency of the performance improvement of the hybrid algorithm when the size of the instances increases.

Table 3. Average RTV values for MS_{RTVP} and *MS+VNS*

		Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
10 s.	$\boldsymbol{MS_{RTVP}}$	**31,847.07**	13.26	52.09	2,582.45	124,740.50
	MS+VNS	**71.07**	10.24	21.58	51.07	201.39
50 s.	$\boldsymbol{MS_{RTVP}}$	**21,390.39**	12.08	44.36	226.90	85,278.25
	MS+VNS	**62.17**	10.24	21.23	47.46	169.76
200 s.	$\boldsymbol{MS_{RTVP}}$	**10,060.19**	11.43	39.50	185.85	40,003.97
	MS+VNS	**58.45**	10.24	21.01	45.54	158.10
500 s.	$\boldsymbol{MS_{RTVP}}$	**4,015.37**	11.10	36.74	171.40	15,842.23
	MS+VNS	**56.25**	10.24	20.97	43.97	149.83
1,000 s.	$\boldsymbol{MS_{RTVP}}$	**1,378.58**	10.93	35.48	160.67	5,307.25
	MS+VNS	**54.95**	10.24	20.94	43.26	145.35

Finally, the results obtained with *DPSOpoi-*c_p*dyn* and PSO+*VNS* are shown in Table 4. Similar to the comparison between the MS algorithm and its hybrid version, the benefits of hybridizing the PSO algorithm with the VNS algorithms are clear. For all class of instances, *PSO+VNS* has better performance. After 1,000 computing seconds, the results obtained with *PSO+VNS* are 27.18%, 52.74%, 67.49% and 97.57% better than the results of PSO_{RTVP} for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. The convergence of PSO+VNS is also better.

Table 4. Average RTV values for *DPSOpoi-c$_p$dyn* and *PSO+VNS*

		Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
10 s.	***DPSOpoi-c$_p$dyn***	**9,108.66**	17.36	82.37	1,512.67	34,822.26
	PSO+VNS	**81.84**	10.49	22.65	51.18	243.04
50 s.	***DPSOpoi-c$_p$dyn***	**4,625.46**	16.42	51.34	610.34	17,824.04
	PSO+VNS	**60.03**	10.47	22.42	49.57	161.91
200 s.	***DPSOpoi-c$_p$dyn***	**2,757.89**	15.47	48.88	262.78	10,704.43
	PSO+VNS	**57.37**	10.46	22.26	48.07	148.68
500 s.	***DPSOpoi-c$_p$dyn***	**1,964.62**	14.61	48.18	168.82	7,626.87
	PSO+VNS	**56.31**	10.45	22.03	47.55	145.22
1,000 s.	***DPSOpoi-c$_p$dyn***	**1,537**	14.35	46.55	143.95	5,944.51
	PSO+VNS	**55.86**	10.45	22.00	46.80	144.22

4.2. Hybrid metaheuristics versus best methods

The two most efficient methods to solve the RTVP proposed in the literature up to now are *RVNS$_{RTVP}$* and *TS$_{RTVP}$*. The TS algorithm is slightly better for solving small and medium RTVP instances whereas the VNS algorithm is clearly the best for solving the largest instances. Table 5 show the results obtained with these two algorithms and with the three proposed hybrid algorithms for 10, 50 and 1,000 computing seconds.

Table 5. Average RTV values for *RVNS$_{RTVP}$*, *TS$_{RTVP}$*, *TS+VNS*, *MS+VNS* and *PSO+VNS*

		Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
10 s.	***RVNS$_{RTVP}$***	**68.60**	10.73	23.72	52.87	187.07
	TS$_{RTVP}$	**339.59**	10.42	25.32	128.29	1,194.31
	TS+VNS	**87.18**	10.63	25.43	60.67	251.99
	MS+VNS	**71.07**	10.24	21.58	51.07	201.39
	PSO+VNS	**81.84**	10.49	22.65	51.18	243.04
50 s.	***RVNS$_{RTVP}$***	**63.69**	10.73	23.69	51.80	169.64
	TS$_{RTVP}$	**210.47**	10.26	22.56	73.26	735.78
	TS+VNS	**71.57**	10.38	24.00	53.99	193.83
	MS+VNS	**62.17**	10.24	21.23	47.46	169.76
	PSO+VNS	**60.03**	10.47	22.42	49.57	161.91
1,000 s.	***RVNS$_{RTVP}$***	**62.24**	10.73	23.69	51.40	163.69
	TS$_{RTVP}$	**78.62**	10.24	21.16	48.12	234.96
	TS+VNS	**55.05**	10.24	22.48	47.66	139.84
	MS+VNS	**54.95**	10.24	20.94	43.26	145.35
	PSO+VNS	**55.86**	10.45	22.00	46.80	144.22

Independently of the performance of the original metaheuristic algorithms, the solutions obtained with the hybrid algorithms are very good. Anyway, significant differences (with a confidence level of 95%) are observed after 1,000 computing seconds. For *CAT1* instances, *TS+VNS* and *MS+VNS* are 2.01% better than *PSO+VNS*. For *CAT2* and *CAT3* instances, *MS+VNS* is the algorithm that obtains best solution and is 4.82% and 7.56% better, respectively, than *PSO+VNS*, which is the second best algorithm for solving these instances. Finally, the best algorithm for the *CAT4* instances is *TS+VNS*, which is 3.04% better than the second best algorithm, *PSO+VNS*. With respect to the convergence, all hybrid heuristics converge very fast.

Comparing the proposed hybrid algorithms with the two best methods published in the literature, we can see that, on average, all proposed algorithms outperform them.

Observing the results by class, MS+VNS is able to obtain equal or better results for the best known results up to now of each class (for *CAT1* to *CAT3* instances, the best results were obtained with TS_{RTVP}; for *CAT4* instances, the best results were obtained with $RVNS_{RTVP}$). For the smallest instances, *MS+VNS* obtains equal results than TS_{RTVP}. For *CAT2* and *CAT3* instances, *MS+VNS* results are, on average, 1.04% and 10.10% better than the TS_{RTVP} results. Finally, for *CAT4* instances, *MS+VNS* is 11.20% better than $RVNS_{RTVP}$. And in global *MS+VNS* is 11.71% better than the best method published to date.

Table 5 shows that the proposed hybrid algorithms are able to obtain better solutions and faster than the previous methods proposed in the literature for solving the RTVP. However, we cannot know if the solutions are good. Thus, we have tried to find the optimal solutions but only the smallest instances (*CAT1* instances) were optimally solved. For the remaining instances, the lower bound (LB) proposed in Corominas *et al.* (2007) is used. Table 6 shows the average of the optimal RTV values ($\overline{RTV^*}$) for *CAT1* instances and the averages of the LBs ($\overline{LB}$).

Table 6. Averages of the optimal RTV values and the RTV lower bounds

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
$\overline{LB}$	**21.40**	5.35	10.95	21.15	48.15
$\overline{RTV^*}$	*	10.24	*	*	*
TS+VNS	**55.05**	10.24	22.48	47.66	139.84
MS+VNS	**54.95**	10.24	20.94	43.26	145.35
PSO+VNS	**55.86**	10.45	22.00	46.80	144.22

For all 185 *CAT1* instances, *MS+VNS* and *TS+VNS* achieve the optimal solutions. We can see in Table 6 that the LB calculated as proposed in Corominas *et al.* (2007) is not accurate. For the smallest instances, the ratio between $\overline{RTV^*}$ and $\overline{LB}$ is 1.914. It seems reasonable to assume that this ratio will remain equal or increase for larger instances. Thus, if we assume that the ratio remains equal, a more accurate estimation of the averages of the optimal values for *CAT2*, *CAT3* and *CAT4* instances are obtained by multiplying their $\overline{LB}$ by 1.914; that is, 20.96, 40.48 and 92.16 for *CAT2*, *CAT3* and *CAT4* instances, respectively. According to this assumption, we can ensure that the solutions obtained by the hybrid algorithms for *CAT2* and *CAT3* instances are very good.

5. Conclusions

In this paper, the response time variability problem (RTVP) is solved. This scheduling problem arises in a variety of real-life environments including mixed-model assembly lines, multi-threaded systems, network servers, broadcast of commercial videotapes, periodic machine maintenance and waste collection, among others. The aim of the RTVP is to minimise the variability in the distances between any two consecutive copies of the same symbol.

The RTVP is an NP-hard problem and heuristic and metaheuristic methods are needed to solve real-life instances. Since the first method to solve the RTVP was proposed in

1994, the solution of this problem has been improving. Up to date, the best solutions have been obtained with a reduced variable neighbourhood search (RVNS) algorithm. One of the main shortcoming of RVNS is that may be trapped in a local optimum with respect all neighbourhoods. To overcome this situation, we propose three different hybrid algorithms in which RVNS is hybridized with tabu search (TS), multi-start (MS) and particle swarm optimisation (PSO). Thus, the diversification ability of TS, MS and PSO is combined with the intensification ability of RVNS.

A computational experiment shows the success of our proposals. On average, all three hybrid algorithms are able to improve the best solutions published in the literature. Moreover, we have shown that two of the proposed algorithms obtain the optimal solutions for all 185 smallest test instances (*CAT1* instances) and we can reasonably assume that optimal or near optimal solutions are obtained for small and medium instances (*CAT2* and *CAT3* instances).

Very efficient non-exact methods have been designed in this work to solve the RTVP. A future research can be focused on improving the exact solution of the RTVP by increasing the size of the instances that can be solved in a practical time. In order to achieve this goal, the following lines of research are promising: 1) To improve the best MILP model which is proposed in Corominas *et al*. (2010), and 2) To specifically design a branch and bound algorithm in order to take advantage of all characteristics of the problem.

REFERENCES

Adenso-Díaz, B. and Laguna, M. (2006) ‘Fine-tuning of algorithms using fractional experimental designs and local search’, *Operations Research*, Vol. 54, pp. 99-114.

Anghinolfi, D. and Paolucci, M. (2009) ‘A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times’, *European Journal of Operational Research*, Vol. 193, pp. 73-85.

Anily, S., Glass, C.A. and Hassin, R. (1998) ‘The scheduling of maintenance service’, *Discrete Applied Mathematics*, Vol. 82, pp. 27-42.

Balinski, M.L. and Young, H.P. (1982) *Fair Representation*, Yale University Press, New Haven.

Bar-Noy, A., Nisgav, A. and Patt-Shamir, B. (2002) ‘Nearly optimal perfectly-periodic schedules’, *Distributed Computing*, Vol. 15, pp. 207–220.

Boender, C.G.E., Rinnooy, A.H.G., Stougie, L. and Timmer, G.T. (1982) ‘A Stochastic Method for Global Optimization’, *Mathematical Programming*, Vol. 22, pp. 125-140.

Bollapragada, S., Bussieck, M.R. and Mallik, S. (2004) ‘Scheduling Commercial Videotapes in Broadcast Television’, *Operations Research*, Vol. 52, pp. 679-689.

Brusco, M.J. (2008) ‘Scheduling advertising slots for television’, *Journal of the Operational Research Society*, Vol. 59, pp. 1363-1372.

Corominas, A., Kubiak, W. and Moreno, N. (2007) ‘Response time variability’, *Journal of Scheduling*, Vol. 10, pp. 97-110.

Corominas, A., García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics’, *Special*

Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development, Vol. 184, pp. 128-137.

Corominas, A., García-Villoria, A. and Pastor, R. (2009a) ‘Using Tabu Search for the Response Time Variability Problem’, *3rd International Conference on Industrial Engineering and Industrial Management* (CIO 2009), Barcelona and Terrassa, Spain.

Corominas, A., García-Villoria, A., Pastor, R. (2009b) ‘Resolución del response time variability problem mediante tabu search’, *VIII Evento Internacional de Matemática y Computación* (COMAT’2009), Universidad de Matanzas, Cuba.

Corominas, A., García-Villoria, A. and Pastor, R. (2009c) ‘Solving the Response Time Variable Problem by means of a Variable Neighbourhood Search Algorithm’, *13th IFAC Symposium of Information Control Problems in Manufacturing* (INCOM 2009), Moscow, Russia.

Corominas, A., Kubiak, W. and Pastor, R. (2010) ‘Mathematical programming modeling of the Response Time Variability Problem’, *European Journal of Operational Research*, Vol. 200, pp. 347-357.

Dong, L., Melhem, R. and Mosse, D. (1998) ‘Time slot allocation for real-time messages with negotiable distance constrains requirements’, *Fourth IEEE Real-Time Technology and Applications Symposium* (RTAS'98), Denver, CO. pp. 131-136.

Ekşioğlu, B., Ekşioğlu, S.D. and Pramod, J. (2008) ‘A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods’, *Computers & Industrial Engineering*, Vol. 54, pp. 1-11.

Eiben, A.E., Hinterding, R. and Michalewicz, Z. (1999) ‘Parameter control in evolutionary algorithms’, *IEEE Transactions on evolutionary computation*, Vol. 3, pp. 124-141.

García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics’, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 146, pp. 187-194.

García-Villoria, A., Pastor, R. and Corominas, A. (2007) ‘Solving the Response Time Variability Problem by means of the Cross-Entropy Method’, *International Journal of Manufacturing Technology and Management* (to be published).

García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of a psychoclonal approach’, *Journal of Heuristics*, in press, corrected proof, available online, 16 July 2008, doi:10.1007/s10732-008-9082-2.

García-Villoria, A. and Pastor, R. (2009a) ‘Introducing dynamic diversity into a discrete particle swarm optimization’, *Computers & Operations Research*, Vol. 36, pp. 951-966.

García-Villoria, A. and Pastor, R. (2009b) ‘Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism’, *International Journal of Production Research*, doi: 10.1080/00207540902862545.

García-Villoria, A. and Pastor, R. (2010) ‘Solving the response time variability problem by means of a genetic algorithm’, *European Journal of Operational Research*, Vol. 202, pp. 320-327.

Gendreau, M. (2003) ‘An Introduction to Tabu Search’, Chapter 2 in *Handbook of Metaheuristics*, Kluwer Academic Publishers.

Glover, F. (1986) ‘Future paths for Integer Programming and Links to Artificial Intelligence’, *Computers & Operations Research*, Vol. 5, pp. 533-549.

Han, C.C., Lin, K.J. and Hou, C.J. (1996) ‘Distance-constrained scheduling and its applications in real-time systems’, *IEEE Transactions on Computers*, Vol. 45, pp. 814-826.

Hansen, P. and Mladenovic, N. (1999) ‘An introduction to variable neighborhood search’, In *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pp. 433-458, Kluwer Academic Publishers.

Hansen, P. and Mladenovic, N. (2003) ‘Variable Neighborhood Search’, Chapter 6 in *Handbook of metaheuristics*, Kluwer Academic Publishers

Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report*, University of Maryland, USA. Available at http://hdl.handle.net/1903/7082.

Herrmann, J.W. (2009) ‘Using aggregation to reduce response time variability in cyclic fair sequences’, *Journal of Scheduling*, doi 10.1007/s10951-009-0127-7.

Hoos, H. and Stützle, T. (2005) Stochastic local research: foundations and applications, Morgan Kaufmann Publishers, San Francisco.

Kennedy, J. and Eberhart, R.C. (1995) ‘Particle swarm optimization’, In *IEEE International Conference on Neural Networks*, Australia, pp. 1942-1948.

Kubiak, W. (1993) ‘Minimizing variation of production rates in just-in-time systems: A survey’, *European Journal of Operational Research*, Vol. 66, pp. 259-271.

Kubiak, W. (2004) ‘Fair Sequences’, Chapter 19 in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapman and Hall.

Kubiak, W. (2009) ‘Proportional optimization and fairness’, *International Series in Operations Research & Management Science*, Springer.

Martí, R. (2003) ‘Multi-start methods’, Chapter 12 in *Handbook of Metaheuristics*, Kluwer Academic Publishers.

Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 35, pp. 192-207.

Mladenovic, N. and Hansen, P. (1997) ‘Variable neighbourhood search’, *Computers & Operations Research*, Vol. 24, pp. 1097-1100.

Monden, Y. (1983) ‘Toyota Production Systems’, *Industrial Engineering and Management Press*, Norcross, GA.

Tasgetiren, M.F., Liang, Y.C., Sevkli, M. and Gencyuilmaz, G. (2007) ‘A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem’, *European Journal of Operational Research*, Vol. 177, pp. 1930-1947.

Tchomté, S.K. and Gourgand, M. (2009) ‘Particle swarm optimization: A study of particle displacement for solving continuous and combinatorial optimization problems’, *International Journal of Production Economics*, Vol. 121, pp. 57-67.

Waldspurger, C.A. and Weihl, W.E. (1994) ‘Lottery Scheduling: Flexible Proportional-Share Resource Management’, *First USENIX Symposium on Operating System Design and Implementation*, Monterey, California.

Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Scheduling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528*, Massachusetts Institute of Technology, MIT Laboratory for Computer Science. Available at https://eprints.kfupm.edu.sa/67117

Wei, W.D. and Liu, C.L. (1983) ‘On a periodic maintenance problem’, *Operations Research Letters*, Vol. 2, pp. 90-93.

Xu, J., Sohoni, M, McCleery, M. and Bailey, T.G. (2006) ‘A dynamic neighbourhood based tabu search algorithm for real-world flight instructor scheduling’, *European Journal of Operational Research*, Vol. 169, pp. 978-993.

A2.2. Articles published in other international journals

Solving the Response Time Variability Problem by means of the Cross-Entropy Method

Article published as [International Journal of Manufacturing Technology and Management, Volume 20, 2010, Pages 316-330] © [copyright Inderscience Publishers]

Solving the Response Time Variability Problem by means of the Cross-Entropy Method[†]

Alberto García-Villoria[*]
Institute of Industrial and Control Engineering (IOC),
Technical University of Catalonia (UPC), Barcelona, Spain
E-mail: alberto.garcia-villoria@upc.edu

Albert Corominas
Institute of Industrial and Control Engineering (IOC),
Technical University of Catalonia (UPC), Barcelona, Spain
E-mail: albert.corominas@upc.edu

Rafael Pastor
Institute of Industrial and Control Engineering (IOC),
Technical University of Catalonia (UPC), Barcelona, Spain
E-mail: rafael.pastor@upc.edu

Abstract: The Response Time Variability Problem (RTVP) is an NP-hard combinatorial scheduling problem that has recently appeared in the literature. The RTVP has a wide range of production line systems applications such as sequencing the models to be produced on a mixed-model assembly line in a just-in-time context. This problem occurs whenever several units of different models need to be sequenced so as to minimize the variability of the distance between any two consecutive units of the same model. A mathematical mixed integer linear programming (MILP) model has been presented by another study, but the practical limit for obtaining optimal solutions is around 40 units to be scheduled. Another study has developed five heuristic algorithms to solve non-small RTVP instances. We propose to solve the RTVP by means of the metaheuristic cross-entropy (CE) method, which has been developed recently. We report on the computational experiments in which the CE method is compared with the five heuristic algorithms proposed in the literature.

Keywords: response time variability, fair sequences, mixed-model assembly production line, just-in-time, scheduling, cross-entropy, metaheuristics

Biographical notes: A. García-Villoria is an Informatics Engineer at the Technical University of Catalonia (UPC). Since 2007 he is an assistant professor of the Department of Management (OE). He is currently pursuing a PhD degree at the Advanced Automation and Robotics (AAR) doctoral programme organised by the Institute of Industrial and Control Engineering (IOC) of the UPC and his research is focused on the solution of a sequencing problem.

[†] Sponsored by the Spanish Ministry of Education and Science's projects DPI2004-03472 and DPI2007-61905; co-funded by the FEDER.
[*] Corresponding autor: Alberto García-Villoria, IOC – Institute of Industrial and Control Engineering, Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; e-mail: alberto.garcia-villoria@upc.edu

A. Corominas is a professor of industrial engineering and operational research at the Department of Business Management and the Institute of Industrial and Control Engineering (IOC) at the Technical University of Catalonia (UPC). He holds degrees in engineering from the University of País Vasco (UPV) and in computer science from the Technical University of Madrid; his PhD is also from the UPV. His academic and professional experience has focused on industrial engineering and especially on the development and application of quantitative techniques to the design of production and logistic systems and to operations management.

R. Pastor is a university lecturer of industrial engineering and operations research at the School of Industrial Engineering of Barcelona of the Technical University of Catalonia (UPC). Previously, he worked as a product production manager for Revlon. Dr. Pastor holds a PhD in industrial engineering and a Master's Degree in Logistics Organization. His research focuses on the use of a variety of combinatorial optimization techniques to model and solve real-world applications in production, scheduling and manpower planning.

1 INTRODUCTION

Sequencing several units as *regularly* as possible in mixed-model assembly production lines is a major problem ever since Toyota Motor Corporation developed and implemented the just-in-time (JIT) production system (Monden, 1983). One of the most important JIT objectives is to get rid of all kinds of waste and inefficiency and, according to Toyota, the main waste is due to the stocks.

The key to reducing stocks with JIT, as Monden (1983) says, is to have constant production rates and constant consumption rates of the components involved in the production process. First, the number of units of each model to be produced by the mixed-model assembly production line throughout the production period must be decided. Next, these units must be sequenced as *regularly* as possible. Regularity can be sought in the consumption of the components that arrive to the production line or in the production of the models that leave the production line. Depending on the kind of regularity desired, Kubiak (1993) classifies these sequencing problems into two categories: ORV (Output Rate Variation) problems and PRV (Production Rate Variation) problems.

The ORV problem concentrates on the consumption of the components needed by the models and its aim is to minimize the variations in this consumption in the production period.

On the other hand, the PRV problem concentrates on production rates of the models and its objective is to minimize a function of the discrepancies between the real production rate and the ideal one (i.e., the one that would correspond to a constant rate of production). This kind of regularity is important when production needs to be adjusted to demand. Thus, according to the JIT system, it is possible to satisfy demands for a variety of models without holding large inventories or incurring large waits.

Regularity in the PRV problem can be characterized at least in as many ways as discrepancy functions are defined. The Response Time Variability Problem (RTVP) is a PRV problem in which the regularity consists in preserving the distance between two consecutive units of the same model as constant as possible. The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources. As it is usual in the literature on regular sequences (Monden, 1983; Miltenburg, 1989), we assume in this paper that the processing time of the units does not depend on the model and, therefore, is the same for all the units.

The RTVP was first presented by Corominas et al. (2007), who proposed a mixed integer lineal programming (MILP) model and five greedy heuristic algorithms to solve this problem. An improved MILP presented by Corominas et al. (2006) has a practical limit for obtaining optimal solutions of around 40 units to be scheduled.

In order to improve the results obtained in prior studies, an application based on the Cross Entropy (CE) method is proposed. The CE method was originally created for rare-event simulation by Rubinstein (1997). The new CE method provides an adaptive algorithm for estimating probabilities of rare events in, for example, queuing models or complex stochastic networks (Rubinstein, 1997). Later it was observed that the CE method can be easily adapted for other applications such as general combinatorial and multi-extremal optimization, learning algorithms and neural computation (Rubinstein and Kroese, 2004).

The CE method involves an iterative procedure in which each iteration has two steps. Step 1 generates a random sample of solutions according to a probability distribution. Step 2 modifies the probability distribution according to the sample obtained in the previous step; this change in the probability distribution will increase the probability of generating a better sample in the next iteration. For a discrete optimization problem in which a deterministic objective function is optimized (this is the case of the

RTVP), it is demonstrated that the CE method always converges and the probability of an optimal solution being found can be made arbitrarily close to 1 (Costa et al., 2006).

This paper shows how CE method can be easily adapted to solve the combinatorial scheduling RTVP. Moreover, the results obtained by the CE method and the five greedy heuristics proposed by Corominas et al (2007) are compared and it is shown that the CE method outperforms them clearly.

The rest of this paper is organized as follows. Section 2 exposes a formal definition of the RTVP and briefly explains the five heuristic algorithms presented by Corominas et al. (2007). Section 3 extends the explanation of the CE method and proposes its application to the RTVP. Section 4 gives the results of the computational experiment. Finally, some conclusions are presented in Section 5.

2 THE RESPONSE TIME VARIABILITY PROBLEM (RTVP)

In the Response Time Variability Problem (RTVP) the regularity of production in mixed-model assembly production lines is measured in a new way. The idea is to arrange the units of each model as regularly as possible.

Real-life examples do not occur only in the mixed-model assembly production line context. For example, asynchronous transfer mode networks need to broadcast the video data frames and sound data frames of the applications as constantly as possible (Dong et al., 1998); another example is the periodic machine maintenance problem, if it is considered with equal distances between consecutive services of the same machine (Anily et al., 1998).

These problems are often faced as a distance-constrained scheduling problem, in which the distance between any two given consecutive units of the same model is bounded. The difficulty lies in obtaining a feasible solution which respects the distance constraints. On the other hand, the RTVP always obtains a solution, preserving the main idea of having the distances between any two given consecutive units of the same model as constant as possible.

The RTVP is formulated as follows. Let *n* be the number of models, d_i the number of units of model *i* (*i* = 1,…,*n*) and *D* the total number of units ($D = \sum_{i=1}^{n} d_i$). Let *s* be a solution of an instance in the RTVP that consists of a circular sequence of units ($s = s_1 s_2 \ldots s_D$), where s_j is the unit sequenced in position *j* of sequence *s*. For all models *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions in which units *k* + 1 and *k* of model *i* are found (i.e., the number of positions between them, where the distance between two consecutive positions is considered equal to 1). As the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_i}^i$ is the distance between the first unit of model *i* in a cycle and the last unit of the same model in the preceding cycle. Let $\bar{t}_i$ be the average distance between two consecutive units of model *i* ($\bar{t}_i = D / d_i$). For all models *i* in which $d_i = 1$, t_1^i is equal to $\bar{t}_i$. The objective is to minimize the $RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2$.

For example, let $n = 3$, $d_A = 3$, $d_B = 2$ and $d_C = 2$; thus, $D = 7$, $\bar{t}_A = 7/3$, $\bar{t}_B = 7/2$ and $\bar{t}_C = 7/2$. A feasible solution is the sequence (A, B, A, C, B, A, C), where $RTV = \left(\left(2 - 7/3\right)^2 + \left(3 - 7/3\right)^2 + \left(2 - 7/3\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) = 2/3 + 1/2 + 1/2 = 5/3$.

To solve non-small RTVP instances, five greedy heuristic algorithms were proposed by Corominas et al. (2007): the bottleneck, random, Webster’s, Jefferson’s and insertion sequences. These five heuristic algorithms, which are until now the only algorithms proposed in the literature, are compared with the proposed CE method in Section 4. Next the heuristics are briefly described (for more details, see Corominas et al., 2007).

The bottleneck sequence is obtained by solving the bottleneck problem optimally. Among the proposed algorithms in the literature, the algorithm of Moreno (2002) (see also Moreno and Corominas, 2006) has been chosen.

The random sequence is obtained by randomizing the bottleneck sequence as follows. For each position $p = 1,...,D$, a random number r between 1 and D is obtained; then, the units of the positions p and r are swapped.

Webster's sequence is obtained by applying the parametric method of apportionment (Balinski and Young, 1982) with parameter $\delta = 1/2$. Let x_{il} be the number of model i units in the sequence of length l, l = 0, 1, …; assume $x_{i0} = 0$, $i = 1,\dots,n$. The model to be sequenced in position $l + 1$ can be computed as follows: $i^* = \arg\max_i \left\{ d_i \big/ (x_{il} + \delta) \right\}$.

Jefferson's sequence is obtained by applying the parametric method of apportionment with $\delta = 1$.

The insertion sequence is based on the idea of solving two-model problems. The two-model problems are solved in a polynomial time using an exact algorithm presented by Corominas et al. (2007). Let $d_1 \leq ... \leq d_n$ and let n - 1 two-model problems $P_{n-1} = (d_{n-1}, d_n)$, $P_{n-2} = (d_{n-2}, \sum_{i=n-1}^{n} d_i)$, …, $P_1 = (d_1, \sum_{i=2}^{n} d_i)$. In each of the problems P_{n-2}, P_{n-3}, …, P_1, the second model will be the same fictitious model for all problems, denoted by *. Let sequences S_{n-1}, S_{n-2},…, S_1 be the optimal solution to problems P_{n-1}, P_{n-2},…, P_1 respectively, which are obtained by the algorithm described by Corominas et al. (2007). Note that the sequence S_i, $i = n-2,\dots 1$, is made up of the model i and the fictitious model *. Then the sequence for the original problem is built recursively by first replacing * in S_1 by S_2 to obtain S_1'. Next, * are replaced by S_3 in S_1' to obtain a sequence S_1'', and so on.

3 THE CROSS-ENTROPY METHOD

This section presents the CE method, the application based on this method to solve the RTVP and how the parameters of the CE method have been fine-tuned.

3.1 The general CE method

The CE method was pioneered in 1997 when an adaptive algorithm for estimating probabilities of rare events in complex stochastic networks was presented by Rubinstein (1997). There are a lot of real-life applications in which rare events with very small probabilities need to be estimated: ruins in insurance risk or finance, breakdowns of manufacturing systems, packet losses and buffer overflows in computer and communication networks, false alarms in radar or similar security systems, technical defects, and many others.

Simulating the system for a long time is not a practical way to estimate these small probabilities. A better way is to use the importance sampling (IS) technique, in which the system is simulated under a different probability distribution so that the rare event occurs more frequently. However, an important disadvantage of the IS technique lies in determining the optimal parameters for the probability distribution of the events. The CE method eliminates this disadvantage because it provides an adaptive procedure to estimate the optimal parameters.

It was realized that the CE method could be adapted to solving difficult combinatorial and multi-extremal optimization problems together with learning algorithms and neural computation applications (Rubinstein and Kroese, 2004). The CE method provides a mathematical way which connects the estimation of rare events in simulation to the combinatorial optimization problems (COP).

The CE method is based on an iterative procedure divided into two steps: 1) generate a random sample of solutions according to a probability distribution; and 2) modify the probability distribution depending on the sample obtained in the previous step. The change in the probability distribution will increase the probability of generating a better sample in the next iteration.

To solve a COP with the CE method, the COP is associated with a stochastic optimization problem represented with a weighted directed graph. Depending on the problem, the randomness will be: a) in the nodes, in which case we call it a stochastic node network (SNN) problem; or b) at the edges, in which case we call it a stochastic edge network (SEN) problem. Examples of SNN problems are the max-cut problem and clustering problems; examples of SEN problems are the travelling salesman problem and the quadratic assignment problem (Rubinstein and Kroese, 2004).

As the RTVP is an SEN problem, we will concentrate on the CE method algorithm for solving problems of this type. We merely provide a practical explanation of this algorithm; for a formal definition and its mathematical justification, see the book by Rubinstein and Kroese (2004) or the tutorial by Boer et al. (2005). A SEN problem consists in finding the optimal set of edges in the graph which represents the optimal solution. To find an optimal solution, the algorithm works as follows:

1. Initialize the probabilities of the edges; $it = 1$.

2. Generate N candidate feasible solutions $X_k^{(it)}$, $k = 1,\ldots,N$ according to the probabilities of the edges at the iteration *it*, $pr_e^{(it)}$. Let $B^{(it)}$ be the set of the $\lceil \rho * N \rceil$ best solutions according to the fitness function at the iteration *it*, where ρ is between (0,1].

3. For each edge *e*, update its probability for the next iteration as

$$pr_e^{(it+1)} = \frac{\left|\left\{X \in B^{(it)} : e \in X\right\}\right|}{\left|B^{(it)}\right|}.$$

4. If stopping criterion is reached then stop, otherwise set $it = it + 1$ and go to Step 2.

To avoid a premature convergence, the *smooth* version of the CE method (Rubinstein and Kroese, 2004) is used in the RTVP. Now, a new smoothing parameter α, whose value can be between 0 and 1, is used for updating the probabilities as follows:

$$pr_e^{(it+1)} = (1-\alpha)pr_e^{(it)} + \alpha\left(\frac{\left|\left\{X \in B^{(it)} : e \in X\right\}\right|}{\left|B^{(it)}\right|}\right)$$

The main reason why the *smooth* version updating procedure performs better is that it prevents probabilities of 0s and 1s; once edge has a probability of 0 or 1, it will appear in solutions never or forever, which is undesirable.

Although the CE method bears similarities with other algorithms that work with probabilities, as the Ant Colony Optimization (ACO) method (Dorigo et al., 1999) and the Learning Automata Search Technique (LAST) (Gosavi, 2003), specially when a SEN problem is solved, there are differences between them. The main differences between the CE and the ACO method are: 1) CE uses only the best solutions of the sample whereas ACO uses all the solutions (Kauppila, 2006), and 2) in CE, the generation of future solutions is based on a generic calculation whereas in ACO the generation of solutions is also based on problem dependent heuristics (Boer et al., 2005). The main differences between CE and LAST lies in: 1) at each iteration, CE generates a set of solutions to calculate the probabilities of the next generation, whereas LAST only generate one solution, and 2) CE calculates the probability for each edge *e* according to the number of solutions in which edge *e* appears and to the fitness of the solutions, whereas LAST calculates the probability of edge *e* according to the fitness of the historical best solution in which edge *e* appears (for a detailed explanation, see Gosavi, 2003).

3.2 The application of the CE method to the RTVP

The general CE method algorithm for a SEN problem has been described in the previous subsection. For the application of the CE method to the RTVP, four points of the algorithm need to be specified: 1) the graph that represents the associated stochastic optimization problem, 2) the generation of a candidate feasible solution sequence according to the probabilities, 3) the initial probabilities of the edges and 4) the stopping criterion.

In order to make more understandable the explanation, the example introduced in Section 2 is used: n = 3; $d_A = 3$, $d_B = 2$ and $d_C = 2$; and $D = 7$

3.2.1. Definition of the graph

Let the graph $G = (N', E)$. The set of nodes N' is the union of the sets N_1 and N_2, where $N_1 = \{n_k^{(i)} : 1 \le i \le n-1, 1 \le k \le d_i - 1\}$ and $N_2 = \{t : 1 \le t \le D-1\}$. Notice that the model n is not included in N_1 because the units of this model are fixed when the previous models are sequenced. The node $n_k^{(i)}$ belonging to N_1 represents the unit k of the model i; the node t belonging to N_2 represents a distance t between two units. Thus, in the example we have $N_1 = \left\{n_1^{(A)}, n_2^{(A)}, n_1^{(B)}\right\}$ and $N_2 = \{1,2,3,4,5,6\}$. Let $E \subset N_1 \times N_2$, where the edge $e_{ikt} = (n_k^{(i)}, t)$ represents that the unit $k + 1$ of the model i is sequenced at distance t of the unit k of the model i.

3.2.2. Generation of a feasible solution sequence

We start by setting unit 1 of model 1 to the first position of the sequence (example of the start sequence shown in Figure 1a). Then, an edge has to be randomly chosen from the node $n_1^{(1)}$ of the set $\{e_{1,1,t} : 1 \le t \le D - d_1 + 1\}$. For each unit, the sum of the probabilities of an eligible set of edges from this unit is almost always different than one. The probabilities of the set are therefore normalized for each selection. The initial probabilities of the example are generated as it is explained in Section 3.2.3 and they are shown in Table 1 (before being normalized) and Table 2 (after being normalized). The choice of the edge will fix the unit 2 position (let it be called $p_2^{(1)}$) of model 1 to the value $1+t$. Note that the biggest possible position $p_2^{(1)}$ is $D - d_1 + 2$, which allows the rest of the units of model 1 to be sequenced at the positions $p_2^{(1)}$ + 1, $p_2^{(1)}$ + 2, …, *D*. In the example, it is supposed that the edge randomly chosen is $e_{A,1,2}$ (i.e., t = 2) and, therefore, $p_2^{(A)} = 1+2 = 3$ (see Figure 1b).

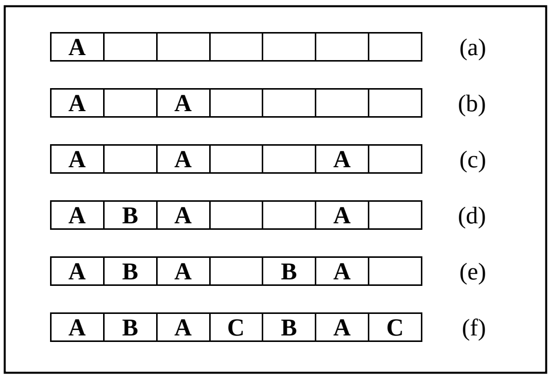


Figure 1. Generation of a feasible solution sequence

	t=1	t=2	t=3	t=4	t=5
$n_1^{(A)} = n_2^{(A)}$	$3/4$	3	$3/2$	$3/5$	$3/8$
$n_1^{(B)}$	$2/5$	$2/3$	2	2	$2/3$

Table 1. Initial probabilities (before being normalized)

	t=1	t=2	t=3	t=4	t=5
$n_1^{(A)} = n_2^{(A)}$	$30/249$	$120/249$	$60/249$	$24/249$	$15/249$
$n_1^{(B)}$	$3/43$	$5/43$	$15/43$	$15/43$	$5/43$

Table 2. Initial probabilities (after being normalized)

Now we choose an edge randomly from node $n_2^{(1)}$ of the set $\{e_{1,2,t} : 1 \le t \le D - (d_1 - 2) - p_2^{(1)} + 1\}$. This process continues for units 3, 4, …,d_1 – 1 of model 1. The set of eligible edges from unit *k* of model 1 is $\{e_{1,k,t} : 1 \le t \le D - (d_1 - k) - p_k^{(1)} + 1\}$, where $p_k^{(1)}$ is the position at the sequence of unit k of model 1. In the example the set would be $\{e_{A,2,t} : 1 \le t \le 7 - (3-2) - 3 + 1\}$, i.e., $\{1,2,3,4\}$, and the probabilities of the edges from $n_2^{(A)}$ are normalized (Table 3a); then, an edge is randomly chosen: for example, $e_{A,2,3}$ (i.e., t = 3). Therefore, $p_3^{(A)} = 3+3 = 6$ (see Figure 1c). Note that the distance between the first unit of model 1 and the last unit of the same model in the preceding cycle is automatically determined when $p_{d_1}^{(1)}$ is fixed.

When all units of model 1 are set, it is the turn of the units of model 2, then model 3, and so on until the penultimate model. The first unit of each model is always set at the first free position of the sequence. The distances between the remaining units must ensure that the units are not set at a taken position. Thus, in the example the first unit of the model *B* is placed in the first free position of the sequence and, therefore, $p_1^{(B)} = 2$ (see Figure 1d). Next, an edge from $n_1^{(B)}$ has to be chosen from the set $\{2,3,5\}$. Notice that the distances 1 and 4 are not included in the set because if the edge $e_{B,1,1}$ is chosen then $p_2^{(B)} = 2+1 = 3$ and the position 3 is already occupied by the model *A* (and analogous for the edge $e_{B,1,4}$). The probabilities of the edges from $n_1^{(B)}$ are normalized (Table 3b) and then the edge is randomly chosen: for example, the edge $e_{B,1,3}$. Therefore, $p_2^{(B)} = 2+3=5$ (see Figure 1e). Finally, the sequence is completed with the units of the last model *C* (see Figure 1f).

	t=1	**t=2**	**t=3**	**t=4**
$n_2^{(A)}$	$30/234$	$120/234$	$60/234$	$24/234$

(a)

	t=1	**t=2**	**t=3**	**t=4**	**t=5**
$n_1^{(B)}$	⊠	$5/25$	$15/25$	⊠	$5/25$

(b)

Table 3. Normalized probabilities

The distance between the last unit and the first unit of a majority model could be forced to be big by the rule of setting the first unit of each model at the first free position of the sequence. To alleviate this, the models are arranged decreasingly by their number of units before the CE method application starts.

3.2.3. Initial probabilities of the edges

The initial probabilities are set so that the nearer a distance is to the ideal distance, the more probable the distance is. For each model i = 1,...,n, let $tmax_i = D - d_i + 1 - (i - 1)$, where $tmax_{\mathrm{i}}$ is the maximal theoretical distance between two consecutive units of model i. Notice that $tmax_i$ depends on i because the first unit of each sequenced model is set at the first free position of the sequence. For each node k of model i, $n_k^{(i)}$, the probabilities of the set of edges which go out from the node are set as follows: $pr_{e_{ikt}}^{(1)} = 1 \big/ \left(\left|\bar{t}_i - t\right| + \varepsilon\right)$, t = 1,…, $tmax_i$, where $\varepsilon = 10^{-6}$ is used to avoid a division by zero if t is equal to $\bar{t}_i$.

Because $\sum_{t=1}^{tmax_i} pr_{e_{ikt}}^{(1)}$ is greater than 1, the probabilities are next normalized (as it has been shown with the previous example in Tables 1 and 2).

3.2.4. Stopping criterion

The CE algorithm stops running after 50 seconds.

3.3 Fine-tuning the CE parameters

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although the value of the parameters is vital because the results of the metaheuristic for each problem are very sensitive to them, the fine-tuning is usually done by intuitively testing several values.

The problem of fine-tuning the parameters of a metaheuristic application can be approached as an optimization problem, for which the solution consists of finding the parameter values that optimize the running of the metaheuristic for the problem to solve. Since the set of instances of a problem is infinite, we must resign ourselves to a representative training set for making the optimization.

The Nelder and Mead (N&M) algorithm (Nelder and Mead, 1965), also named the flexible polyhedron algorithm, has been chosen for solving the fine-tuning problem because it is a direct one (i.e., it uses only the values of the function). There are another algorithms that could be used to solve this fine-tuning optimization problem, but the N&M algorithm has offered good results since its publication and it is referenced by recent papers (Anjos et al., 2004; Chelouah and Siarry, 2005; Corominas, 2005). The N&M algorithm starts from an v-dimensional point, whose coordinates are the v parameters of an objective function, and an initial hyper-tetrahedron is formed. For the fine-tuning problem, the parameters of the metaheuristic are used as the coordinates of the points. It is advisable for one of the initial vertices of the hyper-tetrahedron to be a known good point since the N&M algorithm ensures that the solution found is never worse than the best of the initial vertices. Then, the points of the hyper-tetrahedron are iteratively moved in the v-dimensional space according to the values of the function of each point until a local optimal point is reached. The function to be used by the N&M algorithm for the fine-tuning problem of a metaheuristic is the sum of the objective function values corresponding to the solutions obtained with the metaheuristic application at each instance of the training set. Usually, the bigger an instance is, the bigger the objective function value of its optimal solution is. Therefore, the N&M algorithm will give more relevance to the big instances for fine-tuning the parameters. To avoid this situation, the objective function values are normalized by dividing them by a lower bound of the instance. The lower bound used for the RTVP is the lower bound (let be called *LBT*) introduced by Corominas et al. (2007), which is calculated as follows:

$$LBT = \sum_{i=1}^{n} \left[(D \bmod d_i) \cdot \left(\left\lceil \frac{D}{d_i} \right\rceil - \bar{t}_i \right)^2 + (d_i - D \bmod d_i) \cdot \left(\left\lfloor \frac{D}{d_i} \right\rfloor - \bar{t}_i \right)^2 \right]$$

The CE method needs fine-tuning of three parameters: the number of candidate solutions to be produced at each iteration (N), the proportion of candidate solutions that is part of the best solutions (ρ) and the smooth parameter (α). To set the initial point (initial values of the parameters) of the N&M algorithm, the values recommended by Rubinstein and Kroese (2004) are considered: N bigger than D^2 (D is the number of units), ρ equal to the minimum of 0.01 and $\ln(D)/D$, and α between 0.7 and 1. Note that N could be very big, so we limited this initial value to 500; ρ was initially set to $\ln(500)/500 = 0.0124$ and α to 0.8. To know a good initial point to start the N&M algorithm is another reason to use it because, in this case, the N&M algorithm ensures a good solution, i.e., a good fine-tuning of the CE parameters.

A set of 45 training instances (generated as explained in Section 4) was used to fine-tune the CE method and, to value a point, the CE method was run for 50 seconds each instance. The N&M algorithm was stopped after 42 hours. The fine-tuning values of the parameters are finally N=306, ρ = 0.0206 and α = 0.8868.

4 COMPUTATIONAL EXPERIMENTS

The computational experiments were performed by running 555 instances grouped into three classes (185 instances in each class) according to their size. The instances in the first class (called *CAT1*) were generated using a random value of D (number of units) uniformly distributed between 25 and 50, and a random value of n (number of models) uniformly distributed between 3 and 15; for the second class (called *CAT2*), D was between 50 and 100, and n between 3 and 30; and for the third class (called *CAT3*), D was between 100 and 200 and n between 3 and 65. For all instances and for each model i, a random value of d_i (number of units of the model i) is between 1 and $\left| \frac{D-(n+1)}{2.5} \right|$ such that $\sum_{i=1}^{n} d_i = D$. The instances were randomly generated because there is not any set of benchmark instances published in the literature since the problem is very new.

The CE and the heuristic algorithms were coded in Java and the computational experiments were carried out using a 3.4 GHz Pentium IV with 512 Mb of RAM.

For each instance, the CE algorithm was run for 50 seconds. The time needed by the five heuristic algorithms (*H1* to *H5*) is negligible (always less than a second per instance). Table 4 shows the averages of the RTV values for each class of instances (CAT1 to CAT3).

	CE	H1	H2	H3	H4	H5
CAT1	21.16	107.09	932.13	121.84	147.19	172.69
CAT2	106.15	693.38	4741.55	933.11	1077.88	1254.29
CAT3	2809.81	4369.44	25157.87	8502.80	9106.04	10248.21

Table 4. Averages of the RTV values for each class of instance
H1 = Bottleneck; H2 = Random; H3 = Webster; H4 = Jefferson; H5=Insertion

The CE method gives much better results than the best of the heuristic algorithms (*H1*: bottleneck): 80.24% better for class *CAT1*, 84.69% better for class *CAT2* and 35.69% better for class *CAT3*. However, the improvement for class *CAT3* is not as impressive as the improvement for the other two classes. Table 5 shows: the averages of the number of iterations that the CE algorithm does in 50 seconds, for each class of instance, and the iteration in which the best solution was found.

	Average number of iterations	Best solution iteration
CAT1	527.94	7.37
CAT2	103.57	15.94
CAT3	16.78	14.64

Table 5. Averages of number of iterations and iteration in which the best solution is found

The best solution for CAT1 and CAT2 instances is found long before the last iteration of the CE algorithm. On the other hand, more than 70% of the CAT3 instances found their best solution at the last or penultimate iteration (47% at the last and 23% at the penultimate iteration). This indicates that in many CAT3 instances the CE algorithm stops before it finishes to converge, i.e., the probabilities of the edges are nearly to zero or to one. In spite of this, the CE algorithm improves by 35.69% for class CAT3 when compared to the bottleneck heuristic.

Tables 5 and 6 show other characteristic of the CE method. As opposed to other metaheuristic methods such as genetic algorithms, the CE method converges very fast (though it needs to generate a lot of solutions at each iteration) (Rubinstein and Kroese, 2004). To see how many iterations (on average) the *CAT3* instances need in order to converge, these instances were also run for 100, 200 and 300 seconds. The results are shown in Table 6.

	RTV value	**Number of iterations**	**Best solution iteration**
50 sec.	2809.81	16.78	14.64
100 sec.	1935.76	27.85	19.10
200 sec.	1320.24	54.57	25.73
300 sec.	1183.27	80.91	28.22

Table 6. RTV value and averages of number of iterations and the best solution iteration of the CE application for the *CAT3* instances.

When the *CAT3* instances are run for 100 seconds, there are still 51% of instances which found their best solution at the last or penultimate iteration (31% at the last and 20% at the penultimate). However, the results are 31,11% better than the 50-second results. For 200 seconds, there are 23% of instances which found their best solution at the last or penultimate iteration (13% at the last and 10% at the penultimate). These results are 53.01% better than the 50-second results. Finally, for 300 seconds, there are only 4% of instances which found their best solution at the last or penultimate iteration (3% at the last and 1% at the penultimate). These results are 57.89% better than the 50-second results and 72.92% better than the bottleneck heuristic.

As the execution time of the greedy heuristics is negligible, a better heuristic (let it be called *HB*) can be constructed by merely running the five heuristics and getting the best solution for each instance. Moreover, to solve the RTVP in a real context, an application (let it be called *CE-HB*) can be constructed by running the CE method (over 50 seconds, for example) and *HB* and getting the best solution for each instance. Table 7 shows the averages of the RTV values for the CE application (run for 50 seconds), for the bottleneck heuristic, for the *HB* and for the *CE-HB*.

	CE	**Bottleneck**	**HB**	**CE-HB**
CAT1	21.16	107.09	98.40	19.97
CAT2	106.15	693.38	631.41	86.45
CAT3	2809.81	4369.44	4200.67	1534.16

Table 7. Averages of the RTV obtained values

Although *HB*, obviously, improves on the bottleneck heuristic, the CE application continues to give much better results than *HB*: 78.45% better for class *CAT1*, 83.19% better for class *CAT2* and 33.11% better for class *CAT3*. Table 7 shows that it is a good idea to solve the RTVP with *CE-HB*, especially when the CE application has no time to converge: for the *CAT3* instances, *CE-HB* gets 45.40% better results than the CE method. This is because the CE application needs to converge until the end to obtain good results.

5 CONCLUSIONS AND FUTURE RESEARCH

The RTVP proposes a new metric for measuring the regularity of production in a mixed-model assembly production line. The RTVP tries to minimize the variability of distances between units of the same model. This metric has the advantage that it is very easy to understand for the practitioners, which know the importance of obtaining a good sequence in the mixed-model lines. Therefore, the efforts for minimizing the response time variability are important.

A CE method application has been presented to solve the RTVP, an NP-hard combinatorial optimization problem. The CE method has created recently for rare-event simulation but was soon adapted to solving combinatorial optimization problems. The power and generality of this method consist in the fact that the updating rules are often simple and fast. The first contribution of our paper shows how the CE method is easily adapted to solve the scheduling problem RTVP. Moreover, this adaptation is a good example of how the CE method can be used for other production scheduling problems.

The effectiveness of the CE method is demonstrated by the computational experiments. The CE application is contrasted with the greedy heuristics proposed by Corominas et al. (2007) and the solutions obtained with the CE application are considerably better. Thus, the second contribution of our paper consists in that the RTV values of the obtained sequences are improved. Although the computation times needed by the heuristics are much less than the time needed by the CE application, it is not a disadvantage in a production environment because the scheduled sequence is not updated frequently. Moreover, a metaheuristic, as it is the CE method, has the advantage that the more time is running, better solutions may found and, therefore, it is more probable to obtain the global optimal solution.

At present, we are developing two applications based on the metaheuristics Electromagnetism-like Mechanism and Psycho-Clonal Algorithm in order to find one metaheuristic that improves the CE method results.

As it is said in Section 1, this paper assumes that the processing time of the units does not depend on the model. In a future research, different processing time of the units according to the model will be considered.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to the anonymous reviews for their valuable comments which have improved the quality of this paper.

REFERENCES

Anily, S., Glass, C.A. and Hassin, R. (1998) 'The scheduling of maintenance service', *Discrete Applied Mathematics*, Vol. 82, pp. 27-42.

Anjos, M.F.., Cheng, R.C.H. and Currie, C.S.M.. (2004) 'Maximizing revenue in the airline industry under one-way pricing', *SIAM Journal on Optimization*, Vol. 9, pp. 535-541.

Balinski, M.L. and Young, H.P. (1982) 'Fair Representation: meeting the ideal of one man, one vote', *Yale University Press*, New Haven CT.

Boer, P. de, Kroese, D.P., Mannor, S. and Rubinstein, R.Y. (2005) 'A Tutorial of the Cross-Entropy Method', *Annals of Operations Research*, Vol. 134, pp. 19-67.

Chelouah, R. and Siarry, P. (2005) 'A hybrid method combining continous tabu search and Nelder-Mead simplex algorithms for the global minimization of multiminima functions', *European Journal of Operational Research*, Vol. 161, pp. 636-654.

Corominas, A. (2005) 'Empirically Adjusted Greedy Algorithms (EAGH): A new approach to solving combinatorial optimisation problems', *Working paper IOC-DT-P-2005-22*, Universitat Politècnica de Catalunya.

Corominas, A., Kubiak, W. and Pastor, R. (2006) 'Solving the Response Time Variability Problem (RTVP) by means of mathematical programming', *Working paper IOC-DT*, Universitat Politècnica de Catalunya.

Corominas, A., Kubiak, W. and Moreno, N. (2007) 'Response time variability', *Journal of Scheduling*, Vol. 10, pp. 97-110.

Costa, A., Jones, O.D. and Kroese, D. (2006) 'Convergence properties of the cross-entropy method for discrete optimization', *Operations Research Letters*, In Press, Corrected Proof.

Dong, L., Melhem, R. and Mossel, D. (1998) 'Time slot allocation for real-time messages with negotiable distance constraint requirements', *Real-time Technology and Application Symposium*, RTAS, Denver.

Dorigo, M., Di Caro, G. and Gambardella, L.M. (1999) 'Ant Algorithms for Discrete Optimization', *Artificial Life*, Vol. 5, pp. 137-172.

Gosavi, A., (2003) 'Simulation Based Optimization', *Kluwer Academic Publishers*.

Kauppila, M. (2006) 'Ant Colony Optimization as a Cross Entropy variant – a case study with DNA reconstruction', http://www.rawhed.com/uttumuttu/index.html.

Kubiak, W. (1993) 'Minimizing variation of production rates in just-in-time systems: A survey', *European Journal of Operational Research*, Vol. 66, No. 3, pp. 259-271.

Miltenburg, J. (1989) 'Level schedules for mixed-model assembly lines in just-in-time production systems', *Management Science*, Vol. 35, No. 2, pp. 192-207.

Monden, Y. (1983) 'Toyota Production Systems', *Industrial Engineering and Management Press*, Norcross, GA.

Moreno, N. (2002) 'Solving the Product Rate Variation Problem (PRVP) of large dimensions as an assignment problem', *Doctoral Thesis*, DOE, Universitat Politècnica de Catalunya.

Moreno, N. and Corominas, A. (2006) 'Solving the minmax product rate variation problem (PRVP) as a bottleneck assignment problem', *Computers & Operations Research*, Vol. 33, No. 4, pp. 928-939.

Nelder, J.A. and Mead, R. (1965) 'A simplex method for function minimization', *The Computer Journal*, Vol. 7, pp. 308-313.

Rubinstein, R.Y. (1997) 'Optimization of Computer Simulation Models with Rare Events', *European Journal of Operational Research*, Vol. 99, No. 1, pp. 89-112.

Rubinstein, R.Y. and Kroese, D.P. (2004) 'The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning', Springer-Verlag, New York.

Solving the Response Time Variability Problem by means of metaheuristics[†]

Alberto GARCÍA, Rafael PASTOR and Albert COROMINAS
Institut d'Organització i Control de Sistemes Industrials (IOC), Universitat Politècnica de Catalunya, Barcelona, Spain.

Abstract. The Response Time Variability Problem (RTVP) is a combinatorial NP-hard problem which has a wide range of real-life applications. It has recently appeared in the literature and has therefore not been widely discussed to date. The RTVP has been solved in other works by mixed integer linear programming (for small instances) and heuristics, but metaheuristic procedures have not previously been used. In this paper, a solution to the RTVP by means of multi-start, GRASP and PSO procedures is proposed. We report on our computational experiments and draw conclusions.

Keywords: response time variability, fair sequences, scheduling, metaheuristics.

Introduction

The Response Time Variability Problem (RTVP) consists in sequencing a list of products, events, clients and jobs in such a way that the variability in the time they spend waiting for their next turn to obtain the resources they need is minimized. This problem has recently been defined in the literature and to date very few papers have been published on the subject [1], [2], [3].

Corominas et al. [2] have proved that the RTVP is a combinatorial NP-hard problem and, with the exception of a few special cases, they have in fact found an optimum solution to the problem only for small instances. Therefore, solving the problem by means of heuristic and metaheuristic procedures is entirely justified. In this paper, a solution to the RTVP is put forward by applying the following three procedures: multi-start, GRASP (Greedy Randomized Adaptive Search Procedure) and PSO (Particle Swarm Optimization).

The multi-start method is based on generating initial random solutions and on improving each of them to find a local optimum, which is usually done through a local search procedure.

GRASP, designed by Feo and Resende [5] in 1989, can be considered to be a variant of the multi-start method in which the initial solutions are obtained using directed randomness. They are generated by means of a greedy strategy in which random steps are added and the choice of the elements to be included in the solution is adaptive.

PSO is a metaheuristic procedure designed by Kennedy and Eberhart [6] in 1995. The original algorithm was designed for working with continuous functions of real variables and has obtained good results. Furthermore, it has recently been adapted for the purposes of working with combinatorial problems such as the travelling salesperson problem [7] or the flowshop problem [8]. In spite of these good results, there are not many PSO methods for solving combinatorial optimization problems.

[†] Sponsored by the Spanish Ministry of Education and Science's project DPI2004-03472; co-funded by the FEDER.

The remainder of this paper is set out as follows: Section 1 presents a formal definition of the RTVP; Section 2 briefly describes the methods used and how they were adapted to solve the RTVP; Section 3 explains how the values for the metaheuristic parameters were established; the computational results are shown in Section 4; and finally, the conclusions are put forward in Section 5.

1. Response Time Variability Problem (RTVP)

The Response Time Variability Problem occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources.

The RTVP occurs in a wide range of real-life applications. For example, it is a common occurrence in the automobile industry in the sequencing of models [9] and in the Asynchronous Transfer Mode (ATM) when multimedia systems need to broadcast video or sound at a specific time [10].

These kinds of situations are often considered to be distance-constrained scheduling problems, in which the distance between any two given consecutive units of the same product is bounded. However, in the RTVP the aim is to minimize variability in the distances between any two consecutive units of the same product and to find a feasible solution that optimizes this objective.

The RTVP is formulated as follows. Let *n* be the number of products, d_i the number of units of product *i* and *D* the total number of units ($D=\sum_{i=1}^{n} d_i$). Let *s* be a solution of an instance in the RTVP that consists of a circular sequence of units ($s = s_1 s_2 \ldots s_D$), where s_j is the unit sequenced in position *j* of sequence *s*. For all products *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the units *k*+1 and *k* of product *i* are found (i.e., the number of positions between them). As the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_i}^i$ is the distance between the first unit of product *i* in a cycle and the last unit of the same product in the preceding cycle. Let $\bar{t}_i$ be the average distance between two consecutive units of product *i* ($\bar{t}_i = D/d_i$). For all products *i* in which $d_i = 1$, t_1^i is equal to $\bar{t}_i$. The objective is to minimize the $RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2$.

For example, let $n = 3$, $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\bar{t}_C = 2$. A feasible solution is the sequence (C, A, C, B, C, B, A, C) where $RTV = \left[(5-4)^2 + (3-4)^2\right] + \left[(2-4)^2 + (6-4)^2\right] + \left[(2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2\right] = 2 + 8 + 2 = 12$

Corominas et al. [2] proved that the RTVP is NP-hard. The RTVP was optimally solved by means of mathematical programming, up to 40 units [3], and by means of heuristic procedures plus local optimization [2].

2. Multi-start, GRASP and PSO metaheuristic methods

2.1. Multi-start method

The multi-start method consists in generating random solutions, applying local optimization methods and preserving the best results.

The pseudocode of the adaptation of the multi-start method is

```
1. Let the value of the best solution found be $\overline{Z} = \infty$.
2. While (actual time < execution time), do:
3.          Get a random initial solution X
4.          Apply the local optimization to X and get X'
5.          If value (X') < $\overline{Z}$, then $\overline{Z}$ = value (X')
```

Random solutions are generated as follows. For each position from 1 to *D* in the sequence, we randomly obtain which product will be sequenced with a probability equal to the number of units of that type of product that remain to be sequenced divided by the total number of units that remain to be sequenced.

The local optimization is applied as follows. A local search is performed iteratively in a neighbourhood that is generated by interchanging two consecutive units; the best solution in the neighbourhood is chosen; the optimization ends when no neighbouring solution remains that is better than the current solution.

2.2. Greedy Randomized Adaptive Search Procedure (GRASP) method

Like the multi-start method, GRASP consists in generating solutions, applying local optimizations and preserving the best results. However, the generation of solutions is performed by applying a heuristic with directed randomness, which is usually a random variation of a simple greedy heuristic. At each stage in the heuristic, the next product to be added to the solution is randomly selected from a list of candidates with a probability proportional to the value of an associated index.

The pseudocode of the GRASP adaptation is almost the same as that of the multi-start method: the only difference is the way in which the initial solutions are obtained, which is as follows. For each position from 1 to *D* in the sequence, the product to be sequenced is randomly selected from the candidate list with a probability proportional to the value of its Webster index. This index, defined in [2], is as follows: let $\delta = 1/2$ and let x_{ik} be the number of units of product *i* that have already been sequenced in the sequence of length *k*, *k* = 0, 1, …; the value of the Webster index of product *i* to be sequenced in position $k+1$ is $\frac{d_i}{x_{ik} + \delta}$.

The local optimization used is the same as the optimization used in the multi-start method.

The size of the candidate list was set to 5 candidates.

2.3. Particle Swarm Optimization (PSO) method

Kennedy and Eberhart designed the PSO metaheuristic by establishing an analogy to the social behaviour of flocks of birds when they search for food. Originally, this metaheuristic was designed to optimize continuous functions of real variables [6]. Due to its good performance, it has been adapted for the purposes of working with combinatorial problems [7], [8], [11].

In this kind of algorithm, the *particles*, which correspond to the birds, have a position (a feasible solution) and a velocity (the change in their position), and the set of particles form the *swarm*, which corresponds to the flock.

At each step in the PSO algorithm, the behaviour of a particle is the result of the combination of the following three factors: 1) to continue on the path that it is following, 2) to follow the best solution found and 3) to go to the best position found by the swarm. The formalization of this behaviour is expressed in the following two equations:

$$v_{t+1} = c_1 \cdot v_t \otimes c_2 \cdot (BP_t - X_t) \otimes c_3 \cdot (BSP_t - X_t) \qquad (1)$$

$$X_{t+1} = X_t + v_{t+1} \qquad (2)$$

where v_t is the velocity of the particle at time step *t*; X_t is the position of the particle at time step *t*; BP_t is the best position of the particle up to time step *t*; BSP_t is the best position of the swarm up to time step *t*; and c_1, c_2 and c_3 are the coefficients that weight the importance of the three types of decision.

The values of coefficients c_1, c_2 and c_3 are usually fixed in advance.

To apply the PSO algorithm to the RTVP, the elements and the operations of the equations (1) and (2) have to be defined.

2.3.1. Position of the particle

As mentioned above, a position represents a feasible solution. The position is represented by a D-length array that contains the sequence of *D* units.

2.3.2. Velocity of the particle

The expression ($X_2 - X_1$) represents the difference between two positions and it is the velocity needed to go from position X_1 to X_2. This velocity is an ordered list of transformations (called *movements*) that must be applied to the particle so that it changes from its current position to the other one. Two types of movements, each of which had two variations, were considered.

The first type of movement, called *M1*, is a pair of values (α / j). For each position *s* in the sequence X_1, a check is conducted to determine whether the unit in this position *s* is equal to the unit in position *s* of sequence X_2. If they are different, α is the unit in position *s* of X_2 and *j* is position *s*. Thus, this movement denotes that the unit in position *j* must be exchanged for the first unit that is equal to α and that is to the right of position *s*. This concept is used to solve the CONWIP problem [11].

The second type of movement, called *M2*, is a pair of positions (i, j). These values indicate that the units that are sequenced in positions *i* and *j* have been exchanged.
Two examples of the movements that are needed to move to position X_2 (A-B-C-A-B-C-A-B-C) from position X_1 (A-A-A-B-B-B-C-C-C) are shown below.

M1: movements (B/2), (C/3) and (C/6) are needed.
A-A-A-B-B-B-C-C-C → (B/2) → A-**B**-A-**A**-B-B-C-C-C → (C/3) →
A-B-**C**-A-B-B-**A**-C-C → (C/6) → A-B-C-A-B-**C**-A-**B**-C
M2: movements (2,4), (3,7) and (6,8) are needed.
A-A-A-B-B-B-C-C-C → (2,4) → A-**B**-A-**A**-B-B-C-C-C → (3,7) →
A-B-**C**-A-B-B-**A**-C-C → (6,8) → A-B-C-A-B-**C**-A-**B**-C

There would seem to be no difference between *M1* and *M2*, but when two velocities are added (see Section 2.3.4) then lists of movements that refute this may appear.

The two variations for each movement are: 1) if only the type of product is used to compare two units (this variation is called *T* and it is used in examples above), and 2) if the unit number is used to compare two units and therefore a unit is only equal to itself (this variation is called *F*). For example, in the case of variation *F*, position A1-A2-A3-B1-B2-B3-C1-C2-C3 (in which the number next to each letter is a unit identifier for each product) is different to position A2-A1-A3-B1-B3-B2-C1-C2-C3, but in variation *T* the two positions are equal (they appear as A-A-A-B-B-B-C-C-C).

The difference between two positions using variation *F* will always be greater than or equal to the difference when variation *T* is applied.

2.3.3. External multiplication of a coefficient by a velocity

The coefficients c_1, c_2 and c_3 yield values of between 0 and 1. When a coefficient is multiplied by a velocity, it indicates the probability of each movement that is to be applied. For example, if we multiply velocity [(B/2), (C/3), (C/6)] by coefficient 0.6, three random numbers between 0 and 1 are generated for comparison with coefficient 0.6; if the values are 0.3, 0.8 and 0.4, then movements (B/2) and (C/6) are applied, whereas movement (C/3) is not. The resulting velocity of the multiplication is therefore [(B/2), (C/6)].

2.3.4. Sum of velocities

The sum of two velocities is simply the concatenation of their own list of movements.

2.3.5. Sum of a velocity plus a position

The sum of a velocity plus a position gives the same result as applying each movement of the velocity to the position.

2.3.6. Pseudocode of the algorithm

```
1. Initiate the particles with random positions and empty velocities.
2. While (actual time < execution time), do:
3.        Update the best swarm position.
4.        For each particle:
5.            update its best position and apply the two PSO equations.
```

The random positions are generated in the same way as the random solutions in the multi-start method.

3. Fine-tuning the PSO parameters

Adapting metaheuristics to a specific problem does not end with the definition of the space of solutions or the local search; moreover, it is required to set the parameters. The value of the parameters is vital because the results of the metaheuristic for each problem are very sensitive to them. To fine-tune is very expensive and it is usually done by intuitively testing several values.

For the purposes of this paper, we fine-tuned the parameters using a recent technique called CALIBRA [12]. CALIBRA is an automatic configuration procedure based on statistical analysis techniques (Taguchi's fractional factorial experimental designs) coupled with a local search procedure. A set of 60 representative instances was used to fine-tune the algorithms and a set of 740 units was used to test them. The four parameters to be fine-tuned were the number of particles in the swarm and coefficients c_1, c_2 and c_3. The range of the values used to fine-tune the algorithms was [5,30] for the number of particles and [0,1] for the coefficients. CALIBRA needed 35 hours to fine-tune each algorithm.

4. Computational results

As described in Section 2.3.2, depending on the type of movement (*M1* or *M2*) and the variation (*T* or *F*), we have four PSO algorithms (called *M1-F*, *M1-T*, *M2-F* and *M2-T*), as well as the multi-start algorithm and the GRASP algorithm.

The algorithms ran 740 instances, which were grouped into four classes (185 instances in each class) depending on their size. The instances in the first class (called *CAT1*) were generated using a random value of *D* (number of units) between 25 and 50, and a random value of *n* (number of products) between 3 and 15; for the second class (called *CAT2*), *D* was between 50 and 100, and *n* between 3 and 30; for the third class (called *CAT3*), *D* was between 100 and 200 and *n* between 3 and 65; and for the fourth class (called *CAT4*), *D* was between 200 and 500 and *n* between 3 and 150.

The algorithms were coded in Java and the computational experiments were carried out using a 3.4 GHz Pentium IV with 512 Mb of RAM.

Table 1. Averages of the RTV values to be minimized

	PSO				**Multi-start**	**GRASP**
	M1F	**M1T**	**M2F**	**M2T**		
CAT1	68.79	66.83	83.14	80.93	11.33	13.90
CAT2	445.55	509.89	604.27	517.05	48.10	91.64
CAT3	3050.38	4335.87	4488.44	3888.79	320.63	541.52
CAT4	28955.82	48917.80	37937.76	30029.34	79823.89	57041.74

Firstly, the six algorithms were run for 50 seconds for each instance. Table 1 shows the averages of the RTV values to be minimized for each class of instances.

In Table 1 it can be seen that the best results for the three first classes are given by the multi-start method, followed by the GRASP method, whereas the PSO algorithm yields the worst results. However, in the case of class *CAT4*, in which the instances are largest, the order is the reverse: the four PSO algorithms yield better results than the GRASP method, and the multi-start method gives the worst results. The reason for this is that the multi-start method does not have time to locally optimize a single solution for 87.57% of the instances in the *CAT4* class; this happens in the GRASP method for 84.32% of the instances.

The second computational experiment consisted in locally optimizing the solutions that were obtained with the PSO algorithms in the first computational experiment. The optimization used was the same as the multi-start optimization; it stops after 50 seconds if the optimization has not been completed. Table 2 shows the averages of the RTV values obtained for each class of instances.

Table 2. Averages of the RTV values of the PSO local optimized solutions

	M1F	**M1T**	**M2F**	**M2T**
CAT1	21.61	24.43	23.61	25.65
CAT2	67.42	89.75	77.56	95.14
CAT3	229.32	406.63	302.06	427.09
CAT4	15842.12	29604.35	20560.1	15537.62

The results obtained using *M1F* for the instances in class *CAT3* after local optimization are better than the results obtained using the multi-start method. Moreover, the optimization times for the first two classes are negligible and the average time for the third class is between 4.26 and 5.84 seconds (using *M1F* and *M1T*, respectively). The instances in the first three classes were all locally optimized. However, there was not enough time to optimize all the instances in class *CAT4*: only 60 instances (32.43%) were locally optimized based on the solutions that were obtained using *M1F*.

Finally, the six procedures were re-run for 200 seconds using the instances in class *CAT4*, which are the most difficult to solve. In the case of PSO algorithms, 100 seconds were spent on obtaining a solution and a further 100 seconds, at the most, were spent on locally optimizing the previous solution. Table 3 shows the average of the RTV values obtained for class *CAT4* (the values in parenthesis were obtained using the PSO algorithms before local optimization was applied).

Table 3. Average of the RTV values of the *CAT4* instances

M1F	**M1T**	**M2F**	**M2T**	**multi-start**	**GRASP**
(*24022.52*) *8782.07*	(44697.30) 21432.13	(36445.60) 14892.35	(29838.01) 11984.25	39719.71	30020.35

The results show that all the PSO algorithms give better results than the multi-start and GRASP algorithms. In this last experiment, 97 instances (52.43%) were locally optimized after applying the *M1F* algorithm.

5. Conclusions and future lines of research

In this paper we have presented our solution to the RTVP (a problem that has not been widely researched to date), to which six algorithms were applied: one multi-start, one GRASP and four PSO.

The results show that the best procedure is the multi-start for small instances (between 25 and 100 units and between 3 and 30 products). However, for bigger instances (between 100 and 500 units and between 3 and 150 products) the search should be more specific as the four PSO algorithms are much better than the multi-start and GRASP methods and the latter are better than the multi-start methods. Moreover, as was to be expected, there is a significant improvement in the solutions that were obtained using the PSO algorithm to which local optimization had been applied.

Future research will consist in adapting new metaheuristic procedures, such as for example simulated annealing and tabu search.

References

[1] D. León, A. Corominas, A. Lusa, *Resolución del problema PRV min-var*, Working paper IOC-DT-I-2003-03, UPC, Barcelona, Spain, 2003.

[2] A. Corominas, W. Kubiak, N. Moreno, *Response time variability*, Working paper IOC-DT-P-2004-08. UPC, Barcelona, Spain, 2004.

[3] A. Corominas, W. Kubiak, R. Pastor, *Solving the Response Time Variability Problem (RTVP) by means of mathematical programming,* Working paper IOC-DT, UPC, Barcelona, Spain, 2006.

[4] R Martí, *Multi-start methods*, Handbook of Metaheuristics, Glover and Kochenberger (eds.), Kluwer Academic Publishers, pp. 355-368, 2003.

[5] T.A. Feo, M.G.C. Resende, *A probabilistic heuristic for a computationally difficult set covering problem*, Operations Research Letters, vol. 8, pp. 67-81, 1989.

[6] J. Kennedy, R.C. Eberhart, *Particle swarm optimization*, IEEE International Conference on Neural Networks, Australia, 1995.

[7] B. Secrest, *Travelling salesman problem for surveillance mission using PSO*, PhD thesis, Air Force Institute of Technology, Ohio, USA, 2001.

[8] C.J. Liao, C.T. Tseng, P. Luarn, *A discrete version of PSO for flowshop scheduling problems*, Computers & Operations Research, in press, corrected proof available online, 5 December 2005.

[9] Y. Monden, *Toyota Production Systems*, Industrial Engineering and Management Press, Norcross, GA, 1983.

[10] L. Dong, R. Melhem, D. Mossel, *Time slot allocation for real-time messages with negotiable distance constraint requirements*, Real-time Technology and Application Symposium, RTAS, Denver, 1998

[11] C. Andrés, R. Pastor, J.M. Framiñán, *Optimización mediante cúmulos de partículas del problema de secuenciación CONWIP*, Eighteenth Conference on Statistics and Operations Research SEIO’04, Cádiz, Spain, 2004.

[12] B. Adenso-Díaz, M. Laguna, *Fine-tuning of algorithms using fractional experimental designs and local search,* Operations Research, vol. 54, no. 1, pp. 99-114, 2006.

Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics[†]

Albert COROMINAS, Alberto GARCÍA-VILLORIA[*], Rafael PASTOR
Institute of Industrial and Control Engineering (IOC), Technical University of Catalonia (UPC), Barcelona, Spain

Abstract. The Response Time Variability Problem (RTVP) is an NP-hard scheduling optimization problem that has recently appeared in the literature. This problem has a wide range of real-life applications in, for example, manufacturing, hard real-time systems, operating systems and network environments. The RTVP occurs whenever models, clients or jobs need to be sequenced to minimize variability in the time between the instants at which they receive the necessary resources. The RTVP has been already solved in the literature with a multi-start and a GRASP algorithm. We propose an improved multi-start and an improved GRASP algorithm to solve the RTVP. The computational experiment shows that, on average, the results obtained with our proposed algorithms improve on the best obtained results to date.

Keywords. response time variability, regular sequences, scheduling, multi-start metaheuristic, grasp

Introduction

The Response Time Variability Problem (RTVP) is a scheduling problem that has recently been formalized in [1]. The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources. Although this optimization problem is easy to formulate, it is very difficult to solve optimally (it is NP-hard [1]).

The RTVP has a broad range of real-life applications. For example, it can be used to regularly sequencing models in the automobile industry [2], to allocating resources in computer multi-threaded systems and network servers [3], to broadcasting video and sound data frames of applications over asynchronous transfer mode networks [4], in the periodic machine maintenance problem when the distances between consecutive services of the same machine are equal [5] and in the collection of waste [6].

One of the first problems in which has appeared the importance of sequencing *regularly* is the sequencing on the mixed-model assembly production lines at Toyota Motor Corporation under the just-in-time (JIT) production system. One of the most important JIT objectives is to get rid of all kinds of waste and inefficiency and, according to Toyota, the main waste is due to the stocks. To reduce the stock, JIT production systems require to producing only the necessary models in the necessary quantities at the necessary time. To achieve this, one main goal, as Monden says [2], is scheduling the units to be produced to keep constant consumption rates of the components involved in the production process. Miltenburg [7] deals with this scheduling problem and he assumes that models require approximately the same number and mix of parts. Thus, he considers only the demand rates for the models. In our experience with practitioners of manufacturing industries, we noticed that they usually refer to a good mixed-model sequence in terms of having distances between the units for the same model as regular as

[†] Sponsored by the Spanish Ministry of Education and Science's project DPI2007-61905; co-funded by the FEDER.
[*] Corresponding author: Alberto García-Villoria, IOC – Institute of Industrial and Control Engineering, Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; e-mail: alberto.garcia-villoria@upc.edu

possible. Therefore, the metric used in the RTVP reflects the way in which practitioners refer to a desirable regular sequence.

In [1], a mixed integer lineal programming (MILP) model to solve the RTVP has been proposed. The previous MILP model has been improved in [8], but the practical limit to obtain optimal solutions is 40 units to be scheduled. Thus, the use of heuristic or metaheuristic methods for solving real-life RTVP instances is justified. In [1], five greedy heuristic algorithms have been proposed. Seven metaheuristic algorithms -one multi-start, one GRASP (Greedy Randomized Adaptive Search Procedure) and four PSO (Particle Swarm Optimization) algorithms- have been proposed in [9]. Finally, eleven PSO metaheuristic algorithms were used to solve the RTVP in [10].

The general scheme of the multi-start metaheuristic consists of two phases. In the first phase an initial solution is generated. Then, the second phase improves the obtained initial solution. These two phases are iteratively applied until a stop condition is reached. The GRASP metaheuristic can be considered a variant of the multi-start metaheuristic in with the initial solutions are obtained using direct randomness. They are generated by means of a greedy strategy in which random steps are added and the choice of the elements to be included in the solution is adaptive.

This paper is an extension of the work initialized in [9]. The new research done with the PSO metaheuristic was reported in [10] and the PSO algorithm called *DPSOpoi-c$_p$dyn* by the authors is the best algorithm to date for solving the RTVP. In this paper we propose an improved multi-start algorithm and an improved GRASP algorithm. On average, the proposed algorithms improve strongly on previous results.

The rest of this paper is organized as follows. Section 1 presents a formal definition of the RTVP. Section 2 explains the existing multi-start and GRASP for solving the RTVP and proposes two new improved multi-start and GRASP algorithms. Section 3 provides the computational experiment and the comparison with the best algorithm to solve the RTVP (*DPSOpoi-c$_p$dyn*) and the existing multi-start and GRASP algorithms. Finally, some conclusions are given in Section 4.

1. Response Time Variability Problem (RTVP)

The aim of the Response Time Variability Problem (RTVP) is to minimize the variability of the distances between any two consecutive units of the same model in the sequence.

The RTVP is stated as follows. Let *n* be the number of models, d_i the number of units of the model *i* (*i* = 1,…,*n*) to be scheduled and *D* the total number of units ($D=\sum_{i=1}^{n} d_i$). Let *s* be a solution of an instance of the RTVP. It consists in a circular sequence of units $(s = s_1 s_2 ... s_D)$, where s_j is the unit sequenced in position *j* of sequence *s*. For all model *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the units *k* + 1 and *k* of the model *i* are found (where the distance between two consecutive positions is considered equal to 1). Since the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_i}^i$ is the distance between the first unit of the model *i* in a cycle and the last unit of the same model in the preceding cycle. Let $\overline{t_i}$ be the average distance between two consecutive units of the model *i* $(\overline{t_i} = D / d_i)$. For all model *i* in which $d_i = 1$, t_1^i is equal to $\overline{t_i}$. The objective is to minimize the metric Response Time Variability (RTV), which is defined by the following expression: $RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \overline{t_i})^2$.

For example, let $n = 3$, $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\overline{t_A} = 4$, $\overline{t_B} = 4$ and $\overline{t_C} = 2$. Any sequence such that contains exactly d_i times the symbol *i* $(\forall i)$ is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a solution, where

$$RTV = \left((5-4)^2 + (3-4)^2\right) + \left((2-4)^2 + (6-4)^2\right) + \left((2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2\right) = 1$$

As explained in the introduction, the best RTVP results recorded to date were obtained by using a PSO algorithm called *DPSOpoi-c$_p$dyn* [10]. PSO is a population metaheuristic algorithm based on the social behaviour of flocks of birds when they search for food. The population or swarm is composed of particles (birds), whose attributes are an *n*-dimensional real point (which represents a feasible solution) and a velocity (the movement of the point in the *n*-dimensional real space). The velocity of a particle is typically a combination of three types of velocities: 1) the inertia velocity (i.e., the previous velocity of the particle); 2) the velocity to the best point found by the particle; and 3) the velocity to the best point found by the swarm. These components of the particles are modified iteratively by the algorithm as it searches for an optimal solution. Although the PSO algorithm was originally designed for *n*-dimensional real spaces, *DPSOpoi-c$_p$dyn* is adapted to work with a sequence that represents the solution. Moreover, *DPSOpoi-c$_p$dyn* introduces random modifications to the points of the particles with a frequency that changes dynamically according to the homogeneity of the swarm (for more details, see [10]).

2. The multi-start and GRASP algorithms

2.1. The multi-start algorithm

The multi-start metaheuristic is a general scheme that consists of two phases. The first phase obtains an initial solution and the second phase improves the obtained initial solution. These two phases are applied iteratively until a stop condition is reached. This scheme has been first used at the beginning of 80's [11]. The generation of the initial solution, how to improve them and the stop condition can be very simple or very sophisticated. The combination of these elements gives a wide variety of multi-start methods. For a good review of multi-start methods, see [12] and [13].

The multi-start algorithm proposed in [9] for solving the RTVP is based on generating, at each iteration, a random initial solution and on improving it by means of a local search procedure. The algorithm stops after it has run for a preset time. Random solutions are generated as follows. For each position, a model to be sequenced is randomly chosen. The probability of each model is equal to the number of units of this model that remain to be sequenced divided by the total number of units that remain to be sequenced. The local search procedure used is applied as follows. A local search is performed iteratively in a neighbourhood that is generated by interchanging each pair of two consecutive units of the sequence that represents the current solution; the best solution in the neighbourhood is chosen; the optimization ends when no neighbouring solution is better than the current solution.

If the quality of the initial solutions is low, the computing time required by the local search to find the local optimum is increased. For big RTVP instances, few iterations may be done because of the available execution time. An easy and fast way to obtain better initial solutions without giving up the simplicity of the multi-start algorithm could be generating, at each iteration, *P* random solutions and get as the initial solution the best of them, that is, applying the local search only for the best solution of the *P* random solutions. In this paper we propose a parametric multi-start algorithm to solve the RTVP that has one parameter: the number of random solutions generated at each iteration (*P*). Figure 1 shows the pseudocode of our algorithm.

Figure 1. Pseudocode of the proposed multi-start algorithm

```
1. Set the value of the parameter $P$
2. Let the best solution found $\bar{X}$ initially be void
3. Let the RTV value of the best solution found be $\bar{Z} = \infty$
4. While execution time is not reached do:
5.        Generate $P$ random solutions
6.        Let $X$ the best solution generated at step 5
7.        Apply the local optimization to $X$ and get $X^{opt}$
8.        If $RTV(X^{opt}) < \bar{Z}$, then $\bar{X} = X^{opt}$ and $\bar{Z} = RTV(X^{opt})$
9. End While
```

As it has been mentioned, when the execution time of the algorithm is reached, the algorithm is immediately stopped (that is, the current local optimization is also stopped).

2.2. The GRASP algorithm

The GRASP metaheuristic was designed in 1989 by Feo and Resende [14] and can be considered as a multi-start variant. However, the generation of the initial solution is performed by means of a greedy strategy in which random steps are added and the choice of the elements to be included in the solution is adaptive.

The random step in the GRASP proposed in [9] consists of selecting the next model to be sequenced from a set called candidate list; the probability of each candidate model is proportional to the value of an associated index. The index used in [9] is the Webster index, which is evaluated as follows. Let x_{ik} be the number of units of model i that have been already sequenced in the sequence of length k, $k = 0, 1, \dots$ (assuming $x_{i0} = 0$); the value of the Webster index of model i to be sequenced in position $k + 1$ is $d_i / (x_{ik} + 0.5)$. The local optimization used is the same as the optimization used in the multi-start algorithm.

In this paper we propose to use another index that is evaluated as follows. Let x_{ik} be the number of units of model i that have been already sequenced in the sequence of length k, $k = 0, 1, \dots$ (assuming $x_{i0} = 0$), d_i the number of units of the model i to be sequenced and D the total number of units to be sequenced; the value of our index of the model i to be sequenced in position $k + 1$ is:

$$\frac{(k+1)\cdot d_i}{D} - x_{ik} \qquad (1)$$

If there is a tie, then the models with lower d_i are first added in the candidate list.

2.3. Fine-tuning of the algorithm parameters

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although the parameter values are extremely important because the results of the metaheuristic for each problem are very sensitive to them, the selection of parameter values is commonly justified in one of the following ways [15]: 1) "by hand" on the basis of a small number of experiments that are not specifically referenced; 2) by using the general values recommended for a wide range of problems; 3) by using the values reported to be effective in other similar problems; or 4) by choosing values without any explanation.

Adenso-Díaz and Laguna [15] proposed a new technique called CALIBRA for fine-tuning the parameters of heuristic and metaheuristic algorithms. CALIBRA is based on Taguchi's fractional factorial experimental designs coupled with a local search procedure.

CALIBRA has been chosen for fine-tuning the parameters of our proposed parametric multi-start algorithm, our proposed GRASP algorithm and the GRASP algorithm proposed in [9] (the multi-start algorithm proposed in [9] has not parameters) using a set of 60 representative training instances (generated as explained in Section 3). The following parameter values are obtained: for the parametric multi-start algorithm, P = 1,500, and for both GRASP algorithms, size of the candidate list = 3.

The size of the candidate list used in the GRASP algorithm proposed in [9] was 5, but the computational experiment showed that slightly better results are obtained, on average, using the value returned by CALIBRA. Thus, the results shown in the next section are referred only to the ones obtained using a size of the candidate list equal to 3.

3. Computational experiment

Our two proposed algorithms are compared with the PSO algorithm called *DPSOpoi-$c_p$dyn* [10], which is the most efficient algorithm published to date to solve non-small RTVP instances. We compare also our algorithms with the multi-start and the GRASP algorithms proposed in [9] in order to compare the improvements achieved with the modifications that we have proposed. In what follows in this section, we refer to the multi-start and GRASP algorithms proposed in [9] as *MS-old* and *GR-old*, respectively; and we refer to our proposed multi-start and GRASP algorithms as *MS-new* and *GR-new*, respectively.

The computational experiment was carried out for the same instances used in [9] and [10]. That is, the algorithms ran 740 instances which were grouped into four classes (185 instances in each class) according to their size. The instances in the first class (*CAT1*) were generated using a random value of D (total number of units) uniformly distributed between 25 and 50, and a random value of n (number of models) uniformly distributed between 3 and 15; for the second class (*CAT2*), D was between 50 and 100 and n between 3 and 30; for the third class (*CAT3*), D was between 100 and 200 and n between 3 and 65; and for the fourth class (*CAT4*), D was between 200 and 500 and n between 3 and 150. For all instances and for each model $i = 1,\ldots,n$, a random value of d_i (number of units of model i) was between 1 and $\left|\frac{D-n+1}{2.5}\right|$ such that $\sum_{i=1}^{n} d_i = D$. All algorithms were coded in Java and the computational experiment was carried out using a 3.4 GHz Pentium IV with 1.5 GB of RAM.

For each instance, all algorithms were run for 50 seconds. Table 1 shows the averages of the RTV values to be minimized for the global of 740 instances and for each class of instances (*CAT1* to *CAT4*).

Table 1. Averages of the RTV values for 50 seconds

	MS-new	***GR-new***	***DPSOpoi-$c_p$dyn***	***MS-old***	***GR-old***
Global	**2,106.01**	**2,308.69**	**4,625.54**	**21,390.40**	**14,168.83**
CAT1	11.56	13.00	16.42	12.08	15.47
CAT2	38.02	60.45	51.34	44.36	88.48
CAT3	154.82	270.93	610.34	226.90	510.44
CAT4	8,219.65	8,890.37	17,824.04	85,278.25	56,060.92

For the global of all instances, the results of our multi-start and GRASP algorithm are, on average, 54.47% and 50.09%, respectively, better than *DPSOpoi-$c_p$dyn*, which was to date the best algorithm to solve the RTVP. Moreover, *MS-new* is the best algorithm, on average, for small (*CAT1* and *CAT2*), medium (*CAT3*) and big (*CAT4*) instances. Comparing *MS-new* with *MS-old* by class, we can observe in Table 1 that *MS-new* is 4.30%, 14.29%, 31.77% and 90.36% better than *MS-old* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively; and *GR-new* is 15.97%, 31.68%, 46.92% and 84.14% better than *GR-old* for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. As we expected, the bigger are the instances, the bigger is the improvement obtained.

To complete the analysis of the results, their dispersion is observed. A measure of the dispersion (let it be called σ) of the RTV values obtained by each metaheuristic mh (mh = {*MS-new*, *GR-new*, *DPSOpoi-$c_p$dyn*, *MS-old*, *GR-old*}) for a given instance, ins, is defined as follows:

$$\sigma(mh, ins) = \left(\frac{RTV_{ins}^{(mh)} - RTV_{ins}^{(best)}}{RTV_{ins}^{(best)}}\right)^2 \tag{2}$$

where $RTV_{ins}^{(mh)}$ is the RTV value of the solution obtained with the metaheuristic mh for the instance ins, and $RTV_{ins}^{(best)}$ is, for the instance ins, the best RTV value of the solutions obtained with the four metaheuristics. Table 2 shows the average σ dispersion for the global of 740 instances and for each class of instances.

Table 2. Average σ dispersion regarding the best solution found for 50 seconds

	MS-new	*GR-new*	*DPSOpoi-$c_p$dyn*	*MS-old*	*GR-old*
Global	**2.55**	**6,650.83**	**4,931.36**	**202,910.13**	**268,299.58**
CAT1	0.08	0.26	0.87	0.21	0.79
CAT2	0.03	1.94	0.56	0.18	6.26
CAT3	0.05	3.07	13.80	0.50	14.18
CAT4	10.06	26,598.04	19,710.23	811,639.64	1,073,177.09

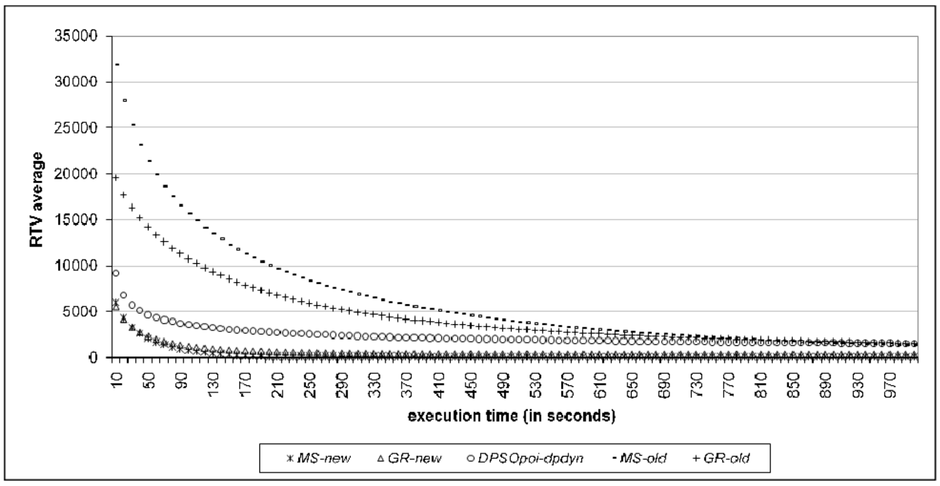
Observing the results in Table 2 by class, we can see that *MS-new* has always a very small dispersion far followed by the other algorithms. That means that *MS-new* has a very stable behaviour independently of the size of the instances. For small and medium instances (*CAT1*, *CAT2* and *CAT3*), *GR-new* has also a stable behaviour, but for some big instances (*CAT4*) *GR-new* obtains very bad RTV values. Note that although the RTV values of the *CAT4* instances obtained with *GR-new* are, on average, better than the values obtained with *DPSOpoi-$c_p$dyn*, the dispersion of *DPSOpoi-$c_p$dyn* is lower than the dispersion of *GR-new*. But comparing the *GR-new* dispersion with the *GR-old* dispersion, we can see *GR-new* has a much more stable behaviour than *GR-old*.

A computing time of 50 seconds may not be long enough to converge for the largest instances (*CAT4* instances). Table 3 shows the averages of the RTV values for the global of all instances and for each class of instances (*CAT1* to *CAT4*) obtained when the algorithms are run for 1000 seconds.

Table 3. Averages of the RTV values for 1,000 seconds

	MS-new	*GR-new*	*DPSOpoi-$c_p$dyn*	*MS-old*	*GR-old*
Global	**169.25**	**301.90**	**1,537.34**	**1,378.59**	**1,495.12**
CAT1	10.51	11.56	14.34	10.93	13.59
CAT2	31.21	50.45	46.55	35.48	75.08
CAT3	123.27	227.50	143.96	160.67	428.86
CAT4	512.02	918.10	5,944.51	5,307.25	5,462.95

Figure 2. Average of the RTV values obtained over the computing time



With 1,000 seconds of execution time, which seems time enough for the convergence of the five algorithms (see Figure 2), *MS-new* is for the global of all instances 43.94%, 88.99%, 87.72% and 88.68% better than the *GR-new*, *DPSOpoi-$c_p$dyn*, *MS-old* and *GR-old*, respectively; and *GR-new* is 80.36%, 78.10% and 79.81% better than *DPSOpoi-$c_p$dyn*, *MS-old* and *GR-old*, respectively. Although *DPSOpoi-$c_p$dyn*, *MS-old* and *GR-old* improve a lot their average results, *MS-new* and *GR-new* are clearly better.

Finally, the real-life industrial example presented in [6] was solved using *MS-new*, *GR-new* and *DPSOpoi-$c_p$dyn*. This example has the following characteristics: n = 14, d = (2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5) and, therefore, D = 46. The three algorithms were run ten times with an execution time limit of 1,000 seconds. *MS-new* found the optimal solution in all cases and the minimum, average and maximum computing times were 3.38, 8.74 and 25.03 seconds, respectively. *GR-new* found also the optimal solution in all cases and the minimum, average and maximum computing times were 1.08, 26.53 and 101.86 seconds, respectively. In contrast, *DPSOpoi-$c_p$dyn* found the optimal solution in only two cases, in computing times of 593.94 and 960.25 seconds.

4. Conclusions and future research

The RTVP occurs in diverse environments as manufacturing, hard real-time systems, operating systems and networks environments. In the RTVP, the aim is to minimize variability in the distances between any two consecutive units of the same model, i.e. to distribute the units as regular as possible. Since it is a NP-hard scheduling optimization problem, heuristic and metaheuristic methods are needed.

This paper is an extension of the work started in [9], in which one multi-start, one GRASP and four PSO algorithms were proposed. New PSO algorithms to solve the RTVP have been published in [10]. The best of them, *DPSOpoi-$c_p$dyn*, obtains the best results to date. In this paper, an improved multi-start algorithm, *MS-new*, and an improved GRASP algorithm, *GR-new*, are proposed to solve the RTVP.

The computational experiment shows clearly that the proposed algorithms obtain, on average, strongly better solutions than *DPSOpoi-$c_p$dyn* independently of the size of the RTVP instance. Moreover, *MS-new*, the proposed algorithm that obtains the best solutions, has always a very stable behaviour. Instead, *GR-new* and *DPSOpoi-$c_p$dyn* have not. Therefore, it is advisable to use always *MS-new* for solving the RTVP.

Although the RTVP is hard to solve, it is interesting to try to solve it by means of exact procedures to know the largest size of the RTVP instances that can be solved optimally in a practical computing time. The two exact procedures proposed in the literature are MILP models [1, 8]. Since the use of Constraint Programming (CP) to solving the RTVP has not been proposed yet in the literature, applying CP to RTVP seems a promising future line of research.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to two anonymous reviewers for their valuable comments, which have improved the quality of this paper.

References

[1] Corominas, A., Kubiak, W. and Moreno, N. (2007) ‘Response time variability’, *Journal of Scheduling*, Vol. 10, pp. 97-110.

[2] Monden, Y. (1983) ‘Toyota Production Systems’, *Industrial Engineering and Management Press*, Norcross, GA.

[3] Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Schedulling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528*, Massechusetts Institute of Technology, MIT Laboratory for Computer Science.

[4] Dong, L., Melhem, R. and Mosse, D. (1998) ‘Time slot allocation for real-time messages with negotiable distance constrains requirements’, *Fourth IEEE Real-Time Technology and Applications Symposium* (RTAS'98), Denver, CO. pp.131-136.

[5] Anily, S., Glass, C.A. and Hassin, R. (1998) ‘The scheduling of maintenance service’, *Discrete Applied Mathematics*, Vol. 82, pp.27-42.

[6] Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report*, University of Maryland, USA.

[7] Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 35, No. 2, pp. 192-207.

[8] Corominas, A., Kubiak, W. and Pastor, R. (2006) ‘Solving the Response Time Variability Problem (RTVP) by means of mathematical programming’, *Working paper IOC-DT*, Universistat Politècnica de Catalunya, Spain.

[9] García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 146, pp.187-194.

[10] García-Villoria, A. and Pastor, R. (2007) ‘Introducing dynamic diversity into a discrete particle swarm optimization’, *Computers & Operations Research*, In Press, Corrected Proof, Avalaible online 7 December 2007, doi:10.1016/j.cor.2007.12.001.

[11] Boender, C.G.E., Rinnooy, A.H.G., Stougie, L. and Timmer, G.T. (1982) ‘A Stochastic Method for Global Optimization’, *Mathematical Programming*, Vol. 22, pp.125-140.

[12] Martí, R. (2003), ‘Multi-start methods’, *Handbook of Metaheuristics*, Glover and Kochenberger (eds.), Kluwer Academic Publishers, pp.355-368.

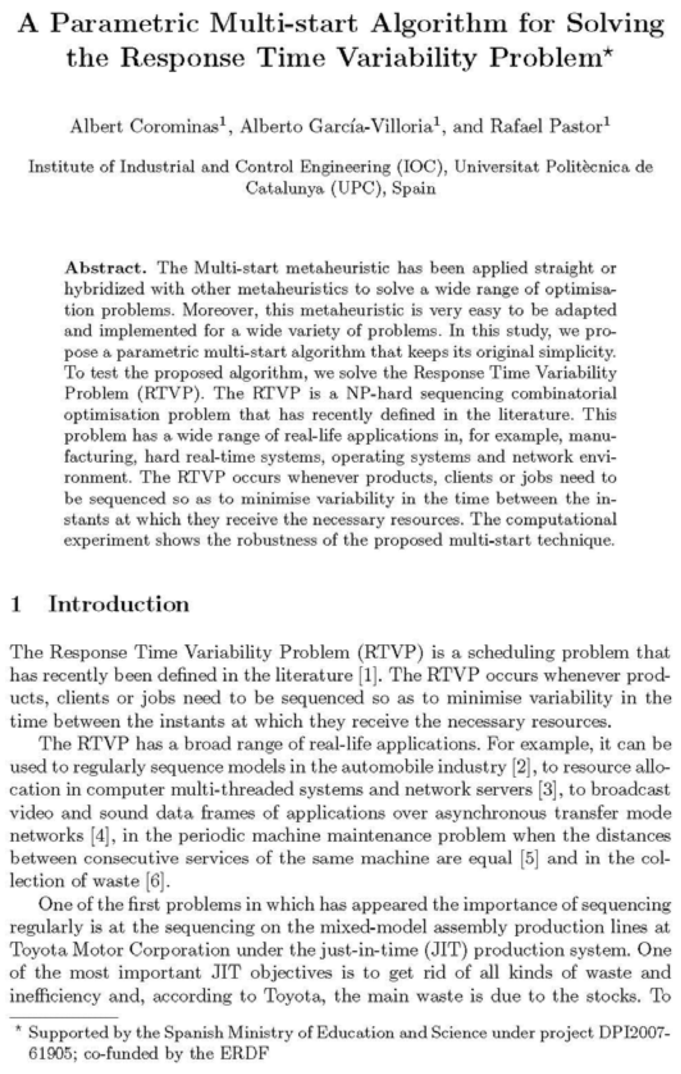
[13] Hoos, H. and Stützle, T. (2005) *Stochastic local research: foundations and applications*, Morgan Kaufmann Publishers, San Francisco.

[14] Feo, T.A. and Resende, M.G.C. (1989) ‘A probabilistic heuristic for a computationally difficult set covering problem’, *Operations Research Letters*, Vol. 8, pp.67-81.

[15] Adenso-Díaz, B. and Laguna, M. (2006) ‘Fine-tuning of algorithms using fractional experimental designs and local search’, *Operations Research*, Vol. 54, pp. 99-114.

A Parametric Multi-start Algorithm for Solving the Response Time Variability Problem

Article published as [Lecture Notes in Computer Science, Volume 5910, 2010, Pages 315-322] © [copyright Springer]



A Parametric Multi-start Algorithm for Solving the Response Time Variability Problem*

Albert Corominas[1], Alberto García-Villoria[1], and Rafael Pastor[1]

Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya (UPC), Spain

Abstract. The Multi-start metaheuristic has been applied straight or hybridized with other metaheuristics to solve a wide range of optimisation problems. Moreover, this metaheuristic is very easy to be adapted and implemented for a wide variety of problems. In this study, we propose a parametric multi-start algorithm that keeps its original simplicity. To test the proposed algorithm, we solve the Response Time Variability Problem (RTVP). The RTVP is a NP-hard sequencing combinatorial optimisation problem that has recently defined in the literature. This problem has a wide range of real-life applications in, for example, manufacturing, hard real-time systems, operating systems and network environment. The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimise variability in the time between the instants at which they receive the necessary resources. The computational experiment shows the robustness of the proposed multi-start technique.

1 Introduction

The Response Time Variability Problem (RTVP) is a scheduling problem that has recently been defined in the literature [1]. The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimise variability in the time between the instants at which they receive the necessary resources.

The RTVP has a broad range of real-life applications. For example, it can be used to regularly sequence models in the automobile industry [2], to resource allocation in computer multi-threaded systems and network servers [3], to broadcast video and sound data frames of applications over asynchronous transfer mode networks [4], in the periodic machine maintenance problem when the distances between consecutive services of the same machine are equal [5] and in the collection of waste [6].

One of the first problems in which has appeared the importance of sequencing regularly is at the sequencing on the mixed-model assembly production lines at Toyota Motor Corporation under the just-in-time (JIT) production system. One of the most important JIT objectives is to get rid of all kinds of waste and inefficiency and, according to Toyota, the main waste is due to the stocks. To

* Supported by the Spanish Ministry of Education and Science under project DPI2007-61905; co-funded by the ERDF

reduce the stock, JIT production systems require to producing only the necessary models in the necessary quantities at the necessary time. To achieve this, one main goal, as Monden [2] says, is scheduling the units to be produced to keep a constant consumption rates of the components involved in the production process. Miltenburg [7] deals with this scheduling problem and assumes that models require approximately the same number and mix of parts. Thus, only the demand rates for the models are considered. In our experience with practitioners of manufacturing industries, we noticed that they usually refer to a good mixed-model sequence in terms of having distances between the units for the same model as regular as possible. Therefore, the metric used in the RTVP reflects the way in which practitioners refer to a desirable regular sequence

The RTVP is a NP-hard combinatorial optimisation problem [1]. Thus, the use of heuristic or metaheuristic methods for solving real-life instances of the RTVP is justified. Two multi-start algorithms to solve the RTVP were proposed in [8] and in [9]. The general scheme of the multi-start metaheuristic consists of two phases. In the first phase an initial solution is generated. Then, the second phase improves the obtained initial solution. These two phases are iteratively applied until a stop condition is reached. The multi-start algorithm proposed in [8] to solve the RTVP consists of generating a random solution in the first phase and then applying to the random solution a local search in the second phase; the stop condition consists in reaching a given execution time.

If the quality of the initial solution is low, the execution time required by the local search to find its local optimum is increased. An easy and fast way to obtain better initial solutions could be generating at each iteration, for example, 5 solutions and applying a local search only for the best solution of them. Given an instance, if the execution time allows running 10 iterations, then 50 solutions are generated and 10 solutions are optimised in total. This idea is applied in the multi-start algorithm proposed in [9] to obtain the initial solutions to be optimised. However maybe a better performance is obtained if all the 50 solutions are generated at first and then are optimised the 10 best of them; or, for example, generating, at each iteration, 10 solutions and optimising the 2 best of them, etc. In this paper we propose a multi-start algorithm to solve the RTVP that has two new parameters: the number of initial solutions generated at each iteration and the number of the best generated solutions that are optimised at each iteration.

The remainder of this paper is organized as follows. Section 2 presents a formal definition of the RTVP; Section 3 proposes a parametric multi-start algorithm for solving the RTVP; Section 4 provides the results of a computational experiment; finally, some conclusions and suggestions for a future research are given in Section 5.

2 The Response Time Variability Problem (RTVP)

The aim of the Response Time Variability Problem (RTVP) is to minimise the variability in the distances between any two consecutive units of the same model.

The RTVP is formulated as follows. Let n be the number of models, d_i the number of units to be scheduled of the model i ($i = 1, \dots, n$) and D the total number of units ($D = \sum_{i=1}^{n} d_i$). Let s be a solution of an instance in the RTVP that consists of a circular sequence of units ($s = s_1 s_2 \dots s_D$), where s_j is the unit sequenced in position j of sequence s. For all unit i in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the units $k+1$ and k of the model i are found (i.e. the number of positions between them, where the distance between two consecutive positions is considered equal to 1). Since the sequence is circular, position 1 comes immediately after position D; therefore, $t_{d_i}^i$ is the distance between the first unit of the model i in a cycle and the last unit of the same model in the preceding cycle. Let $\bar{t}_i$ be the average distance between two consecutive units of the model i ($\bar{t}_i = \frac{D}{d_i}$). For all symbol i in which $d_i = 1$, t_1^i is equal to $\bar{t}_i$. The objective is to minimise the metric Response Time Variability (RTV) which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2 \tag{1}$$

For example, let $n = 3$, $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\bar{t}_C = 2$. Any sequence such that contains exactly d_i times the symbol i ($\forall i$) is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a solution, where $RTV = [(5-4)^2 + (3-4)^2] + [(2-4)^2 + (6-4)^2] + [(2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2] = 2.00 + 8.00 + 2.00 = 12.00$.

3 The multi-start algorithm for solving the RTVP

The multi-start metaheuristic is a general scheme that consists of two phases. The first phase obtains an initial solution and the second phase improves the obtained initial solution. These two phases are applied iteratively until a stop condition is reached. This scheme has been first used at the beginning of 80's ([10], [11]). The generation of the initial solution, how to improve them and the stop condition can be very simple or very sophisticated. The combination of these elements gives a wide variety of multi-start methods. For a good review of multi-start methods, see [12] and [13].

The multi-start algorithm proposed in [8] for solving the RTVP is based on generating, at each iteration, a random solution and on improving it by means of a local search procedure. The algorithm stops after it has run for a preset time. Random solutions are generated as follows. For each position, a model to be sequenced is randomly chosen. The probability of each model is equal to the number of units of this model that remain to be sequenced divided by the total number of units that remain to be sequenced. The local search procedure used is applied as follows. A local search is performed iteratively in a neighbourhood that is generated by interchanging each pair of two consecutive units of the sequence that represents the current solution; the best solution in the neighbourhood is chosen; the optimisation ends when no neighbouring solution is better than the current solution.

To improve the obtained results, an improved multi-start algorithm was proposed in [9]. The difference with respect to the aforementioned multi-start algorithm consists that, at each iteration, several random solutions are generated and only the best of them is improved by means of the local search procedure.

The multi-start technique is also used in [14]. In [14] a multi-start algorithm is compared with other seven metaheuristics (Boltzmann Machine, Evolution Strategy, Genetic Algorithm, Sampling and Clustering, Simulated Annealing, Tabu Search and Immune Networks) when solving the Quadratic Assignment Problem (QAP), which is also a hard combinatorial optimisation problem. Their computational experiment shows the effectiveness of the multi-start approach versus other more complex approaches.

```
1.  Set the values of parameters $P$ and $N$
2.  Let the best solution found $\overline{X}$ initially be void
3.  Let the RTV value of the best solution found be $\overline{Z} = \infty$
4.  While execution time is not reached do:
5.          Generate $P$ random solutions
6.          Let $X_i (i = 1, \dots, P)$ the $i^{th}$ best solution generated at step 5
7.          For each $j = 1, \dots, N$ do:
8.                  Apply the local optimisation to $X_j$ and get $X_j^{opt}$
9.                  If $RTV(X_j^{opt}) < \overline{Z}$, then $\overline{X} = X_j^{opt}$ and $\overline{Z} = RTV(X_j^{opt})$
10.         End For
11. End While
12. Return $\overline{X}$
```

Fig. 1: Pseudocode of the multi-start algorithm

We propose a multi-start algorithm that is an interesting and new extension of those proposed in [8] and in [9]. The local search procedure is the same used in [8] and in [9]. The difference of our proposed algorithm is that, at each iteration, instead of generating one or more random solutions and optimising the best one, our algorithm generates P random solutions and optimises the N best of them. P and N are the two parameters of the algorithm, where $N \leq P$. The number of iterations is limited by the available execution time, as it occurs in [8] and in [9]. To the best of our knowledge, this approach has not been still tested. Note that the algorithms proposed in [8] and in [9] can be though as special cases of our more general parametric algorithm in which $P = N = 1$ and $P \geq 1 \land N = 1$, respectively. Figure 1 shows the pseudocode of our algorithm.

As it has been mentioned, when the execution time of the algorithm is reached, the algorithm is immediately stopped (that is, the current local optimisation is also stopped).

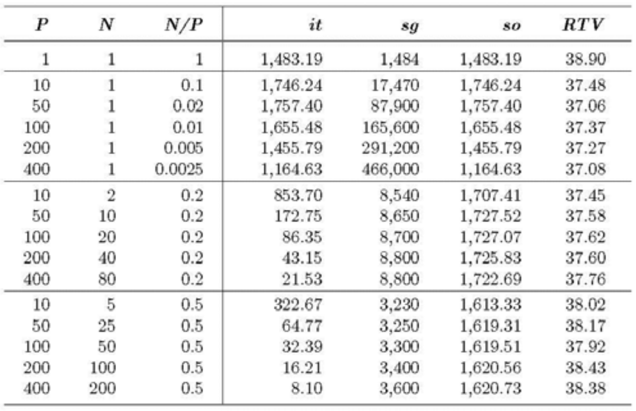
4 Computational experiment

The RTVP instances used in the computational experiment are divided in two groups according to their size. For the first group (called *G1*) 185 instances were generated using a random value of D (number of units) uniformly distributed between 50 and 100, and a random value of n (number of models) uniformly distributed between 3 and 30; for the second group (called *G2*) 185 instances were generated using a random value of D between 100 and 200 and a random value of n between 3 and 65. For all instances and for each model $i = 1, \ldots, n$, a random value of d_i (number of units of the model i) is between 1 and $\lfloor (D - n + 1)/2.5 \rfloor$ such that $\sum_{i=1}^{n} d_i = D$. The multi-start algorithm was coded in Java and the computational experiment was carried out using a 3.4 GHz Pentium IV with 1.5 GB of RAM.

For each instance, the multi-start algorithm was run once for 180 seconds with different values of P (number of random solutions generated at each iteration) and N (number of the best generated solutions optimised at each iteration). We have also used the values $P = N = 1$, which is equivalent to use the multi-start algorithm proposed in [8]. Tables 1 and 2 show the averages per instance of the number of iterations done by the algorithm (it), the total number of random solutions generated (sg), the total number of solutions optimised (so) and the averages per instance of the RTV values of the best solutions obtained for the *G1* and *G2* instances, respectively (RTV).

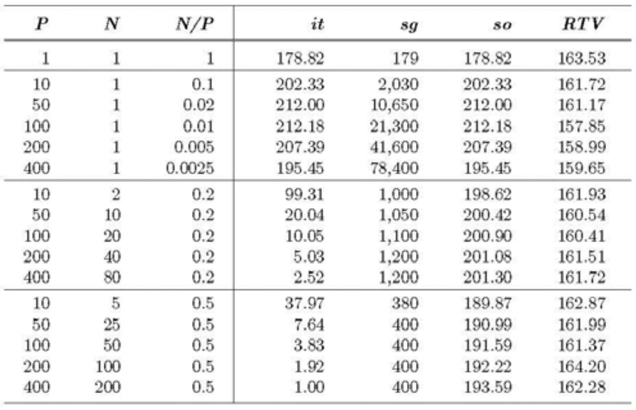
Tables 1 and 2 show that the differences between the average RTV values obtained for different P and N values do not seem enough significant. Thus, the parametric multi-start algorithm seems that is quite insensitive to the values of the parameters P and N. Anyway, note that when $P = N = 1$ or $N/P = 0.5$ worse solutions are obtained than when $N = 1$ or $N/P = 0.2$. One reason is that less solutions are optimised because the initial solutions are more probable to be worse and more time is needed in the local optimisations. Less solutions are also optimised when a considerable execution time is spent on generating random solutions ($P = 400$ and $N = 1$). Therefore, it is recommended to use a quite high value of P ($100 \leq P \leq 200$) and a value of N equal to 1 instead of using the multi-start algorithm proposed in [8] (i.e., $P = N = 1$). We have observed how many times the best initial solution of the N solutions to be optimised gives the best local optimum found by the algorithm during its running. This observation has been also done for the remaining N solutions to be optimised (i.e., the second, the third, …, the N^{th} best initial solution). The results show that all initial solutions have similar probability to be optimised to the best solution found by the algorithm. Therefore, the quality of an initial solution is not a sign of the quality of the local optimum. Although the quality of a local optimum is independent of the quality of the initial solution, notice that it is

Table 1: Averages results of the *G1* instances for 180 seconds



P	N	N/P	it	sg	so	RTV
1	1	1	1,483.19	1,484	1,483.19	38.90
10	1	0.1	1,746.24	17,470	1,746.24	37.48
50	1	0.02	1,757.40	87,900	1,757.40	37.06
100	1	0.01	1,655.48	165,600	1,655.48	37.37
200	1	0.005	1,455.79	291,200	1,455.79	37.27
400	1	0.0025	1,164.63	466,000	1,164.63	37.08
10	2	0.2	853.70	8,540	1,707.41	37.45
50	10	0.2	172.75	8,650	1,727.52	37.58
100	20	0.2	86.35	8,700	1,727.07	37.62
200	40	0.2	43.15	8,800	1,725.83	37.60
400	80	0.2	21.53	8,800	1,722.69	37.76
10	5	0.5	322.67	3,230	1,613.33	38.02
50	25	0.5	64.77	3,250	1,619.31	38.17
100	50	0.5	32.39	3,300	1,619.51	37.92
200	100	0.5	16.21	3,400	1,620.56	38.43
400	200	0.5	8.10	3,600	1,620.73	38.38

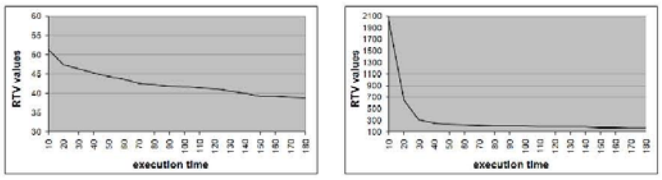
Table 2: Averages results of the *G2* instances for 180 seconds



P	N	N/P	it	sg	so	RTV
1	1	1	178.82	179	178.82	163.53
10	1	0.1	202.33	2,030	202.33	161.72
50	1	0.02	212.00	10,650	212.00	161.17
100	1	0.01	212.18	21,300	212.18	157.85
200	1	0.005	207.39	41,600	207.39	158.99
400	1	0.0025	195.45	78,400	195.45	159.65
10	2	0.2	99.31	1,000	198.62	161.93
50	10	0.2	20.04	1,050	200.42	160.54
100	20	0.2	10.05	1,100	200.90	160.41
200	40	0.2	5.03	1,200	201.08	161.51
400	80	0.2	2.52	1,200	201.30	161.72
10	5	0.5	37.97	380	189.87	162.87
50	25	0.5	7.64	400	190.99	161.99
100	50	0.5	3.83	400	191.59	161.37
200	100	0.5	1.92	400	192.22	164.20
400	200	0.5	1.00	400	193.59	162.28

Table 3: Averages of the number of times that the best solution is obtained from the i^{th} best initial solution

	P	N	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}	7^{th}	8^{th}	9^{th}	10^{th}
G1	10	2	1.40	1.28	*	*	*	*	*	*	*	*
G1	50	10	0.21	0.21	0.26	0.23	0.22	0.22	0.23	0.22	0.22	0.17
G1	10	5	0.43	0.39	0.51	0.57	0.4	*	*	*	*	*
G2	10	2	0.56	0.56	*	*	*	*	*	*	*	*
G2	50	10	0.11	0.09	0.13	0.11	0.11	0.14	0.08	0.12	0.11	0.10
G2	10	5	0.23	0.30	0.17	0.18	0.23	*	*	*	*	*



(a) For *G1* instances (b) For *G2* instances

Fig. 2: Average of the RTV values obtained during the execution time

still advisable to obtain good initial solutions because the time needed for the local optimisation is lower, as it has been explained before. Table 3 shows some examples of the number of times, on average, that the best solution found by the algorithm has been obtained from the i^{th} best initial solution.

Finally, Fig. 2 shows how the averages of the RTV values of the best obtained solutions for the *G1* and the *G2* instances decrease over the execution time for the multi-start algorithm (with $P = 100$ and $N = 1$).

5 Final conclusions

The RTVP occurs in diverse environments as manufacturing, hard real-time systems, operating systems and networks environments. The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimise variability in the time between the instants at which they receive the necessary resources. Since it is a NP-hard combinatorial optimisation problem, heuristic methods are needed for solving real-life instances.

We propose a parametric multi-start algorithm for solving a NP-hard sequencing combinatorial optimisation problem as it is the RTVP. Better solutions, on average, are obtained compared with the solutions of the multi-start algorithm proposed in [8]. The computational experiment also shows that the proposed algorithm seems that is not very sensitive to the parameters. Thus, the multi-start technique is robust. Moreover, the multi-start algorithm is very easy to design and to be implemented.

It is also shown that the best local optimum can be obtained from an initial solution independently of its fitness although, as it is expected, the worst is the initial solution, the more time is spent on the local search.

Our future research will focus on testing our proposed parametric multi-start algorithm in other combinatorial optimisation problem and using other local search methods.

References

1. Corominas, A., Kubiak, W., Moreno, N.: Response Time Variability. Journal of Scheduling 10, 97–110 (2007)
2. Monden, Y.: Toyota Production Systems Industrial Engineering and Management. Press, Norcross, GA (1983)
3. Waldspurger, C.A., Weihl, W.E.: Stride Schedulling: Deterministic Proportional-Share Resource Management. Technical Report MIT/LCS/TM-528, Massechusetts Institute of Technology, MIT Laboratory for Computer Science (1995)
4. Dong, L., Melhem, R., Mossel, D.: Time slot allocation for real-time messages with negotiable distance constraint requirements. Real-time Technology and Application Symposium, RTAS, Denver (1998)
5. Anily, S., Glass, C.A., Hassin, R.: The scheduling of maintenance service. Discrete Applied Mathematics 82, 27–42 (1998)
6. Herrmann, J.W.: Fair Sequences using Aggregation and Stride Scheduling. Technical Report, University of Maryland, USA (2007)
7. Miltenburg, J.: Level schedules for mixed-model assembly lines in just-in-time production systems. Management Science 35, 192–207 (1989)
8. García, A., Pastor, R., Corominas, A.: Solving the Response Time Variability Problem by means of metaheuristics. Artificial Intelligence Research and Development 146, 187–194 (2006)
9. Corominas, A., García, A., Pastor, R.: Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics. Artificial Intelligence Research and Development 148, 128–137 (2008)
10. Boender, C.G.E., Rinnooy, A.H.G., Stougie, L., Timmer, G.T.: A Stochastic Method for Global Optimization. Mathematical Programming 22, 125–140 (1982)
11. Los, M., Lardinois, C.: Combinatorial Programming, Statistical Optimization and the Optimal Transportation Network Problem. Transportation Research 2, 89–124 (1982)
12. Martí, R.: Multi-start methods. Handbook of Metaheuristics. Glover and Kochenberger (eds.), 355–368. Kluwer Academic Publishers (2003)
13. Hoos, H., Stützle, T.: Stochastic local research: foundations and applications. Morgan Kaufmann Publishers, San Francisco (2005)
14. Maniezzo, V., Dorigo, M., Colorni, A.: Algodesk: An experimental comparison of eight evolutionary heuristics applied to the Quadratic Assignment Problem. European Journal of Operational Research 81, 188–204 (1995)

A2.3. Communications to international congresses

Solving the Response Time Variable Problem by means of a Variable Neighbourhood Search Algorithm

In proceedings of the 13th IFAC Symposium of Information Control Problems in Manufacturing (INCOM 2009), Moscow, Russia, June 3-5, 2009.

Solving the Response Time Variable Problem by means of a Variable Neighbourhood Search Algorithm

Corominas, Albert*. García-Villoria, Alberto. Pastor, Rafael*****

**Institute of Industrial and Control Engineering, Universitat Politècnica de Catalunya, Spain; e-mail: albert.corominas@upc.edu*
***Institute of Industrial and Control Engineering, Universitat Politècnica de Catalunya, Spain; e-mail: alberto.garcia-villoria@upc.edu*
****Institute of Industrial and Control Engineering, Universitat Politècnica de Catalunya, Spain; e-mail: rafael.pastor@upc.edu*

Abstract: The Response Time Variability Problem (RTVP) is a NP-hard combinatorial scheduling problem which has recently reported and formalised in the literature. This problem has a wide range of real-world applications in mixed-model assembly lines, multi-threaded computer systems, network environments and others. The RTVP arises whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the points at which they receive the necessary resources is minimized. The best results in the literature for the RTVP were obtained with a psychoclonal algorithm. We propose a Variable Neighbourhood Search (VNS) algorithm for solving the RTVP. The computational experiment shows that, on average, the results obtained with the proposed algorithm improve strongly on the best obtained results to date.

1. INTRODUCTION

The Response Time Variability Problem (RTVP) is a combinatorial scheduling problem that has been first time reported in Waldspurger and Weihl (1994) and was first time formalised in Corominas et al. (2007). The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources. Although this combinatorial optimization problem is easy to formulate, it is NP-hard (Corominas et al., 2007).

The RTVP has a broad range of real-life applications. For example, it can be used to regularly sequence models in the automobile industry (Monden, 1983), to resource allocation in computer multi-threaded systems and network servers (Waldspurger and Weihl, 1994, 1995), to broadcast video and sound data frames of applications over asynchronous transfer mode networks (Dong et al., 1998), in the periodic machine maintenance problem when the distances between consecutive services of the same machine are equal (Anily et al., 1998) and in the collection of waste (Herrmann, 2007).

One of the first problems in which has appeared the importance of sequencing *regularly* is at the sequencing on the mixed-model assembly production lines at Toyota Motor Corporation under the just-in-time (JIT) production system. One of the most important JIT objectives is to get rid of all kinds of waste and inefficiency and, according to Toyota, the main waste is due to the stocks. To reduce the stock, JIT production systems require to producing only the necessary models in the necessary quantities at the necessary time. To achieve this, one main goal, as Monden (1983) says, is scheduling the units to be produced to keep constant consumption rates of the components involved in the production process. Miltenburg (1989) deals with this scheduling problem and assumes that models require approximately the same number and mix of parts. Thus, only the demand rates for the models are considered. In our experience with practitioners of manufacturing industries, we noticed that they usually refer to a good mixed-model sequence in terms of having distances between the units for the same model as regular as possible. Therefore, the metric used in the RTVP reflects the way in which practitioners refer to a desirable regular sequence

Corominas et al. (2007) proposed a mixed integer linear programming (MILP) model to solve the RTVP. Corominas et al. (2009) proposed an improved MILP model and increased the practical limit for obtaining optimal solutions from 25 to 40 units to be scheduled. Thus, the use of heuristic or metaheuristic methods for solving real-life instances of the RTVP is justified. Waldspurger and Weihl (1995) used the Jefferson method of apportionment (Balinski and Young, 1982), a greedy heuristic algorithm which they renamed as the stride scheduling technique. Herrmann (2007) solved the RTVP by applying a heuristic algorithm based on the stride scheduling technique. Corominas et al. (2007) proposed four other greedy heuristic algorithms. García et al. (2006) proposed six metaheuristic algorithms: a multi-start, a greedy randomized adaptive search procedure (GRASP) and four variants of a discrete particle swarm optimization (PSO) algorithm. Other ten discrete PSO algorithms were proposed in García-Villoria and Pastor (2007). A cross-entropy method approach was used in García-Villoria et al. (2007). The Electromagnetism-like Mechanism (EM) was proposed to solve the RTVP in García-Villoria and Pastor (2008a). Finally, the best results recorded to date have been obtained with a Psychoclonal algorithm (García-Villoria and Pastor, 2008b).

To improve the results obtained in prior studies, we propose to use a Variable Neighbourhood Search (VNS)-based algorithm for solving the RTVP. VNS is a metaheuristic used to solve combinatorial optimization problems (Mladenović and Hansen, 1997), as it is the RTVP. This metaheuristic is based on changing systematically the neighbourhood during a local search. The proposed VNS algorithm is compared with the most efficient procedure for solving non-small instances published in the literature, which is a psychoclonal algorithm proposed in García-Villoria and Pastor (2008b). On average, the proposed VNS algorithm improves more than 61% on the best previous results reported in the literature.

The remainder of the paper is organized as follows: Section 2 presents a formal definition of the RTVP and describes briefly the psychoclonal algorithm used for solving the problem. Section 3 proposes a VNS algorithm for solving the RTVP. Section 4 presents the computational experiment and the comparison between our algorithm and the psychoclonal algorithm. Finally, the conclusions are given in Section 5.

2. THE RESPONSE TIME VARIABILITY PROBLEM

The RTVP is designed to minimize variability in the distances between any two consecutive units of the same model and is formulated as follows. Let n be the number of models, d_i the number of units of model i to be scheduled (i = 1,…,n), and D the total number of units ($D = \sum_{i=1..n} d_i$). Let s be a solution of an instance in the RTVP. It consists in a circular sequence of units ($s = s_1 s_2 ... s_D$), where s_j is the unit sequenced in position j of sequence s. For each model i in which $d_i \geq 2$, let t_k^i be the distance between the positions in which units k + 1 and k of model i are found. We consider the distance between two consecutive positions to be equal to 1. Since the sequence is circular, position 1 comes immediately after position D; therefore, $t_{d_i}^i$ is the distance between the first unit of model i in a cycle and the last unit of the same model in the preceding cycle. Let $\overline{t_i}$ be the average distance between two consecutive units of model i ($\overline{t_i} = D/d_i$). Note that for each model i in which $d_i = 1$, t_1^i is equal to $\overline{t_i}$. The aim is to minimize the metric response time variability (RTV) which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} \left(t_k^i - \overline{t_i} \right)^2. \tag{1}$$

For example, let n = 3, d_A = 2, d_B = 2 and d_C = 4; thus, D = 8, $\bar{t}_A = 4$, $\bar{t}_B = 4$ and $\bar{t}_C = 2$. Any sequence that contains model i $(\forall i)$ exactly d_i times is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a feasible solution, where:

$$RTV = \left((5-4)^2 + (3-4)^2\right) + \left((2-4)^2 + (6-4)^2\right) + \left((2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2\right) = 1\quad 2\quad .$$

As has been introduced in Section 1, the psychoclonal algorithm proposed in García-Villoria and Pastor (2008b) is the best procedure to date for solving the RTVP. Psychoclonal is an evolutionary metaheuristic first time proposed in Tiwari et al. (2005). According to the authors, this metaheuristic inherits its characteristics from the need hierarchy theory of Maslow (1954) and the clonal selection principle (Gaspar and Collard, 2000). The basic scheme of the psychoclonal metaheuristic is the following: 1) An initial population of solutions is generated and a function to evaluate the fitness of a solution is given; 2) The best solutions are selected and cloned in a number proportional to their fitness; 3) The generated clones are hypermutated (hypermutation is an operator that modifies the solution with a rate inversely proportional to the fitness of the solution); 4) A new population is formed by the best clones and by new solutions generated at random; 5) Steps 2-4 are repeated until a stop condition is reached. This metaheuristic was adapted to solve the RTVP (for a more detailed explanation, see García-Villoria and Pastor, 2008b).

3. A VNS ALGORITHM FOR SOLVING THE RTVP

Variable Neighbourhood Search (VNS) is a metaheuristic proposed recently in Mladenović and Hansen (1997) for combinatorial optimization. The basic idea of VNS is applying a systematic change of neighbourhood within a local search method (Mladenović and Hansen, 1997). According to the strategies used in changing neighbourhoods and in selecting the neighbour to be the current solution, several extensions have been proposed, but most of them keep the simplicity of the basic idea (Mladenović et al., 2003). VNS is based on the following three simple facts (Hansen and Mladenović, 2003): 1) a local minimum with respect to one neighbourhood structure is not necessarily so with another, 2) a global minimum is a local minimum with respect to all possible neighbourhood structures, and 3) It have been observed empirically that for many problems local minima with respect to one or several neighbourhood structures are relatively close to each other.

In the basic VNS proposed in (Mladenović and Hansen, 1997) there is a local search step, which can be costly for large instances of some problems (Hansen and Mladenović, 2003). In Hansen and Mladenović (1998) is proposed the Reduced VNS (RVNS), in which the local search step is removed. In this paper we propose a RVNS-based algorithm for solving the RTVP because it is shown in García et al. (2006) that the local search proposed in their paper for large RTVP instances is very costly. The general scheme of RVNS is shown in Fig. 1.

```
1.  Select the set of neighbourhood structures $N_k$
    ($k$=1..$k_{max}$), where $k_{max}$ is the cardinality of the set
2.  Let $S$ an initial solution
3.  While stopping condition is not reached do:
4.      Set $k$ = 1
5.      While $k \leq k_{max}$ do:
6.          Select a solution $S'$ at random from $N_k(S)$
7.          If the acceptance criteria is satisfied, then set $S$
              = $S'$ and set $k$ = 1; otherwise set $k$ = $k$ + 1
8.      End While
9.  End While
10. Return $S$
```

Fig. 1. General scheme of RVNS

For the proposed RVNS algorithm, we have selected the following three neighbourhood structures: 1) interchanging each pair of two consecutive units of the sequence that represents the current solution (N_1), 2) interchanging each

pair of consecutive or no-consecutive units of the sequence (N_2), and 3) inserting each unit in each position of the sequence (N_3). Note that all local optima with respect N_2 are always local optima with respect N_1 because the neighbourhood of a solution *S* with respect to N_1 is a subset of the neighbourhood of *S* with respect to N_2. Therefore, if there is not a neighbour of *S* with respect to N_2 that is better than *S*, there is not either a neighbour of *S* with respect to N_1 better than *S*. Thus, it seems that the first neighbourhood is unnecessary according to the aforementioned first and second facts in which are based VNS. To justify the addition of this neighbourhood, Section 4 will show the benefits of adding N_1 to our RNVS algorithm. The initial RTVP solution is generated as in the psychoclonal algorithm (García-Villoria and Pastor, 2008b). That is, for each position, a model to be sequenced is randomly chosen. The probability of each model is equal to the number of units of this model that remain to be sequenced divided by the total number of units that remain to be sequenced. The stopping condition of the algorithm is a preset time run. The original acceptance criteria used in Hansen and Mladenović (1998) is that the neighbour solution *S*' was better than the current solution *S*. But the chosen acceptance criteria for our algorithm is that the neighbour solution *S*' was better than or equal to the current solution *S*, as it is done in Tasgetiren et al. (2007). Its aim is to facilitate escaping from local optima.

4. COMPUTATIONAL EXPERIMENT

The psychoclonal algorithm proposed in García-Villoria and Pastor (2008b) is the most efficient algorithm in the literature for solving non-small RTVP instances. Therefore, we compared the performance of our proposed RVNS algorithm with that psychoclonal algorithm. In what follows of this section, we refer to our RVNS algorithm as $\text{RVNS}_{(1,2,3)}$ and the psychoclonal algorithm as Psycho. In order to justify the use of the neighbourhood N_1, we run also a RVNS algorithm without this neighbourhood structure (i.e., only N_2 and N_3 are used); we refer to this algorithm as $\text{RVNS}_{(2,3)}$.

The computational experiment was carried out for the same instances and conditions that were used in García-Villoria and Pastor (2008b). That is, the algorithms were run for 740 instances which were grouped into four classes (185 instances in each class) according to size. The instances in the first class (*CAT1*) were generated using a random value of *D* (number of units) distributed uniformly between 25 and 50, and a random value of *n* (number of models) distributed uniformly between 3 and 15; for the second class (*CAT2*), *D* was between 50 and 100 and *n* between 3 and 30; for the third class (*CAT3*), *D* was between 100 and 200 and *n* between 3 and 65; and for the fourth class (*CAT4*), *D* was between 200 and 500 and *n* between 3 and 150. For all instances and for each unit *i* = 1,…,*n*, a random value of d_i (number of units of model *i*) was between 1 and $\left|(D-n+1)/2.5\right|$ so that $\sum_{i=1..n} d_i = D$. The two algorithms were coded in Java and the computational experiment was carried out using a 3.4 GHz Pentium IV with 1.5 GB of RAM.

All algorithms were run for 50 seconds for each instance. Table 1 shows the averages of the RTV values to be minimized for the total of 740 instances and for each class of instances (*CAT1* to *CAT4*) obtained with the algorithms.

Table 1. Average RTV values for 50 seconds

	$\text{RVNS}_{(1,2,3)}$	$\text{RVNS}_{(2,3)}$	Psycho
Total	63.96	86.78	235.68
CAT1	10.73	10.63	14.92
CAT2	23.69	23.23	44.25
CAT3	51.80	53.39	137.07
CAT4	169.64	259.86	746.50

Table 1 shows that our proposed algorithm $\text{RVNS}_{(1,2,3)}$ is, on average,72.86% better than the results obtained using the best method proposed in the literature. Moreover, for each type of class of instances, the $\text{RVNS}_{(1,2,3)}$ algorithm always obtains better results than Psycho: 28.08%, 46.46%, 62.21% and 77.28% for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. We can see that the larger is the RTVP instance (and, therefore, harder to be solved), better is $\text{RVNS}_{(1,2,3)}$ compared with Psycho. Comparing $\text{RVNS}_{(1,2,3)}$ with $\text{RVNS}_{(2,3)}$, it is observed in Table 1 that very similar results are obtained for the small and medium instances (*CAT1*, *CAT2* and *CAT3* instances); on the other hand, an improvement of 34.72% is obtained for the largest instances (*CAT4* instances) when the neighbourhood N_1 is used.

Table 2 shows the number of times that each algorithm reaches the best RTV value obtained by either one. The results are shown for the total number of 740 instances and for each class.

Table 2. Number of times that the best solution is reached for 50 seconds

	RVNS$_{(1,2,3)}$	RVNS$_{(2,3)}$	Psycho
Total	587	443	57
CAT1	162	168	51
CAT2	140	144	6
CAT3	124	94	0
CAT4	161	37	0

As expected from the results in Table 1, Table 2 shows that RVNS$_{(1,2,3)}$ reaches the best solution more times. For the total number of instances, the best solution is obtained in 79.32%, 59.86% and 7.7% of cases by RVNS$_{(1,2,3)}$, RVNS$_{(2,3)}$ and Psycho, respectively.

To complete the analysis of the results, their dispersion is observed. A measure of the dispersion (let it be called σ) of the RTV values obtained by each algorithm alg = { RVNS$_{(1,2,3)}$, RVNS$_{(2,3)}$, Psycho } for a given instance, *ins*, is defined as follows:

$$\sigma(alg, ins) = \left(\frac{RTV_{ins}^{(alg)} - RTV_{ins}^{(best)}}{RTV_{ins}^{(best)}} \right)^2 \tag{2}$$

where $RTV_{ins}^{(alg)}$ is the RTV value of the solution obtained with the algorithm *alg* for the instance *ins*, and $RTV_{ins}^{(best)}$ is, for the instance *ins*, the best RTV value of the solutions obtained with the three algorithms. Table 3 shows the average σ dispersion for the total of 740 instances and for each class of instances. The low dispersion of the two RVNS algorithms for all classes of instances means that both algorithms have a very stable behaviour. That is, when an RVNS algorithm does not obtain the best RTV value for a given instance, it obtains a value that is close to it. Psycho-RTVP has a quite stable behaviour, but its dispersion is much bigger than the dispersion of the RVNS algorithms because the Psycho performance is worse.

Table 3. Average σ dispersion regarding the best solution found for 50 seconds

	RVNS$_{(1,2,3)}$	RVNS$_{(2,3)}$	Psycho
Total	0.018	0.162	8.059
CAT1	0.030	0.020	1.003
CAT2	0.024	0.009	1.748
CAT3	0.015	0.029	5.442
CAT4	0.004	0.592	24.043

The difference of the results obtained with the three algorithms may be due to that 50 seconds is not time enough for the convergence of the algorithms for all instances, especially the largest ones. Fig. 2 shows that 1,000 computing seconds seems long enough for all algorithms to converge.

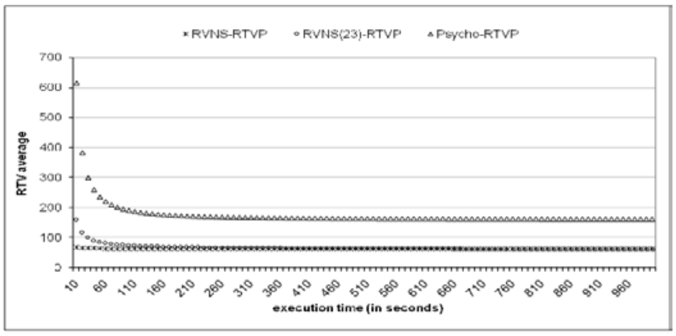


Fig. 2. Average RTV values over the computing time

Tables 4 and 5 shows the average RTV values and the average σ dispersion, respectively, for the total of 740 instances and for each class of instances obtained for 1,000 seconds.

Table 4. Average RTV values for 1,000 seconds

	$RVNS_{(1,2,3)}$	$RVNS_{(2,3)}$	Psycho
Total	62.24	62.06	161.60
CAT1	10.73	10.63	14.90
CAT2	23.29	23.19	39.90
CAT3	51.40	51.46	122.38
CAT4	163.15	162.95	469.23

Table 5. Average σ dispersion regarding the best solution found for 1,000 seconds

	RVNS	$RVNS_{(2,3)}$	Psycho
Total	0.026	0.019	4.100
CAT1	0.030	0.020	0.994
CAT2	0.024	0.008	1.256
CAT3	0.024	0.024	3.984
CAT4	0.026	0.026	10.166

Using 1,000 seconds of computing time, Psycho improves its average RTV value a 31.43% regarding the values obtained with 50 computing seconds. Nevertheless, $RVNS_{(1,2,3)}$ is still 61.49% better on average for all instances than Psycho, and 27.99%, 41.63%, 58.00% and 65.23% better for *CAT1*, *CAT2*, *CAT3* and *CAT4* instances, respectively. Moreover, we can see in Table 5 that $RVNS_{(1,2,3)}$ still obtains the best solutions or solutions very close to the best. Note that 50 seconds is almost enough time for $RVNS_{(1,2,3)}$ to converge, since it improves, on average, only 2.69% with 1,000 computing seconds.

Comparing $RVNS_{(1,2,3)}$ versus $RVNS_{(2,3)}$ for the total of all instances and for each class of instances, we can see in Table 4 and 5 that there are not significant differences between the quality of the solutions obtained with both algorithms. We expected that $RVNS_{(1,2,3)}$ and $RVNS_{(2,3)}$ give similar results when both algorithms have time to converge. The reason is that the only difference between the two algorithms is that the neighbourhood structure N_1 is not included in $RVNS_{(2,3)}$ but, as it has been explained in Section 3, all local optima with respect the neighbourhood structure N_2 (used in both algorithms) are always local optima with respect N_1, that is, N_1 is *dominated* by N_2.

Thus, the great advantage of using N_1 in $RVNS_{(1,2,3)}$ is that it helps to the algorithm to converge very fast without detrimental of its performance. This is very useful for large instances or when little computational time is available. For example, $RVNS_{(1,2,3)}$ obtains an average RTVP value for the largest instances (*CAT4*) with 10 seconds equal to 187.07, whereas the average value obtained with $RVNS_{(2,3)}$ for *CAT4* instances with 10 seconds is 550.50. The reason is because, at the beginning of the search, it is easier to find a neighbour better than the current solution using the neighbourhood structure N_1 instead of N_2. To demonstrate that, we run two times the VNS algorithm for 5 seconds for all 185 *CAT4* instances. The first time only N_1 was used ($RVNS_{(1)}$); the second time only N_2 was used ($RVNS_{(2)}$). During the 5 seconds, $RVNS_{(1)}$ generated, on average, for the total of *CAT4* instances 134,112.25 solutions, where 2.67% (3,578.74), 16.32% (21,881.05) and 81.02% (108,652.46) were better, equal and worse than the current solution, respectively. On the other hand, $RVNS_{(2)}$ generated, on average, for the total of CAT4 instances 145,364.11 solutions, where 0.42% (604.86), 6.43% (9,343.08) and 93.16% (135,416.17) were better, equal and worse than the current solution, respectively.

Finally, we compare the MILP model proposed by Corominas et al. (2009) with our $RVNS_{(1,2,3)}$ algorithm and with the psychoclonal algorithm. Corominas et al. (2009) solved 60 small RTVP instances, with a D value of between 20 and 40 and a p value of between 3 and 15, with the MILP model. We have repeated the experiment by setting the maximum execution time at 2,000 seconds and 55 instances were solved optimally. The results obtained are shown in Table 6.

Table 6. Averages of the RTV values and the execution time (in seconds)

	MILP	$RVNS_{(1,2,3)}$		Psychoclonal	
RTV	9.86	10.06	10.06	14.49	12.49
Time	188.19	0.1	10	0.1	10

Table 6 shows that RVNS$_{(1,2,3)}$ is able to converge very quickly to near optimal solutions for small instances. With only 0.1 seconds of computing time, the quality of the solutions obtained with RVNS$_{(1,2,3)}$ is very close to that obtained with MILP (only 1.99% worse). On the other hand, the psychoclonal algorithm needs 10 seconds to obtain solutions that are, on average, 21.06% worse than those from MILP.

5. CONCLUSIONS

The Response Time Variability Problem is a scheduling problem that has been acquiring a greater importance on the mixed-model assembly production lines since Toyota popularized the just-in-time production system (Monden, 1983; Miltenburg, 1989). RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources. Other real-life applications of the RTVP shown in the literature are present in computer multi-threaded systems and network servers (Waldspurger and Weihl, 1994, 1995; Dong et al., 1998), in periodic machine maintenances (Anily et al., 1998) and in the collection of waste (Herrmann, 2007).

The computational experiment shows the following two points:

1. A straightforward implementation of an algorithm based on the simple metaheuristic RVNS improves strongly all the methods published in the literature, including also the algorithms based on more complex metaheuristics as Particle Swarm Optimization (García et al., 2006; García-Villoria and Pastor, 2007), Cross-Entropy method (García-Villoria et al., 2007), Electromagnetism-like Mechanism (García-Villoria and Pastor, 2008a) and Psychoclonal approach (García-Villoria and Pastor, 2008b).
2. The addition of the dominated neighbourhood structure N_1 in our RVNS algorithm makes it to converge faster to solutions of good quality. This observation may be extended to other problems and VNS algorithms, in which the addition of dominated neighbourhood structures can help them to be more efficient.

The VNS metaheuristic is very easy to be hybridized with any another metaheuristic. Since the good results obtained in the literature (Hansen and Mladenović, 2003), the hybridization of VNS with other metaheuristics proposed in the literature to solve the RTVP as PSO (García-Villoria and Pastor, 2007), EM (García-Villoria and Pastor, 2008a) or Psychoclonal (García-Villoria and Pastor, 2008b) seems a promising future line of research.

ACKNOWLEDGEMENTS

The authors are grateful to the anonymous reviews for his valuable comments which have helped to enhance this paper. This paper was supported by the Spanish Ministry of Education and Science under project DPI2007-61905 and co-funded by the European Regional Development Fund (ERDF).

REFERENCES

S. Anily, C.A. Glass and R. Hassin (1998). The scheduling of maintenance service. *Discrete Applied Mathematics,* **82**, 27-42.

M.L. Balinski, and H.P. Young (1982). Fair Representation. Yale University, USA.

A. Corominas, W. Kubiak and N. Moreno (2007). Response time variability. *Journal of Scheduling,* **10**, 97-110.

A. Corominas, W. Kubiak and R. Pastor (2009). Mathematical Programming Modeling of the Response Time Variability Problem. *European Journal of Operational Research.* doi: 10.1016/j.ejor.2009.01.014

L. Dong, R. Melhem and D. Mosse (1998). Time slot allocation for real-time messages with negotiable distance constrains requirements. *Fourth IEEE Real-Time Technology and Applications Symposium,* 131-136. Denver.

A. García, R. Pastor and A. Corominas (2006). Solving the Response Time Variability Problem by means of metaheuristics. *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development,* **146**, 187-194.

A. García-Villoria and R. Pastor (2007). Introducing dynamic diversity into a discrete particle swarm optimization. *Computers & Operations Research,* In Press, Corrected Proof, available online, doi:10.1016/j.cor.2007.12.001.

A. García-Villoria, R. Pastor and A. Corominas (2007). Solving the Response Time Variability Problem by means of the Cross-Entropy Method. *Special Issue on Production Line Systems: Concepts, Methods and Applications of the International Journal of Manufacturing Technology and Management,* In Press, Corrected Proof, to be published.

A. García-Villoria and R. Pastor (2008a). Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism. *Working paper IOC-DT-P-2008-03,* Technical University of Catalonia, Spain.

A. García-Villoria and R. Pastor (2008b). Solving the Response Time Variability Problem by means of a psychoclonal approach. *Journal of Heuristics,* In Press, Corrected Proof, available online, doi:10.1007/s10732-008-9082-2.

A. Gaspar and P. Collard (2000). Two models of immunization for time dependent optimization. In: *Proceeding of the IEEE International Conference on Systems Manufacturing and Cybernetics*, 113-118.

P. Hansen and N. Mladenović (1998). *Meta-heuristics, Advances and Trends in Local Search Paradigms for Optimization*, 433-458. Kluwer Academic Publishers.

P. Hansen and N. Mladenović (2003). *Handbook of metaheuristics*, Chapter 6. Kluwer Academic Publishers.

J.W. Herrmann (2007). Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling. *Technical Report,* University of Maryland. USA.

A.H. Maslow (1954). Motivation and personality. Harper & Bros, USA.

N. Mladenović and P. Hansen (1997). Variable neighbourhood search. *Computers & Operations Research,* **24**, 1097-1100.

N. Mladenović, J. Petrović, V. Kovačević-Vujčić and M. Čangalović (2003). Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. *European Journal of Operational Research,* **151**, 389-399.

J. Miltenburg (1989). Level schedules for mixed-model assembly lines in just-in-time production systems. *Management Science,* **35**, 192-207.

Y. Monden (1983). *Toyota Productions Systems*. Industrial Engineering and Management Press. Norcross.

M.F. Tasgetiren, Y-C. Liang, M. Sevkli and G. Gencyilmaz (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research,* **177**, 1930-1947.

M.K. Tiwari, A. Prakash, A. Kumar and A.R. Mileham (2005). Determination of an optimal sequence using the psychoclonal algorithm. *ImechE, Part B: Journal of Engineering Manufacture,* **219**, 137-149.

C.A. Waldspurger and W.E. Weihl (1995). Stride Scheduling: Deterministic Proportional-Share Resource Management. *Technical Report MIT/LCS/TM-528,* Massachusetts Institute of Technology. USA.

C.A. Waldspurger and W.E. Weihl (1994). Lottery Scheduling: Flexible Proportional-Share Resource Management. *First USENIX Symposium on Operating System Design and Implementation*.

Using Tabu Search for the Response Time Variability Problem

In proceedings of the 3rd International Conference on Industrial Engineering and Industrial Management (CIO 2009), Ed. IOS Press, ISBN 1-58603-925-7, Barcelona and Terrassa, Spain, September 2-4, 2009.

Using Tabu Search for the Response Time Variability Problem[*]

Albert Corominas[1], Alberto García-Villoria[1], Rafael Pastor[1]

[1] Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya, Av. Diagonal, 647, 08028. Barcelona, Spain. albert.corominas@upc.edu, alberto.garcia-villoria@upc.edu, rafael.pastor@upc.edu

Keywords: response time variability, tabu search, scheduling, regular sequences

1. Introduction

The concept of *fair sequence* has emerged independently from scheduling problems of diverse environments. The common aim of these scheduling problems, as defined in Kubiak (2004), is to build a fair sequence using n symbols, where symbol i ($i = 1,...,n$) must occur d_i times in the sequence. The fair sequence is the one which allocates a fair share of positions to each symbol i in any subsequence. This fair or ideal share of positions allocated to symbol i in a subsequence of length k is proportional to the relative importance (d_i) of symbol i with respect to the total copies of competing symbols (equal to $\sum_{i=1..n} d_i$). There is no a universal definition of fairness because several reasonable metrics can be defined according to the specific problem considered.

Among the different definitions of fairness, several fair sequencing problems have emerged, among them the Response Time Variability Problem (RTVP). This problem has been first time reported in Waldspurger and Weihl (1994) and originally formalised in Corominas et al. (2007). In the RTVP, the fair sequence is the one which minimises the sum of the variability in the distances between any two consecutive copies of the same symbol. In other words, the distance between any two consecutive copies of the same symbol should be as regular as possible (ideally constant).

The RTVP arises whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources (Corominas et al., 2007). This problem has a broad range of real-world applications. These include, for instance, the sequencing on mixed-model assembly lines under JIT (Kubiak, 1993; Miltenburg, 1989), the resource allocation in computer multi-threaded systems such as operating systems, network servers and media-based applications (Dong et al., 1998; Waldspurger and Weihl, 1995), the periodic machine maintenance problem when the times between consecutive services of the same machine are equal (Anily et al., 1998; Wei and Liu, 1983), the collection of waste

[*] Supported by the Spanish Ministry of Education and Science under project DPI2007-61905; co-funded by the ERDF.

(Herrmann, 2007) and the schedule of commercial videotapes for television (Bollapragada et al., 2004; Brusco, 2008).

The RTVP is NP-hard (Corominas et al., 2007). Since this problem is a difficult combinatorial optimisation problem, several heuristic and metaheuristic algorithms has been proposed in the literature to solve it. Waldspurger and Weihl (1995) used the Jefferson method of apportionment (Balinski and Young, 1982), a greedy heuristic algorithm which they renamed as the stride scheduling technique. Herrmann (2007) solved the RTVP by applying a heuristic algorithm based on the stride scheduling technique. Corominas et al. (2007) proposed also the Jefferson method together with other four greedy heuristic algorithms. García et al. (2006) proposed six metaheuristic algorithms: a multi-start, a greedy randomized adaptive search procedure (GRASP) and four variants of a discrete particle swarm optimization (PSO) algorithm. An enhanced multi-start algorithm and an enhanced GRASP algorithm were proposed in Corominas et al. (2008), and other ten discrete PSO algorithms were proposed in García-Villoria and Pastor (2009a). A cross-entropy method approach was used in García-Villoria et al. (2007) and a psychoclonal algorithm was used to solve the RTVP in García-Villoria and Pastor (2008). Finally, an algorithm based on Electromagnetism-like Mechanism (EM) was proposed in García-Villoria and Pastor (2009b). The best results recorded to date have been obtained with the psychoclonal algorithm (García-Villoria and Pastor, 2008) and the enhanced multi-start algorithm (Corominas et al., 2008).

To date, no tabu search (TS) approach has been proposed to solve the RTVP. In this study we propose a TS algorithm for the RTVP which improves the best results reported in the literature.

The remainder of the paper is organized as follows: Section 2 presents a formal definition of the RTVP and describes briefly the two best algorithms up to now for solving the problem. Section 3 proposes a TS algorithm to solve the RTVP. Section 4 presents the results of a computational experiment. Finally, some conclusions and suggestions for future research are given in Section 5.

2. The Response Time Variability Problem

The RTVP is formulated as follows. Let n be the number of symbols, d_i the number of copies to be sequenced of symbol i (i = 1,…,n) and D the total number of copies ($\sum_{i=1..n} d_i$). Let s be a solution of an instance in the RTVP that consists of a circular sequence of copies ($s = s_1 s_2 \dots s_D$), where s_j is the copy sequenced in position j of sequence s. For each symbol i in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the copies k + 1 and k of symbol i are found. We consider the distance between two consecutive positions to be equal to 1. Since the sequence is circular, position 1 comes immediately after position D; therefore, $t_{d_i}^i$ is the distance between the first copy of symbol i in a cycle and the last copy of the same symbol in the preceding cycle. Let $\overline{t_i}$ be the desired average distance between two consecutive copies of symbol i ($\overline{t_i} = D / d_i$). The objective is to minimise the metric called response time variability (RTV), which is defined by the sum of the square errors with respect to the

$\overline{t_i}$ distances. Since the symbols *i* such that $d_i = 1$ do not intervene in the computation of RTV, we assume that for each of these symbols t_1^i is equal to $\overline{t_i}$. Thus, RTV is given by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} \left(t_k^i - \overline{t_i} \right)^2 . \qquad (1)$$

For example, let $n = 3$ with symbols A, B and C. Also consider $d_A = 2$, $d_B = 2$ and $d_C = 4$; thus, $D = 8$, $\overline{t_A} = 4$, $\overline{t_B} = 4$ and $\overline{t_C} = 2$. Any sequence such that contains symbol *i* ($\forall i$) exactly d_i times is a feasible solution. For example, the sequence (C, A, C, B, C, B, A, C) is a feasible solution, and has an $RTV = \left((5-4)^2 + (3-4)^2 \right) + \left((2-4)^2 + (6-4)^2 \right) + \left((2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2 \right)$ = 12.

As it has been introduced in Section 1, the psychoclonal algorithm proposed in García-Villoria and Pastor (2008) and the multi-start algorithm proposed in Corominas et al. (2008) are the best procedures to date to solve the RTVP.

Psychoclonal is an evolutionary metaheuristic first time proposed in Tiwari et al. (2005). According to the authors, this metaheuristic inherits its characteristics from the need hierarchy theory of Maslow (1954) and the clonal selection principle (Gaspar and Collard, 2000). The basic scheme of the psychoclonal metaheuristic is the following: 1) An initial population of solutions is generated and a function to evaluate the fitness of a solution is given; 2) The best solutions are selected and cloned in a number proportional to their fitness; 3) The generated clones are hypermutated (hypermutation is an operator that modifies the solution with a rate inversely proportional to the fitness of the solution); 4) A new population is formed by the best clones and by new solutions generated at random; 5) Steps 2-4 are repeated until a stop condition is reached. This metaheuristic was adapted to solve the RTVP (for a more detailed explanation, see García-Villoria and Pastor, 2008).

The general scheme of the multi-start metaheuristic consists of two phases. In the first phase an initial solution is generated. Then, the second phase improves the obtained initial solution. These two phases are iteratively applied until a stop condition is reached. Thus, the multi-start algorithm proposed in Corominas et al. (2008) to solve the RTVP consists of, at each iteration, generating an initial solution by a random mechanism (first phase) and then applying it a local search (second phase); the stop condition consists in reaching a given computing time (for a more detailed explanation, see Corominas et al., 2008)

3. A Tabu Search algorithm to solve the RTVP

Local search methods have the great disadvantage that the local optimum found is often a fairly mediocre solution (Gendreau, 2003). To overcome this limitation, the Tabu Search metaheuristic (TS) has been proposed by Glover (1986). TS is based on applying a local search in which non-improving movements are allowed. To avoid cycling back to visited solutions, the most recent history of the search is recorded in a tabu list of tabu (forbidden) solutions. The complete tabu solutions could be recorded in the tabu list, but this may require a lot of memory, make it expensive to check whether a solution is tabu or not and, above all, does not diversify sufficiently the search. Thus, it is common to record only the last moves (transformations) performed on the current solution and forbidding reverse transformations (Gendreau, 2003). The tabu lists are usually implemented as a list of fixed length with a FIFO (First In, First Out) policy. A tabu solution can be overridden if a suitable *aspiration criterion* is met. The general scheme of TS is show in Figure 1.

```
1. Define the neighbourhood structure $N$
2. Let $S$ an initial solution and $S^* := S$
3. While stopping condition is not reached do:
4.     Let $S'$ the best solution from $N(S)$ which is non-tabu or allowed by aspiration
5.     If $S'$ is better than $S^*$, then $S^* := S'$
6.     Add the current move in the tabu list (removing its last move if the list is full)
7.     $S := S'$
8. End While
9. Return $S^*$
```

Figure 1. General scheme of TS

We propose an algorithm based on the general scheme of TS to solve the RTVP. The elements of the proposed TS algorithm are defined as follows:

– *Initial solution*. A solution is represented by the sequence of the copies of the symbols to be sequenced. The initial solution is obtained from the best solution returned by the five heuristics proposed in Corominas et al. (2007).

– *Neighbourhood*. The neighbourhood of a solution is obtained by swapping each pair of consecutive or non-consecutive positions of the sequence that represents the solution.

– *Tabu moves*. A forbidden move of the tabu list consists of two pairs of position/symbol. For instance, the move [(3, A), (5, B)] means that all solutions with the symbol A sequenced in position 3 and the symbol B sequenced in position 5 are considered tabu.

– *Aspiration criterion*. The aspiration criterion is that the move produces a solution better than the best solution found in the past.

– *Stopping condition*. The TS algorithm stops once it has run for a preset time.

The TS algorithm has only one parameter whose value has to be set and it is the size of the tabu list. Although the parameter values are extremely important because the results of the metaheuristic for each problem are very sensitive to them, the selection of parameter values is commonly justified in one of the following ways (Eiben et al., 1999; Adenso-Díaz and Laguna, 2006): 1) "by hand" on the basis of a small number of experiments that are not specifically referenced; 2) by using the general values recommended for a wide range of problems; 3) by using the values reported to be effective in other similar problems; or 4) by choosing values without any explanation.

Adenso-Díaz and Laguna (2006) proposed a new technique called CALIBRA specifically designed for fine-tuning the parameters of heuristic and metaheuristic algorithms. CALIBRA was used in García-Villoria and Pastor (2008) and in Corominas et al. (2008) to set the parameter values of the psychoclonal and the multi-start algorithms, respectively. We used also CALIBRA to set the size of the tabu list of our TS algorithm. CALIBRA was applied to a training set. The training set has 60 instances which were generated as explained in the introduction of Section 4. The optimal value of the size tabu list returned by CALIBRA was 75.

4. Computational experiment

The psychoclonal and the multi-start algorithms proposed in García-Villoria and Pastor (2008) and in Corominas et al. (2008), respectively, are the most efficient algorithms in the literature to solve the RTVP. Therefore, we compare the performance of our proposed TS algorithm with these two algorithms. In what follows of this section, we refer to our TS algorithm as *TS*, the psychoclonal algorithm as *Psycho* and the multi-start algorithm as *MS*.

All algorithms are coded in Java and executed on a 3.4 GHz Pentium IV with 1.5 GB of RAM. The same 60 training instances and 740 test instances used in García-Villoria and Pastor (2008) and in Corominas et al. (2008) are also used in this paper. These instances were grouped into four classes (from *CAT1* to *CAT4* with 15 training instances and 185 test instances in each class) according to their size. The instances were generated using the random values of D (total number of copies) and n (number of symbols) shown in Table 1. For all instances and for each symbol $i = 1,\ldots,n$, a random value of d_i (number of copies to be sequenced of model i) is between 1 and $\left|(D-n+1)/2.5\right|$ such that $\sum_{i=1..n} d_i = D$.

Table 1. Uniform distribution for generating the *D* and *n* values

	CAT1	*CAT2*	*CAT3*	*CAT4*
D	U(25, 50)	U(50, 100)	U(100, 200)	U(200, 500)
n	U(3, 15)	U(3, 30)	U(3, 65)	U(3, 150)

The algorithms were run for 50 and 1,000 seconds for each instance. Table 2 and Table 3 shows the overall average RTV values for the 740 test instances and for each class of instances (*CAT1* to *CAT4*) obtained with the three algorithms, respectively.

Table 2. Average RTV values for a computing time of 50 seconds

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
TS	202.42	10.30	22.40	109.38	667.59
Psycho	235.68	14.92	44.25	137.07	746.50
MS	2,106.01	11.56	38.02	154.82	8,219.65

Table 3. Average RTV values for a computing time of 1,000 seconds

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
TS	113.31	10.24	21.46	106.21	315.33
Psycho	161.60	14.90	39.90	122.38	469.23
MS	169.25	10.51	31.21	123.27	512.02

Tables 2 and 3 shows that the multi-start algorithm converges much slower than the other two algorithms when big instances (*CAT4* instances) are solved (Figure 2 shows how the algorithms converge during the computing time). Therefore, we analyse the results obtained by the algorithms after 1,000 seconds of computing time.

The global average RTV values of *TS* with 1,000 computing time seconds for all test instances are 29.88% and 33.05% better than the results obtained using *Psycho* or *MS*, respectively. If we consider the results by class, we can see that *MS* performs better than *Psycho* for the two smallest instances (*CAT1* and *CAT2*), both performs very similar for the medium instances (*CAT3*) and *Psycho* performs better than *MS* for the biggest instances (*CAT4*). On the other hand, *TS* performs better than the other two algorithms for all type of instance: *TS* is 2.57% and 31.24% better than *MS* for *CAT1* and *CAT2* instances, respectively, and 13.21% and 32.80% better than *Psycho* for *CAT3* and *CAT4* instances, respectively.

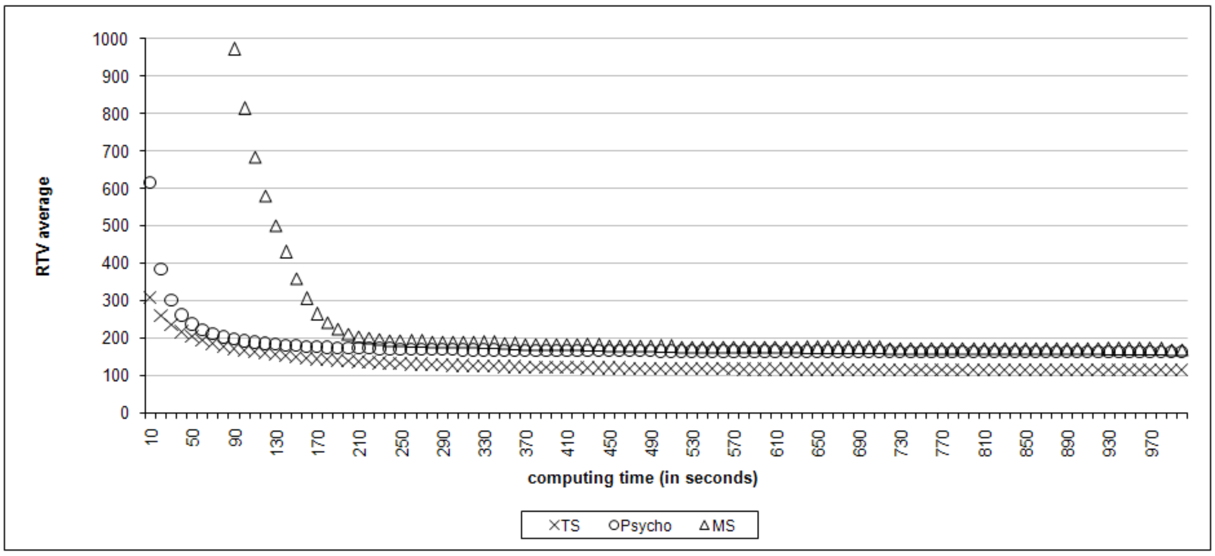


Figure 2. Average RTV values over the computing time

Table 4 shows the number of times that each algorithm reaches the best RTV value for each instance obtained using the three algorithms. The results are shown for the 740 instances overall and for each class of instance.

As could be expected from the results in Table 3, Table 4 shows that *TS* reaches the best solution the greatest number of times (in 79.32% of the instances overall). Moreover, if the results are observed by class, it can be seen that *TS* always obtains the best solution for *CAT1* and *CAT2* instances. On the other hand, *MS* obtains more times the best solution than *Psycho*, even for the *CAT4* instances. This is surprising since *MS* obtains a worst RTV values average for *CAT4* instances than *Psycho*.

Table 4. Number of times that the best solution is reached

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
TS	587	185	185	113	104
Psycho	104	52	7	37	8
MS	305	164	18	48	75

To complete the analysis of the results, we examined the dispersion of the results. A measure of the dispersion (let it be called σ) of the RTV values obtained by each algorithm *alg* = {*TS*, *Psycho*, *MS*} is defined for a given instance, *ins*, according to the following expression:

$$\sigma(alg, ins) = \left(\frac{RTV_{ins}^{(alg)} - RTV_{ins}^{(best)}}{RTV_{ins}^{(best)}} \right)^2 \tag{2}$$

where $RTV_{ins}^{(alg)}$ is the RTV value of the solution obtained with the algorithm *alg* for the instance *ins*, and $RTV_{ins}^{(best)}$ is the best RTV value of the solutions obtained with the three algorithms for the instance *ins*. Table 5 shows the average σ dispersion for the total number of instances and for each class.

Table 5. Average σ dispersion with respect to the best solution found

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
TS	0.42	0.00	0.00	0.21	1.18
Psycho	1.90	1.08	1.68	0.19	4.63
MS	0.48	0.02	0.43	0.19	1.30

Table 5 shows that *TS* has the lowest average σ dispersions for the total number of cases and for each instance class (except for *CAT3*, in which it is also low but slightly worse than the dispersions of the other two algorithms). That is, when *TS* does not obtain the best RTV value for a given instance, it obtains a value that is very close to it. *MS* has also a low dispersion for each instance class. On the other hand, Psycho has a quite worst dispersion for *CAT4* instances than *MS*, although the RTV average obtained by *Psycho* for the *CAT4* instances is better than the RTV average obtained by *MS*. This means that although *Psycho* obtains a better performance, on average, for the *CAT4* instances, the *MS* is a more robust algorithm than *Psycho*. Anyway, the TS algorithm that we propose is the one that obtains, on average, the better solutions and the one that has the most stable behaviour.

5. Conclusions and future lines of research

The RTVP is a scheduling problem which has a broad range of real-world applications. Since the RTVP is NP-hard, several heuristic and metaheuristics have been proposed to solve it. Among them, the two algorithms with which the best results have been achieved are the psychoclonal algorithm proposed in García-Villoria and Pastor (2008) and the multi-start algorithm proposed in Corominas et al. (2008). We propose a straight application of the TS metaheuristics to solve the RTVP. The computational experiment shows that the proposed TS algorithm improves by far the best results published in the literature. Moreover, the TS algorithm is very stable; that is, when it does not obtain the best RTV value for a given instance, it obtains a value that is very close to it.

The definition of the neighbourhood structure is a very critical decision in the design of any TS algorithm (Gendreau et al., 2003). In this study we propose to generate the neighbourhood of a solution by swapping each pair of consecutive or non-consecutive positions of the solution sequence. Because of the good performance of the TS algorithm, a promising line of research is testing other neighbourhood structures in the proposed algorithm to solve the RTVP. Other candidate ways of generating the neighbourhood of a solution are, for example: 1) by swapping each pair of only consecutive positions of the sequence, and 2) by inserting each position in the sequence.

References

Adenso-Díaz, B., Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research, Vol. 54, pp. 99-114.

Anily, S., Glass, C.A., Hassin, R. (1998). The scheduling of maintenance service. Discrete Applied Mathematics, Vol. 82, pp. 27-42.

Balinski, M.L. and Young, H.P. (1982). Fair Representation: meeting the ideal of one man, one vote. Yale University Press, New Haven CT.

Bollapragada, S., Bussieck, M.R., Mallik, S. (2004). Scheduling Commercial Videotapes in Broadcast Television. Operations Research, Vol. 52, pp. 679-689.

Brusco, M.J. (2008). Scheduling advertising slots for television. Journal of the Operational Research Society, Vol. 59, pp. 1363-1372.

Corominas, A., Kubiak, W., Moreno, N. (2007). Response time variability. Journal of Scheduling, Vol. 10, pp. 97-110.

Corominas, A., García-Villoria, A., Pastor, R. (2008). Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics. Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development, Vol. 184, pp. 128-137.

Dong, L., Melhem, R., Mosse, D. (1998). Time slot allocation for real-time messages with negotiable distance constrains requirements. Fourth IEEE Real-Time Technology and Applications Symposium (RTAS'98), Denver, CO. pp. 131-136.

Eiben, A.E., Hinterding, R., Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. IEEE Transactions on evolutionary computation, Vol. 3, pp. 124-141.

García, A., Pastor, R., Corominas, A. (2006). Solving the Response Time Variability Problem by means of metaheuristics. Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development, Vol. 146, pp.187-194.

García-Villoria, A., Pastor, R., Corominas, A. (2007). Solving the Response Time Variability Problem by means of the Cross-Entropy Method. Special Issue on Production Line Systems: Concepts, Methods and Applications of the International Journal of Manufacturing Technology and Management, to be published.

García-Villoria, A., Pastor, R. (2008). Solving the Response Time Variability Problem by means of a psychoclonal approach. Journal of Heuristics, doi:10.1007/s10732-008-9082-2.

García-Villoria, A., Pastor, R. (2009a). Introducing dynamic diversity into a discrete particle swarm optimization. Computers & Operations Research, Vol. 36, pp. 951-966.

García-Villoria, A., Pastor, R. (2009b). Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism. International Journal of Production Research, doi: 10.1080/00207540902862545.

Gaspar, A., Collard, P. (2000). Two models of immunization for time dependent optimization. IEEE International Conference on Systems Manufacturing and Cybernetics, pp. 113-118.

Gendreau, M. (2003). An Introduction to Tabu Search. Chapter 2 in Handbook of Metaheuristics, Eds. Glover and Kochenberger, Kluwer Academic Publishers, pp. 37-54.

Glover, F., (1986). Future paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research, Vol. 5, pp. 533-549.
Herrmann, J.W. (2007). Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling. Technical Report TR 2007-12, University of Maryland, USA. Available at http://hdl.handle.net/1903/7082.

Kubiak, W. (1993). Minimizing variation of production rates in just-in-time systems: A survey. European Journal of Operational Research, Vol. 66, pp. 259-271.

Kubiak, W. (2004). Fair Sequences. Chapter 19 in Handbook of Scheduling: Algorithms, Models and Performance Analysis, Chapman and Hall.

Maslow, A.H. (1954). Motivation and personality. New York: Harper & Bros.

Miltenburg, J. (1989). Level schedules for mixed-model assembly lines in just-in-time production systems. Management Science, Vol. 35, pp. 192-207.

Tiwari, M.K., Prakash, A., Kumar, A., Mileham, A.R. (2005). Determination of an optimal sequence using the psychoclonal algorithm. ImechE, Part B: Journal of Engineering Manufacture, Vol. 219, pp. 137-149.

Waldspurger, C.A., Weihl, W.E. (1994). Lottery Scheduling: Flexible Proportional-Share Resource Management. First USENIX Symposium on Operating System Design and Implementation.

Waldspurger, C.A., Weihl, W.E. (1995). Stride Scheduling: Deterministic Proportional-Share Resource Management. Technical Report MIT/LCS/TM-528, Massachusetts Institute of Technology, MIT Laboratory for Computer Science. Available at https://eprints.kfupm.edu.sa/67117.

Wei, W.D., Liu, C.L. (1983). On a periodic maintenance problem. Operations Research Letters, Vol. 2, pp. 90-93.

Resolución del response time variability problem mediante tabu search

In proceedings of the VIII International Event of Mathematics and Computation (COMAT'2009) of the IV International Scientific Convention of the Universidad de Matanzas (CIUM'09), Universidad de Matanzas, Cuba, June 16-18, 2009.

RESOLUCIÓN DEL RESPONSE TIME VARIABILITY PROBLEM MEDIANTE TABU SEARCH[*]

Dr. Albert Corominas, Ing. Alberto García-Villoria y Dr. Rafael Pastor
Profesor Catedrático, Ayudante y Titular, respectivamente
Instituto de Organización y Control de Sistemas Industriales (IOC).
Universidad Politécnica de Cataluña (UPC).
Av. Diagonal, 647, 08028. Barcelona.
{albert.corominas/alberto.garcia-villoria/rafael.pastor@upc.edu}

Resumen

El Response Time Variability Problem (RTVP) es un problema combinatorio de scheduling publicado recientemente en la literatura. Dicho problema de optimización combinatoria es muy fácil de formular pero muy difícil de resolver de forma exacta (es NP-hard). El RTVP se presenta cuando productos, clientes o tareas se han de secuenciar minimizando la variabilidad entre los instantes de tiempo en los que reciben los recursos que ellos necesitan. Este problema tiene una gran cantidad de aplicaciones reales: secuenciación de modelos en líneas de montaje mixtas, asignación de recursos a sistemas multiprocesadores, mantenimiento continuo, recogida de basuras o la secuenciación de anuncios en televisión. La Inteligencia Artificial dispone de herramientas eficientes, tales como las metaheurísticas, para resolver problemas combinatorios de scheduling complejos. En trabajos previos, el RTVP ha sido resuelto mediantes varios algoritmos metaheurísticos provenientes de la Inteligencia Artificial (entre otros, las metaheurísticas multi-start, PSO y GRASP). En este trabajo se propone un algoritmo de búsqueda tabu (tabu seach), el cual mejora los mejores resultados referenciados en la literatura.

Palabras clave: response time variability, metaheurísticas, tabu search, scheduling

[*] Investigación subvencionada por el Ministerio de Educación y Ciencia de España, proyecto DPI2007-61905, cofinanciado por FEDER.

SOLVING THE RESPONSE TIME VARIABILITY PROBLEM BY MEANS OF A TABU SEARCH APPROACH[□]

Abstract

The Response Time Variability Problem (RTVP) is a combinatorial scheduling problem that has recently appeared in the literature. Although this combinatorial optimisation problem is easy to formulate, it is very difficult to solve (it is NP-hard).The RTVP occurs whenever products, clients or jobs need to be sequenced so as to minimize variability in the time between the instants at which they receive the necessary resources. This problem has a broad range of real-world applications: to sequence on mixed-model assembly lines, to resource allocation in computer multi-threaded systems, in the periodic machine maintenance problem, in the collection of waste and in the schedule of commercial videotapes for television. The field of Artificial Intelligence has provided us with efficient tools such as metaheuristic techniques for solving complex combinatorial scheduling problems. In previous studies, several metaheuristic algorithms (among others, a multi-start, a PSO and a GRASP algorithm) were proposed to solve the RTVP. In this study we propose a tabu search algorithm for the RTVP which improves the best results reported in the literature.

Keywords: response time variability, metaheuristics, tabu search, scheduling

1. Introducción

El concepto de secuencia *fair* (secuencia justa, imparcial, buena, ideal, regular) ha surgido de forma independiente en problemas de *scheduling* de diversos entornos. El objetivo común de ese tipo de problemas de *scheduling*, como es definido en Kubiak (2004), consiste en construir una secuencia *fair* utilizando *n* símbolos, de forma que el símbolo *i* (*i* = 1,...,*n*) se presente d_i veces en la secuencia. Una secuencia *fair* es aquella que asigna, repartiendo de forma *fair*, las posiciones a cada símbolo *i* en cualquier subsecuencia. Este reparto *fair* o ideal de posiciones asignadas al símbolo *i* en una subsecuencia de longitud *k* es proporcional a la importancia relativa del símbolo *i* (d_i) respecto al total de copias de los diferentes símbolos a repartir (que, obviamente, es igual a $\sum_{i=1..n} d_i$). No existe una definición universal de *fairness*, ya que se pueden definir diversas métricas razonables en función del problema específico que se está considerando.

Considerando las diferentes definiciones de *fairness*, varios problemas de secuenciación *fair* han sido propuestos en la literatura, entre ellos el *Response Time Variability Problem* (RTVP). Este problema fue expuesto por primera vez en Waldspurger and Weihl (1994) y formalizado en Corominas et al. (2007). En el RTVP, la secuencia *fair* es aquella que minimiza la suma de la variabilidad

[□] Supported by the Spanish Ministry of Education and Science under project DPI2007-61905; co-funded by the ERDF.

de las distancias entre cualquier pareja de copias consecutivas de un mismo símbolo. En otras palabras, la distancia entre cualquier pareja de copias consecutivas de un mismo símbolo debería ser tan regular como sea posible (e, idealmente, constante).

El RTVP se presenta cuando productos, clientes o tareas se han de secuenciar minimizando la variabilidad entre los instantes de tiempo en los que reciben los recursos que ellos necesitan (Corominas et al., 2007). Este problema tiene una gran cantidad de aplicaciones reales: la secuenciación de modelos en líneas de montaje mixtas (Kubiak, 1993; Miltenburg, 1989); la asignación de recursos en sistemas multiprocesadores, tales como los servidores de redes o las aplicaciones de transmisión de videos (Dong et al., 1998; Waldspurger and Weihl, 1995); el mantenimiento continuo, cuando el tiempo entre servicios consecutivos en una misma máquina debe ser el mismo (Anily et al., 1998; Wei and Liu, 1983); la recogida de basuras (Herrmann, 2007); o la secuenciación de anuncios en prensa o televisión (Bollapragada et al., 2004; Brusco, 2008).

El RTVP es un problema *NP-hard* (Corominas et al., 2007). Debido a que es un problema de optimización combinatoria difícil de resolver de forma óptima, en la literatura se han propuesto para su resolución diversos procedimientos heurísticos y metaheurísticos. Waldspurger and Weihl (1995) utilizan el método de distribución de escaños de Jefferson (Balinski and Young, 1982), un algoritmo heurístico *greedy* que ellos denominan como la técnica *stride scheduling*. Herrmann (2007) resuelve el RTVP aplicando un algoritmo heurístico basado en la técnica *stride scheduling*. Corominas et al. (2007) también proponen el método de Jefferson junto a otros cuatro algoritmos heurísticos. García et al. (2006) diseñan seis algoritmos metaheurísticos: un *multi-start*, un *greedy randomized adaptive search procedure* (GRASP) y cuatro variantes de un algoritmo basado en el *particle swarm optimization* (PSO) discreto. Un algoritmo *multi-start* mejorado, así como un nuevo algoritmo GRASP, también mejorado, son propuestos en Corominas et al. (2008); y otros diez algoritmos PSO discretos son propuestos en García-Villoria and Pastor (2009a). Un procedimiento *cross-entropy* es utilizado en García-Villoria et al. (2007). Un algoritmo basado en *Electromagnetism-like Mechanism* (EM) es propuesto en García-Villoria and Pastor (2009b). Finalmente, un algoritmo *Psychoclonal* es utilizado para resolver el RTVP en García-Villoria and Pastor (2008). Los mejores resultados reportados en la literatura hasta el momento se han conseguido con el algoritmo *psychoclonal* (García-Villoria and Pastor, 2008) y el algoritmo *multi-start* mejorado (Corominas et al., 2008).

Hasta la fecha, ningún algoritmo *tabu search* (TS) ha sido propuesto para resolver el RTVP. En este trabajo se propone un algoritmo TS para el RTVP que mejora los mejores resultados publicados hasta el momento en la literatura.

El resto del artículo está organizado como sigue: la Sección 2 presenta una definición formal del RTVP y describe, brevemente, los dos mejores procedimientos metaheurísticos publicados hasta el momento para resolver el problema de estudio; la Sección 3 propone un algoritmo TS para resolver el RTVP; la Sección 4 presenta el resultado del experimento computacional realizado; finalmente, varias conclusiones y sugerencias de trabajo futuro son expuestas en la Sección 5.

2. El Response Time Variability Problem

El RTVP puede ser formulado como sigue. Sea *n* el número de símbolos, d_i el número de copias a ser secuenciadas del símbolo *i* (*i* = 1,…,*n*) y *D* en número total de copias a secuenciar ($\sum_{i=1..n} d_i$). Sea *s* una solución de un ejemplar del RTVP, que consiste en una secuencia circular de copias ($s = s_1 s_2 \ldots s_D$), donde s_j es la copia secuenciada en la posición *j* de la secuencia *s*. Para cada símbolo *i* con $d_i \geq 2$, sea t_k^i la distancia entre las posiciones en las que se encuentran las copias *k* + 1 y *k* del símbolo *i* (considerando que la distancia entre dos posiciones consecutivas de la secuencia es igual a 1). Como se ha introducido, la secuencia es circular, de esta forma la posición 1 viene inmediatamente después de la posición *D*, y $t_{d_i}^i$ es la distancia entre la primera copia del símbolo *i* en un ciclo y la última copia de ese mismo símbolo en el ciclo precedente. Sea $\overline{t_i}$ la distancia media deseada entre dos copias consecutivas del símbolo *i* ($\overline{t_i} = D/d_i$). El objetivo es minimizar la métrica llamada *response time variability* (*RTV*), que se define como la suma de los cuadrados de los errores respecto a las distancias $\overline{t_i}$. Como los símbolos *i* con $d_i = 1$ no intervienen en el cálculo del RTV, se asume que para cada uno de esos símbolo t_1^i es igual a $\overline{t_i}$. De esta forma, el RTV se obtiene con la expresión

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} \left(t_k^i - \overline{t_i} \right)^2 .$$

Por ejemplo, sea $n = 3$ y los símbolos A, B y C; considérese, adicionalmente, que $d_A = 2$, $d_B = 2$ y $d_C = 4$; así, $D = 8$, $\overline{t_A} = 4$, $\overline{t_B} = 4$ y $\overline{t_C} = 2$. Cualquier secuencia que contenga el símbolo *i* ($\forall i$) exactamente d_i veces es una solución factible. Por ejemplo, la secuencia (C, A, C, B, C, B, A, C) es una solución factible con el siguiente valor:

$$RTV = \left((5-4)^2 + (3-4)^2 \right) + \left((2-4)^2 + (6-4)^2 \right) + \left((2-2)^2 + (2-2)^2 + (3-2)^2 + (1-2)^2 \right) = 1 \quad 2 \quad .$$

Como se ha introducido en la Sección 1, el algoritmo *psychoclonal* propuesto en García-Villoria and Pastor (2008) y el procedimiento *multi-start* presentado en Corominas et al. (2008) son los mejores procedimientos publicados hasta la fecha para resolver el RTVP.

El procedimiento *psychoclonal* es una metaheurística evolutiva propuesta por primera vez en Tiwari et al. (2005). De acuerdo con los autores, esta metaheurística obtiene sus fundamentos de la teoría de las necesidades jerárquicas de Maslow (1954) y del principio de selección por clonación (Gaspar and Collard, 2000). El esquema básico de la metaheurística *psychoclonal* es el siguiente: 1) Se genera una población inicial de soluciones y se dispone de una función que evalúa el *fitness* (la adecuación) de cualquier solución; 2) Las mejores soluciones son seleccionadas y clonadas un número de veces que es proporcional a su propio *fitness*; 3) Los clones generados son hipermutados (la hipermutación es un operador que modifica la solución con un ratio inversamente proporcional a su *fitness*); 4) Se forma una nueva población seleccionando los mejores clones e incorporando nuevas soluciones generadas de forma aleatoria; 5) Se repiten las etapas 2 a 4 hasta que se cumple un criterio de parada. Esta metaheurística fue adaptada y probada en la resolución del RTVP (para un mayor detalle, se recomienda García-Villoria and Pastor, 2008).

El esquema general de la metaheurística *multi-start* consta de dos fases. En la primera fase se genera una solución inicial, la cual es mejorada en la segunda fase del procedimiento. Dichas dos fases se aplican de forma iterativa hasta que se cumple un criterio de parada. En el algoritmo *multi-start* propuesto en Corominas et al. (2008) para resolver el RTVP, la generación de la solución inicial se realiza de forma alteatoria y la fase de mejora con un procedimiento de optimización local; por otro lado, el criterio de parada consiste en alcanzar un tiempo de computación preestablecido (para un mayor detalle, se recomienda Corominas et al., 2008).

3. Un algoritmo Tabu Search para resolver el RTVP

Los métodos de búsqueda local presentan la gran desventaja que una vez se alcanza el óptimo local, éste suele ser una solución mediocre (Gendreau, 2003) y, además, el método finaliza la búsqueda y no permite evolucionar a nuevos óptimos locales. Para superar estas limitaciones, Glover (1986) propone la metaheurística Tabu Search (TS). TS se basa en aplicar una búsqueda local en la cual se permiten movimientos de no- mejora (es decir, que proporcionan soluciones con un valor peor que el valor de la solución de partida). Para evitar que el procedimiento entre en un ciclo infinito entre soluciones ya generadas, la historia más reciente de la búsqueda es guardada en una lista de soluciones tabu (prohibidas). En la lista tabu se podrían guardar las soluciones tabú completas (toda la secuencia que, en el RTVP, las identifica), pero esta estrategia podría necesitar mucha memoria de ordenador, podría hacer lento el comprobar si una solución es o no tabu y, sobretodo, podría no diversificar suficientemente la búsqueda. De esta forma, es común guardar únicamente los últimos movimientos (transformaciones) realizados en la solución de partida y prohibir las transformaciones inversas que, nuevamente, llevarían a (generarían) la solución de partida (Gendreau, 2003). La lista tabu usualmente es implementada como una lista de longitud fija con una política FIFO (*First In, First Out*). Una solución tabu puede ser utilizada (es decir, se cancela la prohibición de ser considerada) si satisface un criterio de aspiración (que, habitualmente, suele ser que proporcione un valor de la función de evaluación mejor que el valor de la mejor solución generada hasta el momento en todo el proceso de búsqueda). La Figura 1 muestra el esquema general del procedimiento TS.

```
Sea $S$ una solución inicial
Definir la estructura de vecindario $N$
Mientras no se cumpla la condición de parada hacer:
    Sea $S'$ la mejor solución de $N$ que cumple el criterio de aspiración o no tabu
    Si $S'$ es mejor que la mejor generada hasta el momento S*, hacer $S^* := S'$
    Hacer $S := S'$
    Añadir el movimiento actual a la lista tabu (y, tal vez, borrar el último)
Fin mientras
Devolver $S^*$
```

Figura 1. Esquema general del procedimiento TS

En este trabajo se propone un algoritmo basado en el esquema anterior de TS para resolver el RTVP. A continuación se especifican los elementos del TS propuesto:

– *Solución inicial*. Una solución es representada por la secuencia de copias de los símbolos a ser secuenciados. Se toma como solución inicial la mejor de las proporcionadas por los cinco procedimientos heurísticos propuestos en Corominas et al. (2007).

– *Vecindario*. El vecindario de una solución se obtiene insertando el símbolo asignado a cada posición entre el resto de posiciones de la secuencia que representa a dicha solución.

– *Movimientos tabu*. Un movimiento prohibido de la lista tabu consiste en una pareja “posición/símbolo”. Por ejemplo, el movimiento (3, A) significa que se consideran tabu todas las soluciones con el símbolo A secuenciado en la posición 3 de la secuencia.

– *Criterio de aspiración*. El criterio de aspiración consiste en que el movimiento proporcione una solución que sea mejor que la mejor solución generada hasta el momento en todo el proceso de búsqueda.

– *Condición de parada*. El algoritmo TS finaliza la búsqueda cuando se alcanza un tiempo de computación preestablecido (como en los algoritmos psychoclonal y *multi-start* propuestos en García-Villoria and Pastor (2008) y Corominas et al. (2008), respectivamente).

El algoritmo TS presenta un único parámetro cuyo valor ha de ser fijado de partida: el tamaño de la lista tabu. Aunque el valor de este tipo de parámetros es extremadamente importante, ya que los resultados de las metaheurísticas para cada problema son muy sensibles a ellos, la decisión de cómo determinarlos es comúnmente justificada de una de las siguiente maneras (Eiben et al., 1999; Adenso-Díaz and Laguna, 2006): 1) "a mano", en base a un pequeño número de experimentos que no son específicamente referenciados; 2) utilizando valores generales, recomendados para un amplio rango de diferentes tipos de problemas; 3) utilizando los valores que han sido efectivos en otros problema similares; o 4) seleccionando dichos valores sin ninguna explicación.

Adenso-Díaz and Laguna (2006) proponen una nueva técnica, llamada CALIBRA, diseñada específicamente para ajustar el valor de los parámetros de algoritmos heurísticos y metaheurísticos. CALIBRA es utilizada en García-Villoria and Pastor (2008) y en Corominas et al. (2008) para determinar los valores de los parámetros de los algoritmos *psychoclonal* y *multi-start*, respectivamente. En este trabajo también se ha utilizado CALIBRA para decidir el valor de la lista tabu del algoritmo TS propuesto: CALIBRA fija el valor de la lista tabu en 38.

4. Experimento computacional

Los algoritmos *psychoclonal* y *multi-start*, propuestos en García-Villoria and Pastor (2008) y en Corominas et al. (2008), son los algoritmos más eficientes publicados en la literatura hasta el momento para resolver el RTVP. De esta forma, se compara la calidad del algoritmo TS propuesto con la de dichos dos algoritmos. En lo que resta de esta sección, el algoritmo TS será referido como *TS*, el algoritmo psychoclonal como *Psycho* y el algoritmo multi-start como *MS*.

Los tres algoritmos han sido codificados en Java y han sido ejecutados en un ordenador Pentium IV de 3.4 GHz con 1.5 GB de memoria RAM. En este trabajo se utilizan los mimos 60 ejemplares de entrenamiento y los 740 ejemplares de prueba que son utilizados en García-Villoria and Pastor (2008) y Corominas et al. (2008). Los 60 ejemplares de entrenamiento son utilizados por CALIBRA para fijar el valor de los parámetros de los algoritmos. Todos los ejemplares pueden ser agrupados en cuatro clases (de la *CAT1* a la *CAT4,* con 15 ejemplares de entrenamiento y 185 de prueba en cada clase) de acuerdo a su tamaño. Los ejemplares fueron generados utilizando las distribuciones uniformes mostradas en la Tabla 1 para generar, de forma aleatoria, los valores de *D* (número total de copias) y *n* (número de símbolos). Para todos los ejemplares y para cada símbolo $i = 1,\ldots,n$, se generó un valor aleatorio de d_i (número de copias del símbolo *i* a ser secuenciadas) entre 1 y $\left|(D-n+1)/2.5\right|$, cumpliendo, obviamente, que $\sum_{i=1..n} d_i = D$.

	CAT1	***CAT2***	***CAT3***	***CAT4***
D	U(25, 50)	U(50, 100)	U(100, 200)	U(200, 500)
n	U(3, 15)	U(3, 30)	U(3, 65)	U(3, 150)

Tabla 1. Distribución uniforme para generar los valores de *D* y de *n*

Los tres algoritmos fueron ejecutados, para cada ejemplar, durante 50 y 1000 segundos. Las Tablas 2 y 3 muestran los valores promedio del RTV, obtenidos con cada uno de los tres algoritmos, para el total de los 740 ejemplares de prueba (*Global*) y para cada clase de ejemplares (*CAT1* a *CAT4*)

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
TS	210.47	10.26	22.56	73.26	735.78
Psycho	235.68	14.92	44.25	137.07	746.50
MS	2106.01	11.56	38.02	154.82	8219.65

Tabla 2. Valor promedio del RTV con 50 segundos de tiempo de computación

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
TS	78.62	10.24	21.16	48.12	234.96
Psycho	161.60	14.90	39.90	122.38	469.23
MS	169.25	10.51	31.21	123.27	512.02

Tabla 3. Valor promedio del RTV con 1000 segundos de tiempo de computación

Las Tablas 2 y 3 muestran que el algoritmo multi-start converge más lentamente que los otros dos algoritmos, en global y cuando se resuelven ejemplares de gran tamaño, los de la categoría *CAT4* (la Figura 2 muestra la convergencia de los algoritmos en función del tiempo de computación). De esta manera, se analizan los resultados obtenidos por los algoritmos después de 1000 segundos de tiempo de computación.

El valor promedio del RTV para *TS*, con 1000 segundos de tiempo de cálculo y para el conjunto de los 740 ejemplares (Global), es un 51.35% y un 53.55% mejor que los valores obtenidos utilizando los algoritmos *Psycho* y *MS*, respectivamente. Si se consideran los resultados por clases, *MS* presenta mejores resultados que *Psycho* para las clases de ejemplares de menor tamaño (*CAT1* y *CAT2*), resultados muy semejantes para los ejemplares de tamaño medio (*CAT3*) y peores resultados para la clase de ejemplares de mayor tamaño (*CAT4*). De todas formas, el algoritmo *TS* obtiene mejores resultados que los otros dos algoritmos en todas las clases de ejemplares: *TS* es un 2.57% y un 32.21% mejor que *MS* para las clases *CAT1* y *CAT2*, respectivamente, y un 60.68% y un 49.93% mejor que *Psycho* para las clases *CAT3* y *CAT4*, respectivamente.

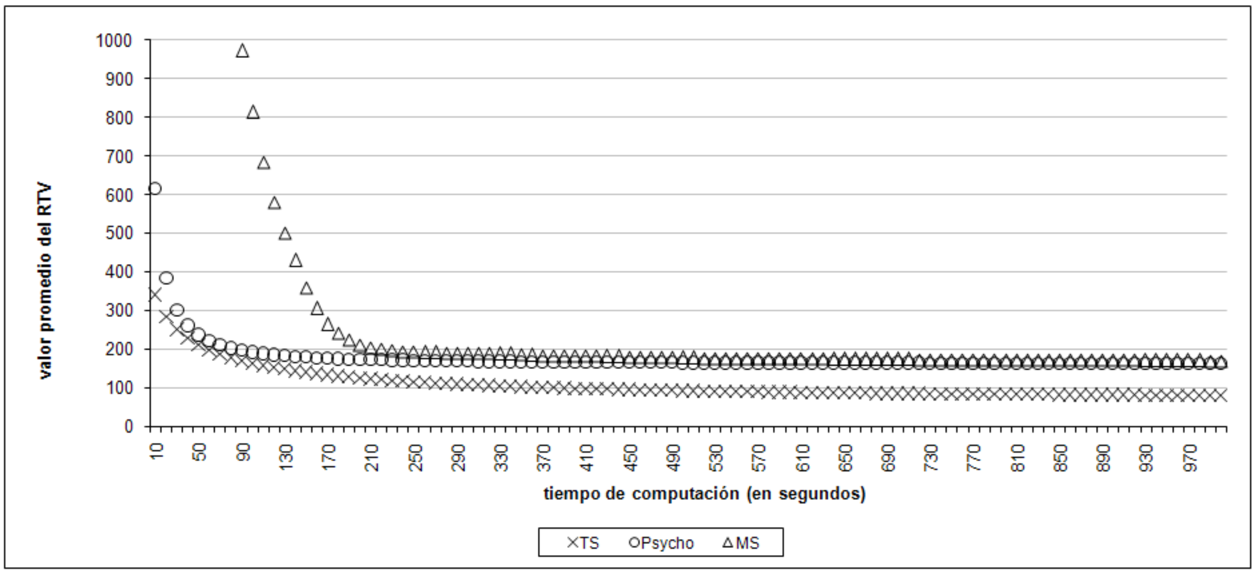


Figura 2. Valor promedio del RTV en función del tiempo de computación

La Tabla 4 muestra el número de veces que cada algoritmo obtiene el mejor valor RTV para cada ejemplar, considerando los tres algoritmos. Los resultados se presentan para el total de los 740 ejemplares (Global) y para cada clase de ejemplares.

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
TS	678	185	185	185	123
Psycho	58	51	7	0	0
MS	244	163	17	1	63

Tabla 4. Número de veces en que se obtiene la mejor solución

Como se podría esperar de los resultados de la Tabla 3, la Tabla 4 muestra que *TS* obtiene la mejor solución el mayor número de veces (en un 91.62% de ejemplares en global). Además, si se analizan los resultados por clases, se observa que *TS* siempre obtiene las mejores soluciones en las clases *CAT1*, *CAT2* y *CAT3*. Por otro lado, *MS* obtiene más veces que *Psycho* la mejor solución, incluso para los ejemplares de la clase *CAT4*. Este resultado puede sorprender ya que, para esta clase de ejemplares, *MS* obtiene un valor promedio del RTV peor que el obtenido por el algoritmo *Psycho*.

Para completar el análisis de los resultados, se calcula su dispersión. Se define una medida de dispersión (aquí denominada *σ*) del valor del RTV obtenido por cada algoritmo, *alg* = {*TS*, *Psycho*, *MS*}, en cada ejemplar, *ins*, según la expresión $\sigma(alg,ins)=\left(\frac{RTV_{ins}^{(alg)}-RTV_{ins}^{(best)}}{RTV_{ins}^{(best)}}\right)^2$; donde $RTV_{ins}^{(alg)}$ es el valor del RTV de la solución obtenida con el algoritmo *alg* para el ejemplar *ins*, y $RTV_{ins}^{(best)}$ es el mejor valor del RTV, para el ejemplar *ins*, de las soluciones obtenidas con los tres algoritmos. La Tabla 5 expone la dispersión *σ* promedio para el total de ejemplares y por clases.

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
TS	0.10	0.00	0.00	0.00	0.39
Psycho	3.00	1.08	1.76	3.67	5.49
MS	1.77	0.02	0.47	3.47	3.11

Tabla 5. Dispersión *σ* promedio

La Tabla 5 muestra que *TS* presenta la menor dispersión promedio para el total de ejemplares y para cada una de las clases. Para las tres primeras clases (*CAT1*, *CAT2* y *CAT3*), la dispersión es nula ya que siempre obtiene las mejores soluciones. En cambio, para los ejemplares más grandes (*CAT4*) la dispersión ya no es nula aunque continúa siendo muy baja. Este resultado significa que cuando *TS* no obtiene el mejor valor del RTV para un ejemplar, obtiene un valor muy cercano. *MS* también obtiene una baja dispersión para las dos clases más pequeñas (*CAT1* y *CAT2*). Por otro lado, *Psycho* presenta una dispersión peor que *MS* para todas las clases, aunque los valores promedios del RTV obtenidos por *Psycho* para *CAT3* y *CAT4* es mejor que el obtenido por *MS*; esto indica que aunque *Psycho* obtiene un mejor funcionamiento para *CAT3* y *CAT4*, en promedio, *MS* es un algoritmo más robusto que *Psycho*. De todas formas, el algoritmo TS propuesto en este trabajo es el que obtiene, en promedio, las mejores soluciones con un comportamiento muy estable.

5. Conclusiones y futuras líneas de investigación

El RTVP es un problema de *scheduling* que se presenta en una gran cantidad de aplicaciones reales. Como el RTVP es un problema *NP-hard*, varios procedimientos heurísticos y metaheurísticos han sido propuestos para su resolución. Entre ellos, los dos algoritmos que han obtenido mejores resultados son el algoritmo psychoclonal propuesto en García-Villoria and Pastor (2008) y el algoritmo multi-start presentado en Corominas et al. (2008). En este trabajo se presenta una aplicación basada en la metaheurística *tabu search* (TS) para la resolución del RTVP. Los resultados del experimento computacional realizado muestran que el algoritmo TS propuesto mejora los mejores resultados publicados en la literatura hasta el momento. Además se comprueba que el algoritmo TS es muy estable: cuando no obtiene el mejor resultado para un ejemplar del RTVP, obtiene un valor muy cercano al mejor.

La definición del vecindario es una decisión crítica en el diseño de cualquier metaheurística, en general, y del algoritmo TS, en particular (Gendreau et al., 2003). En este trabajo se genera el vecindario de una solución insertando el símbolo asignado a cada posición entre el resto de posiciones de la secuencia que representa a dicha solución. Debido al buen funcionamiento del algoritmo TS, se pretende seguir investigando para intentar mejorarlo. Para ello se propone diseñar y probar nuevas definiciones del vecindario de una solución; por ejemplo intercambiado los símbolos asignados a cada pareja de posiciones consecutivas de la secuencia que representa a una solución o intercambiando los símbolos asignados a cada pareja de posiciones consecutivas y no consecutivas de la secuencia.

Bibliografía

1. Kubiak, W. Fair Sequences. Chapter 19 in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapman and Hall, 2004.
2. Waldspurger, C.A.; Weihl, W.E. Lottery Scheduling: Flexible Proportional-Share Resource Management. *First USENIX Symposium on Operating System Design and Implementation*, 1994.
3. Corominas, A.; Kubiak, W.; Moreno, N. Response time variability. *Journal of Scheduling* 10: 97-110, 2007.
4. Kubiak, W. Minimizing variation of production rates in just-in-time systems: A survey. *European Journal of Operational Research* 66: 259-271, 1993.
5. Miltenburg, J. Level schedules for mixed-model assembly lines in just-in-time production systems. *Management Science* 35: 192-207, 1989.
6. Dong, L.; Melhem, R.; Mosse, D. Time slot allocation for real-time messages with negotiable distance constrains requirements. *Fourth IEEE Real-Time Technology and Applications Symposium (RTAS'98)*, Denver, CO, 131-136, 1998.

7. Waldspurger, C.A.; Weihl, W.E. Stride Scheduling: Deterministic Proportional-Share Resource Management. *Technical Report MIT/LCS/TM-528*, Massachusetts Institute of Technology, 1995.
8. Anily, S.; Glass, C.A.; Hassin, R. The scheduling of maintenance service. *Discrete Applied Mathematics* 82: 27-42, 1998.
9. Wei, W.D.; Liu, C.L. On a periodic maintenance problem. *Operations Research Letters* 2: 90-93, 1983.
10. Herrmann, J.W. Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling. *Technical Report TR 2007-12*, University of Maryland, USA, 2007.
11. Bollapragada, S.; Bussieck, M.R.; Mallik, S. Scheduling Commercial Videotapes in Broadcast Television. *Operations Research* 52: 679-689, 2004.
12. Brusco, M.J. Scheduling advertising slots for television. *Journal of the Operational Research Society* 59: 1363-1372, 2008.
13. Balinski, M.L.; Young, H.P. Fair Representation: meeting the ideal of one man, one vote. Yale University Press, New Haven CT, 1982.
14. García, A.; Pastor, R.; Corominas, A. Solving the Response Time Variability Problem by means of metaheuristics. *Frontiers in Artificial Intelligence and Applications* 146: 187-194, 2006.
15. Corominas, A.; García-Villoria, A.; Pastor, R. Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics. *Frontiers in Artificial Intelligence and Applications* 184: 128-137, 2008.
16. García-Villoria, A.; Pastor, R. Introducing dynamic diversity into a discrete particle swarm optimization. *Computers and Operations Research* 36: 951-966, 2009a.
17. García-Villoria, A.; Pastor, R.; Corominas, A. Solving the Response Time Variability Problem by means of the Cross-Entropy Method. *International Journal of Manufacturing Technology and Management*, to be published, 2007.
18. García-Villoria, A.; Pastor, R. Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism. *International Journal of Production Research*, doi: 10.1080/00207540902862545, 2009b.
19. García-Villoria, A.; Pastor, R. Solving the Response Time Variability Problem by means of a psychoclonal approach. *Journal of Heuristics*, doi:10.1007/s10732-008-9082-2, 2008.
20. Tiwari, M.K.; Prakash, A.; Kumar, A.; Mileham, A.R. Determination of an optimal sequence using the psychoclonal algorithm. *ImechE, Part B: Journal of Engineering Manufacture* 219: 137-149, 2005.
21. Maslow, A.H. *Motivation and personality*. New York: Harper & Bros, 1954.
22. Gaspar, A.; Collard, P. Two models of immunization for time dependent optimization. *IEEE International Conference on Systems Manufacturing and Cybernetics*, 113-118, 2000.
23. Gendreau, M. An Introduction to Tabu Search. Chapter 2 in *Handbook of Metaheuristics*, Eds. Glover and Kochenberger, Kluwer Academic Publishers, 37-54, 2003.
24. Glover, F. Future paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research* 5: 533-549, 1986.

25. Eiben, A.E.; Hinterding, R.; Michalewicz, Z. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation* 3: 124-141, 1999.
26. Adenso-Díaz, B.; Laguna, M. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research* 54:, 99-114, 2006.

A2.4. Technical reports

A branch and bound approach for the response time variability problem

Technical report IOC-DT-P-2009-05, Digital version available at UPCommons (http://upcommons.upc.edu), 2009.

A branch and bound approach for the response time variability problem[†]

Alberto GARCÍA-VILLORIA[a*], Albert COROMINAS[a], Xavier DELORME[b], Alexandre DOLGUI[b], Wieslaw KUBIAK[c], and Rafael PASTOR[a]
[a] Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya (UPC), 647, Av. Diagonal, 08028, Barcelona, Spain
[b] Centre for Industrial Engineering and Computer Science, Ecole des Mines de Saint Etienne, 158, Cours Fauriel, 42023 Saint Etienne Cedex, France
[c] Faculty of Business Administration, Memorial University, St. John's, Canada
{alberto.garcia-villoria / albert.corominas / rafael.pastor}@upc.edu, {delorme / dolgui }@emse.fr, wkubiak@mun.ca

Abstract. The response time variability problem (RTVP) is a NP-hard scheduling problem which has recently formalised in the literature. This problem has a wide range of real-world applications in mixed-model assembly lines, multi-threaded computer systems, network environments and others. The RTVP arises whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the points at which they receive the necessary resources is minimised. The best exact method to solve this problem is a mixed integer linear programming (MILP) which is able to solve optimally instances up to 40 units to be scheduled in a practical time. The objective of this work is to increase the size of the instances that can be solved optimally. We propose a branch and bound (B&B) algorithm that takes advantage of the characteristics of the problem. In particular, several dominated and equivalent solutions are detected to avoid exploring them. A computational experiment shows that the proposed B&B algorithm is able to solve larger instances up to 55 units to optimally.

Keywords: response time variability, scheduling, fair sequences, branch and bound

1. Introduction

The Response Time Variability Problem (RTVP) is an optimisation sequencing problem which was first reported in Waldspurger and Weihl (1994) and formally formulated by Corominas *et al*. (2007). This problem occurs in real-life situations in which jobs, clients, products or events need to be sequenced in such a way that the variability in the time between the turns at which they receive their necessary resources is minimised.

[†] Supported by the Spanish Ministry of Education and Science under project DPI2007-61905; co-funded by the ERDF.
[*] Corresponding author: Alberto García-Villoria, Institute of Industrial and Control Engineering (IOC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; tel.: +34 93 4054010; e-mail: alberto.garcia-villoria@upc.edu.

One of the first situations in which the idea of the regular sequence appeared was the sequencing on mixed-model assembly lines at Toyota Motor Corporation under the just-in-time (JIT) production system. Since Toyota popularized the JIT production systems, the problem of sequencing on mixed-model assembly lines has acquired high relevance. One of the main aims of JIT is to eliminate sources of waste and inefficiency. In the case of Toyota, the main source of waste was the production of excessive volumes of stock. To solve this problem, JIT systems produce only the specific models required and in the quantities needed at any given time. According to Monden (1983), in this type of system the units should be scheduled in such a way that the consumption rates of the components in the production process remain constant. Under certain assumptions introduced in Miltenburg (1989), this scheduling problem can be solved considering only the demand rates for the models (Miltenburg, 1989; Kubiak, 1993).

The RTVP also appears in computer multithreaded systems (Waldspurger and Weihl, 1994 and 1995; Dong *et al.*, 1998). Multithreaded systems (operating systems, network servers, media-based applications, etc.) do different tasks to attend to the requests of client programs that take place concurrently. These systems need to manage the scarce resources in order to service the requests of *n* clients. For example, multimedia systems must not display video frames too early or too late, because this would produce jagged motion perceptions (Corominas *et al*., 2007). Waldspurger and Weihl, considering that resource rights could be represented by *tickets* and that each client had its own number of tickets, suggested the RTV metric to evaluate the sequence of resource rights.

Two real-life cases of RTVP applications were reported in the literature. In Bollapragada *et al*. (2004), the study is motivated by the problem faced by the National Broadcasting Company (BNC) of U.S., one of the main firms in the television industry. Major advertisers buy to BNC hundreds of time slots to air commercials. The advertisers ask to BNC that the airings of their commercials are evenly spaced as much as possible over the broadcast season. The same problem is also solved in Brusco (2008). In Herrmann (2007), the author came up with the RTVP while working with a healthcare facility that needed to schedule the collection of waste from waste collection rooms throughout the building. Based on data about how often a waste collector had to visit each room and in view of the fact that different rooms require a different number of visits per shift, the facility manager wanted these visits to occur as regular as possible so that excessive waste would not collect in any room. For instance, if a room needed four visits per eight-hour shift, it should be ideally visited every two hours.

Other contexts in which the RTVP can be applied are the design of sales catalogs (Bollapragada *et al*., 2004), the periodic machine maintenance problem (Wei and Liu, 1983; Anily *et al*., 1998) as well as other distance-constrained problems (e.g., see Han *et al*., 1996).

The abovementioned applications are examples of a very common situation, in manufacturing and in services, in which a resource must be used successively by different units and it is important to schedule them in such a way that the different types of units share the resource in some fair manner (see Kubiak (2009) who provides an extensive overview on fair sequences). The RTVP proposes a new universal measure of fairness: to minimise the variability of the distance (measured, for example, in number of slot times) between any two consecutive units of the same product, event, job or client; i.e., to have the distances between any two given consecutive units of the same product as constant as possible. Several other measures have been proposed for the fairness of the sequence of models on assembly lines, either based on the difference between ideal and actual productions (Miltenburg, 1989; Kubiak, 1993; Steiner and Yeomans, 1993) or on the difference between ideal and actual production dates (Inman and Bulfin, 1991; Bautista *et al.*, 1997). The new measure of fairness is easier to understand by practitioners, since it only uses a simple concept: the distance. Moreover it has the characteristic that the value of the measure does not depend on the position of those products with only one unit to be sequenced.

The RTVP has been demonstrated to be NP-hard (Corominas *et al.*, 2007). Thus, this problem has been mostly solved by means of heuristic procedures (Waldspurger and Weihl, 1994, 1995; Corominas *et al.*, 2007; Herrmann, 2007, 2009) as well as metaheuristic procedures (García *et al.*, 2006; Corominas *et al.*, 2008, 2009a, 2009b, 2009c, 2009d; García-Villoria and Pastor, 2008, 2009a, 2009b, 2010; García-Villoria *et al.*, 2010).

Anyway, there are two works in which the RTVP is solved optimally for small instances (Corominas *et al.*, 2007, 2010). In both works a Mixed Integer Linear Programming (MILP) approach is used. The best MILP model (Corominas *et al.*, 2010) obtains optimal solutions with a practical time of 40 units to be sequenced.

The disadvantage of the MILP approach is that general software is used to solve the MILP model and it is difficult to take advantage of all characteristics of the problem. In order to solve optimally larger instances in a practical time, we suggest using the branch and bound (B&B) approach. In this paper we proposed an algorithm based on the B&B technique that is specifically designed to solve the RTVP. A computational experiment shows that the proposed B&B algorithm is able to solve optimally instances up to 55 copies to be sequenced (that is, the size of the instances that can be optimally solved has been increased 37.5% with respect to the best exact method published in the literature).

The remainder of the paper is organized as follows. Section 2 presents a formal definition of the RTVP. Section 3 proposes our B&B algorithm to solve the RTVP. Section 4 presents the results of a computational experiment. Finally, some concluding remarks are given in Section 5.

2. The Response Time Variability Problem (RTVP)

The RTVP is formulated as follows. Let n be the number of symbols to be sequenced (that represent products, jobs, clients, events, …), where symbol i (i = 1,...,n) is to be copied d_i times in the sequence. Let D be the total number of copies to be sequenced ($D = \sum_{i=1..n} d_i$). Let s be a solution of an instance in the RTVP that consists of a circular sequence of copies ($s = s_1 s_2 \ldots s_D$), where s_j is the copy sequenced in position j of sequence s. For each symbol i in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the copies k + 1 and k of symbol i are found. We consider the distance between two consecutive positions to be equal to 1. Since the sequence is circular, position 1 comes immediately after the last position D; therefore, $t_{d_i}^i$ is the distance between the first copy of symbol i in a cycle and the last copy of the same symbol in the preceding cycle. Let $\overline{t_i}$ be the desired average distance between two consecutive copies of symbol *i* ($\overline{t_i} = D/d_i$). Note that for each symbol *i* in which $d_i = 1$, t_1^i is equal to $\overline{t_i}$. The objective is to minimise the metric called response time variability (RTV), which is defined by the sum of the square errors with respect to the $\overline{t_i}$ distances. This is given in the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \overline{t_i})^2 \qquad (1)$$

For example, let $n = 3$ with symbols A, B and C, and consider, $d_A = 3$, $d_B = 2$ and $d_C = 2$; thus, $D = 7$, $\overline{t_A} = 7/3$, $\overline{t_B} = 7/2$ and $\overline{t_C} = 7/2$. Any sequence such that contains symbol *i* ($\forall i$) exactly d_i times is a feasible solution. For instance, the sequence (A, B, A, C, B, A, C) is a feasible solution, where:

$$RTV = \left(\left(2 - 7/3\right)^2 + \left(3 - 7/3\right)^2 + \left(2 - 7/3\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) = 5/3$$

3. A Branch and Bound algorithm for the RTVP

Branch and Bound (B&B) is a general technique for solving combinatorial optimisation problems, as it is the RTVP. B&B is based on dividing (branching) a feasible region of the solution space into several subregions. The divisions of the (sub)regions of the solution space are represented in a tree, in which each node represents a (sub)region of the space. The terminal nodes of the tree are regions of the space that contains only one solution. To avoid examining all (sub)regions of the space of solutions, the tactics of pruning the nodes with not better bounds than the value of the best current solution found, and eliminating dominated or equivalent nodes are usually used. For a good overview of enumerative procedures in tree representations, see Pastor and Corominas (2000).

We propose an algorithm based on the B&B technique. In Sections 3.1 to 3.12 the characteristics of the proposed algorithm are explained. Finally, in Section 3.13 the fine-tuning of the algorithm parameters is shown.

3.1. Objective

The objective is to obtain one solution that minimises the RTV value defined by Eq. 1

3.2. Initial solution

The RTV value of an initial solution is used as an initial upper bound of the optimal solution. The initial solution is generated as follows.

First, a solution is obtained by applying the Reduced Variable Neighbourhood Search (RVNS) algorithm proposed in Corominas *et al.* (2009b), which is the best heuristic method published in the literature to solve the RTVP. The RVNS algorithm works as follows. The start solution is generated using the lottery scheduling (Waldspurger and Weihl, 1994) as it is done in previous works published in the literature when an start solution is required. That is, for each position, a symbol to be sequenced is randomly chosen. The probability of each symbol is equal to the number of copies of this symbol that remain to be sequenced divided by the total number of units that remain to be sequenced. The neighbourhoods (N_1, N_2 and N_3) used in this RVNS algorithm are the following. N_1 is generated by swapping each pair of two consecutive positions of the sequence. N_2 is a generalisation of N_1 where the move is not restricted to consecutive positions. N_3 is generated by inserting each position in the sequence. At each iteration, a neighbour of the current solution is obtained at random from the current neighbourhood. If the neighbour is worse, then the neighbourhood is changed to the following neighbourhood; otherwise, the current neighbourhood is changed to the first one (N_1). The stop condition of the RVNS algorithm used in this paper is that the current solution does not improve in a certain number of consecutive iterations. A previous computational experiment has shown that 500.000 iterations are far enough for the convergence of the RVNS algorithm. The pseudo-code of the algorithm is shown in Figure 1.

```
1. Let $S$ an initial solution
2. $k := 1$
3. $it := 0$
4. While $it < 500.000$ do:
5.      $it := it + 1$
6.     Select a solution $S'$ at random from $N_k$
7.     If $RTV(S) \leq RTV(S')$ then:
            $S := S'$ and $k := 1$;
            If $RTV(S) < RTV(S')$ then $it := 0$
       Otherwise:
            $k := (k \bmod 3) + 1$
8. End While
9. Return $S$
```

Figure 1. Pseudo-code of the RVNS algorithm

The RVNS algorithm is able to obtain a good solution very quickly, although it does not ensure that the returned solution is a local optimum with respect to the neighbourhoods N_1, N_2 and N_3. Thus, we apply a composite hill climbing method to improve (if possible) the solution returned by the RVNS algorithm. The composite hill climbing method consists of applying iteratively three local search procedures (*LS1*, *LS2* and *LS3*) to the solution until there is no improvement of the solution (i.e., until a local optimum with respect all three neighbourhoods is reached). The local searches are performed iteratively as follows: the best solution in the neighbourhood is chosen at each iteration; the optimisation ends when no neighbouring solution is better than the current solution. The neighbourhoods used by *LS1*, *LS2* and *LS3* are N_1, N_2 and N_3, respectively.

3.3. Nodes of the tree

The algorithm starts from an empty sequence and, at each iteration, allocates a copy of one symbol that still has unallocated copies to the first empty position. Therefore, at each node of the tree we have a partially filled sequence, excepting at the terminal nodes, where we have a complete sequence.

3.4. Preprocessing to avoid examining equivalent nodes

Note that a RTVP solution is defined by the relative distances, for each symbol *i*, between the consecutive copies of symbol *i* instead of the absolute positions in which the copies are sequenced. The two following preprocesses are applied to avoid examining equivalent nodes.

a) A copy of the symbol *i* with the largest d_i (number of copies to be sequenced), i^*, is fixed in the first position of the sequence. The aim is to avoid some equivalent isomorphic solutions. Two solutions are isomorphic if one sequence can be reduced to the other by means of a partial rotation or if considering one of them clockwise is equal to the other considered counterclockwise.

   For example, let symbol A the one with the largest number of copies to be sequenced (i^* = A). Then, the sequence (A, B, A, C, B, A, C) will be generated, but not the sequences (B, A, C, B, A, C, A) and (C, A, B, C, A, B, A). Note that the second one can be reduced to the first one by means of a partial rotation, and the third one can be reduced to the first one considered counterclockwise.

b) To replace the symbols that have only one copy to be sequenced by a fictitious symbol. The number of times that the fictitious symbol is sequenced is equal to the number of symbols that have only one copy to be sequenced. The copies of the fictitious symbol will not contribute in the RTV value of the sequence.

   For example, the sequences (A, B, **D**, A, C, B, **E**, A, C) and (A, B, **E**, A, C, B, **D**, A, C) are equivalent. But if the suggested preprocess is applied, only the equivalent sequence (A, B, *****, A, C, B, *****, A, C) will be generated, where * is the fictitious symbol.

3.5. Branching

Given a non terminal node, its child nodes are generated by allocating one copy of the symbols that still have unallocated copies at the first free position of the sequence.

3.6. Node selection

A dynamic search strategy is used for selecting the next node to be explored (Pastor and Corominas, 2000). According to the current available RAM memory, the following two possibilities are applied:

a) If the available RAM is over than certain threshold, the search strategy consists of selecting the node that has the minimum value according to the following expression:

$$\eta \cdot LB + \lambda \cdot UB + \mu \cdot \upsilon \qquad (2)$$

where η, λ, μ are parameters, LB is the lower bound of the node, UB is the upper bound of the node and υ is the number of empty positions in the partial sequence corresponding to the node.

b) If the available RAM is below than certain threshold, the search strategy consists of selecting the node that has the minimum value according to Eq. 2 and then applying a depth-first search in the subtree that has the selected node as root (i.e., in the exploration of that subtree the values of the parameters are $\eta = \lambda = 0$ and $\mu = 1$).

3.7. Bounding

A node is pruned when the following condition is true:

$$LB > \left(RTV^* - 2\right) \tag{3}$$

where RTV* is the RTV value of the current best solution found (i.e., the global upper bound of the problem). The reason is because the difference between the RTV values corresponding to any pair of feasible solutions is an even integer number (see Corominas *et al*., 2007).

3.8. Lower bound (LB)

The following lower bound was suggested in Corominas *et al.* (2007). Let a decomposition vector of D into d_i components be defined as follows: $\lambda_i = \left(\lambda_1, ..., \lambda_{d_i}\right)$ of d_i positive integers that add up to D and $\lambda_1 \geq ... \geq \lambda_{d_i}$. The components of vector λ_i are the distances between the d_i copies of symbol i. Thus, the minimum RTV value for symbol i, RTV_i, can be obtained when $D \bmod d_i$ and $d_i - D \bmod d_i$ components of λ_i are equal to $\left\lceil \frac{D}{d_i} \right\rceil$ and $\left\lfloor \frac{D}{d_i} \right\rfloor$, respectively. For example, let $D = 24$, $n = 4$, $d = (9, 7, 5, 3)$ and $\overline{t} = (2.67, 3.43, 4.8, 8)$. The decomposition vectors $\lambda_1 = (3,3,3,3,3,3,2,2,2)$, $\lambda_2 = (4,4,4,3,3,3,3)$, $\lambda_3 = (5,5,5,5,4)$ and $\lambda_4 = (8,8,8)$ provide the minimum values of RTV_i $(i = 1,...,4)$. A lower bound on the value of RTV_i, $RTVLB_i$, and a lower bound on the value of RTV, $RTVLB$, can be defined as follows: $RTVLB_i = \left(D \bmod d_i\right) \cdot \left(\left\lceil \frac{D}{d_i} \right\rceil - \bar{t}_i \right)^2 + \left(d_i - D \bmod d_i\right) \cdot \left(\left\lfloor \frac{D}{d_i} \right\rfloor - \bar{t}_i \right)^2$ and $RTVLB = \sum_{i=1}^{n} RTVLB_i$:

$$RTVLB = \left[6 \cdot (3 - 2.67)^2 + 3 \cdot (2 - 2.67)^2\right] + \left[3 \cdot (4 - 3.43)^2 + 4 \cdot (3 - 3.43)^2\right]$$
$$+ \left[4 \cdot (5 - 4.8)^2 + 1 \cdot (4 - 4.8)^2\right] + \left[0 \cdot (8 - 8)^2 + 3 \cdot (8 - 8)^2\right] = 4.51$$

In this case, however, a lower bound, PLB, is needed for a partial solution, that is, a solution in which the first *k* positions of the sequence has been filled.

A lower bound for a partial solution, PS, can be obtained by adding, for all symbols in which $d_i \geq 2$, the sum of RTV_{PS} (the value associated with the distances between the copies of the symbols allocated in [1,…, *k*], if any) and RTV_{REM} (a bound corresponding to the assignment of the remaining copies, if any, to the free positions).

Let i be a non-fictitious symbol whose copies have not all been assigned in the partial solution PS. Three cases must be distinguished:

- Case 1. No copy of symbol i has been assigned in the *k* positions: We must distribute D positions among d_i distances (between two copies of symbol i), guaranteeing that one distance be greater than or equal to $k+1$.
- Case 2. Only one copy of symbol i has been assigned to position h ($\leq k$): We must distribute D positions among d_i distances, guaranteeing that one distance be greater than or equal to $k-h+1$ and another be greater or equal to *h*.
- Case 3. *p* copies of symbol i have been assigned in the *k* positions, the first in the sequence in position h_f and the last one in position h_l: We must distribute $D-h_l+h_f$ positions among d_i-p+1 distances, guaranteeing that one distance be greater than or equal to $k-h_l+1$ and another be greater than or equal to h_f. Case 2 can be reduced to Case 3 taking into account that $h_f=h_l=h$ and $p=1$. Case 1 can be reduced to Case 3 taking into account that $h_f=h_l=h=0$ and $p=1$.

Thus, the problem consists in distributing $D-h_l+h_f$ “units of distance” among d_i-p+1 distances t_j^i $\left(j=1,...,d_i-p+1\right)$, taking into account that two distances are lower bounded by $k-h_l+1$ and h_f, respectively, and the others are lower bounded by 1, with the objective of minimising a function of the discrepancy between the distances and the average distance $\overline{t_i}$. That is, it is the apportionment problem with lower bounds. Bautista *et al*. (2001) propose a general optimisation procedure for a convex, nonnegative (symmetric or not) discrepancy function and such that $f\left(0\right)=0$. For the discrepancy function considered here (the quadratic discrepancy), the resulting procedure is as follows:

```
$t_1^i = k - h_l + 1$
$t_2^i = h_f$
for $j = 3$ to $d_i - p + 1$
        $t_j^i = 1$
next $j$
for $j = 1$ to $D - k + p - d_i$    $\left(= D - (h_f + k - h_l + 1) - (h_l - h_f) - (d_i - p - 1)\right)$
        find $s^* \left| t_{s^*}^i = \min_s \left(t_s^i\right)\right.$; malot $t_{s^*}^i = t_{s^*}^i + 1$
next $j$
```

For the instance defined by $n=4$ and $d=(9,7,5,3)$ and the partial solution $PS=(\text{A,C,B,A,A,C,C,C,A,,,,,,,,,,,,,,,})$, we have:

$$RTV_{PS}=\left[(3-2.67)^2+(1-2.67)^2+(4-2.67)^2\right]+\left[(4-4.8)^2+(1-4.8)^2+(1-4.8)^2\right]=34.19$$

And, applying the procedure described above, the distances (3,3,3,3,2,2), (7,3,3,3,3,3,2), (9,9) and (10,7,7) are obtained for the symbols A, B, C and D, respectively. The value corresponding to these distances, RTV_{REM}, is:

$$RTV_{REM}=\left[4\cdot(3-2.67)^2+2\cdot(2-2.67)^2\right]+\left[1\cdot(7-3.43)^2+5\cdot(3-3.43)^2+1\cdot(2-3.43)^2\right]$$
$$+\left[2\cdot(9-4.8)^2\right]+\left[1\cdot(10-8)^2+2\cdot(7-8)^2\right]=58.33$$

And, finally, $PLB=RTV_{PS}+RTV_{REM}=34.19+58.33=92.52$

3.9. Upper bound (UB)

To calculate the upper bound of a node, the partial solution associated to the node is completed applying a constructive heuristic based on the priority rule proposed in Corominas *et al*. (2008), and the RTV value of the completed solution is taken as the upper bound.

The constructive heuristic that we propose works as follows. Let x_{ik} be the number of copies of symbol i that have been already sequenced in the sequence of length k, k = 0, 1, … (asumme x_{i0} = 0); the symbol to be sequenced in position k + 1 is $i^*=\arg\max_i\left\{\frac{(k+1)\cdot d_i}{D}-x_{ik}\right\}$. If there is a tie, then the symbol i with the lowest d_i is sequenced. If there is also a tie, then use lexicographical order.

In the case that the obtained full solution is better than the best current solution found, then the global upper bound (RTV*) is updated.

3.10. Improving the solutions of terminal nodes

When a terminal node is examined, the composite hill climbing method explained in Section 3.2 is applied to the solution associated with the node. In the case that the obtained improved solution is better than the best current solution found, then the global upper bound (RTV*) is updated.

3.11. Dominances

We will say that a node A dominates another node B if all the following conditions are fulfilled:

a) The partial sequences of both nodes contain the same total number of copies.

b) The partial sequences of both nodes contain the same number of copies of the fictitious symbol.

c) Excluding the fictitious symbol and the symbols such that all their copies are included in the partial sequence: the first and last positions occupied for each symbol present in the partial sequences coincide in both nodes.

d) The partial RTV value (RTV_{PS}) for node A is less (strict dominance) or equal (non-strict dominance) than for node B.

Note that the definition includes the possibility of mutual (non-strict) dominance.

When A dominates strictly B, the latter can be fathomed. When there is a mutual dominance between A and B, any of them (but not both, of course) can be fathomed.

3.12. Avoiding examining equivalent isomorphic nodes

A first preprocess method to avoid examining equivalent solutions is proposed in Section 3.4. In this section, we propose a method to avoid examining other isomorphic solutions during the search.

In order to make more understandable the explanation, the following simple example is introduced: let n = 3; d_A = 5, d_B = 4 and d_C = 3; thus, D = 12. The exploration tree that will be generated (taking into account the preprocess methods explained in Section 3.4) is shown in Figure 2.

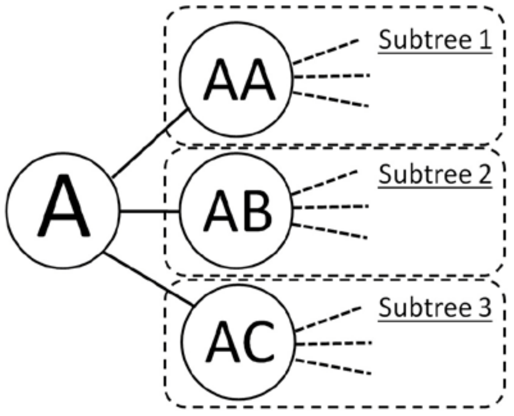


Figure 2. Exploration tree example

All sequences that start with subsequence (A, A) are generated (explicitly or implicitly) from the subtree that has the node *AA* as its root (Subtree 1 in Figure 2).

Thus, there is not necessity to generate any sequence that contains the subsequence (A, A) from the subtree that has the node *AB* as its root (Subtree 2 in Figure 2). The reason is that each one of these type of solutions is symmetric with respect of one of the solutions generated in Subtree 1. For example, the sequence (A, B, ?, ?, ?, A, A, ?, ?, ?, ?, ?) is symmetric with respect to the sequence (A, A, ?, ?, ?, ?, ?, A, B, ?, ?, ?), which could be generated in Subtree 1, where ? is a copy of symbols A, B or C.

Similarly, there is not necessity to generate any sequence that contains any of the sequences (A, A), (A, B) or (B, A) from the subtree that has the node *AC* as its root (Subtree 3 in Figure 2). For example, the sequence (A, ?, ?, ?, ?, ?, ?, B, A, ?, ?, ?) is symmetric with respect the sequence (A, B, ?, ?, ?, ?, ?, ?, A, ?, ?, ?), which could be generated in Subtree 2.

The idea to avoid generating symmetries has been exemplified using *tabu subsequences* with a length equal to 2. This idea can be generalized for a length equal to or greater than 2.

Before formalizing the introduced idea, let first introduce some additional nomenclature. Let $s = s_1 s_2 \ldots s_L$ and $s' = s'_1 s'_2 \ldots s'_L$ be two (sub)sequences of length *L*, where s_j and s'_j are the copies sequenced in position *j* of sequences *s* and *s*', respectively. We define $\underset{SYM}{<}$ and $\underset{SYM}{=}$ as two comparison operators between symbols, where $s_j \underset{SYM}{<} s_k$ returns if symbol s_j is less than s_k, and $s_j \underset{SYM}{=} s_k$ returns if symbol s_j is the same symbol than s_k; the order of the symbol set used is the lexicographical order (the order criterion will not have influence in the performance of the algorithm). We also define $\underset{SEQ}{<}$ as a comparison operator between two (sub)sequences of the same length, where $s \underset{SEQ}{<} s'$ is defined as follows:

$$s \underset{SEQ}{<} s' = \left\{ \begin{array}{cc} false & \text{, if } L = 0 \\ \left[SYM(s_1) \underset{SYM}{<} SYM(s'_1)\right] \vee & \\ \left[\left(SYM(s_1) \underset{SYM}{=} SYM(s'_1)\right) \wedge \left(s_2 s_3 \ldots s_L \underset{SEQ}{<} s'_2 s'_3 \ldots s'_L\right)\right] & \text{, if } L \geq 1 \end{array} \right\},$$

where SYM(s_j) returns the symbol of copy s_j.

Let suppose that the length of the tabu subsequences is *L*. Given a node *N* of the exploration tree of the level *L*, let its associated partial sequence be $sn = sn_1 sn_2 \ldots sn_L$, where sn_1 is a copy of the symbol with the largest number of copies to be sequenced, i^* (see Section 3.4). The set of tabu subsequences of node *N*, *TS*(*N*), is defined as follows:

$$TS(N) = TS_1(N) \cup TS_2(N) \tag{4}$$

$$TS_1(N) = \left\{ \begin{array}{c} sn_1 s_2 \ldots s_L : \left[\underset{j=2..L}{\forall} SYM(s_j) \in \{1..n\}\right] \wedge \left[\underset{i=1..n:i \neq i^*}{\forall} \left(\sum_{j=2..L:SYM(s_j)=i} 1 \leq d_i \right)\right] \wedge \\ \left[1 + \sum_{j=2..L:SYM(s_j)=i^*} 1 \leq d_{i^*}\right] \wedge \left[sn_1 s_2 \ldots s_L \underset{SEQ}{<} sn_1 sn_2 \ldots sn_L\right] \end{array} \right\} \tag{4'}$$

$$TS_2(N) = \left\{ s_L s_{L-1} \ldots s_2 sn_1 : \left[sn_1 s_2 \ldots s_{L-1} s_L \in TS_1(N)\right] \wedge \left[s_L \neq i^*\right] \right\} \tag{4''}$$

The definition of the set $TS_1(N)$ includes as tabu subsequences all subsequences that are feasible (that is, the symbols allocated are comprised between 1 and n, and the number of times that each symbol i is sequenced is not greater than d_i) and are *lower* (according to the operator $\underset{SEQ}{<}$) than the partial sequence *sn*. The definition of the set $TS_2(N)$ includes all sequences of $TS_1(N)$ considered counterclockwise in which the symbol of its last copy is different of symbol i^*.

For example, let $L = 4$, $n = 3$, $A \underset{SYM}{<} B \underset{SYM}{<} C$ and $d_A = 10$, $d_B = 6$, $d_C = 9$; thus, $i^* = A$. The set of tabu subsequences of node *ABAC* is {(A, A, A, A), (A, A, A, B), (A, A, A, C), (A, A, B, A) , (A, A, B, B), (A, A, B, C), (A, A, C, A) , (A, A, C, B), (A, A, C, C), (A, B, A, A) , (A, B, A, B)} $\cup$ {(B, A, A, A), (C, A, A, A), (B, B, A, A), (C, B, A, A), (B, C, A, A), (C, C, A, A), (B, A, B, A)}

During the branching procedure, only the child nodes whose associated sequence has not any tabu subsequence are examined. The more greater is the value of L, the less equivalent solutions will be examined although the cpu time needed to check if a partial solution contains a tabu subsequence increases exponentially. Thus, the suitable value of the parameter L has to be set empirically.

3.13. Fine-tuning the algorithm parameters

Fine-tuning the parameters of an algorithm is almost always a difficult task. Although the parameter values may have a very strong effect on the performance of the algorithm for each problem, they are often selected using one of the following methods, which are not sufficiently thorough (Eiben et al., 1999; Adenso-Díaz and Laguna, 2006): 1) “by hand”, based on a small number of experiments that are not referenced; 2) using the general values recommended for a wide range of problems; 3) using the values reported to be effective in other similar problems; or 4) with no apparent explanation.

Adenso-Díaz and Laguna (2006) proposed a new technique called CALIBRA for fine-tuning the parameters of algorithms. CALIBRA is based on using conjointly Taguchi’s fractional factorial experimental designs and a local search procedure.

We propose to use CALIBRA for setting the parameter values of our algorithm. CALIBRA was applied to a training set of 30 instances. These instances were generated as follows (the same way that is used in the literature; e.g., Corominas et *al*., 2010). D was randomly selected with a discrete uniform distribution between 20 and 30, between 30 and 35 and between 35 and 40, for instances T1 to T10, T11 to T20 and T21 to T30, respectively. For instances T1 to T10, n and d_i were randomly selected with a discrete uniform distribution between 3 and $\lceil D/2 \rceil$ and between 1 and $\lceil (D-n+1)/2 \rceil$ (with $\sum_{i=1}^{n} d_i = D$), respectively. For instances T11 to T30, n and d_i were randomly selected with a discrete uniform distribution between 3 and 12 and between 1 and $\lceil (D-n+1)/2.5 \rceil$ (with $\sum_{i=1}^{n} d_i = D$), respectively. To evaluate the effectiveness of a given set of parameter values we consider the number of the training instances that can be solved optimally within 2,000 seconds. To break the tie the average computing time spent for solving all training instances is used. The following parameter values were returned by CALIBRA:

- $\eta = 1$, $\lambda = 0$, $\mu = 675$ (parameters defined in Section 3.6)
- $L = 3$ (parameter defined in Section 3.12)

CALIBRA points that the UB value is useless for the node selection. With the values of the parameters returned by CALIBRA, Eq. 2 takes the form $LB + 675 \cdot \upsilon$. Thus, in the final proposed B&B algorithm the UB of each node is not calculated to save cpu time. Notice that using the η, λ and μ values returned by CALIBRA is almost equivalent to use a depth-first node selection strategy breaking the ties with the LB value.

4. Computational experiment

We first solved the 120 instances used in Corominas *et al*. (2010). These instances were generated as follows. D was randomly selected with a discrete uniform distribution between 20 and 30, between 30 and 35 and between 35 and 40, for instances 1 to 40, 41 to 80 and 81 to 120, respectively. For instances 1 to 40, n and d_i were randomly selected with a discrete uniform distribution between 3 and $\lceil D/2 \rceil$ and between 1 and $\lceil (D-n+1)/2 \rceil$ (with $\sum_{i=1}^{n} d_i = D$), respectively. For instances 41 to 120, n and d_i were randomly selected with a discrete uniform distribution between 3 and 12 and between 1 and $\lceil (D-n+1)/2.5 \rceil$ (with $\sum_{i=1}^{n} d_i = D$), respectively.

The B&B algorithm was coded in Java and the computational experiment was carried out on a PC 3.00 GHz Intel Pentium IV with 1.5 GB of RAM. Because the Corominas *et al*. (2010) computational experiment was carried out in a slower machine, their computing times showed in this paper are multiplied by a corrective factor in order to be compared fairly with our computing times. The applied corrective factor is 0.5 and it is calculated according to the public CPU benchmark provided by PassMark Software (http://www.cpubenchmark.net/).

Table 1 summarises the results obtained with a maximum calculation time of 10,000 seconds for each instance. The columns *#Opt* and *#Fea* show the number of instances that have been solved optimally and the number of instances in which a solution has been found but its optimality has not been demonstrated, respectively. The average computing time (in seconds) is shown between parentheses.

Table 1. Comparison between the MILP and the proposed B&B method

	#Opt	***#Fea***
MILP	114 (278 s.)	6 (10,000 s.)
B&B	114 (7.47 s.) + 6 (316.21 s.)	0

The results show that the B&B algorithm is able to solve all 120 instances including the six ones that could not be solved with the MILP method. Moreover, the computing time needed to solve the instances presents a huge improvement. For the 116 instances that has been solved by the two methods, the MILP method needs, on average, 278 seconds whereas the proposed B&B method only needs 7.47 seconds. On the other hand, the 6 instances solved by the B&B algorithm but not by the MILP model need, on average, 316.21 computing seconds.

We have expanded the computational experiment solving larger instances with the B&B algorithm. The instances were generated as follows. D was randomly selected with a discrete uniform distribution between 40 and 45, between 45 and 50, between 50 and 55, between 55 and 60 and between 60 and 65 for instances 121 to 160, 161 to 200, 201 to 240, 241 to 280 and 281 to 320, respectively. The n and d_i values are generated as it was done for instances 41 to 120 (all test and training instances can be downloaded at https://www.ioc.upc.edu/EOLI/research/)

Table 2 shows the results obtained for instances 1 to 320 with a maximum calculation time of 10,000 seconds for each instance. Column D shows the range size of the instances, column T shows the average time (in seconds) to solve an instance, column T_{S0} shows the time (in seconds) to obtain the initial solution (see Section 3.2), column *RTV* shows the average of the best RTV values found and column *#Opt* shows the number of instances that have been solved optimally.

Table 2. Results obtained with the B&B method

Instances	***D***	***T***	T_{S0}	***RTV***	***#Opt***
1-40	20-30	2.15	2.08	6.23	40
41-80	30-35	5.89	2.83	9.24	40
81-120	35-40	60.69	3.05	13.47	40
121-160	40-45	785.98	2.08	14.43	38
161-200	45-50	1,589.13	2.83	16.49	37
201-240	50-55	2,973.90	3.06	18.51	34
241-280	55-60	5,090.25	3.60	20.48	23
281-320	60-65	5,910.49	4.00	24.87	18

As it has been mentioned before, the B&B algorithm is able to solve all instances up to 40 copies to be sequenced. Between 40 and 45 copies, 45 and 50 copies and 50 and 55 copies the B&B algorithm solves the 95%, 92.5% and 85% of instances, respectively. For larger instances, the number of solved instances decrease quickly. However, the algorithm is still able to solve around 50% of instances that has between 55 and 65 copies to be sequenced.

Table 3 shows the results focused on the instances that have been optimally solved within 10,000 computing seconds, where column T_B shows the instant at which the best solution was found (in seconds) and column RTV^* shows the average of the optimal RTV values. As it is usual in exact methods, we can see that the most computing time for non-small instances is spent not on finding an optimal solution but on ensuring its optimality.

Table 3. Optimal results obtained with the B&B method

Instances	*T*	T_B	T_{S0}	*RTV**	*#Opt*
1-40	2.15	2.08	2.08	6.23	40
41-80	5.89	2.83	2.83	9.24	40
81-120	60.69	12.49	3.05	13.47	40
121-160	301.03	19.83	2.08	14.38	38
161-200	907.17	232.26	2.81	16.21	37
201-240	1,734.00	442.36	3.03	17.34	34
241-280	1,461.30	3.62	3.61	18.17	23
281-320	912.21	142.91	4.00	19.17	18

5. Conclusions

This paper deals with the exact solution of the response time variability problem (RTVP) by means of a branch and bound (B&B) algorithm. The RTVP is a scheduling problem that arises in a wide range of real-life problems. The exact solution of this problem is, in general, very difficult because it is NP-hard. A mathematical programming model proposed in Corominas *et al*. (2010) was the best exact method to solve it with a practical limit for obtaining optimal solution of 40 copies to be sequenced.

We have analysed the characteristics of the problem to propose a specially designed B&B algorithm. In particular, we have tried to avoid exploring dominated and equivalent solutions as much as possible. The proposed algorithm improves the best published exact method. All instances proposed in Corominas et *al*. (2010) are solved optimally and much faster. Moreover, the size of the instances that can be solved to optimality increased from 40 to 55 units. Thus, not only larger instances can be optimally solved but also it is useful to found new optimal solutions to the RTVP that can be used to compare the results obtained with heuristic and metaheuristic methods.

REFERENCES

Adenso-Díaz, B. and Laguna, M. (2006) 'Fine-tuning of algorithms using fractional experimental designs and local search', *Operations Research*, Vol. 54, pp. 99-114.

Anily, S., Glass, C.A. and Hassin, R. (1998) 'The scheduling of maintenance service', *Discrete Applied Mathematics*, Vol. 82, pp. 27-42.

Bautista, J., Companys, R. and Corominas, A. (1997) 'Modelling and solving the production rate variation problem (PRVP)', *TOP*, Vol. 5, pp. 221-239.

Bautista, J., Companys, R. and Corominas, A. (2001) 'Solving the generalized apportionment problem through the optimization of discrepancy functions', *European Journal of Operational Research*, Vol. 131, pp. 676-684.

Bollapragada, S., Bussieck, M.R. and Mallik, S. (2004) 'Scheduling Commercial Videotapes in Broadcast Television', *Operations Research*, Vol. 52, pp. 679-689.

Brusco, M.J. (2008) ‘Scheduling advertising slots for television’, *Journal of the Operational Research Society*, Vol. 59, pp. 1363-1372.

Corominas, A., García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics’, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 184, pp. 128-137.

Corominas, A., García-Villoria, A. and Pastor, R. (2009a) ‘A Parametric Multi-start Algorithm for Solving the Response Time Variability Problem’, 7th *International Conference Large-Scale Scientific Computations* (LSSC 2009), Sozopol, Bulgaria.

Corominas, A., García-Villoria, A. and Pastor, R. (2009b) ‘Solving the Response Time Variable Problem by means of a Variable Neighbourhood Search Algorithm’, *13th IFAC Symposium of Information Control Problems in Manufacturing* (INCOM 2009), Moscow, Russia.

Corominas, A., García-Villoria, A. and Pastor, R. (2009c) ‘Using Tabu Search for the Response Time Variability Problem’, *3rd International Conference on Industrial Engineering and Industrial Management* (CIO 2009), Barcelona and Terrassa, Spain.

Corominas, A., García-Villoria, A., Pastor, R. (2009d) ‘Resolución del response time variability problem mediante tabu search’, *VIII Evento Internacional de Matemática y Computación* (COMAT’2009), Universidad de Matanzas, Cuba.

Corominas, A., Kubiak, W. and Moreno, N. (2007) ‘Response time variability’, *Journal of Scheduling*, Vol. 10, pp. 97-110.

Corominas, A., Kubiak, W. and Pastor, R. (2010) ‘Mathematical programming modeling of the Response Time Variability Problem’, *European Journal of Operational Research*, Vol. 200, pp. 347-357.

Dong, L., Melhem, R. and Mosse, D. (1998) ‘Time slot allocation for real-time messages with negotiable distance constrains requirements’, *Fourth IEEE Real-Time Technology and Applications Symposium* (RTAS'98), Denver, CO. pp. 131-136.

Eiben, A.E., Hinterding, R. and Michalewicz, Z. (1999) ‘Parameter control in evolutionary algorithms’, *IEEE Transactions on evolutionary computation*, Vol. 3, pp. 124-141.

García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics’, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 146, pp. 187-194.

García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of a psychoclonal approach’, *Journal of Heuristics*, in press, corrected proof, available online, 16 July 2008, doi:10.1007/s10732-008-9082-2.

García-Villoria, A. and Pastor, R. (2009a) ‘Introducing dynamic diversity into a discrete particle swarm optimization’, *Computers & Operations Research*, Vol. 36, pp. 951-966.

García-Villoria, A. and Pastor, R. (2009b) ‘Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism’, *International Journal of Production Research*, doi: 10.1080/00207540902862545.

García-Villoria, A. and Pastor, R. (2010) ‘Solving the response time variability problem by means of a genetic algorithm’, *European Journal of Operational Research*, Vol. 202, pp. 320-327.

García-Villoria, A., Pastor, R. and Corominas, A. (2010) ‘Solving the Response Time Variability Problem by means of the Cross-Entropy Method’, *International Journal of Manufacturing Technology and Management*, Vol. 20, pp. 316-330.

Han, C.C., Lin, K.J. and Hou, C.J. (1996) ‘Distance-Constrained Scheduling and Its Applications in Real-Time Systems’ *IEEE Trans. on Computers,* Vol. 45, pp. 814-826.

Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report TR 2007-12*, University of Maryland, USA. Available at http://hdl.handle.net/1903/7082.

Herrmann, J.W. (2009) ‘Using aggregation to reduce response time variability in cyclic fair sequences’, *Journal of Scheduling*, doi 10.1007/s10951-009-0127-7.

Inman R.R. and Bulfin, R.L. (1991) ‘Sequencing JIT mixed-model assembly lines’, *Management Science*, Vol. 37, pp. 901-904.

Kubiak, W. (1993) ‘Minimizing variation of production rates in just-in-time systems: A survey’, *European Journal of Operational Research*, Vol. 66, pp. 259-271.

Kubiak, W. (2009) ‘Proportional optimization and fairness’, Springer.

Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 35, pp. 192-207.

Monden, Y. (1983) ‘Toyota Production Systems’, *Industrial Engineering and Management Press*, Norcross, GA.

Pastor, R. and Corominas, A. (2000) ‘Branch and Win: OR Tree Search Algorithms for Solving Combinatorial Optimisation Problems’, *TOP*, Vol. 12, pp. 169-191.

Steiner, G. and Yeomans S. (1993) ‘Level Schedules for Mixed-Model, Just-in-Time Processes’, *Management Science*, Vol. 39, pp. 728-735.

Waldspurger, C.A. and Weihl, W.E. (1994) ‘Lottery Scheduling: Flexible Proportional-Share Resource Management’, *First USENIX Symposium on Operating System Design and Implementation*.

Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Scheduling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528*, Massachusetts Institute of Technology, MIT Laboratory for Computer Science.

Wei, W.D. and Liu, C.L. (1983) ‘On a periodic maintenance problem’, *Operations Research Letters*, Vol. 2, pp. 90-93.

Using an Ant Colony System to solve the Response Time Variability Problem[†]

Albert Corominas, Alberto García-Villoria and Rafael Pastor

Universitat Politècnica de Catalunya, IOC Research Institute, Av. Diagonal 647, Edif. ETSEIB, p.11, 08028 Barcelona, Spain
{albert.corominas, alberto.garcia-villoria, rafael.pastor}@upc.edu

Abstract. The response time variability problem (RTVP) is a combinatorial optimization problem which has recently appeared in the literature. This problem has a wide range of real-life applications in mixed-model assembly lines, multi-threaded computer systems, network environments and others. The RTVP arises whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the points at which they receive the necessary resources is minimized. This problem is very complex. Swarm intelligence research has proposed some metaheuristics for solving complex optimization problems: among others, particle swarm optimization (PSO) and ant colony optimization (ACO). A PSO algorithm called *DPSOpoi-$c_p$dyn* has been proposed in the literature to solve efficiently the RTVP. We propose an ACS algorithm—which is an ACO variant—for solving the RTVP. A computational experiment is carried out and it is shown that, on average, the ACS algorithm produces better results than *DPSOpoi-$c_p$dyn*.

Keywords: response time variability, scheduling, ant colony optimization, ant colony system, swarm intelligence

1. Introduction

The *fair sequence* concept emerged independently of scheduling problems in a range of environments including, among others, mixed-model assembly lines, multi-threaded computer systems and network environments. The common aim of these scheduling problems is to build a fair sequence using *n* symbols, where symbol *i* ($i = 1,...,n$) must occur d_i times in the sequence. The fair sequence is the one which allocates a fair share of positions to each symbol *i* in any subsequence. This fair or ideal share of positions allocated to symbol *i* in a subsequence of length *k* is proportional to the relative importance (d_i) of symbol *i* with respect to the total copies of competing symbols (equal to $\sum_{i=1..n} d_i$). There is no a universal definition of fairness because several reasonable metrics can be defined according to the specific problem considered.

In the response time variability problem (RTVP), the fair sequence is the one which minimizes variability in the distances between any two consecutive copies of the same symbol [1]. In other words, the distance between any two consecutive copies of the same symbol should be as regular as possible (ideally, constant).

The RTVP has a broad range of real-life applications. For example, it can be used to regularly sequence models in the automobile industry [2], to resource allocation in computer multi-threaded systems and

[†] Supported by the Spanish Ministry of Education and Science under project DPI2007-61905; co-funded by the FEDER.

network servers [3, 4], in the periodic machine maintenance problem [5], in the collection of waste [6] and to broadcast commercial videotapes in television [7].

The RTVP is NP-hard [1]. Eleven algorithms based on the particle swarm optimization (PSO) metaheuristic have been proposed in the literature for solving the RTVP [8, 9]. The PSO metaheuristic (as well as the ant colony optimization (ACO) metaheuristic) falls under the scope of *swarm intelligence* research, which studies algorithms inspired by the observed behaviour of swarms [10]. Swarm behaviour is determined by the interactions of single agents in the group who exchange information with the rest of the group.

In this paper we propose to use an ACO approach to solve the RTVP. The ACO metaheuristic is inspired by the behaviour of ants when searching for food and was introduced by Dorigo [11] as a tool for solving hard combinatorial optimization problems (COPs) such as the RTVP. Real ants secrete pheromones that they put on the ground as they move between the anthill and sources of food. Ants smell the pheromones and tend to follow a trail according to its intensity. In ACO algorithms, artificial ants 'walk' over the COP states which are visited by other ants while building solutions. The path taken by an ant is a sequence of states that represents a solution to the problem. Each state is associated with a pheromone trail which is represented by a numeric value. The pheromone trails for each state are modified iteratively according to the fitness of the paths (solutions) built by the ants. At each step, an ant chooses a state of the problem using a probability which depends on its associated pheromone trail.

Several variants of ACO have been developed in the literature, including Ant System [11], Ant-Q [12], Ant Colony System [13], MAX-MIN Ant System [14], and Rank-Based Ant System [15]. These variants differ principally in the procedure by which the pheromone trail is updated [16]. In this paper, the ant colony system (ACS) is used to solve the RTVP because it is one of the most successful ACO variants in practice [16].

The proposed ACS algorithm for solving the RTVP was compared with the best PSO algorithm published in the literature, which is called *DPSOpoi-c$_p$dyn* [9]. On average, the proposed ACS algorithm improves on the best previous results.

The remainder of the paper is organized as follows: Section 2 presents a formal definition of the RTVP and briefly presents the *DPSOpoi-c$_p$dyn* algorithm. Section 3 describes the basic scheme of the ACS metaheuristic. Section 4 proposes an ACS algorithm for solving the RTVP. Section 5 presents the computational experiments and the comparison between our algorithm and *DPSOpoi-c$_p$dyn*. Finally, some conclusions are given in Section 6.

2. The Response Time Variability Problem (RTVP)

The RTVP is designed to minimize variability in the distances between any two consecutive copies of the same symbol and is formulated as follows. Let n be the number of symbols, d_i the number of copies of the symbol i to be scheduled (i = 1,…,n), and D the total number of copies ($D = \sum_{i=1..n} d_i$). Let s be a solution of an instance in the RTVP that consists of a circular sequence of copies ($s = s_1 s_2 \dots s_D$), where s_j is the copy sequenced in position j of sequence s. For all symbol i in which $d_i \geq 2$, let t_k^i be the distance between the positions in which copies k + 1 and k of symbol i are found (i.e. the number of positions between them, where the distance between two consecutive positions is considered equal to 1). Since the sequence is circular, position 1 comes immediately after position D; therefore, $t_{d_i}^i$ is the distance between the first copy of symbol i in a cycle and the last copy of the same symbol in the preceding cycle. Let $\bar{t}_i$ be the average distance between two consecutive copies of symbol i ($\bar{t}_i = D/d_i$). For all symbol i in which d_i =1, t_1^i is equal to $\bar{t}_i$. The aim is to minimize the metric response time variability (RTV) which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \overline{t}_i)^2 \quad \textbf{(1)}$$

For example, let $n = 3$, $d_A = 3$, $d_B = 2$ and $d_C = 2$; thus, $D = 7$, $\overline{t}_A = 7/3$, $\overline{t}_B = 7/2$ and $\overline{t}_C = 7/2$. Any sequence such that contains exactly d_i times the symbol i $(\forall i)$ is a feasible solution. For example, the sequence (A, B, A, C, B, A, C) is a feasible solution, where: $RTV = \left(\left(2 - 7/3\right)^2 + \left(3 - 7/3\right)^2 + \left(2 - 7/3\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) + \left(\left(3 - 7/2\right)^2 + \left(4 - 7/2\right)^2 \right) = 5/3$.

As explained above, the best PSO algorithm for solving the RTVP is *DPSOpoi-$c_p$dyn* [9]. PSO is a populational metaheuristic algorithm designed by Kennedy and Eberhart [17] which is based on swarm intelligence obtained from the observed social behaviour of flocks of birds when they search for food. The population or swarm is composed of particles (birds), whose attributes are an *m*-dimensional real point (which represents a feasible solution) and a velocity (the movement of the point in the *m*-dimensional real space). The velocity of a particle is typically a combination of three types of velocities: 1) the inertia velocity (i.e., the previous velocity of the particle); 2) the velocity to the best point found by the particle; and 3) the velocity to the best point found by the swarm. These components of the particles are modified iteratively by the algorithm as it searches for an optimal solution. Although the PSO algorithm was originally designed for *m*-dimensional real spaces, *DPSOpoi-$c_p$dyn* is adapted to work with a sequence that represents the solution. Moreover, *DPSOpoi-$c_p$dyn* introduces random modifications to the points of the particles with a frequency that changes dynamically according to the homogeneity of the swarm (for more details, see [9]).

3. ACO and Ant Colony System

The ACO metaheuristic was initially designed by Dorigo to solve the traveling salesman problem [11]. ACO algorithms have been used to solve many combinatorial optimization problems (COPs) [10]. ACO was initially inspired by the biological behaviour of ants but was soon modified to solve COPs more efficiently. ACO differs from real ants in the following ways [10]: 1) artificial ants move through a discrete environment (i.e., through a finite set of states of the problem); 2) heuristic information is also considered when the solutions are being built; 3) the pheromone update is performed only by some ants and often after a solution has been constructed; and 4) ACO may include artificial mechanisms such as local search and look-ahead.

The first step in solving a COP with ACO is to associate a graph *G* = (*N*, *E*), called *construction graph*, with the problem. The nodes in the set *N* are usually components of the solution, and the artificial ants build a solution incrementally by moving from node to node along the edges of the set *E*. Each edge has an associated pheromone trail value and a heuristic value. The ants combine the pheromone and the heuristic information to select the next edge probabilistically. Fig. 1 shows a classical scheme of the ACO metaheuristic, which consists in setting an initial value (τ_0) for each pheromone trail and then looping over the following three components until a stop condition is reached: 1) the construction of a solution by the ants; 2) a local search from some or all the solutions (this component is optional); and 3) the update of the pheromone trail values.

```
1. Set the values of the ACO parameters
2. Initialize the pheromone trail values
3. While stopping condition is not reached do:
4.         ConstructAntSolutions
5.         ApplyLocalSearch [optional]
6.         UpdatePheromones
7. End While
```

Fig. 1. Scheme of the ACO metaheuristic

The first ACO metaheuristic proposed in the literature was Ant System [11]. Several other ACO metaheuristics have been introduced to improve the performance of Ant System. All ACO metaheuristics use the scheme shown in Fig. 1, but they contain different definitions for constructing solutions and updating pheromones (Steps 4 and 6 in Fig. 1). For more extensive information about ACO, see the book by Dorigo and Stützle [18].

Dorigo and Blum [16] found that ACS is one of the most successful ACO metaheuristics in practice. Therefore, we decided to use ACS to solve the RTVP. In the following sections we explain how the solution construction, the local search and the pheromone update are applied in ACS [10].

ConstructAntSolutions. Given a construction graph $G = (N, E)$, each ant constructs a solution starting with an empty partial solution s^p. Then, a component from N is added to s^p at each construction step until the solution is complete. The next component to be added is determined by selecting at random an edge from the set $E(s^p)$, which is the subset of E composed of the eligible edges for the partial solution s^p. The probability that an edge e_{ij} (where i is the last component added to s^p) will be chosen is given by the following equation:

$$p(e_{ij} \mid s^p) = \begin{cases} 1 & \text{if } q \le q_0 \text{ and } j = \underset{k \in N \mid e_{ik} \in E(s^p)}{\arg\max}\ \tau_{ik}^{\alpha} \cdot \eta(e_{ik})^{\beta} \\ 0 & \text{if } q \le q_0 \text{ and } j \ne \underset{k \in N \mid e_{ik} \in E(s^p)}{\arg\max}\ \tau_{ik}^{\alpha} \cdot \eta(e_{ik})^{\beta} \\ \dfrac{\tau_{ij}^{\alpha} \cdot \eta(e_{ij})^{\beta}}{\sum\limits_{e_{ik} \in E(s^p)} \tau_{ik}^{\alpha} \cdot \eta(e_{ik})^{\beta}} & \text{if } q > q_0 \end{cases} \qquad \textbf{(2)}$$

where q is a random number distributed uniformly over [0,1], q_0 is a parameter of ACS, τ_{ij} is the pheromone trail values associated with the edge e_{ij}, $\eta(e_{ij})$ is the heuristic information that indicates how desirable it is to choose the edge e_{ij}, and α and β are two positive parameters of ACS that weight the importance of the pheromone value and the heuristic information, respectively.

ApplyLocalSearch. Optional actions called *daemon actions* could be performed once the solutions have been constructed. The most commonly used daemon action is to apply a local search to the solutions. Although this component is optional, ACO algorithms and their variants perform better if a local search is applied [19].

UpdatePheromones. Pheromones are updated according to the locally optimized solutions. This component is designed to increase the pheromone trail values associated with the edges used by good solutions and to decrease the pheromone trail values associated with the edges used by bad solutions. ACS applies two pheromone updates: the offline pheromone update and the local pheromone update. The offline pheromone update is applied at the edges belonging to the best solution *bs* (either the best current solution or the best solution found by the algorithm) using the following formula:

$$\tau_{ij} = \begin{cases} (1-\rho) \cdot \tau_{ij} + \rho \cdot {}^{1}\!/_{f(bs)} & \text{if } e_{ij} \text{ belong to } bs \\ \tau_{ij} & \text{otherwise} \end{cases} \qquad \textbf{(3)}$$

where $\rho \in (0,1]$ is a parameter called the evaporation rate and f is the objective function of the problem to be minimized.
The local pheromone update is performed by all ants when an edge e_{ij} is chosen according to the following formula:

$$\tau_{ij} = (1-\varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0 \qquad \textbf{(4)}$$

where $\varphi \in (0,1)$ is a parameter called the pheromone decay coefficient and τ_0 is the initial value of the pheromone trails. The local update is intended to diversify the search performed by subsequent ants in the current iteration of ACS by reducing the pheromone value of the edges that are chosen by the previous ants. Note that τ_{ij} only decreases if τ_0 is smaller than the current τ_{ij}; consequently, τ_0 is usually set to a low value [20].

4. Using ACS to Solve the RTVP

The ACS scheme was described in the previous section. Five main points need to be specified when applying ACS to the RTVP. Section 4.1 presents the construction graph associated with the RTVP and explains how the graph is covered by the ants in order to build a solution. Section 4.2 explains the heuristic information used in Equation (2). Section 4.3 describes the local search procedure. Section 4.4 gives the best solution used in Equation (3). Section 4.5 explains the stop condition. Finally, Section 4.6 explains the fine-tuning of the parameter values of the ACS algorithm.

4.1. Defining the Construction Graph and Building a Solution

In order to make the explanation more understandable, the example introduced in Section 2 is used: $n = 3$; $d_A = 3$, $d_B = 2$, $d_C = 2$; and $D = 7$.
Let the construction graph $G = (N, E)$. The set of nodes N is the union of the sets N_1 and N_2, where $N_1 = \left\{ n_k^i : 1 \le i \le n-1,\ 1 \le k \le d_i - 1 \right\}$ and $N_2 = \left\{ t : 1 \le t \le D-1 \right\}$. Note that the symbol n is not included in N_1 because the positions of the copies of this symbol are fixed when the previous symbols are sequenced. The node n_k^i belonging to N_1 represents the copy k of the symbol i; the node t belonging to N_2 represents a distance between two copies of the same symbol. Therefore, in the example we have $N_1 = \left\{ n_1^A, n_2^A, n_1^B \right\}$ and $N_2 = \left\{ 1,2,3,4,5,6 \right\}$. Let $E \subset N_1 \times N_2$, where the edge $e_{ikt} = \left(n_k^i, t \right)$ represents that the copy k + 1 of the symbol i is sequenced at distance t of the copy k of the symbol i.

An ant starts to generate a solution sequence by setting copy 1 of symbol 1 to the first position of the sequence (see the current s^p of the example in Fig. 2a). Then, an edge has to be chosen at random from the set $E(s^p) = \left\{ e_{1,1,t} : 1 \le t \le D - d_1 + 1 \right\}$ using the probabilities defined by Equation (2). The choice of the edge will fix the position (let it be called p_2^1) of the second copy of symbol 1 to the value 1 + t. Note that the highest possible position p_2^1 is $D - d_1 + 2$, so the remaining copies of symbol 1 can be sequenced at the positions $p_2^1 + 1, p_2^1 + 2, \ldots, D$. In the example, let us suppose that the edge chosen at random is $e_{A,1,2}$ (i.e., t = 2) and, therefore, $p_2^A = 1 + 2 = 3$ (see current s^p in Fig. 2b). The ant then chooses an edge at random from the set $E(s^p) = \left\{ e_{1,2,t} : 1 \le t \le D - (d_1 - 2) - p_2^1 + 1 \right\}$. This process continues for copies 3, 4, …, d_1 -1 of symbol 1. The set of eligible edges when copy k of symbol 1 has been sequenced is $E(s^p) = \left\{ e_{1,k,t} : 1 \le t \le D - (d_1 - k) - p_k^1 + 1 \right\}$, where p_k^1 is the position at the sequence of copy k of symbol 1. In the example, $E(s^p) = \left\{ e_{A,2,t} : 1 \le t \le 7 - (3-2) - 3 + 1 \right\}$, i.e., $\left\{ e_{A,2,1}, e_{A,2,2}, e_{A,2,3}, e_{A,2,4} \right\}$. Next, an edge is chosen at random: for example, $e_{A,2,3}$ (i.e., t = 3). Therefore, $p_3^A = 3 + 3 = 6$ (see Fig. 2c). Note that the distance between the first copy of symbol 1 and the last copy of the same symbol in the preceding cycle is determined when $p_{d_1}^1$ is fixed.

When all copies of symbol 1 have been sequenced, the process is repeated for the copies of symbol 2, then symbol 3, and so on up to the penultimate symbol. The first copy of each symbol is always sequenced at the first free position in the sequence. The other copies of each symbol are sequenced in the same way as those of symbol 1, but the eligible edges must be chosen in such as way that the copies are not sequenced at an occupied position. In the example, the first copy of symbol B is placed in the first free position of the sequence, and therefore $p_1^B = 2$ (see Fig. 2d). Next, an edge has to be chosen from $E(s^p) = \left\{ e_{B,1,2}, e_{B,1,3}, e_{B,1,5} \right\}$. Note that edges $e_{B,1,1}$ and $e_{B,1,4}$ are not eligible because if edge $e_{B,1,1}$ is chosen (i.e., t = 1), $p_2^B = 2 + 1 = 3$ and position 3 is already occupied by symbol A (and analogously for edge

$e_{B,1,4}$). Let us suppose that edge $e_{B,1,3}$ is chosen at random. Therefore, $p_2^B = 2 + 3 = 5$ (see Fig. 2e). Finally, the sequence is completed with the copies of symbol *C* (see Fig. 2f).

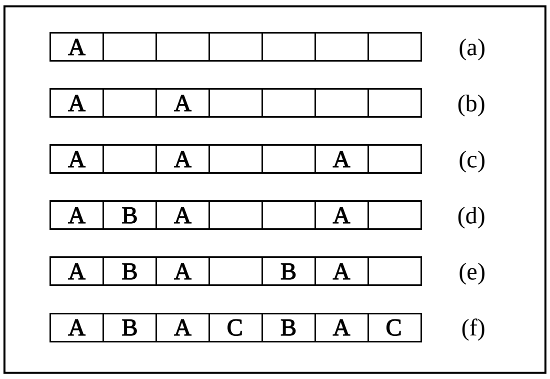


Fig. 2. Construction of a solution sequence

4.2. Heuristic Information

The heuristic information for the edge $e_{i,k,t}$ is given by the following equation:

$$\eta(e_{i,k,t}) = \frac{1}{(t - \bar{t}_i)^2 + \varepsilon} \tag{5}$$

which expresses the desirability for distance *t* to be equal to the ideal distance of symbol *i*; *ε* is a small value (10^{-6}) to prevent a division by zero if the two distances are equal.

4.3. Local Search

The local search procedure is applied as follows. A local search is performed iteratively in a neighbourhood that is generated by interchanging each pair of consecutive copies of the sequence that represents the current solution. The best solution in the neighbourhood is then chosen, and the optimization stops when a certain number of iterations are reached or when no neighbouring solution is better than the current solution.

During the first iterations of the ACS algorithm we observed that the solutions constructed by the ants were still relatively poor. Consequently, it may be very computationally expensive to apply a local search until a local optimum is found when large instances are being solved, so only a few iterations of the ACS algorithm can be run. Therefore, the number of iterations (which is a parameter of the algorithm that we shall call *lsiter*) is limited to reduce the time spent on the local search.

4.4. Best Solution Used in the Offline Pheromone Update

Only the best solution is considered for the offline pheromone update (Equation (3)). This best solution can be either the best solution obtained in the current iteration or the best solution obtained by the algorithm during its execution. We conducted a brief experiment and found that the best solution obtained in the current iteration is clearly the preferable option. Therefore, in this paper we only consider the ACS algorithm that uses the first option.

4.5. Stop Condition

The ACS algorithm stops once it has run for a preset time.

4.6. Fine-tuning the ACS Algorithm Parameters

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although the parameter values may have a very strong effect on the results of the metaheuristic for each problem, they are often selected using one of the following methods, which are not sufficiently thorough [21, 22]: 1) "by hand", based on a small number of experiments that are not referenced; 2) using the general values recommended for a wide range of problems; 3) using the values reported to be effective in other similar problems; or 4) with no apparent explanation.

Adenso-Díaz and Laguna [22] proposed a new technique called CALIBRA for fine-tuning the parameters of heuristic and metaheuristic algorithms. CALIBRA is based on using conjointly Taguchi's fractional factorial experimental designs and a local search procedure.

CALIBRA was used in [9] to fine-tune *DPSOpoi-*c_p*dyn*, and we used the same technique to fine-tune our ACS algorithm. The following parameter values were obtained: number of ants = 20, q_0 = 0.9, α = 1.5, β = 1.75, τ_0 = 0.00013, ρ = 0.87, φ = 0.13 and *lsiter* = 50.

Since CALIBRA cannot fine-tune more than five parameters, the ACS algorithm was fine-tuned in two steps. In the first step, α and β were set to 1 (that is, the pheromone and heuristic information had the same weight), τ_0 was set to a small value (0.01), as is commonly done in the literature [20], and the remaining parameters (number of ants, q_0, ρ, φ and *lsiter*) were fine-tuned. In the second step, the number of ants, q_0 and *lsiter* were set to the values obtained in the first step and the remaining parameters (α, β, τ_0, ρ, and φ) were fine-tuned.

5. Computational Experiment

The computational experiment was carried out for the same instances that were used in [9]. That is, the algorithms were run for 740 instances which were grouped into four classes (185 instances in each class) according to size. The instances in the first class (*CAT1*) were generated using a random value of D (number of copies) distributed uniformly between 25 and 50, and a random value of n (number of symbols) distributed uniformly between 3 and 15; for the second class (*CAT2*), D was between 50 and 100 and n between 3 and 30; for the third class (*CAT3*), D was between 100 and 200 and n between 3 and 65; and for the fourth class (*CAT4*), D was between 200 and 500 and n between 3 and 150. For all instances and for each symbol $i = 1,\ldots,n$, a random value of d_i (number of copies of symbol i) was between 1 and $\left|\frac{D-n+1}{2.5}\right|$ so that $\sum_{i=1..n} d_i = D$. Both algorithms were coded in Java and the computational experiment was carried out using a 3.4 GHz Pentium IV with 1.5 GB of RAM.

The algorithms were run for 50 seconds for each instance. Table 1 shows the average RTV values to be minimized for the global of 740 instances and for each class of instances (*CAT1* to *CAT4*) obtained with the two algorithms.

Table 1. Average RTV values for a computing time of 50 seconds

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
ACS algorithm	**1,651.48**	10.92	36.83	504.84	6,053.31
DPSOpoi-*c_p*dyn	**4,625.54**	16.42	51.34	610.34	17,824.04

Table 1 shows that the global average results of the ACS algorithm for all the instances considered are 64.23% better than the results of *DPSOpoi-*c_p*dyn*. If we consider the results by class, the ACS algorithm also obtains better results than *DPSOpoi-*c_p*dyn*: the results obtained with the ACS algorithm are 33.50%, 28.26%, 17.29% and 66.04% better for *CAT1* instances, *CAT2* instances, *CAT3* instances and *CAT4* instances, respectively. Considerable improvements are observed in all classes, particularly for the biggest instances (*CAT4*), which are the most difficult to solve.

We examined also the dispersion of the results. A measure of the dispersion (let it be called σ) of the RTV values obtained by each algorithm *alg* = {ACS algorithm, *DPSOpoi-*c_p*dyn* } was defined for a given instance, *ins*, according to the following expression:

$$\sigma(alg, ins) = \left(\frac{\mathrm{RTV}_{ins}^{(alg)} - \mathrm{RTV}_{ins}^{(best)}}{\mathrm{RTV}_{ins}^{(best)}} \right)^2 \tag{6}$$

where $\mathrm{RTV}_{ins}^{(alg)}$ is the RTV value of the solution obtained with the algorithm *alg* for the instance *ins*, and $\mathrm{RTV}_{ins}^{(best)}$ is the best RTV value of the solutions obtained with the two algorithms for the instance *ins*. Table 2 shows the average σ dispersion for the total number of instances and for each class.

Table 2. Average σ dispersion for the best solution found

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
ACS algorithm	**0.59**	≈ 0.00	0.03	2.22	0.11
DPSOpoi-$c_p$dyn	**2.66**	0.82	0.61	1.70	7.53

Table 2 shows that the ACS algorithm produces the lowest average σ dispersion for the total number of cases. That is, the ACS algorithm not only obtains the best RTV values but also exhibits more stable behaviour. If we consider the results in Table 2 by class, we can see that the *CAT3* instances are an exception to this pattern. Surprisingly, the dispersion of results obtained with *DPSOpoi-$c_p$dyn* for *CAT3* instances is lower than that obtained with the ACS algorithm, although the ACS algorithm produces better RTV values. This is due to the presence of an outlier, since the solution obtained with the ACS algorithm is much worse than that obtained with *DPSOpoi-$c_p$dyn* for only one *CAT3* instance. If this outlier is disregarded, the average σ dispersions of the produced by the ACS algorithm and *DPSOpoi-$c_p$dyn* are 0.89 and 1.70, respectively.

A computing time of 50 seconds may not be long enough for the algorithms to converge for the largest instances (*CAT4* instances). Table 3 shows the average RTV values for the total number of instances and for each class of instances (*CAT1* to *CAT4*) when the algorithms are run for 1,000 seconds.

Table 3. Average RTV values for a computing time of 1,000 seconds

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
ACS algorithm	**1,208.81**	10.46	31.17	337.31	4,456.32
DPSOpoi-$c_p$dyn	**1,537.34**	14.35	46.55	143.96	5,944.51

When a computing time of 1,000 seconds is used—which seems to be long enough for both algorithms to converge (see Fig. 3)—the ACS algorithm is 21.37% better than *DPSOpoi-$c_p$dyn* for the total number of instances. If we consider the results by class, the ACS algorithm is 27.11%, 33.04% and 25.03% better than *DPSOpoi-$c_p$dyn* for *CAT1*, *CAT2* and *CAT4* instances, respectively. However, *DPSOpoi-$c_p$dyn* performs better for the *CAT3* instances.

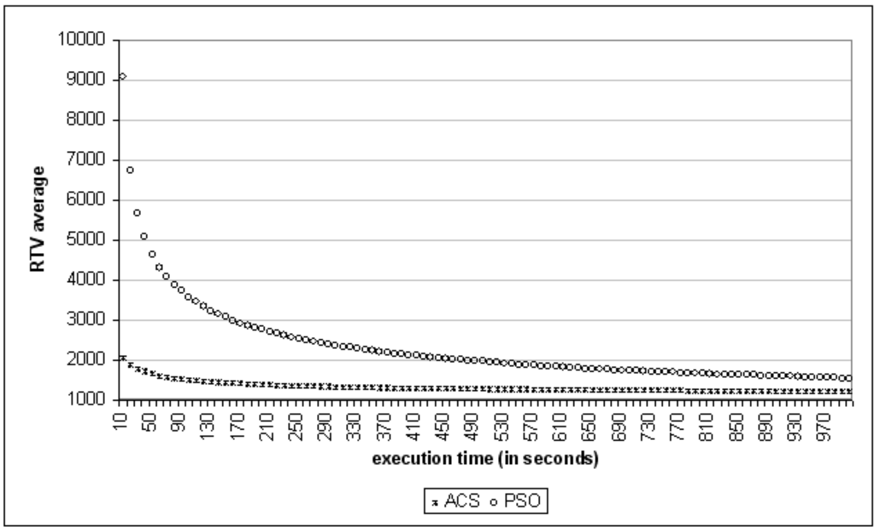


Fig. 3. Average RTV values over the computing time

6. Conclusions

In this paper, the response time variability problem (RTVP) is solved. This scheduling problem arises in a variety of real-life environments including mixed-model assembly lines, multi-threaded systems, network servers, periodic machine maintenance, waste collection and television broadcast. The aim of the RTVP is to minimize the variability in the distances between any two consecutive copies of the same symbol.

Since the RTVP is an NP-hard problem, heuristic and metaheuristic methods are needed to solve real-life instances. Several metaheuristics have been developed for solving hard optimization problems based on biological swarm intelligence, including the PSO and ACO metaheuristics. The best PSO algorithm for solving the RTVO is called *DPSOpoi-*c_p*dyn* [9]. An ant colony system (ACS), which is a variant of the ACO metaheuristic, has been successfully applied to combinatorial optimization problems. We propose an ACS algorithm for solving the RTVP, and the computational experiment showed that the ACS algorithm improves *DPSOpoi-*c_p*dyn*.

References

1. Corominas, A., Kubiak, W., Moreno, N.: Response time variability. Journal of Scheduling, 10, 97--110 (2007)
2. Monden, Y.: Toyota Production Systems. Industrial Engineering and Management Press, Norcross, GA (1983)
3. Waldspurger, C.A., Weihl, W.E.: Stride Scheduling: Deterministic Proportional-Share Resource Management. Technical Report MIT/LCS/TM-528. Massachusetts Institute of Technology, USA (1995)
4. Dong, L., Melhem, R., Mosse, D.: Time slot allocation for real-time messages with negotiable distance constrains requirements, In: 4th IEEE Real-Time Technology and Applications Symposium, 131--136. Denver (1998)
5. Anily, S., Glass, C.A., Hassin, R.: The scheduling of maintenance service. Discrete Applied Mathematics, 82, 27--42 (1998)
6. Herrmann, J.W.: Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling. Technical Report. University of Maryland, USA (2007)
7. Brusco, M.J.: Scheduling advertising slots for television. Journal of Operational Research Society, 59, 1363--1372 (2008)
8. García, A., Pastor, R., Corominas, A.: Solving the Response Time Variability Problem by means of metaheuristics. Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development, 146, 187--194 (2006)
9. García-Villoria, A., Pastor, R.: Introducing dynamic diversity into a discrete particle swarm optimization. Computers & Operations Research, 36, 951—966 (2009)
10. Dorigo, M., Socha, K.: An Introduction to Ant Colony Optimization. In: Handbook of Approximation Algorithms and Metaheuristics. T. F. Gonzalez (2007)
11. Dorigo, M.: Optimization, learning and natural algorithms. Ph.D. Thesis. Department of Electronics, Politecnico di Milano, Italy (1992)
12. Gambardella, L.M., Dorigo, M.: Ant-Q: A reinforcement learning approach to the traveling salesman problem. In: 20th international conference on machine learning, 252--260 , Prieditis and Russell, Palo Alto (1995)
13. Gambardella, L.M., Dorigo, M.: Solving symmetric and asymmetric TSPs by ant colonies. In: 1996 IEEE international conference on evolutionary computation, 622--627, IEEE Press, New York (1996)
14. Stützle, T., Hoos, H.: MAX-MIN Ant System. Future Generation Computer Systems, 16, 889--914 (1996)
15. Bullnheimer, B., Hartl, R.F., Strauss, C.: A new rank-based version of the ant system: A computational study. Central European Journal of Operations Research and Economics, 7, 25--38 (1999)
16. Dorigo, M., Blum, C.: Ant colony optimization theory: A survey. Theoretical Computer Science, 344, 243--278 (2005)
17. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: IEEE International Conference on Neural Networks. Australia (1995)
18. Dorigo, M., Stützle, T.: Ant colony optimization. MIT Press, Cambridge (2004)

19. Yagmahan, B., Yenisey, M.M.: Ant colony optimization for multi-objective flow shop scheduling problem. Computer and Industrial Engineering, 58, 411--420 (2008)
20. Lo, S-T., Chen, R-M., Huang, Y-M., Wu C-L.: Multiprocessor system scheduling with precedence and resource constraints using an enhanced and colony system. Expert Systems with Applications, 34, 2071--2081 (2008)
21. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. In: IEEE Transactions on evolutionary computation, 3, 124--141 (1999)
22. Adenso-Díaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research, 54, 99--114 (2006)

An enhanced metaheuristic for solving the response time variability problem[†]

Albert COROMINAS, Alberto GARCÍA-VILLORIA[*]and Rafael PASTOR
Institute of Industrial and Control Engineering (IOC)
Universitat Politècnica de Catalunya (UPC)
{albert.corominas / alberto.garcia-villoria / rafael.pastor}@upc.edu

Abstract. The response time variability problem (RTVP) is a hard scheduling problem which has been recently formalised in the literature. This problem has a wide range of real-world applications in mixed-model assembly lines, multi-threaded computer systems, machine maintenance, waste collection and others. The RTVP arises whenever products, clients or jobs need to be sequenced in such a way that the variability in the time between the points at which they receive the necessary resources is minimised. The RTVP is a NP-hard problem, heuristic and metaheuristic techniques are needed to solve it. The best results in the literature for the RTVP were obtained with a hybrid metaheuristic. We propose three algorithms based on simulated annealing to solve the RTVP. A computational experiment is carried out and it is shown that, on average, two of the three proposed algorithms improve the best methods published in the literature.

Keywords: response time variability, regular sequences, simulated annealing, multi-start, metaheuristics

1. Introduction

The response time variability problem (RTVP) is a combinatorial optimisation problem that occurs whenever products, clients or jobs need to be sequenced so as to minimise variability in the time between the instants at which they receive the necessary resources. This problem has a broad range of real-world applications: among others, to sequencing on mixed-model assembly lines under JIT (Monden, 1983; Miltenburg, 1989; Kubiak, 1993), resource allocation in computer multi-threaded systems such as operating systems, network servers and media-based applications (Waldspurger and Weihl, 1994 and 1995; Dong *et al.*, 1998; Bar-Noy *et al.*, 2002), in the collection of waste (Herrmann, 2007), in the schedule of commercial videotapes for television (Bollapragada *et al.*, 2004), in the design of sales catalogues (problem introduced in Bollapragada *et al.*, 2004),and in the periodic machine maintenance problem (Wei and Liu, 1983; Anily *et al.*, 1998). These real-life problems are usually considered as distance-constrained scheduling problems (Han et al., 1996; Dong et al., 1998). Although the main objective of the distance-constrained problem and the RTVP is to

[†] Supported by the Spanish Ministry of Education and Science under project DPI2007-61905; co-funded by the ERDF.
[*] Corresponding author: Alberto García-Villoria, Institute of Industrial and Control Engineering (IOC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; tel.: +34 93 4054010; e-mail: alberto.garcia-villoria@upc.edu

find as regular a sequence as possible, the advantage of the RTVP is that it will always come up with a feasible solution, contrary to the distance-constrained problem as well as other distance-constrained problems (e.g., see Han *et al*., 1996). For a good introduction to the RTVP, see Corominas *et al*. (2009a)

The RTVP is formulated as follows. Let *n* be the number of symbols, d_i the number of copies to be scheduled of symbol *i* (*i* = 1,…,*n*) and *D* the total number of copies ($D = \sum_{i=1..n} d_i$). Let *s* be a solution of a RTVP instance that consists of a circular sequence of copies ($s = s_1 s_2 \ldots s_D$), where s_j is the copy sequenced in position *j* of sequence *s*. For all symbol *i* in which $d_i \geq 2$, let t_k^i be the distance between the positions in which the copies *k* + 1 and *k* of symbol *i* are found (i.e. the number of positions between them, where the distance between two consecutive positions is considered equal to 1). Since the sequence is circular, position 1 comes immediately after position *D*; therefore, $t_{d_i}^i$ is the distance between the first copy of symbol *i* in a cycle and the last copy of the same symbol in the preceding cycle. Let $\bar{t}_i$ be the average or ideal distance between two consecutive copies of symbol *i* ($\bar{t}_i = D/d_i$). For all symbol *i* in which $d_i = 1$, t_1^i is equal to $\bar{t}_i$. The objective is to minimise the metric response time variability (RTV) which is defined by the following expression:

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - \bar{t}_i)^2 \qquad (1)$$

The RTVP has been demonstrated to be NP-hard (Corominas *et al*., 2007).

The objective of this work is to improve the solution of the RTVP. To achieve it, we have used simulated annealing (SA). The remainder of the paper is organized as follows. The state of the art with the methods proposed in the literature to solve the RTVP is given in Section 3. Three SA-based algorithms are proposed in Section 3 to improve the solution of the RTVP. A computational experiment is carried whose results are shown and discussed in Section 4. Finally, some conclusions are given in Section 5.

2. State of the art

Although the RTVP is in general NP-hard, the two-symbol case can be optimally solved with a polynomial algorithm proposed in Corominas *et al*. (2007). For the other cases, Corominas *et al*. (2007) proposed a mixed-integer linear programming (MILP) model, which was enhanced in Corominas *et al*. (2010). Anyway, only small instances can be solved optimally in a practical time (the limit size is 40 copies to be sequenced).

The RTVP problem has been first time solved in Waldspurger and Weihl (1994) using a method that authors called *lottery scheduling*, which consists on generating a solution at random. Later, Waldspurger and Weihl (1995) used the Jefferson method of apportionment (Balinski and Young, 1982), a greedy heuristic algorithm which they renamed as the stride scheduling technique. Herrmann (2007) solved the RTVP by applying a heuristic algorithm based on the stride scheduling technique. An aggregation approach was used in Herrmann (2009). Corominas *et al*. (2007) proposed also the Jefferson method together with other four greedy heuristic algorithms and a local search method.

Metaheuristic approaches have been intensively proposed during the last three years. García *et al*. (2006) proposed a multi-start (MS), a greedy randomized adaptive search procedure (GRASP) and four variants of a discrete particle swarm optimisation (PSO) algorithm. An enhanced multi-start algorithm and an enhanced GRASP algorithm were proposed in Corominas *et al*. (2008), and other ten discrete PSO algorithms were proposed in García-Villoria and Pastor (2009a). A cross-entropy method (CE) algorithm, a psychoclonal algorithm, an electromagnetism-like mechanism (EM) algorithm, and a genetic algorithm (GA) were used in García-Villoria *et al*. (2007) and García-Villoria and Pastor (2008, 2009b, 2010), respectively. Two tabu search (TS) algorithms and a variable neighbourhood search (VNS) algorithm were proposed in Corominas *et al*. (2009b, 2009c, 2009d), respectively. Finally, three hybrid algorithms (MS+VNS, TS+VNS and PSO+VNS), have been proposed in Corominas *et al*. (2009e).

All these metaheuristic algorithms, except the CE algorithm, have been tested on the same set of benchmark instances. The set is composed of 740 instances which were grouped into four classes (from *CAT1* to *CAT4* with 185 test instances in each class), where *CAT1* instances are the smallest instances and *CAT4* instances are the largest instances. Tables 1 and 2 show the average RTV values of the solutions obtained with the algorithms for 50 and 1,000 computing seconds, respectively (if there are more than one algorithm based on the same metaheuristic, only the results of the best of them are shown). The results are shown for the 740 instances and for each class of instances (*CAT1* to *CAT4*).

Table 1. Average RTV values for a computing time of 50 seconds

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
MS+VNS	62.17	10.24	21.23	47.46	169.76
TS+VNS	71.57	10.38	24.00	53.99	197.90
PSO+VNS	60.03	10.47	22.42	49.37	157.86
VNS	63.96	10.73	23.69	51.80	169.64
TS	210.47	10.26	22.56	73.26	735.78
Psycho	235.68	14.92	44.25	137.07	746.50
MS	2,106.01	11.56	38.02	154.82	8,219.65
GRASP	2,308.69	13.00	60.45	270.93	8,890.37
EM	3,747.05	19.14	54.54	260.79	14,653.72
PSO	4,625.54	16.42	51.34	610.34	17,824.04

The best results have been achieved with the three hybrid algorithms Corominas *et al*. (2009e). The algorithms are based on hybridizing the TS algorithm proposed in Corominas *et al*. (2009c), the MS algorithm proposed in Corominas *et al*. (2008) and a PSO algorithm proposed in García-Villoria and Pastor (2009a) with the VNS algorithm proposed in Corominas *et al*. (2009d), respectively. All three algorithms obtain very similar results, but the MS+VNS algorithm is slightly better than the TS+VNS and

PSO+VNS algorithms. The MS+VNS algorithm consist on embedding the VNS algorithm in a MS scheme. That is, several executions of the VNS algorithm proposed in Corominas *et al*. (2009d) are done from a different initial, random starting solution.

Table 2. Average RTV values for a computing time of 1,000 seconds

	Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
MS+VNS	54.95	10.24	20.94	43.26	145.35
TS+VNS	55.05	10.24	22.48	47.66	139.84
PSO+VNS	55.86	10.45	22.00	46.80	144.22
VNS	62.24	10.73	23.69	51.40	163.15
TS	78.62	10.24	21.16	48.12	234.96
Psycho	161.60	14.90	39.90	122.38	469.23
MS	169.25	10.51	31.21	123.27	512.02
GRASP	301.90	11.56	50.45	227.50	918.10
EM	330.29	18.64	52.97	157.20	1,092.36
PSO	1,537.34	14.35	46.55	143.96	5,944.51

3. Three SA-based algorithms for the RTVP

Bollapragada *et al*. (2004) presented a real-life case of a problem that can be considered as a variant of the RTVP. There are two differences between this variant with respect to the RTVP defined in the Introduction: 1) the problem faced by Bollapragada *et al*. is a non-cyclic problem instead of a cyclic one; and 2) the discrepancies between real and ideal distances is penalized linearly instead of using a square penalization. The metric to be minimised in Bollapragada *et al*. (2004) is $\sum_{i=1}^{n}\sum_{k=1}^{d_i-1}\left|t_k^i - \overline{t_i}\right|$.

To solve the non-cycling variant of the RTVP in which the television advertising slots are scheduled, Brusco (2008) proposed a SA algorithm. However, no SA approach has been proposed to solve the RTVP to date. We propose three algorithms based on SA. The first algorithm is a straightforward application of the classical SA. The second algorithm is a hybridization of the first proposed SA algorithm with a MS scheme. Finally, the third algorithm is an extension of the second one in which several neighbourhood structures are used.

We first introduce in Subsection 3.1 the SA metaheuristic. Then the three SA-based algorithms are explained in Subsections 3.2, 3.3 and 3.4, respectively. Finally, the fine-tuning of the parameters of all algorithms is explained in Subsection 3.5.

3.1. Simulated Annealing

The Simulated Annealing metaheuristic (SA) was proposed in Kirkpatrick *et al*. (1983) to solve complex combinatorial optimisation problems, as it is the RTVP. Since then, SA has been successfully applied for solving a wide range of combinatorial optimisation problems (Henderson *et al*., 2003).

SA can be seen as a variant of a local search procedure in which is allowed moving to a worse solution in small probability. The objective of accepting worse solutions is to avoid being trapped into a local optimum. The metaheuristic starts from an initial solution, which is initially the current solution. Then, at each iteration, a new solution from the neighbourhood of the current solution is considered. If the neighbour is not worse than the current solution, then the neighbour becomes the current solution; in the case that is worse, the neighbour can become also the current solution with a probability that depends on: 1) how worse is the neighbour, and 2) the value of a parameter called *temperature*, which is decreased every certain number of iterations. The pseudo-code of SA (when minimising the objective function) is shown in Figure 1.

```
Let f(s) be the objective function to be minimised of the solution s
Let N(s) the neighbourhood of the solution s
Let A(t) the new temperature calculated from the temperature t

0.  Set the parameters:
        $t_0$ (initial temperature)
        itt (number of iterations during the temperature remains equal)
1.  $t := t_0$;
2.  s := Generation of the initial solution
3.  While stopping criterion is not reached do:
4.      i :=0
5.      While i < itt do:
6.          s' := choose at random a solution from N(s)
7.          $\Delta := f(s') - f(s)$
8.          If $\Delta \leq 0$ Then s := s'
9.          If $\Delta > 0$ Then s := s' with probability $\exp(-\Delta/t)$
10.         i := i + 1
11.     End while
12.     t := A(t)
13. End while
14. Return the best solution found
```

Figure 1. Pseudo-code of SA

3.2. A straightforward SA algorithm (SA$_{N1}$)

Several decisions have to be taken before applying the general scheme of SA to solve the RTVP. Some of these decisions are general and the others are specific for the problem to solve. Specific decisions for the RTVP are the representation of solutions and the neighbourhood of each solution ($N(s)$), the generation of the initial solution and the objective function ($f(s)$). General decisions are the way to decrease the temperature ($A(t)$) and the stopping criterion of the algorithm. Moreover, the parameters of the algorithm need to be fine-tuned before the execution (it is explained in Section 3.5).

3.1.1. Representation and neighbourhood of solutions

The representation of a solution is the sequence of symbols, in which each symbol i appears d_i (the number of copies of symbol i to be sequenced) times. The neighbourhood of a solution is generated interchanging each pair of two consecutive units of the sequence that represents the solution (let this neighbourhood be called N_1). This neighbourhood has been successfully applied when solving the RTVP with a

Multi-start and GRASP algorithm (Corominas *et al*., 2008) and a VNS algorithm (Corominas *et al*., 2009d).

3.2.2. Initial solution

The initial solution is generated using the lottery scheduling (Waldspurger and Weihl, 1994) as it is done in previous works published in the literature when an initial solution is required. That is, for each position, a symbol to be sequenced is randomly chosen. The probability of each symbol is equal to the number of copies of this symbol that remain to be sequenced divided by the total number of units that remain to be sequenced. The random generation of the initial solution for a SA algorithm is usually done in the literature (Dowsland and Adenso-Díaz, 2003).

3.2.3. Objective function

In the case of the RTVP, the objective function to be minimised is the RTV value of the solution (Equation 1).

3.2.4. Decreasing the temperature

The temperature of the SA algorithm has influence on the probability of acceptance of worse neighbouring solutions. The higher the temperature, the more probable; and vice versa, the lower the temperature, the less probable (Step 9 in Figure 1). The most popular way in the literature that obtains good results is the geometric reduction, that is, $A(t) = t.\alpha$, where $\alpha < 1$ (Dowsland and Adenso-Díaz, 2003, Henderson *et al*., 2003). The α value has to be set; thus, α becomes another parameter of the algorithm.

3.2.5. Stopping criterion

The algorithm stops when it has run for a preset available time (the same criterion has been usually used in previous proposed metaheuristic methods for the RTVP).

3.3. A SA algorithm embedded in a multi-start scheme (MS+SA$_{N1}$)

When the temperature is too low then the probability of accepting worse neighbour solutions is negligible. Thus, in practice, the previous proposed SA algorithm may be trapped in a local optimum after certain computing time. To overcome this situation, we propose to embed a variant of the previous SA algorithm in a MS scheme as follows. During a preset time, the SA variant algorithm is iteratively run. The variant is equal to the previous SA algorithm, except for its stopping criterion. The embedded SA stops when the current temperature is lower than a threshold (final temperature); then, and according to the MS scheme, the SA is launched again. Although it seems natural to apply a MS scheme to the SA metaheuristic after a certain computing time, up to our knowledge this idea does not appear in the literature. The second proposed SA algorithm is shown in Figure 2.

```
0.  Set the parameters:
        $t_0$ (initial temperature)
        $t_f$ (final temperature)
        itt (number of iterations during the temperature remains equal)
1.  While current runtime < maximum runtime do:
2.      $t := t_0$;
3.      $s$ := Generation of the initial solution
4.      While $t \geq t_f$ do:
5.          $i$ :=0
6.          While i < itt do:
7.              $s'$ := choose at random a solution from $N_1(s)$
8.              $\Delta := f(s') - f(s)$
9.              If $\Delta \leq 0$ Then $s := s'$
10.             If $\Delta > 0$ Then $s := s'$ with probability $\exp(-\Delta/t)$
11.             i := i + 1
12.         End while
13.         $t := A(t)$
14.     End while
15. End while
16. Return the best solution found
```

Figure 2. Pseudo-code of $MS+SA_{N1}$

3.4. A SA algorithm embedded in a multi-start scheme with multiple neighbourhoods ($MS+SA_{N1,2,3}$)

The previous second algorithm is extended with the following idea inspired from the SA algorithm proposed in Brusco (2008) to solve the non-cyclic variant of the RTVP. He defines two neighbourhoods, N_2 and N_3 (which are also used frequently in the literature). N_2 consists on interchanging each pair of consecutive or no-consecutive units of the sequence. N_3 consists on inserting each unit in each position of the sequence. Then, at each iteration of the Brusco's algorithm, it is selected from which of the two neighbourhoods will be obtained the neighbour of the current solution. The selection of the neighbourhood is at random with equal probabilities.

We propose a new algorithm by incorporating the selection of the neighbourhood at the algorithm explained in Section 4.3. The available neighbourhoods are N_1, N_2 and N_3. Moreover, instead of using equal probabilities for each neighbourhood, we put these values as parameters of the algorithm because using non equal probabilities may improve the performance of the algorithm. The third proposed SA algorithm is shown in Figure 3.

```
0.  Set the parameters:
          $t_0$ (initial temperature)
          $t_f$ (final temperature)
          $itt$ (number of iterations during the temperature remains equal)
          $p_1$, $p_2$, $p_3$ (probability of selection neighbourhood $N_1$, $N_2$, $N_3$)
1.  While current runtime < maximum runtime do:
2.        $t := t_0$;
3.        $s$ := Generation of the initial solution
4.        While $t \geq t_f$ do:
5.              $i$ :=0
6.              While i < $itt$ do:
7.                    Let $N$ be the neighbourhood selected at random between $N_1$, $N_2$ and $N_3$
8.                    $s'$ := choose at random a solution from $N(s)$
9.                    $\Delta := f(s') - f(s)$
10.                   If $\Delta \leq 0$ Then $s := s'$
11.                   If $\Delta > 0$ Then $s := s'$ with probability exp(-Δ/t)
12.                   i := i + 1
13.             End while
14.             $t := A(t)$
15.       End while
16.  End while
17.  Return the best solution found
```

Figure 3. Pseudo-code of $MS{+}SA_{N1,2,3}$

3.5. Fine-tuning the algorithm parameters

Fine-tuning the parameters of a metaheuristic algorithm is almost always a difficult task. Although the parameter values may have a very strong effect on the results of the metaheuristic for each problem, they are often selected using one of the following methods, which are not sufficiently thorough (Eiben *et al.*, 1999; Adenso-Díaz and Laguna, 2006): 1) "by hand", based on a small number of experiments that are not referenced; 2) using the general values recommended for a wide range of problems; 3) using the values reported to be effective in other similar problems; or 4) with no apparent explanation.

Adenso-Díaz and Laguna (2006) proposed a new technique called CALIBRA for fine-tuning the parameters of algorithms. CALIBRA is based on using conjointly Taguchi's fractional factorial experimental designs and a local search procedure. We propose to use CALIBRA for setting the parameter values of our algorithms. CALIBRA was applied to a representative training set of 60 instances which were generated as explained in the Section 4. The following parameter values were obtained:

- SA_{N1}: $t_0 = 13$, $itt = 1762$ and $\alpha = 0.9875$.
- $MS{+}SA_{N1}$: $t_0 = 25$, $t_f = 0.008$, $itt = 1525$ and $\alpha = 0.9875$.
- $MS{+}SA_{N1,2,3}$: $t_0 = 88$, $t_f = 0.007$, $itt = 1750$, $\alpha = 0.9875$, $p_1 = 0.37$ and $p_2 = 0.25$.

Since CALIBRA cannot fine-tune more than five parameters, $MS{+}SA_{N1,2,3}$ (which have six parameters to be fine-tuned) is fine-tuned in two steps. Note that the value of p_3 is not calibrated because it depends on the values of p_1 and p_2 ($p_3 = 1 - p_1 - p_2$). In the first step, the value of t_f is set to a small value (0.01), as it is usually done in the literature (e.g., Brusco, 2008), and the remaining parameters (t_0, *itt*, α, p_1 and p_2) are fine-tuned. In the second step the value of t_0 is set at the value obtained in the first step and the remaining parameters (t_f, *itt*, α, p_1 and p_2) are fine-tuned.

4. Computational experiment

The MS+VNS hybrid algorithm proposed in Corominas *et al*. (2009e) is the most efficient algorithm in the literature for solving non-small RTVP instances. Therefore, we compare the performance of our proposed algorithms with that MS+VNS algorithm (let it be called *MS+VNS*).

All algorithms are coded in Java and executed on a PC 3.4 GHz Intel Pentium IV with 1.5 GB of RAM. The same 60 training instances and 740 test instances used in Corominas *et al*. (2009e) and in previous works are also used in this paper (all instances can be found at https://www.ioc.upc.edu/EOLI/research/). These instances were grouped into four classes (from *CAT1* to *CAT4* with 15 training instances and 185 test instances in each class) according to their size. The instances were generated using the random values of *D* (number of copies) and *n* (number of symbols) shown in Table 3. For all instances and for each model $i = 1,\dots,n$, a random value of d_i (number of copies of symbol *i*) is between 1 and $\lceil (D-n+1)/2.5 \rceil$ such that $\sum_{i=1..n} d_i = D$.

Table 3. Uniform distributions for generating the *D* and *n* values

	CAT1	***CAT2***	***CAT3***	***CAT4***
D	U(25, 50)	U(50, 100)	U(100, 200)	U(200, 500)
n	U(3, 15)	U(3, 30)	U(3, 65)	U(3, 150)

The stop condition of all algorithms is to be run for a preset time. We run the algorithms for 10, 50, 200 and 1,000 seconds. Table 4 shows the average RTV values to be minimised for the global of 740 instances and for each class of instances (*CAT1* to *CAT4*) obtained with the algorithms and Figure 4 shows the evolution of the average RTV values for the global of all instances during the computing time.

After 1,000 computing seconds, the best overall RTV average is obtained with $MS{+}SA_{N1}$. It is worth to point that Corominas *et al*. (2009e) showed that the solutions obtained with their MS+VNS algorithm for all *CAT1* instances are optimal solutions. Thus, we can see that $MS{+}SA_{N1}$ and $MS{+}SA_{N1,2,3}$ are also able to solve optimally all *CAT1* instances. For *CAT2* instances, $MS{+}SA_{N1}$ is 0.10%, 0.14% and 3.46% better than *MS+VNS*, $MS{+}SA_{N1,2,3}$ and SA_{N1}, respectively, but without significant difference between *MS+VNS* and $MS{+}SA_{N1,2,3}$. For *CAT3* instances, $MS{+}SA_{N1}$ is 6.19.%, 6.77% and 9.47% better than $MS{+}SA_{N1,2,3}$, *MS+VNS* and SA_{N1}, respectively. Finally, for *CAT4* instances, $MS{+}SA_{N1}$ is 9.19.%, 20.94% and 33.77% better than SA_{N1}, *MS+VNS* and $MS{+}SA_{N1,2,3}$, respectively. To sum up, we can see that two of the SA algorithms, SA_{N1} and $MS{+}SA_{N1}$, are able to obtain, on average, better solutions (7.64% and 15.20% better, respectively) than VNS hybridises with MS (*MS+VNS*). This tendency grows with the size of the instances to be solved.

Table 4. Average RTV values for *MS+VNS* and SA based algorithms

		Global	***CAT1***	***CAT2***	***CAT3***	***CAT4***
10 s.	***MS+VNS***	**71.07**	10.24	21.58	51.07	201.39
	SA_{N1}	**108.46**	10.26	21.67	45.68	356.24
	$MS+SA_{N1}$	**278.53**	10.25	22.03	91.88	989.95
	$MS+SA_{N1,2,3}$	**144.21**	10.35	22.72	55.61	488.17
50 s.	***MS+VNS***	**62.17**	10.24	21.23	47.46	169.76
	SA_{N1}	**50.87**	10.26	21.67	44.57	126.98
	$MS+SA_{N1}$	**51.84**	10.24	21.19	43.57	132.35
	$MS+SA_{N1,2,3}$	**73.12**	10.24	21.52	47.34	213.37
200 s.	***MS+VNS***	**58.45**	10.24	21.01	45.35	157.22
	SA_{N1}	**50.78**	10.26	21.67	44.56	126.62
	$MS+SA_{N1}$	**48.52**	10.24	20.95	41.59	121.30
	$MS+SA_{N1,2,3}$	**66.60**	10.24	21.15	44.84	190.17
1,000 s.	***MS+VNS***	**54.95**	10.24	20.94	43.26	145.35
	SA_{N1}	**50.75**	10.26	21.67	44.55	126.54
	$MS+SA_{N1}$	**46.60**	10.24	20.92	40.33	114.91
	$MS+SA_{N1,2,3}$	**61.92**	10.24	20.95	42.99	173.51

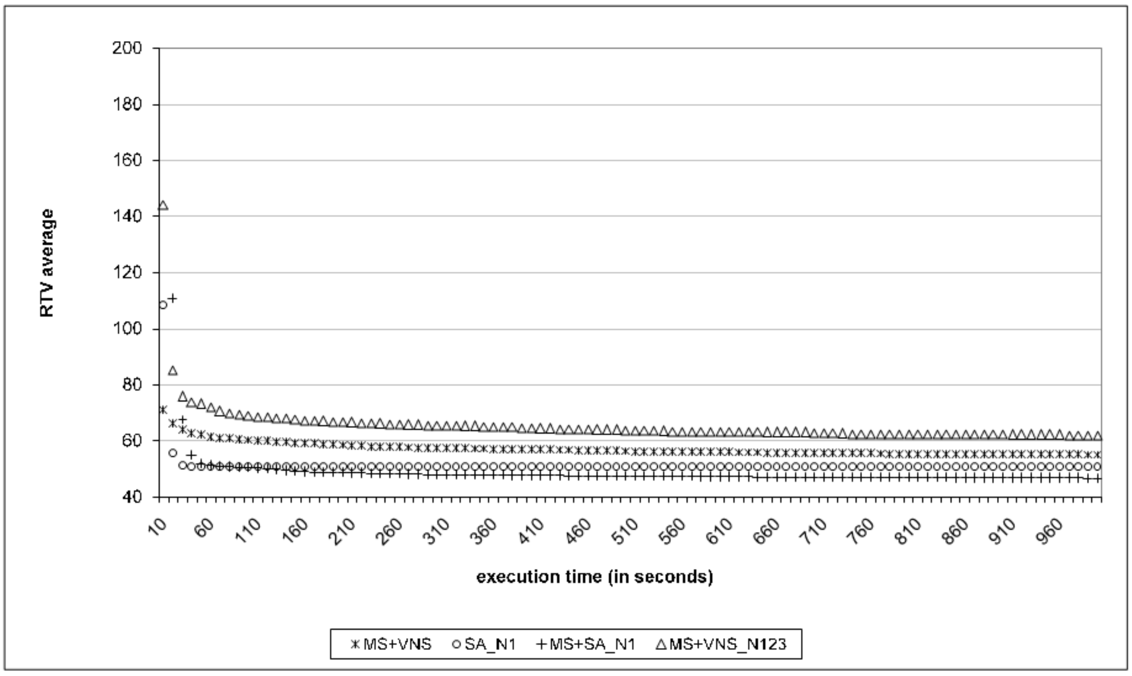


Figure 4. Average RTV values over the computing time

Table 5 shows the number of times that each algorithm reaches the best RTV value obtained by either one after 1,000 computing seconds. The results are shown for the total number of 740 instances and for each class. As expected from the results in Table 4, Table 5 shows that *$MS+SA_{N1}$* is the algorithm that more time reaches the best solution. For the total number of instances, *$MS+SA_{N1}$* obtains the best solution in 97% of times.

Table 5. Number of times that the best solution is reached

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
MS+VNS	**458**	185	183	70	20
SA$_{N1}$	**396**	183	128	39	46
MS+SA$_{N1}$	**721**	185	184	179	173
MS+SA$_{N1,2,3}$	**455**	185	181	72	17

To complete the analysis of the results, we examined the dispersion of the results with respect to the best solution obtained by either algorithm. A measure of the dispersion (let it be called σ) of the RTV values obtained by each algorithm *alg* = {*MS+VNS, SA$_{N1}$, MS+SA$_{N1}$, MS+SA$_{N1,2,3}$*} was defined for a given instance, *ins*, according to the following expression:

$$\sigma(alg, ins) = \left(\frac{\mathrm{RTV}_{ins}^{(alg)} - \mathrm{RTV}_{ins}^{(best)}}{\mathrm{RTV}_{ins}^{(best)}} \right)^2 \tag{2}$$

where $\mathrm{RTV}_{ins}^{(alg)}$ is the RTV value of the solution obtained with the algorithm *alg* for the instance *ins*, and $\mathrm{RTV}_{ins}^{(best)}$ is the best RTV value of the solutions obtained with the four algorithms for the instance *ins*. Table 6 shows the maximum σ dispersion for the total number of instances and for each class. We can see that low dispersions are obtained for the total number of cases and for each instance class with all algorithms (except for *MS+SA$_{N1,2,3}$* when the largest instances are solved). That is, when an algorithm does not obtain the best RTV value for a given instance, it obtains a value that is very close to it (especially true for *MS+SA$_{N1}$*).

Table 6. Maximum σ values with respect to the best solutions found

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
MS+VNS	**0.68**	0.00	0.03	0.18	0.68
SA$_{N1}$	**0.34**	0.09	0.34	0.17	0.26
MS+SA$_{N1}$	**0.02**	0.00	0.02	0.01	≈0.00
MS+SA$_{N1,2,3}$	**5.07**	0.00	0.02	0.10	5.07

In order to see how close are the solutions obtained with the best method (*MS+SA$_{N1}$*) with respect to the optimal solutions, the lower bound (LB) proposed in Corominas *et al*. (2007) is used. Table 6 shows the average of the RTV values obtained with *MS+SA$_{N1}$* after 1,000 computing seconds ($\overline{RTV}$) and the averages of the LBs ($\overline{LB}$).

Table 7. Averages of the optimal RTV values and the RTV lower bounds

	Global	*CAT1*	*CAT2*	*CAT3*	*CAT4*
$\overline{LB}$	**21.40**	5.35	10.95	21.15	48.15
$\overline{RTV}$	**46.60**	10.24	20.92	40.33	114.91

As it has been said, all 185 *CAT1* instances were solved optimally with *MS+SA$_{N1}$*. We can see in Table 7 that the LB is not accurate. For the smallest instances, the ratio between $\overline{RTV}$ and $\overline{LB}$ is 1.914. It seems reasonable to assume that this ratio will remain equal or increase for larger instances. Thus, if we assume that the ratio remains equal, a more accurate estimation of the averages of the optimal values for *CAT2*, *CAT3* and *CAT4* instances are obtained by multiplying their $\overline{LB}$ by 1.914; that is, 20.96, 40.48 and

92.16 for *CAT2*, *CAT3* and *CAT4* instances, respectively. According to this assumption, we could ensure that the solutions obtained by the hybrid algorithms for *CAT2* and *CAT3* instances are very good.

5. Conclusions

In this paper, the response time variability problem (RTVP) is solved. This scheduling problem arises in a variety of real-world environments including mixed-model assembly lines, multi-threaded systems, periodic machine maintenance and waste collection, among others. The aim of the RTVP is to minimise the variability in the distances between any two consecutive copies of the same symbol.

The RTVP is an NP-hard problem and heuristic and metaheuristic methods are needed to solve real-world, large instances. Several metaheuristic algorithms have been developed for solving this hard combinatorial optimization problem. The most efficient algorithm to date for solving the RTVP was a hybrid algorithm in which VNS is embedded in a MS scheme (Corominas *et al*., 2009e). The existing MS+VNS algorithm is a very efficient one to solve the RTVP and it was shown that it was able to solve optimally all test instances up to 50 copies to be sequenced.

In this study we propose three SA-based algorithms. The best of them, $MS{+}SA_{N1}$, improves on average the MS+VNS algorithm and obtains an average RTV value 15.20% better. Moreover, $MS{+}SA_{N1}$ is very stable for all type of instances.

REFERENCES

Adenso-Díaz, B. and Laguna, M. (2006) ‘Fine-tuning of algorithms using fractional experimental designs and local search’, *Operations Research*, Vol. 54, pp. 99-114.

Anily, S., Glass, C.A. and Hassin, R. (1998) ‘The scheduling of maintenance service’, *Discrete Applied Mathematics,* Vol. 82, pp. 27-42.

Balinski, M.L. and Young, H.P. (1982) *Fair Representation*, Yale University Press, New Haven.

Bar-Noy, A., Nisgav, A. and Patt-Shamir, B. (2002) ‘Nearly optimal perfectly-periodic schedules’, *Distributed Computing*, Vol. 15, pp. 207–220.

Bollapragada, S., Bussieck, M.R. and Mallik, S. (2004) ‘Scheduling Commercial Videotapes in Broadcast Television’, *Operations Research*, Vol. 52, pp. 679-689.

Brusco, M.J. (2008) ‘Scheduling advertising slots for television’, *Journal of the Operational Research Society*, Vol. 59, pp. 1363-1372.

Corominas, A., Kubiak, W. and Moreno, N. (2007) ‘Response time variability’, *Journal of Scheduling*, Vol. 10, pp. 97-110.

Corominas, A., Kubiak, W. and Pastor, R. (2010) ‘Mathematical programming modeling of the Response Time Variability Problem’, *European Journal of Operational Research*, Vol. 200, pp. 347-357.

Corominas, A., García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of Multi-start and GRASP metaheuristics’, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development,* Vol. 184, pp. 128-137.

Corominas, A., García-Villoria, A. and Pastor, R. (2009a) ‘The Response Time Variability Problem: A Review’, *3rd International Conference on Industrial Engineering and Industrial Management* (CIO 2009), Barcelona and Terrassa, Spain.

Corominas, A., García-Villoria, A. and Pastor, R. (2009b) ‘Using Tabu Search for the Response Time Variability Problem’, *3rd International Conference on Industrial Engineering and Industrial Management* (CIO 2009), Barcelona and Terrassa, Spain.

Corominas, A., García-Villoria, A. and Pastor, R. (2009c) ‘Resolución del response time variability problem mediante tabu search’, *VIII Evento Internacional de Matemática y Computación* (COMAT’2009), Universidad de Matanzas, Cuba.

Corominas, A., García-Villoria, A. and Pastor, R. (2009d) ‘Solving the Response Time Variable Problem by means of a Variable Neighbourhood Search Algorithm’, *13th IFAC Symposium of Information Control Problems in Manufacturing* (INCOM 2009), Moscow, Russia.

Corominas, A., García-Villoria, A. and Pastor, R. (2009e) ‘Metaheuristic algorithms hybridized with variable neighbourhood search for solving the response time variability problem’, *Technical report IOC-DT-P-2009-04*, Universitat Politècnica de Catalunya, Spain.

Dong, L., Melhem, R. and Mosse, D. (1998) ‘Time slot allocation for real-time messages with negotiable distance constrains requirements’, *Fourth IEEE Real-Time Technology and Applications Symposium* (RTAS'98), Denver, CO. pp. 131-136.

Dowsland, K.A. and Adenso-Díaz, B. (2003) ‘Heuristic design and fundamentals of the Simulated Annealing’, *Inteligencia Artificial*, Vol. 19, pp. 93-102.

Eiben, A.E., Hinterding, R. and Michalewicz, Z. (1999) ‘Parameter control in evolutionary algorithms’, *IEEE Transactions on evolutionary computation*, Vol. 3, pp. 124-141.

García, A., Pastor, R. and Corominas, A. (2006) ‘Solving the Response Time Variability Problem by means of metaheuristics’, *Special Issue of Frontiers in Artificial Intelligence and Applications on Artificial Intelligence Research and Development*, Vol. 146, pp. 187-194.

García-Villoria, A., Pastor, R. and Corominas, A. (2007) ‘Solving the Response Time Variability Problem by means of the Cross-Entropy Method’, *International Journal of Manufacturing Technology and Management* (to be published).

García-Villoria, A. and Pastor, R. (2008) ‘Solving the Response Time Variability Problem by means of a psychoclonal approach’, *Journal of Heuristics*, in press, corrected proof, available online, 16 July 2008, doi:10.1007/s10732-008-9082-2.

García-Villoria, A. and Pastor, R. (2009a) ‘Introducing dynamic diversity into a discrete particle swarm optimization’, *Computers & Operations Research*, Vol. 36, pp. 951-966.

García-Villoria, A. and Pastor, R. (2009b) ‘Solving the Response Time Variability Problem by means of the Electromagnetism-like Mechanism’, *International Journal of Production Research*, doi: 10.1080/00207540902862545.

García-Villoria, A. and Pastor, R. (2010) ‘Solving the response time variability problem by means of a genetic algorithm’, *European Journal of Operational Research*, Vol. 202, pp. 320-327.

Han, C.C., Lin, K.J. and Hou, C.J. (1996) ‘Distance-constrained scheduling and its applications in real-time systems’, *IEEE Transactions on Computers*, Vol. 45, pp. 814-826.

Henderson, D., Jacobson, S.H. and Johnson, A.W. (2003) ‘The Theory and Practice of Simulated Annealing’, Chapter 10 in *Handbook of Metaheuristics*, Eds. Glover and Kochenberger, Kluwer Academic Publishers, pp. 287-319.

Herrmann, J.W. (2007) ‘Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling’, *Technical Report*, University of Maryland, USA. Available at http://hdl.handle.net/1903/7082.

Herrmann, J.W. (2009) ‘Using aggregation to reduce response time variability in cyclic fair sequences’, *Journal of Scheduling*, doi 10.1007/s10951-009-0127-7.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) ‘Optimization by simulated annealing’, *Science*, Vol. 220, pp. 671-680.

Kubiak, W. (1993) ‘Minimizing variation of production rates in just-in-time systems: A survey’, *European Journal of Operational Research*, Vol. 66, pp. 259-271.

León, D., Corominas, A. and Lusa, A. (2003) ‘Resolución del problema PRV min-var’, *Technical report IOC-DT-I-2003-03*, Universitat Politècnica de Catalunya, Spain.

Miltenburg, J. (1989) ‘Level schedules for mixed-model assembly lines in just-in-time production systems’, *Management Science*, Vol. 35, pp. 192-207.

Monden, Y. (1983) ‘Toyota Production Systems’, *Industrial Engineering and Management Press*, Norcross, GA.

Waldspurger, C.A. and Weihl, W.E. (1994) ‘Lottery Scheduling: Flexible Proportional-Share Resource Management’, *First USENIX Symposium on Operating System Design and Implementation*, Monterey, California.

Waldspurger, C.A. and Weihl, W.E. (1995) ‘Stride Scheduling: Deterministic Proportional-Share Resource Management’, *Technical Report MIT/LCS/TM-528*, Massachusetts Institute of Technology, MIT Laboratory for Computer Science. Available at https://eprints.kfupm.edu.sa/67117

Wei, W.D. and Liu, C.L. (1983) ‘On a periodic maintenance problem’, *Operations Research Letters*, Vol. 2, pp. 90-93.