UNIVERSITAT POLITÈCNICA DE CATALUNYA

Tesis doctoral:

METALGORITMO DE OPTIMIZACIÓN COMBINATORIA MEDIANTE LA EXPLORACIÓN DE GRAFOS

Director de la tesis:

Dr. D. Albert Corominas Subias

Para optar al Grado de Doctor Ingeniero Industrial, presenta:

Rafael Pastor Moreno

Barcelona, Junio 1999

5. PROCEDIMIENTOS GENERALIZADOS DE EXPLORACIÓN.

En la extensa literatura publicada sobre la resolución de problemas de optimización combinatoria mediante la exploración de grafos de estados, se han presentado diversos procedimientos generales de búsqueda. Entre todos éstos, se puede considerar que algunos son más claros, accesibles, generales e integradores que otros, e incluso en alguna formalización, como la de Corrêa (1995), se llega a exponer otras técnicas de búsqueda como casos particulares del procedimiento general *branch and bound* propuesto: los procedimientos *branch and bound* basados en programación lineal, el algoritmo A* y la técnica *branch and cut.*

En los apartados siguientes se exponen y analizan brevemente los procedimientos presentados por los autores referenciados. De todas maneras, y en general, se pueden enumerar las siguientes carencias que afectan, en mayor o menor grado, a la mayoría de formulaciones generales de *branch and bound* expuestas:

- En algunas ocasiones se utiliza una terminología poco familiar en el área de la investigación operativa (sobretodo en referencias provenientes del área de la inteligencia artificial), que repercute negativamente en la comprensión y aplicabilidad práctica de la formulación propuesta. Además, generalmente se trabaja con una escasa concreción de los elementos utilizados.

- Muchos procedimientos de búsqueda en grafos de estados no son considerados, sobretodo los aparecidos más recientemente. Además, la mayoría de los existentes tampoco son especificados como casos particulares de los procedimientos generales propuestos.

- Generalmente no se incluye el uso de técnicas de reformulación y preproceso (las técnicas de reducción), ni en el problema original, ni en los subproblemas parciales.

- No se incorporan técnicas de optimización local ni de exploración de entornos, ni en los subproblemas intermedios ni en los finales.

- Tampoco se impone el cálculo de nuevas cotas superiores (en caso de minimizar) en los subproblemas parciales.

- Usualmente no se considera la posibilidad de utilizar más de un procedimiento de acotado, reformulación, reducción, etc., en función del estado de evolución de la arborescencia, del tiempo restante de cálculo, etc.

- Las estrategias de guiado de la búsqueda son normalmente estáticas. Y, además, usualmente no se tienen en cuenta indicadores generales que consideren de manera explícita el cumplimiento de criterios.

5.1. Corominas & Companys (1977).

Antes de estudiar el trabajo de Corominas & Companys (1977), cabe destacar que existen algunas otras aportaciones expuestas con anterioridad, por ejemplo las formalizaciones presentadas por Agin en 1966, Hervé en 1968, Roy en 1969 y Mitten en 1970, además del *survey* de Lawler & Wood de 1966[24]. De todas maneras, éstas no se estudian en la presente tesis debido a que ya han sido analizadas en la mayoría de aportaciones que sí son incluidas.

El objetivo del trabajo de Corominas & Companys (1977) consiste, según los propios autores, en elaborar una formulación de los procedimientos *branch and bound* que, por su generalidad, comprenda como casos particulares a los detectados en la literatura hasta ese momento y que, por ello, constituya un marco teórico en el que inscribir las diversas implementaciones existentes y otras que puedan derivarse, inclusive las de tipo heurístico, por relajación de alguna o algunas de las condiciones.

Entre todas las características concretas de la formulación general descrita, a continuación se presentan aquéllas que se pueden considerar como más sobresalientes y que, por tanto, constituyen los puntos fuertes de la misma:

- Se distingue claramente entre el conjunto de soluciones F (donde tienen lugar los procedimientos de separación) y un subconjunto de éste, el conjunto de soluciones factibles E, que es el conjunto donde realmente se realiza la búsqueda.

- Posibilidad de finalizar la búsqueda con soluciones ε-óptimas respecto al conjunto de soluciones factibles no halladas (obteniendo, de esta manera, procedimientos de carácter heurístico).

- Presenta reglas de eliminación de vértices basadas, según los autores, en consideraciones razonables (por ejemplo, limitaciones en la memoria y/o el tiempo de cálculo), con el objetivo de obtener nuevamente soluciones heurísticas.

[24] Las referencias concretas se corresponden con Agin (1966), Hervé (1968), Roy (1969), Mitten (1970) y Lawler & Wood (1966).

- Los vértices se definen como una pareja formada por el conjunto de soluciones incluidas en una partición de F, más un número natural; esta representación permite trabajar con arborescencias y, si se desea, la eliminación de vértices equivalentes.

- Propone la posibilidad de utilizar más de un procedimiento de acotado de los vértices, de diferente dificultad de cálculo y grado de ajuste.

- Presenta una estrategia de guiado de la búsqueda potencialmente dinámica, que permite variar dinámicamente la estrategia de exploración al definir una función de búsqueda que depende del tiempo transcurrido y/o de la información obtenida en el curso del propio desarrollo del algoritmo: cota de los vértices, su profundidad, su accesibilidad (por ejemplo, su ubicación en memoria interna o externa), la disponibilidad de los resultados de los cálculos (usualmente, si el vértice es el último que se ha explorado o no) y de su grado de separación.

- Tiene una consideración dinámica del proceso de construcción de la arborescencia: posibilidad de tener en cuenta la información que se va obteniendo a medida que avanza el algoritmo para modificar las variables que definen el estado de los vértices ya examinados. Concretamente, dicha información puede ser el tiempo transcurrido (o mejor, el tiempo permitido del que se dispone), la información que se ha ido obteniendo en el curso del propio algoritmo, una intervención humana o un intercambio de información hombre-máquina.

- Comprende como casos particulares a todos los procedimientos generales relatados en la literatura hasta ese momento, además de permitir derivar otros nuevos procedimientos de búsqueda.

El análisis de la formulación general presentada en Corominas & Companys (1977), también permite enumerar un conjunto de puntos débiles de la misma:

- En el proceso de separación no se contempla la separación parcial, únicamente se trabaja con particiones de los subconjuntos de soluciones de F (aunque posteriormente sí se comenta la posibilidad de que no lo sean).
- En la formulación no se expone el hecho de utilizar técnicas de reformulación y preproceso (técnicas de reducción), ni en el problema original, ni en los subproblemas parciales.
- No se presentan con concreción las relaciones de dominancia entre vértices, ni las propiedades de obligado cumplimiento por todas o alguna solución óptima.
- No se incorporan técnicas de optimización local ni de exploración de entornos, ni en los vértices intermedios ni en los finales.

- No se incluye el cálculo de cotas superiores, en caso de minimizar, en los nuevos vértices que se van generando.
- Muchos procedimientos de búsqueda actuales no son considerados.
- En las estrategias de selección del próximo vértice a expandir, no tienen en cuenta indicadores generales que consideren de manera explícita el cumplimiento de criterios.

5.2. Kumar & Kanal (1983a).

Kumar & Kanal (1983a) presentan una formalización del procedimiento *branch and bound* para la resolución de árboles AND/OR, que, según los propios autores, es más general y simple que las anteriores "... *we have developed a formulation which is more general and also much simpler than existing formulations;* ..." (Kumar & Kanal 1983a, p. 180).

De todas maneras, el objetivo que se plantean consiste en intentar aclarar las confusiones existentes en la literatura procedente del área de la inteligencia artificial, concernientes a las relaciones entre diferentes algoritmos de búsqueda en árboles AND/OR, representación de los problemas de optimización combinatoria que, como se expone en las hipótesis de trabajo (apartado 1.2.), queda fuera del ámbito de estudio de esta tesis. Además, y como los mismos autores especifican, la formulación expuesta carece de suficiente grado de detalle "... *a number of details heve been left out.*" (Kumar & Kanal 1983a, p. 187). Por otro lado, no aporta ningún elemento que no sea introducido por alguna de las otras referencias consideradas, y no es suficientemente general ya que no considera un gran número de procedimientos de búsqueda enumerativos, entre otros, las técnicas de programación dinámica.

5.3. Kumar & Kanal (1983b).

En el presente apartado se expone la formalización que realizan Kumar & Kanal (1983b) con el objetivo de unificar diversos procedimientos de búsqueda, como la programación dinámica y los *branch and bound.* Dicha formalización es utilizada para la resolución de grafos AND/OR, fuera de los límites de estudio impuestos en este trabajo. De todas formas se analiza para poner de manifiesto los puntos débiles que presentaría en la resolución de árboles OR, como una clase particular de árboles AND/OR, y para mostrar las grandes diferencias existentes con la terminología usualmente utilizada en el área de la investigación operativa, en esta clase de procedimientos de búsqueda.

Kumar & Kanal (1983b) presentan el concepto de proceso de decisión compuesto como una formulación unificada de diversos procedimientos de búsqueda: búsqueda heurística, programación dinámica y procedimientos *branch and bound.* Por otro lado, comentan que la metodología desarrollada permite ver muchos de estos procedimientos de forma unificada, además de ayudar a sintetizar nuevas variantes y generalizaciones de los procedimientos de búsqueda existentes.

Sea un problema de optimización combinatoria que se puede formular como encontrar el elemento de menor coste en un conjunto X, usualmente estructurado. A menudo, es posible ver X como un conjunto de políticas en un proceso de decisión multietapa (que sugieren estrategias de búsqueda de programación dinámica), o como un conjunto de caminos entre dos estados de un espacio de estados (procedimientos de búsqueda heurística), o como un conjunto de árboles (técnicas *branch and bound*). Kumar & Kanal (1983b) desarrollan el concepto de proceso de decisión compuesto como un modelo general de formulación de problemas de optimización combinatoria; además, también desarrollan un procedimiento sistemático para explotar el conocimiento especifico del problema en el descubrimiento eficaz del elemento óptimo de X, y, según ellos mismos, muestran que muchos de los procedimientos de búsqueda existentes son casos especiales de este procedimiento.

Un proceso de decisión compuesto (CDP) es una tripleta C = (G, t, c), donde:

- G = (V, N, S, P) es una gramática de contexto libre, en la que V, N, P, y S son, respectivamente, los conjuntos de símbolos finales, de símbolos no finales, de producciones, y el símbolo inicial (en una terminología más propia de investigación operativa deben corresponder con el conjunto de vértices finales, el de no finales, las reglas de ramificación y el vértice raíz).
- t denota el conjunto de atributos de coste asociados con las producciones de G: un valor del atributo de coste $t_p(., ..., .)$ está asociado con cada producción p = "w $\rightarrow$ w_1 ...w_k" de G (por tanto, se debe referir al incremento en la función objetivo que suministra cada símbolo hijo generado).
- c es un valor de la función de coste definido sobre el conjunto de símbolos terminales de G.

El conjunto de árboles analizados con raíz el símbolo S de G, representa el conjunto X del problema de optimización formulado por C. Se asigna un coste f(T) a cada árbol analizado T de G, en función del conjunto de atributos de coste t y de la función c.

Se define un valor para la función c_T sobre los vértices n de un árbol analizado T de G, como:

(i) Si n es un símbolo terminal, entonces $c_T = c(n)$.
(ii) Si n es un símbolo no terminal y n_1, ..., n_k son sus descendientes en T (situación que implica que G tiene una producción p = "$n \rightarrow n_1 ... n_k$"), entonces $c_T(n) = t_p(c_T(n_1), ..., c_T(n_k))$.

De esta manera, si n es el símbolo raíz de T, entonces $f(T) = c_T(n)$.

Para un vértice del árbol analizado T, $c_T(n)$ es el coste del subárbol de T con raíz n. Para una producción p: "$n \rightarrow n_1 ... n_k$", $t_p(x_1, ..., x_k)$ es el coste derivado de un árbol T con raíz n, si los costes de los subárboles de T con raíz n_1, ..., n_k son, respectivamente, x_1, ..., x_k. Por tanto, el coste del árbol analizado es recursivamente definido en términos del coste de sus subárboles.

De esta forma, un problema de minimización para el proceso de decisión compuesto C puede ser relatado como sigue: encontrar un árbol analizado T* con raíz el símbolo S tal que $f(T^*) = \min\{f(T) \mid T$ es el árbol analizado con raíz S$\}$.

Un proceso de decisión compuesto monótono (MCDP) es una clase de proceso de decisión compuesto (CDP), para el que el problema de minimización se puede reducir al problema de resolver un sistema de ecuaciones recurrentes. Se dice que un CDP, C = (G(V, N, S, P), t, c), es un MCDP, si todos los atributos de coste t_p asociados con las producciones p: $n \rightarrow n_1 ... n_k$, son monótonamente no decrecientes en cada variable: $x_1 \leq y_1$ & ... & $x_k \leq y_k \Rightarrow t_p(x_1, ..., x_k) \leq t_p(y_1, ..., y_k)$.

Para un símbolo n de G, sea c*(n) el mínimo de los costes de los árboles analizados con raíz n. El siguiente teorema establece que para un MCDP, c*(n) puede ser definido recursivamente en términos de $\{c^*(m) \mid m$ es una parte de un string directamente derivable de n$\}$.

Teorema 1: para un MCDP, C = (G, t, c), se cumple la siguiente ecuación recursiva:

(i) Si n es un símbolo terminal, entonces c*(n) = c(n).
(ii) Si n es un símbolo no terminal, entonces $c^*(n) = \min\{t_p(c^*(n_1), ..., c^*(n_k) \mid p: n \rightarrow n_1 ... n_k$ es una producción en G$\}$.

Existen muchos casos especiales para los que se han desarrollado algoritmos que resuelven las ecuaciones para calcular c*(S). A menudo, esos algoritmos pueden ser fácilmente modificados para construir el árbol analizado de menor coste con raíz S; en

tal caso, el problema de minimización de un MCDP es equivalente a resolver un sistema de ecuaciones recursivas.

Una vez definidos los MCDPs, Kumar & Kanal (1983b) exponen las relaciones existentes con la programación dinámica. La resolución de un problema de optimización mediante la programación dinámica de Bellman, convierte la resolución del problema de optimización en la resolución de un conjunto de ecuaciones recursivas. Como muchos problemas de optimización combinatoria resolubles mediante programación dinámica pueden ser formulados en un formato de MCDP, se puede considerar el MCDP como una formulación general de la programación dinámica, y las ecuaciones recursivas del Teorema 1 como las ecuaciones recursivas generalizadas de la programación dinámica.

En cuanto a la extensa aplicabilidad del CDP, Kumar & Kanal (1983b) comentan que es obvia cuando se anuncia la correspondencia natural entre la gramática libre de contexto y los grafos AND/OR. Debido a esta correspondencia, exponen que la especificación de problemas mediante grafos AND/OR puede ser vista como su especificación en términos de CDP, y el problema de encontrar el árbol de mínimo coste de un grafo AND/OR llega a ser equivalente al problema de minimización en su correspondiente CDP.

Para la resolución del problema de minimización de un MCDP, estos autores comentan que el árbol de menor coste de G con raíz en S puede ser identificado, generando y evaluando únicamente un número finito de árboles de G. Para ello, se centran en la programación dinámica y en los procedimientos *branch and bound.*

a) Programación dinámica generalizada. Un camino consiste en encontrar sucesivamente c*(n) para los símbolos n de G, hasta encontrar c*(S). Visto en términos de los grafos AND/OR, los problemas grandes son sucesivamente resueltos comenzando por los problemas más pequeños. Históricamente, muchos de estos procedimientos se han denominado programación dinámica.

b) *Branch and bound* generalizado. Se comienza con la representación del total de conjuntos de árboles para los que se necesita encontrar el árbol de menos coste. Este conjunto se parte repetidamente, eliminando todos los miembros de la partición para los que se puede demostrar que, después de su eliminación, todavía existe un árbol de mínimo coste en el resto de conjuntos no eliminados. Este procedimiento es conocido con el nombre de *branch and bound.*

Después de analizar el procedimiento presentado por Kumar & Kanal (1983b), se pueden exponer un conjunto de puntos débiles que presenta dicho procedimiento:

- Terminología poco familiar en el área de la investigación operativa, que repercute negativamente en una muy complicada comprensión y difícil aplicabilidad práctica, además de una escasa concreción de los elementos expuestos.
- Muchos procedimientos enumerativos de búsqueda no son considerados.
- No se utilizan técnicas de reformulación y preproceso, ni en el problema original, ni en los subproblemas parciales.
- No se incorporan técnicas de optimización local ni de exploración de entornos, ni en los problemas parciales intermedios ni en los finales.
- No se especifica el cálculo de cotas superiores, en caso de minimizar, ni tan siquiera en el problema inicial.
- Las estrategias de guiado de la búsqueda no son dinámicas. Además, no se tienen en cuenta indicadores generales que consideren de manera explícita el cumplimiento de criterios.
- No se contempla la posibilidad de utilizar diversos procedimientos de acotado, reformulación, reducción, etc., en función del estado de evolución de la arborescencia, del tiempo restante de cálculo, etc.

5.4. Nau et al. (1984).

Nau et al. (1984) exponen una formulación de un algoritmo *branch and bound* general que, según sus propias palabras, incluye formulaciones previas como casos especiales de ésta "... *This paper presents a formulation of B&B general enough to include previous formulations as special cases, ...*" (Nau et al. 1984, p. 29). El esquema que proponen estos autores es de los más integradores de los que se comentan en este apartado, y, en particular, especifican como casos particulares de éste: los procedimientos *branch and bound*, el *best-first branch and bound*, los *branch and bound* con relaciones de dominancia y el procedimiento A*.

Después de definir los problemas de optimización combinatoria y la necesidad de distinguir entre los conjuntos de soluciones y su representación explícita (en un sentido semejante a la distinción que se hace en Corominas & Companys (1977), entre el conjunto de soluciones F y el conjunto de soluciones factibles E), estos autores unifican, como técnicas de podado, cualquier procedimiento que permite eliminar un vértice si se puede demostrar que al menos uno de los vértices que son conservados contiene una solución objetivo (según los propios autores, con esta definición incorporan de una manera fácil las técnicas de podado definidas hasta ese momento, aunque sólo

especifican las de acotación). Según la definición anterior, parece que sólo se busca una solución objetivo (en la terminología habitual en el campo de la investigación operativa, una solución óptima).

A continuación, Nau et al. (1984) exponen mediante un sencillo algoritmo su formulación general de los procedimientos *branch and bound*: un procedimiento compuesto por un proceso iterativo de selección, separación y podado (en el sentido amplio descrito anteriormente). Como características más interesantes cabe destacar que en este procedimiento se permite la selección y separación de varios vértices en cada iteración del algoritmo, y, por otro lado, que no se conserva la mejor solución alcanzada.

Como casos particulares de su esquema anterior, los autores proponen los siguientes procedimientos:

a) Un procedimiento de *branch and bound* como el anterior, al que, además, incorporan el hecho de conservar la mejor solución factible; con ésta el proceso de podado es más potente.

b) El algoritmo *best-first branch and bound*, formado por el esquema presentado en a) en el que la función de selección es la búsqueda primero el mejor (*best first search*).

c) Un *branch and bound* "convencional", constituido por la formulación presentada en a) excepto la posibilidad de seleccionar y separar más de un vértice, y con un proceso de podado constituido únicamente por la comparación entre la cota inferior y el valor de la solución preferible (de esta manera, cualquier procedimiento *branch and bound* con un proceso de podado más elaborado deja de ser considerado "convencional").

d) Un *branch and bound* "convencional" aumentado con el uso de relaciones de dominancia para podar. Como su propia nombre indica, se trata del esquema presentado en c) al que se adiciona el podado debido a las relaciones de dominancia. En este punto parece curioso el hecho de que estos autores no introduzcan inmediatamente las relaciones existentes con la programación dinámica, y cómo ésta podría ser un caso particular de su procedimiento general en el que el podado sólo se realizara gracias a las relaciones de dominancia.

e) El procedimiento A* es considerado un esquema b) en el que la bondad de un vértice se define como se expone en el apartado 3.6.8. para el A* (como se puede comprobar, en este procedimiento no incorporan las relaciones de dominancia).

f) Para finalizar se especifica el algoritmo AO*, un procedimiento para grafos AND/OR y que, por tanto, no es objeto de estudio de esta tesis.

El análisis de la formulación general presentada por Nau et al. (1984), permite enumerar un conjunto de puntos fuertes y débiles de la misma:

a) Puntos fuertes observados: el más destacable consiste en que el esquema general comprende como casos particulares a diversos procedimientos relatados de la literatura; además, se muestran claramente las relaciones entre los procedimientos incorporados y cómo la concreción del modelo general permite obtenerlos.

b) Puntos débiles del procedimiento:

- La terminología utilizada no es muy usual en el área de la investigación operativa. Pero es la muy escasa concreción de los elementos utilizados, la que dificulta la comprensión y aplicabilidad del esquema propuesto.
- Aunque los autores exponen que otros procedimientos pueden ser considerados casos especiales de su esquema general, no los especifican; de esta manera, existen muchos procedimientos de búsqueda en grafos de estados no son considerados explícitamente.
- En la dirección del comentario anterior, no se especifica la posibilidad de desarrollar procedimientos enumerativos de carácter heurístico.
- Tampoco se especifica la posibilidad de utilizar técnicas de reformulación y preproceso ni en el problema original, ni en los subproblemas parciales; ni la incorporación de técnicas de optimización local y de exploración de entornos.
- Las estrategias de guiado de la búsqueda no son dinámicas. Además, no se especifica el tener en cuenta indicadores generales que consideren de manera explícita el cumplimiento de criterios.

5.5. Ibaraki (1988).

La formulación general que se expone en Ibaraki (1988) ha sido tomada como hilo conductor en la descripción de la técnica *branch and bound* (apartado 3.5.1.), y, por tanto, ya ha sido ampliamente comentada. De esta manera, en el presente apartado únicamente se detallan aquellas características que no han sido suficientemente descritas o que no han sido especificadas.

Ibaraki (1988) descompone las técnicas de búsqueda en grandes funciones, operaciones o elementos, para, posteriormente, formular su procedimiento generalizado como una combinación de todos éstas: operaciones de ramificación, estrategias de búsqueda, funciones de acotado inferior, funciones de acotado superior y relaciones de dominancia.

Este autor describe un algoritmo *branch and bound* para la obtención de una única solución óptima (véase el apartado 3.5.1.1.), y mediante una leve variación del proceso de finalización obtiene, además, el cuerpo de un algoritmo *branch and bound* diseñado para obtener todas las soluciones óptimas -como aclara, eliminando únicamente los peores subproblemas para las relaciones de dominancia y el test de cota inferior, y no los que son de igual valor que los ya explorados-.

En función de la concreción de diversos componentes de su algoritmo *branch and bound* general, Ibaraki (1988) especifica diferentes procedimientos de búsqueda: la búsqueda en profundidad (apartado 3.3.3.), la búsqueda en anchura (apartado 3.3.4.), la búsqueda primero el de mejor cota (apartado 3.3.6.), la búsqueda primero el mejor (apartado 3.3.7.) y procedimientos de potencia heurística (apartado 3.6.13.) que Ibaraki (1988) considera parte integrante de las técnicas de búsqueda primero el mejor.

Además expone otros procedimientos enumerativos que se obtienen realizando búsquedas en subconjuntos de subproblemas que cumplen alguna propiedad concreta (ser los de mejor indicador, ser los más profundos, etc.), y que, en función de los parámetros de control que éstos incorporan, permiten obtener algunas de las técnicas de exploración expuestas en los apartados anteriores: la búsqueda en profundidad-m (apartado 3.6.17.), el *parallel depth first search* (apartado 3.3.10.), la búsqueda en anchura limitada (apartado 3.6.18.), el *floating down search* (apartado 3.3.11.) y el *jump backtracking* (apartado 3.3.12.). Por último expone los procedimientos *barried method* (apartado 3.6.19.): técnicas de búsqueda utilizadas cuando existen limitaciones en la memoria disponible.

A pesar de describir la poda por dominancias, en la formulación general no se hace ninguna referencia a la utilización de técnicas de reformulación y preproceso. Este tema es comentado al exponer procedimientos particulares de resolución de dos problemas de optimización combinatoria: problemas de programación entera (en los que dichas técnicas se aplican antes de comenzar la resolución del problema) y problemas de cubrimiento (en los que se puede reducir los subproblemas por preproceso de los datos).

A partir de su formulación general Ibaraki (1988) describe el algoritmo A*, señalando que se trata de un algoritmo *branch and bound* con una estrategia de búsqueda primero el mejor, en la que se selecciona para expandir el vértice P_i activo con menor valor de la función $f(P_i)$, definida ésta como se expone en el apartado 3.6.8. para el A*; por otro lado, en este procedimiento no incorpora el test de dominancia.

En cuanto a los procedimientos de búsqueda basados en programación dinámica (apartado 3.4.), este autor comenta que es posible interpretarlos como casos especiales de *branch and bound* que sólo utilizan el test de dominancia; aunque, al mismo tiempo,

los *branch and bound* pueden ser interpretados como técnicas de programación dinámica a las que se adiciona un test de cota inferior.

Por otro lado introduce brevemente la implementación y ejecución paralela de los algoritmos *branch and bound*, pero sin especificarla en su formulación general.

Ibaraki (1988) también describe diversos algoritmos heurísticos *branch and bound*, derivados directamente de su formulación general y como consecuencia de realizar la búsqueda en un área restringida. De esta forma, expone tres procedimientos heurísticos puros: método de la ε-asignación, método del T-corte y método del M-corte, y también diversos métodos híbridos de los anteriores: ε+T, ε+M, T+M y ε+T+M; además, muestra una generalización del método ε-asignación, como consecuencia de relajar la efectividad del test de dominancia (apartados 3.1.2. y 3.9.1.).

Para evaluar el funcionamiento de los procedimientos *branch and bound*, este autor define diversos parámetros de control: T (número total de subproblemas generados), F (número de problemas parciales generados hasta obtener la primera solución factible), B (número de subproblemas generados hasta obtener la solución óptima) y M (número máximo de vértices activos en cualquier momento de la ejecución). Según Ibaraki (1988), calcular teóricamente los parámetros T, B y M, es deseable pero muy difícil, por lo que se puede recurrir a un modelo de simulación de la ejecución del *branch and bound*: se construye un árbol de ramificación y se generan los valores $f(P_i)$ (valor de la función objetivo del vértice P_i), $g(P_i)$ (cota inferior del vértice), $u(P_i)$ (cota superior del vértice) y $h(P_i)$ (estimación heurística del valor exacto de $f(P_i)$), aleatoriamente de acuerdo con funciones apropiadas de distribución de probabilidad. Las funciones que propone dependen de la dificultad del problema, de la precisión de la función de acotado inferior y de la función heurística; por otro lado, no incorpora las relaciones de dominancia ya que dependen fuertemente de la estructura particular del problema[25].

De esta manera Ibaraki (1988) prueba diferentes estrategias de búsqueda (búsqueda primero el de mejor cota, búsqueda primero el mejor, búsqueda en profundidad, búsqueda en profundidad-m y los algoritmos de búsqueda *barried method*), observando el comportamiento del algoritmo *branch and bound* en función de los resultados de los parámetros T, B y M. Con esta información, Ibaraki construye un árbol de decisión para seleccionar la mejor estrategia de búsqueda, aunque, como él mismo reconoce, las características del problema y las necesidades de programación también deben ser tenidas en cuenta; además se puede argumentar que el modelo puede ser muy simple comparándolo con situaciones reales, ya que las hipótesis de partida son muy restrictivas: el modelo únicamente utiliza árboles binarios y de profundidad constante de

[25] Las funciones de distribución de probabilidad utilizadas se pueden encontrar en Ibaraki (1976a) y en Ibaraki (1988).

los vértices terminales, como mucho únicamente se calcula un valor de la cota superior en el vértice raíz, no se incorporan las relaciones de dominancia, y existen hasta cuatro parámetros iniciales que el usuario debe fijar de partida (tamaño del problema, dificultad del ejemplar, precisión de la función de acotado inferior y la precisión de la función heurística h) sobre cuya calibración no se realiza ningún comentario. Por otro lado, ya en 1976 este mismo autor -en Ibaraki (1976b)- trabaja sobre este tema y comenta que los resultados obtenidos son cualitativos y que sería interesante hallar resultados de forma cuantitativa.

Otros autores también han trabajado en esta dirección, aunque los resultados que aportan no son concluyentes. Por ejemplo, Lenstra & Rinnooy Kan (1978) puntualizan varias imprecisiones en la exposición de Bellmore & Malone, en la que utilizan la simulación para la resolución de un problema de TSP asimétrico: Lenstra & Rinnooy Kan consideran que la elección de las funciones de probabilidad no son fáciles y son interdependientes entre los diferentes vértices, critican la función de probabilidad utilizada (dudando de que sea una cota inferior de lo que pretende ser), y comentan que los argumentos dados son insuficientemente rigurosos y que las hipótesis de Bellmore & Malone se basan en resultados empíricos. Como se puede comprobar, existe una gran dificultad en hallar leyes de probabilidad correctas y en obtener conclusiones serias sobre este tema.

Después de analizar el procedimiento general presentado por Ibaraki (1988), se pueden exponer los siguientes puntos fuertes y débiles del mismo:

a) Puntos fuertes observados:

- De muy fácil comprensión e inmediata aplicabilidad. Además, se muestran numerosos ejemplos de aplicaciones a diversos problemas de optimización combinatoria, mostrando con gran concreción los elementos utilizados.
- Enumeración de diversas variantes del procedimiento de acotado, de la estrategia de selección del próximo vértice a explorar, de la forma de determinar la variable de ramificación, etc.
- Se permite el cálculo de nuevas cotas superiores, en caso de minimizar, en todos los subproblemas parciales.
- Se muestran claras relaciones entre algunos procedimientos y cómo la concreción de unos permite obtener otros.

b) Puntos débiles del procedimiento:

- Algunos procedimientos derivados se obtienen realizando búsquedas en subconjuntos de subproblemas, lo que se aparta de la formulación general.

- Existen muchos procedimientos que no son considerados.
- No incluye en la formulación general, el hecho de utilizar técnicas de reducción ni en el problema original, ni en los subproblemas parciales.
- En las estrategias de selección del próximo vértice a expandir no se tienen en cuenta indicadores generales que consideren de manera explícita el cumplimiento de criterios, únicamente en los procedimientos de potencia heurística (apartado 3.6.13.) se incluye una función de ponderación entre la estimación heurística del vértice y su profundidad.
- No se incorporan técnicas de optimización local.
- No se considera la posibilidad de utilizar más de un procedimiento de acotado, reformulación, reducción, etc., en función del estado de evolución de la arborescencia, del tiempo restante de cálculo, etc.

5.6. Helman (1989).

Helman (1989) presenta un nuevo esquema que formaliza los algoritmos de programación dinámica y *branch and bound*, aunque, como ocurre en la formalización de Kumar & Kanal (1983b), éste presenta una terminología poco clara, accesible y usual, en el campo de la investigación operativa.

La formulación de Helman (1989) agrupa a los procedimientos citados anteriormente (programación dinámica y *branch and bound*), comentando que ambos utilizan relaciones de dominancia (que el autor divide en relaciones de comparabilidad -en el sentido habitual de agrupar vértices equivalentes- y relaciones de acotado) para actuar sobre el orden de enumeración en un espacio de soluciones finito, y que, esencialmente, son estrategias del mismo tipo. Como el mismo autor señala, la característica principal de este esquema consiste en el uso que los procedimientos de búsqueda enumerativos hacen de estas relaciones de dominancia.

El modelo general propuesto incorpora ambas clases de dominancias, y, según se utilicen las relaciones de comparabilidad o las de acotado, se definen los procedimientos de programación dinámica o los algoritmos *branch and bound*. Por otro lado, Helman comenta que su formulación general también incluye variantes de los algoritmos clásicos de programación dinámica y *branch and bound*: uno de éstos es un procedimiento híbrido de programación dinámica y *branch and bound* que se obtiene al incorporar ambos tipos de relaciones de dominancias; como se puede comprobar, continúan siendo procedimientos basados en las relaciones de dominancia. Principalmente por este motivo, estas variantes parecen estar muy limitadas: no se hace ninguna referencia a otros posibles procedimientos a utilizar (técnicas de reducción, de

exploración de entornos, etc.) ni a la obtención de procedimientos heurísticos enumerativos derivados directamente de la formulación general.

Helman (1989) también introduce las relaciones de optimalidad de las soluciones (en el sentido introducido en el apartado 3.5.1.4.) y expone brevemente que su formulación va bien para implementaciones y ejecuciones en paralelo, aunque ésta no es incluida en el modelo.

Después de comentar y analizar brevemente la formulación general presentada en Helman (1989), es posible enumerar los siguientes puntos fuertes y débiles de la misma:

a) Puntos fuertes observados:

- Muestra con claridad las relaciones existentes entre los procedimientos de programación dinámica y los de *branch and bound.*
- Se presentan algunos ejemplos de aplicaciones a problemas de optimización combinatoria.

b) Puntos débiles del procedimiento:

- Terminología insuficientemente familiar en el área de la investigación operativa, que repercute en una difícil comprensión y aplicabilidad práctica.
- La mayoría de procedimientos de búsqueda no son considerados.
- En la formulación presentada no se expone el hecho de utilizar técnicas de reducción ni en el problema original, ni en los subproblemas parciales.
- Las estrategias de selección del próximo vértice a tratar no son dinámicas. Por otro lado, tampoco se tiene en cuenta indicadores generales que consideren de manera explícita el cumplimiento de criterios.
- No se incorporan técnicas de optimización local ni de exploración de entornos, ni en los vértices intermedios ni en los finales.
- No se considera la posibilidad de utilizar más de un procedimiento de acotado, reformulación, reducción, etc., en función del estado de evolución de la arborescencia, del tiempo restante de cálculo, etc.

5.7. Corrêa (1995).

Corrêa (1995) propone una formulación general de los algoritmos *branch and bound*, para la búsqueda de soluciones (sub)óptimas en un espacio de soluciones discreto; introduce la implementación paralela de su esquema general; y muestra cómo varios de los algoritmos de búsqueda más concocidos (A*, *branch and bound* basados en

programación lineal y *branch and cut*) pueden ser considerados casos particulares de su procedimiento. Además, comenta que su esquema general permite obtener soluciones subóptimas, constituyendo de esta forma un procedimiento general heurístico.

Este autor define un problema de optimización discreto restringido, introduciendo la distinción entre el conjunto de soluciones y el conjunto de soluciones factibles, que es el que realmente se hace explícito (como ya hacen Corominas & Companys 1977). Por otro lado, únicamente busca una solución óptima.

En primer lugar, define una función de selección heurística del próximo vértice a tratar que identifica, como ejemplo, con la búsqueda el primero el mejor y la búsqueda en profundidad. Además introduce un operador de evaluación, que en realidad consiste en una operación de separación y de resolución heurística en cada subproblema generado. Por otro lado, Corrêa (1995) unifica como técnicas de podado, cualquier procedimiento que permite eliminar vértices que no son necesarios en la búsqueda, de todas maneras esta operación es incorporada únicamente como un podado por cotas inferiores y dominancias. A continuación, Corrêa (1995) expone mediante un sencillo algoritmo su formulación general de los procedimientos *branch and bound.*

Para ilustrar la generalidad del esquema presentado, el autor propone los siguientes procedimientos como casos particulares de su formulación:

a) El algoritmo A* puede ser visto como un caso especial de su formulación general, si se considera que el espacio de búsqueda está representado por un grafo de estados (representación utilizada de manera exclusiva en la presente tesis): la estrategia de guía de la exploración es la búsqueda primero el mejor (en este caso, el de mejor cota inferior); se incluye la poda por dominancias pero no es especificada la poda por cota inferior; y, sobre todo, la función de acotado del procedimiento es la usualmente definida para el algoritmo A* (apartado 3.6.8.).

b) Para problemas de programación entera, Corrêa (1995) considera a los *branch and bound* basados en programación lineal, como aquellos procedimientos que para la obtención de cotas en los vértices de la arborescencia utilizan la programación lineal; en este caso incluye la poda por dominancias y por cota inferior.

c) La búsqueda *branch and cut* también puede obtenerse del modelo general planteado, al generar un conjunto relativamente pequeño de restricciones, mediante planos de corte, que adicionadas al problema relajado reduzcan sustancialmente el *gap* entre el valor de sus soluciones óptimas.

Para finalizar su exposición, Corrêa (1995) introduce como contribución más destacable de su artículo, según sus propias palabras, la implementación paralela de los algoritmos *branch and bound*, comentando que la formulación en paralelo de un algoritmo *branch and bound* consiste, únicamente, en la definición paralela de la lista de subproblemas a tratar mediante una multilista paralela. Este autor sí especifica brevemente los cambios que se deberían realizar en algunas de las etapas del algoritmo secuencial para obtener un procedimiento general en paralelo; de todas formas, éstos dependen de la estructura de paralelización seleccionada (en su caso de alto nivel o granularidad gruesa).

El análisis del procedimiento general presentado por Corrêa (1995), permite enumerar un conjunto de puntos fuertes y débiles del mismo:

a) Puntos fuertes observados:

- Se distingue entre el conjunto de soluciones y el conjunto de soluciones factibles que se hace explícito.
- Posibilidad de obtener procedimientos de carácter heurístico, permitiendo la obtención de soluciones subóptimas.
- Comprende como casos particulares a diversos procedimientos relatados en la literatura; además, muestra claramente las relaciones entre dichos procedimientos y cómo la concreción del modelo general permite obtenerlos.
- Se introduce el hecho de calcular cotas superiores, en caso de minimizar, en los subproblemas parciales.
- Se comentan las relaciones de dominancia entre vértices, pero no las propiedades de obligado cumplimiento por todas o alguna solución óptima.

b) Puntos débiles del procedimiento:

- La terminología utilizada no es usual en el área de la investigación operativa. Pero es la escasa concreción de los elementos utilizados, la que dificulta la aplicabilidad del esquema propuesto.
- Existe una gran cantidad de procedimientos de búsqueda que no son especificados.
- No se expone claramente el hecho de utilizar técnicas de reducción; de todas formas en su formulación estas técnicas podrían tener cabida.
- En el esquema general no se incorporan técnicas de optimización local ni de exploración de entornos, ni en los vértices intermedios ni en los finales.
- No se contempla la posibilidad de utilizar diversos procedimientos de acotado, reformulación, reducción, etc., en función del estado de evolución de la arborescencia.

- Las estrategias de guiado de la búsqueda no son dinámicas. Por otro lado, no tienen en cuenta indicadores generales que consideren de manera explícita el cumplimiento de criterios.

6. ANÁLISIS DEL ESTADO DEL ARTE Y CONCLUSIONES PRELIMINARES.

Una vez descritos los diversos procedimientos de búsqueda de la literatura referenciada y después de analizar sus relaciones y semejanzas, a continuación se presenta una serie de conclusiones, críticas y aseveraciones, sobre el tema de la resolución de problemas de optimización combinatoria mediante estos procedimientos enumerativos de exploración:

a) Se puede plantear una fuerte crítica desde el punto de vista industrial, ya que la mayoría de procedimientos de búsqueda utilizados habitualmente, parece que no han sido diseñados para resolver situaciones reales sino más bien problemas teóricos o de dimensiones nada interesantes.

b) Existe una gran confusión en las definiciones de los procedimientos y en la terminología empleada, en parte debida a que éstos han sido utilizados tanto en el área de la investigación operativa como en el de la inteligencia artificial. Como resultado se presenta una dispersión, confusión e incluso duplicidad entre dichas técnicas, que dificulta en gran medida la selección de los procedimientos para resolver problemas de optimización combinatoria.

c) Los procedimientos descritos poseen diversos elementos en común, cinco o seis, cuya especificación permite obtenerlos: la combinación de un escaso número de elementos permite obtener la mayoría de los algoritmos referenciados.

d) Como algunos autores exponen, fácilmente se puede comprobar cómo algunos procedimientos derivan de otros; de todas formas, estas derivaciones no son todo lo generales que podrían llegar a ser.

e) En el caso extremo, un procedimiento de este tipo enumeraría y crearía todo el árbol de búsqueda, resultando una técnica muy ineficaz. Para reducir la duración de la búsqueda, evitando explorar zonas donde no se encuentra la solución deseada, existen, en definitiva, tres tácticas generales de reducción del espacio de estados:

- eliminación de estados dominados;
- poda de vértices del árbol de búsqueda mediante el cálculo de cotas y de propiedades de la solución óptima;
- y, por último, detección de vértices vacíos o con algunos o todos sus descendientes vacíos, mediante técnicas de reducción.

f) Las técnicas de consistencia pueden incorporarse, y de hecho ya lo hacen en algunos procedimientos, en algoritmos *branch and bound*, lo que relaciona y une, aún en mayor

grado, las técnicas de búsqueda provenientes del área de la investigación operativa y de la inteligencia artificial.

g) Hay algunos elementos de vital importancia que no son tenidos en cuenta en los procedimientos relatados. Por ejemplo, la mayoría de estrategias de selección únicamente son función del vértice considerado y no tienen en cuenta ni el estado de la exploración ni las condiciones de entorno en las que se resuelve el problema.

De esta manera, parece conveniente establecer una formulación general que englobe a todos estos procedimientos. Esta necesidad ya ha sido detectada por varios autores que han expuesto diferentes versiones de procedimientos generales de resolución de problemas de optimización combinatoria mediante la exploración de grafos de estados. Pero dichas formulaciones presentan un conjunto de carencias que, en mayor o menor grado, afectan a la mayoría de ellas:

- En algunas ocasiones se utiliza una terminología poco familiar en el área de la investigación operativa, que repercute negativamente en su comprensión y aplicabilidad práctica. Además, generalmente se trabaja con una escasa concreción de los elementos utilizados.

- Muchos procedimientos de búsqueda en grafos de estados no son considerados, sobretodo los aparecidos más recientemente.

- Las estrategias que guían las búsquedas usualmente son estáticas.

- Generalmente no se incluyen los elementos del siguiente conjunto de técnicas de búsqueda: uso de la reformulación y preproceso (tanto en el problema original como en los subproblemas parciales); incorporación de las técnicas de optimización local y de exploración de entornos (en los subproblemas intermedios y en los finales); imposición del cálculo de nuevas cotas superiores (en caso de minimizar) en los subproblemas parciales; y uso de más de un procedimiento de acotado, reformulación, reducción, etc., en función del estado de evolución de la arborescencia, del tiempo restante de cálculo, etc.

7. *BRANCH AND WIN*: METALGORITMO DE RESOLUCIÓN DE PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA MEDIANTE LA EXPLORACIÓN DE ÁRBOLES OR.

7.1. Introducción.

Las conclusiones obtenidas en el apartado anterior al analizar el estado del arte de los procedimientos enumerativos de búsqueda en árboles OR, evidencian la necesidad de diseñar un nuevo procedimiento general, realista y con una terminología habitual en el área de la investigación operativa, para la resolución de problemas de optimización combinatoria mediante este tipo de técnicas.

Para cubrir esta necesidad se ha diseñado un procedimiento al que se ha denominado *branch and win*, un metalgoritmo para la resolución de problemas de optimización combinatoria mediante la exploración de árboles OR. Cabe comentar que el antecedente más cercano de *branch and win*, tomado como punto de partida de este metalgoritmo, es el procedimiento generalista presentado en Corominas (1996), que a su vez está fundamentado en el trabajo de Corominas & Companys (1977).

Entre las características de *branch and win* destacan las siguientes:

a) Puede incorporar aspectos importantes en la resolución de problemas combinatorios de tamaño industrial: tiempo de cálculo permitido, tiempo restante, cantidad de memoria utilizada, etc.

b) Usa una terminología unificadora y clara.

c) Utiliza los diversos elementos en común que poseen todos estos procedimientos de búsqueda para, en función de su especificación y combinación, obtener dichas técnicas enumerativas.

d) Permite el uso de procedimientos de acotación, reducción, resolución heurística y exploración de entornos, en todos y cada uno de los subproblemas generados.

e) Incorpora una función general de evaluación y selección del próximo vértice a explorar, como una estrategia general de guía de la búsqueda, que tiene en cuenta elementos de vital importancia que no son considerados en las técnicas referenciadas: se toma en consideración el estado de evolución de la exploración y las condiciones de entorno en las que se resuelve el problema.

f) Y, por último, permite diseñar nuevos procedimientos de búsqueda, como resultado de realizar nuevas combinaciones de los elementos que integran a los procedimientos referenciados.

Es decir, *branch and win* constituye un procedimiento general que resuelve las carencias que presentan las diversas formulaciones de procedimientos generales de resolución de problemas de optimización combinatoria mediante la exploración de árboles OR, referenciadas en esta tesis.

A continuación, se expone un conjunto de definiciones previas de los elementos que intervienen en *branch and win* (apartado 7.2.); se formaliza el metalgoritmo *branch and win* (apartado 7.3.); y, para finalizar, se detallan los procedimientos utilizados, así como algunas características del metalgoritmo propuesto (apartado 7.4.).

7.2. Definiciones previas.

En primer lugar se definen una serie de elementos que intervienen en la formalización del metalgoritmo *branch and win*: problema a resolver, objetivo, vértices, procedimientos de separación, de acotación, etc.

7.2.1. Problema a resolver. Objetivo.

El problema que se desea resolver responde al esquema general ya presentado en el apartado 2.1. y definido para poder tratar la programación lineal mixta, uno de los problemas de optimización combinatoria más importante. Se propone una definición general de problema de optimización combinatoria, en la que se formaliza que una vez definido el valor de las variables discretas se dispone de un procedimiento para resolver óptimamente el resto del problema. De esta forma, el problema original se puede reducir de forma inmediata al siguiente esquema:

$$[\text{MIN}]\ Z = f(X)$$
$$X \in E \subseteq F,$$

donde:

F es un conjunto finito de soluciones (habitualmente el producto cartesiano de los dominios de las variables del problema),

E es el conjunto finito de soluciones factibles,

X es una solución,

$f: E \rightarrow \kappa$, donde κ es un conjunto ordenado (normalmente los números enteros o reales).

7.2.2. Vértice.

Sean P(F) el conjunto de las partes de F y N el conjunto de los números naturales; se define un vértice V_i de la siguiente manera:

$$V_i \in P(F) \times N,$$

es decir, $V_i = (S_i, i)$ y consta de las dos partes siguientes: $S_i \in P(F)$, $i \in N$.

7.2.3. Vértice vacío y vértice terminal.

Un vértice V_i es vacío $\Leftrightarrow S_i \cap E = \varnothing$ (es un vértice que no contiene ninguna solución factible).

Un vértice V_i es terminal $\Leftrightarrow |S_i| = 1$ (es un vértice que contiene una única solución, factible o no).

7.2.4. Procedimiento de separación.

Se denomina procedimiento de separación primitivo (designado por M_0), a un procedimiento que hace corresponder a los elementos de P(F) con cardinal igual o superior a dos, dos o más partes de F. El procedimiento de separación primitivo, M_0, separa las partes de F, y además lo hace sin eliminar ninguno de los elementos de P(F). Con este objetivo, si $M_0(S) \neq \varnothing$ y $S \in P(F)$, M_0 cumple las siguientes propiedades:

1) $Y \in M_0(S) \Rightarrow Y \neq S$,
2) $\cup_{\{\forall Y \in M_0(S)\}} Y = S$,
3) $\forall Y, Z \in M_0(S) \Rightarrow Y \cap Z \neq Y$ (se permite que Y y Z sean conjuntos disjuntos o que tengan cierta intersección, pero no que uno incluya al otro).

Una vez definido un procedimiento de separación primitivo M_0, se puede definir un procedimiento de separación (designado por M), como una correspondencia que a un vértice no terminal le hace corresponder un subconjunto de vértices (terminales o no). Es decir, un procedimiento de separación M consiste en aplicar un procedimiento de separación primitivo M_0 a S_i, y en asignar un número natural a los S_{ij} creados a partir de S_i, de forma que nunca se asigne el mismo número natural más de una vez.

La aplicación de este procedimiento de separación se inicia en el vértice $V_1 = (F, 1)$, que se denomina vértice inicial o raíz, y continúa en los demás vértices proporcionando una aborescencia (en el sentido definido en el apartado 1.1.2.).

A continuación se describen dos propiedades que presentan los procedimientos de separación M:

1) Como se desprende de la segunda propiedad de M_0, el procedimiento de separación M no produce pérdida de vértices, ni factibles ni no factibles.
2) De la definición de M_0 y de su primera propiedad, se desprende inmediatamente que la arborescencia es finita: en un conjunto finito, se realizan separaciones que generan conjuntos de soluciones no coincidentes.

Por otro lado, se supone que un vértice terminal es reconocible como tal y resulta inmediato determinar si éste es vacío o no. De esta forma, si V_i es terminal o se ha detectado que es vacío, entonces no tiene sucesores (término introducido en el apartado siguiente).

7.2.5. Vértice sucesor o precedente y vértice descendiente o antecesor.

Definido un procedimiento de separación M, se pueden definir los siguientes conceptos:

1) V_j es un vértice sucesor o descendiente inmediato de V_i, y V_i es un vértice precedente o antecesor inmediato de $V_j \Leftrightarrow V_j \in M(V_i)$.
2) V_j es un vértice descendiente de $V_i \Leftrightarrow V_j \in M(V_i) \cup M^2(V_i) \cup M^3(V_i) \cup ...$
3) V_j es un vértice antecesor de $V_i \Leftrightarrow V_j \in M^{-1}(V_i) \cup M^{-2}(V_i) \cup M^{-3}(V_i) \cup ...$

Cuando se aplica la correspondencia M y se obtiene un vértice sucesor, se dice que dicho vértice ha sido generado en el proceso de separación.

7.2.6. Vértice separado y totalmente separado.

Un vértice V_i está separado $\Leftrightarrow$ se ha generado uno o algunos de sus sucesores.

Un vértice V_i está totalmente separado $\Leftrightarrow$ se han generado todos sus sucesores.

7.2.7. Procedimiento de acotación.

Se dice que p_e es el procedimiento de acotación e, si a cada vértice V_i le hace corresponder $C_e(V_i) \in \kappa$, de forma que $C_e(V_i) \leq f(X)$, $\forall X \in S_i \cap E$ y $\forall p_e \in P$ (donde P es un conjunto finito de procedimientos p_e). Entonces se dice que $C_e(V_i)$ es una cota

inferior del valor de las soluciones factibles contenidas en el vértice V_i, obtenida con el procedimiento de acotación p_e. El hecho de trabajar con varios procedimientos de acotación usualmente permite disponer de una sucesión de procedimientos que se pueden ordenar de menor a mayor dificultad de cálculo, pero que, por contra, normalmente proporcionan una calidad creciente del ajuste.

En el conjunto P existe un procedimiento p_α tal que, si V_i es un vértice terminal no vacío, $C_\alpha(V_i) = f(X)$, $X \in S_i$ (entre los procedimientos de acotación se dispone de al menos uno, p_α, que proporciona el valor de la función objetivo en los vértices terminales no vacíos).

Dados dos procedimientos de acotación p_e y p_e', diremos que p_e' es más potente que p_e: $p_e \leq p_e' \Leftrightarrow C_e(V_i) \leq C_e'(V_i)\ \forall\ V_i$ (es decir, p_e' siempre proporciona cotas tanto o más ajustadas que p_e).

Sea P_i el conjunto de procedimientos de acotación que se han aplicado en el vértice V_i; sea C la mejor cota asociada a un vértice V_i, es decir $C(V_i) = \max\{C_e(V_i)\}$, $\forall\ p_e \in P_i$; y sea $\overline{C}(V_i) = \max\{C(V_i), \upsilon\}$, donde υ es una cota obtenida al tener información del vértice precedente o de los sucesores. De esta forma:

1) $V_j \in M(V_i) \Rightarrow \overline{C}(V_j) = \max\{C(V_j), C(V_i)\}$.
2) $\overline{C}(V_i) = \max\{C(V_i), \min\{C(V_j)\}\}$, $\forall\ V_j \in M(V_i)$; por tanto, para este cálculo es necesario que V_i esté totalmente separado.

7.2.8. Procedimiento de reducción.

Un procedimiento de reducción e, $r_e \in R$ (donde R es un conjunto finito de procedimientos r_e), consiste en sustituir un vértice $V_i = (S_i, i)$ por un nuevo vértice $V_i' = (S_{ie}', i)$, de forma que $S_{ie}' \subset S_i$ y, si además V_i no es vacío, $\exists\ X^*_{Si} \in S_{ie}'$ en el caso de que sólo se busque una solución óptima (donde X^*_{Si} representa una solución óptima en S_i) o no exista $X^*_{Si} \in S_i \setminus S_{ie}'$ si se desean todas las soluciones óptimas.

La reducción puede concluir en fijar una serie de variables o en la disminución de sus rangos de valores, aunque también se puede detectar una solución óptima de S_i, lo cual, si sólo se busca una solución óptima, permite reducir el conjunto S_i a esa solución y convertir el vértice en terminal.

Dados dos procedimientos de reducción r_e y r_e', diremos que r_e' es más potente que r_e: $r_e \leq r_e' \Leftrightarrow S_{ie}' \subseteq S_{ie}'$ (y como consecuencia, r_e' siempre proporciona partes de F de menor o igual cardinal que r_e).

En este procedimiento también es posible transmitir la información resultante del proceso de reducción, de vértices sucesores a precedentes y viceversa:

1) $V_j \in M(V_i) \Rightarrow S_j' = S_j \cap S_i'$.
2) $S_i' = \cup_{\forall V_j \in M(Vi)} S_j'$. Hay que tener presente que esta propiedad sólo se puede utilizar si V_i está totalmente separado.

De esta definición, fácilmente se puede comprobar cómo entre los procedimientos de reducción se encuentran todas las técnicas descritas en el apartado 3.8.

7.2.9. Procedimiento de resolución heurístico.

Se dice que $h_e \in H$ es un procedimiento de resolución heurístico (donde H es un conjunto finito de procedimientos h_e), si a cada vértice V_i intenta hacerle corresponder una solución factible $X \in S_i \cap E$; entonces $U_e(V_i) = f(X)$ es una cota superior del valor de las soluciones factibles contenidas en el vértice V_i, obtenida con el procedimiento de resolución heurístico h_e.

Dados dos procedimientos de resolución heurísticos h_e y h_e', diremos que h_e' es más potente que h_e: $h_e \leq h_e' \Leftrightarrow U_e(V_i) \geq U_e'(V_i)\ \forall\ V_i$ (es decir, h_e' siempre proporciona soluciones factibles tanto o más próximas a la solución óptima que h_e).

Sea H_i el conjunto de procedimientos de resolución heurísticos que se han aplicado al vértice V_i; sea $U(V_i)$ el valor de la mejor solución factible encontrada asociada al vértice V_i, es decir $U(V_i) = \min\{U_e(V_i)\},\ \forall\ h_e \in H_i$; y sea $\overline{U}(V_i) = \min\{U(V_i), \upsilon\}$, donde υ es el valor de una solución obtenida al tener información de sus vértices sucesores. De esta forma:

$$\overline{U}(V_i) = \min\{U(V_i), \min\{U(V_j)\}\},\ \forall\ V_j \in M(V_i)$$

Los procedimientos de resolución heurísticos h_e no garantizan encontrar siempre una solución factible, $X \in S_i \cap E$; para unos problemas particulares siempre se encuentra una solución factible (por ejemplo, problemas de secuencias), pero para otros puede resultar extremadamente difícil. De todas formas, lo más usual es que sí se obtengan soluciones factibles.

Cabe tener presente que se pueden definir procedimientos de optimización local o de exploración de entornos e_e (introducidos en el apartado 7.2.14.), como parte integrante de un procedimiento más elaborado de resolución heurístico h_e.

7.2.10. Procedimiento de examen.

Examinar un vértice V_i consiste en aplicarle procedimientos de acotación y/o de reducción y/o de resolución heurísticos, con el objetivo de mejorar la información de que se dispone sobre dicho vértice y, a veces, también sobre otros.

7.2.11. Procedimiento de evaluación.

Se denomina procedimiento de evaluación, a una correspondencia que a cada vértice V_i le hace corresponder $\varphi(V_i) \in R$, donde φ constituye la función de evaluación y selección del próximo vértice a tratar.

7.2.12. Relaciones de dominancia.

Se dice que un vértice V_i domina a otro V_j, $V_i \succ V_j$, si $f(X^*_{Si}) \leq f(X^*_{Sj})$ si sólo se desea una solución óptima o $f(X^*_{Si}) < f(X^*_{Sj})$ si se buscan todas las soluciones óptimas. En particular, se puede asegurar que se cumple esta condición si $\forall X_{Sj} \in (S_j \cap E)\ \exists X_{Si} \in (S_i \cap E) \mid f(X_{Si}) \leq f(X_{Sj})$ si sólo se desea una solución óptima o $f(X_{Si}) < f(X_{Sj})$ si se buscan todas las soluciones óptimas.

A modo de aclaración del concepto anterior, a continuación se define una relación de dominancia para el problema del viajante de comercio, definido éste en un esquema polietápico que consiste en decidir iterativamente la próxima ciudad a visitar a partir de la última visitada: generados dos vértices V_a y V_b, con idéntica ciudad de partida, visitando las mismas ciudades y con igual ciudad en último lugar, el vértice V_a domina al vértice V_b, $V_a \succ V_b$, si el valor del camino acumulado desde el vértice origen hasta V_a es menor o igual que el valor acumulado desde el vértice origen hasta V_b.

7.2.13. Propiedades de la solución óptima.

Estudiando las características del problema a resolver, en algunas ocasiones se pueden determinar algunas propiedades que han de cumplir las soluciones óptimas. Se pueden presentar dos situaciones:

a) Toda solución óptima cumple una propiedad Π.
b) Siempre existe una solución óptima que cumple la propiedad Π'.

7.2.14. Procedimiento de exploración de entornos.

Para un vértice V_i terminal no vacío, se define un procedimiento de exploración de entornos $e_e \in EN$ (donde EN es un conjunto finito de procedimientos e_e), como un procedimiento que inspecciona soluciones factibles del entorno de la solución en curso (en primera instancia, la solución contenida en V_i), hasta que se cumple alguna condición de fin.

Los procedimientos de exploración de entornos que aquí se contemplan, pueden ser tanto las técnicas usualmente denominadas de optimización local (por ejemplo, k-intercambios), como aquellos procedimientos habitualmente conocidos como metaheurísticas y que incluyen al recocido simulado, la búsqueda tabú, los algoritmos genéticos, los algoritmos GRASP, los algoritmos de hormigas, etc.

Los procedimientos de exploración de entornos pueden resultar especialmente interesantes si se aplican en vértices que contienen una solución factible de cierta calidad (por ejemplo, pasa a ser la solución preferible o está muy cerca del valor de ésta). Y se puede alcanzar un doble resultado:

- si se alcanza una nueva solución preferible, se actualiza y se obtienen las consecuencias subsiguientes (posibilidad de descartar -estado de un vértice definido en el apartado siguiente- nuevos vértices);
- puede permitir mejorar la cota superior de otro u otros vértices con las consecuencias derivadas (posibilidad de descartar dicho vértice si las cotas inferior y superior coinciden).

7.2.15. Estados de un vértice.

Una vez definidos los diferentes procedimientos que se pueden aplicar en un vértice, a continuación se describen los estados que éstos pueden presentar como consecuencia de aplicarles dichos procedimientos:

1) Generado o no generado (obtenido, o todavía no obtenido, en el proceso de separación).
2) Descartado o no descartado: para descartar un vértice se debe tener la certeza de que no contiene ninguna solución que pueda interesar; un vértice descartado no es tenido en cuenta y para descartarlo se debe dar, como mínimo, una de las siguientes condiciones:
   - ser vacío;
   - ser terminal y no vacío, y conocer el valor de la solución f(X) si $X \in E$;

- $\overline{C}(V_i) \geq \overline{Z}$ si sólo se desea una solución óptima o $\overline{C}(V_i) > \overline{Z}$ si se buscan todas las soluciones óptimas, donde $\overline{Z}$ es el valor de la solución preferible;
- $\overline{C}(V_i) = \overline{U}(V_i)$;
- estar dominado por otro vértice V_j: $V_j \succ V_i$;
- no contener soluciones que cumplan la propiedad Π', en caso de sólo buscar una sola solución óptima, o la propiedad Π, en caso de buscar todas las soluciones óptimas;
- estar totalmente separado y con todos sus sucesores descartados;
- tener a su precedente descartado;
- no cumplir una condición de permanencia del procedimiento que lo convierte en un procedimiento heurístico: no estar entre los mejores cuando se agota la memoria, cuando se alcanza un número máximo de vértices generados, etc.

3) Virtualmente examinado por un procedimiento de reducción r_e: se ha determinado S_{ie}' que no puede ser reducido por la aplicación de r_e.

4) Virtualmente examinado por un procedimiento de acotación p_e aplicado en S_i: se conoce una cota de S_i que no se puede mejorar por la aplicación de p_e en S_i.

5) Virtualmente examinado por un procedimiento de resolución heurístico h_e aplicado en S_i: se ha determinado una solución $X \in S_i \cap E$, que no puede ser mejorada con h_e.

6) Totalmente examinado: vértice virtualmente examinado por todos los procedimientos de reducción, de acotación y de resolución heurísticos.

7) Cerrado o no cerrado: un vértice está cerrado, si está descartado o bien si todos sus sucesores, si los tiene, están totalmente examinados.

8) Evaluado o no evaluado: un vértice V_i está evaluado si se conoce el valor de $\varphi(V_i)$. Para ciertas funciones de evaluación este estado es permanente (por ejemplo, la profundidad), pero para otras es efímero y únicamente se puede utilizar para realizar una selección, ya que una vez realizada pierde todo su valor indicativo (por ejemplo, al depender del tiempo restante permitido para finalizar la búsqueda).

7.3. Formalización del metalgoritmo *branch and win*.

Una vez definidos los elementos principales que intervienen en el metalgoritmo *branch and win*, a continuación se introduce su formalización. Antes, sin embargo, se definen una serie de elementos que se utilizan en el metalgoritmo, y, después, se concretan las subrutinas del mismo.

7.3.1. Definiciones.

$\overline{X}$: solución preferible.
$\overline{Z} = f(\overline{X})$: valor de la función objetivo para la solución preferible.
n: número natural asociado al vértice generado.
$V_1 = (F, 1)$: vértice raíz.
L: lista de vértices generados, no descartados y no cerrados.
$\hat{V}$, $\hat{V}'$ y $\hat{V}''$: vértices.

7.3.2. Formalización de *branch and win*.

Cabe destacar que, en la formalización que se expone seguidamente, el objetivo consiste en minimizar y, además, solamente se busca una única solución óptima, considerándose obvios los cambios que es necesario realizar en el caso de maximizar o buscar todas las soluciones óptimas.

Branch and win se formaliza en tres fases que, brevemente, tienen las siguientes funciones:

> *Fase 1.* Inicialización del algoritmo: obtener una primera cota superior de todas las soluciones factibles, o, en su defecto, inicializar ésta a $+\infty$; inicializar el contador de vértices generados a 1; generar el vértice raíz de la arborescencia; inicializar, con este vértice, la lista de vértices candidatos a ser tratados con alguno de los procedimientos de examen definidos.
>
> *Fase 2.* Cuerpo del algoritmo: seleccionar el próximo vértice a tratar; tratar el vértice en función de que sea un vértice terminal, un vértice no terminal totalmente separado u otro tipo de vértice.
>
> *Fase 3.* Finalización del algoritmo: mostrar los resultados requeridos.

A continuación se describe el metalgoritmo *branch and win*:

Fase 1. INICIO.

Obtener una solución inicial $\overline{X}$

$\overline{Z} = f(\overline{X})$, o, si se desconoce $\overline{X}$, $\overline{Z} = +\infty$

Hacer n = 1

Generar $V_1 = (F, 1)$

Hacer $L = \{V_1\}$

Fase 2. CUERPO.

Hacer hasta que $L = \varnothing$

SELECCIONAR_DE_L ($\hat{V}$)

Según $\hat{V}$ hacer

caso $\hat{V}$ es terminal

EXAMINAR_TERMINAL ($\hat{V}$)

caso $\hat{V}$ está totalmente separado

SELECCIONAR_DE_FAMILIA ($\hat{V}$')

EXAMINAR_GENERAL ($\hat{V}$')

otro caso

NUEVO_SUCESOR (Nd)

Si Nd = 1 entonces

GENERAR ($\hat{V}$'')

EXAMINAR_GENERAL ($\hat{V}$'')

sino

SELECCIONAR_DE_FAMILIA ($\hat{V}$')

EXAMINAR_GENERAL ($\hat{V}$')

fin si

fin según

fin hacer

Fase 3. EXHIBIR.

Mostrar los resultados que sean necesarios

7.3.3. Subrutinas.

En los apartados siguientes se describen las subrutinas que forman parte de *branch and win*.

7.3.3.1. Subrutina *SELECCIONAR_DE_L ( )*.

SELECCIONAR_DE_L ($\hat{V}$):

Hacer para todo vértice $V_i \in L$

Recalcular los sumandos de $\varphi(V_i)$ para los que se desconoce su valor actualizado.

Calcular $\varphi(V_i)$.

fin hacer

Tomar un vértice $\hat{V}$ de entre los de mejor valor $\varphi(V_i)$.

Según sea en cada momento la forma de la función dinámica de evaluación $\varphi(V_i)$, es decir, según qué elementos la compongan (valor de la cota inferior del vértice, de la cota superior, etc.), se debe calcular aquellos valores activos que la forman (están ponderados por parámetros diferentes de 0) y para los que, en ese momento, se desconoce su valor actualizado. Este cálculo puede implicar, por ejemplo, el utilizar en este momento algún procedimiento de resolución heurístico.

7.3.3.2. Subrutina *EXAMINAR_TERMINAL ( )*.

EXAMINAR_TERMINAL ($\hat{V}$):

Seleccionar un procedimiento de examen.

Aplicar el procedimiento de examen.

Si se dispone de f(X) entonces

Si f(X) < $\overline{Z}$ entonces

$\overline{X}$ = X, $\overline{Z}$ = f(X)

fin si

Si se cumple la condición de exploración de entornos hacer

EXPLORAR_ENTORNO ($\hat{V}$)

fin si

Cerrar el vértice y, por consiguiente, L = L \ { $\hat{V}$ }

fin si

Deducir las variaciones en el estado del vértice.

Deducir las variaciones en los otros vértices.

Un procedimiento de examen puede incluir como parte integrante de dicho procedimiento, uno o varios procedimientos de acotación, uno o varios procedimientos de reducción y uno o varios procedimientos de resolución heurísticos. Por otro lado, también es posible diseñar un procedimiento de examen que utilice procedimientos de acotación, reducción y/o resolución heurísticos, de forma iterativa, con lo que, por ejemplo, es posible obtener un procedimiento de examen cota-reducción-cota-

reducción-... que finaliza cuando no se puede reducir o acotar más el subproblema. Además cabe recordar que se pueden definir procedimientos de exploración de entornos, como parte integrante de un procedimiento de resolución heurístico (apartado 7.2.9.). En esta subrutina, el procedimiento de examen únicamente incluye procedimientos de acotación.

Si el vértice no está cerrado, su estado puede variar como consecuencia de encontrarse un valor de una cota que permita descartarlo o por el incumplimiento de las propiedades de la solución óptima.

Por otro lado, es posible que se produzcan variaciones en los estado de otros vértices: al mejorar el valor de la solución preferible $\overline{Z}$ (gracias al examen del vértice $\hat{V}$ o a un proceso de exploración de entornos en $\hat{V}$); si se obtienen nuevas cotas superiores de algunos vértices, al realizar un proceso de exploración de entornos; al propagar el valor de la cota o de la solución, a su vértice precedente; al variar el estado de un vértice pariente: un vértice padre cerrado $\Rightarrow$ todos sus sucesores cerrados, todos los vértices hijos cerrados $\Rightarrow$ su precedente cerrado.

Cuando se deducen las variaciones en los vértices, cualquier vértice que resulta cerrado es, obviamente, eliminado de L.

7.3.3.3. Subrutina *EXPLORAR_ENTORNO ( )*.

EXPLORAR_ENTORNO ($\hat{V}$):

Seleccionar un procedimiento de exploración de entornos.

Aplicar el procedimiento de exploración de entornos.

7.3.3.4. Subrutina *SELECCIONAR_DE_FAMILIA ( )*.

SELECCIONAR_DE_FAMILIA ($\hat{V}$'):

En el conjunto formado por el vértice $\hat{V}$ y todos sus sucesores $\in$ L, tomar un vértice, $\hat{V}$', no totalmente examinado.

7.3.3.5. Subrutina *EXAMINAR_GENERAL ( )*.

EXAMINAR_GENERAL ($\hat{V}$'):

Si $\hat{V}$' es terminal entonces

EXAMINAR_TERMINAL ($\hat{V}$')

sino

Seleccionar un procedimiento de examen.
Aplicar el procedimiento de examen.
Deducir las variaciones en el estado del vértice.
Deducir las variaciones en los otros vértices.

fin si

En esta subrutina, los procedimientos de examen pueden incluir a uno o diversos procedimientos de acotación, de reducción y/o de resolución heurísticos, y, además, de forma iterativa o no.

Si el vértice es cerrado también se descartan (y por tanto se cierran) todos sus descendientes; además, en algunas ocasiones este estado puede propagarse hacia vértices antecesores. Por otro lado, se pueden producir las mismas variaciones de estados que con la subrutina EXAMINAR_TERMINAL ( ), pero además se incluyen aquellas variaciones resultado de comprobar las relaciones de dominancia entre vértices.

Por supuesto, si se detecta que el vértice es vacío, es descartado y cerrado.

7.3.3.6. Subrutina *NUEVO_SUCESOR ( )*.

NUEVO_SUCESOR (Nd):

Si todos los elementos del conjunto $\{\hat{V} \cup [M(\hat{V}) \in L]\}$ están totalmente examinados entonces

Nd = 1

sino

Decidir el valor de Nd (1 si se genera un nuevo sucesor y 0 en caso contrario)

fin si

Si en el conjunto de vértices formado por el vértice $\hat{V}$ y todos sus sucesores $\in$ L, no hay ningún vértice que todavía permita ser examinado, entonces se debe generar de forma forzada otro vértice sucesor; sino, se permite decidir si se genera o no un nuevo vértice sucesor.

7.3.3.7. Subrutina *GENERAR ( )*.

GENERAR ($\hat{V}$ ''):

Concretar el procedimiento de separación.
Concretar la variable que se utiliza para realizar la separación.
n = n + 1.
Generar $\hat{V}$ '' = ($\hat{S}$ '', n).
L = L ∪ {$\hat{V}$ ''}.

Como ya se ha comentado, en el momento de la separación un vértice terminal es reconocible como tal y resulta inmediato determinar si es vacío o no: si lo es, se cierra y no se incorpora a la lista L.

7.4. Detalle de los procedimientos y características de *branch and win*.

Una vez formalizado el metalgoritmo *branch and win*, a continuación se comentan algunas de sus características y se detallan y concretan los procedimientos utilizados.

Ante todo, cabe recordar las siguientes características de *branch and win*: el grafo OR con el que se trabaja es una arborescencia, ya que todos los vértices son diferentes al estar caracterizados por (S, i), donde i es un número natural que coincide con el orden de creación del vértice; *branch and win* es un procedimiento finito, ya que la separación, y por tanto la arborescencia, es finita (se realizan separaciones no coincidentes en un conjunto finito); además, el número de operaciones también es finito y seguro que se encuentra la solución óptima (si un vértice generado no esta cerrado, está en la lista L, y, de esta manera, podrá ser examinado y finalmente cerrado).

Los procedimientos de examen pueden estar formados por procedimientos de reducción que, como se especifica en el apartado 3.8., incluyen técnicas procedentes del área de la investigación operativa: las técnicas de preproceso (apartado 3.8.1.), y técnicas de reducción procedentes del área de la inteligencia artificial: las técnicas de consistencia o de satisfacción de restricciones (apartado 3.8.2.).

En la formulación de *branch and win* no se especifica el uso del paralelismo debido a que, una vez expuestas las posibles tácticas y estrategias a utilizar en la paralelización de un procedimiento de búsqueda enumerativo, éste puede ser considerado en el momento de la programación del algoritmo -buscando un uso eficiente de los procesadores y las comunicaciones-, en vez de integrarlo en la formalización del mismo -como tampoco hacen los autores referenciados, por ejemplo Helman (1989) y Corrêa (1995)-. De todas formas, no conviene olvidar las múltiples posibilidades que el

paralelismo puede presentar en el momento de utilizar *branch and win*, ni tampoco que la ejecución de programas en paralelo y en particular de algoritmos de búsqueda en grafos de estados paralelizados, es un tema actualmente en investigación y desarrollo que ha ido variando rápida y continuamente a lo largo de los últimos años y que, sin duda, lo continuará haciendo.

En los apartados siguientes se detallan con mayor grado de concreción los procedimientos utilizados en *branch and win*.

7.4.1. Inicialización de *branch and win*.

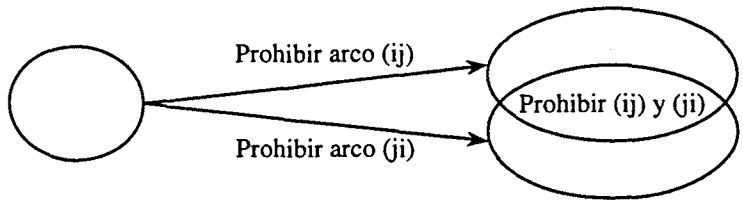
El elemento más importante de la primera fase de *branch and win* consiste en obtener una solución inicial factible, que en principio puede ser cualquiera o, incluso, no facilitarse. De todas formas (recuérdese la exposición realizada en el apartado 3.5.1.5.), usualmente la calidad de la solución inicial tiene una gran influencia en el tiempo necesario para finalizar la búsqueda, ya que puede permitir iniciar desde el primer instante, un eficiente y potente proceso de podado. De esta manera, la idea consiste en incluir en el metalgoritmo la búsqueda de una solución heurística inicial, factible y de calidad, con la que inicializar *branch and win*; cabe destacar que el tiempo que se invierte en buscar una solución inicial de calidad, puede llegar a constituir un factor de éxito a considerar.

Para obtener una solución inicial de calidad se pueden utilizar uno o varios de los procedimientos de resolución heurísticos definidos para el problema (que pueden ser de cualquier tipo, incluso algoritmos heurísticos derivados del propio *branch and win*), completados con algún procedimiento de exploración de entornos. Los procedimientos heurísticos a utilizar y los parámetros de control de los procedimientos de exploración de entornos, deben ser calibrados para cada problema concreto.

7.4.2. Procedimiento de separación.

Como se introduce en el apartado 3.5.1.3., se puede considerar que el procedimiento de separación es uno de los elementos más importantes de *branch and win*, ya que para un mismo problema es posible diseñar diferentes estrategias de separación que pueden influir decisivamente en la eficacia de la búsqueda.

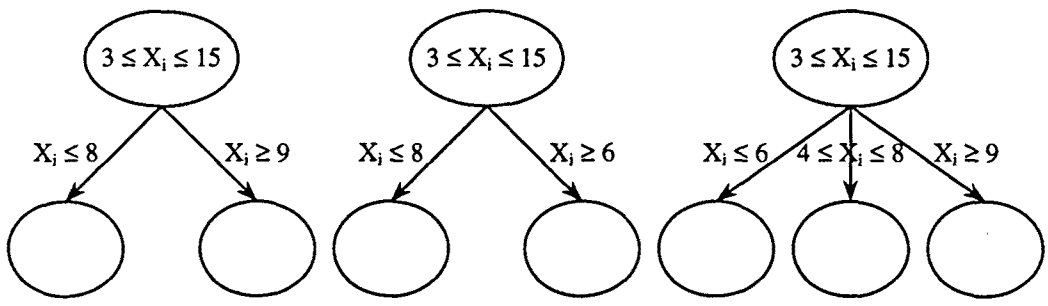
Habitualmente, el procedimiento de separación proporciona particiones del conjunto F; de todas formas, también puede proporcionar separaciones que no constituyen una partición de F. A continuación se expone un procedimiento de separación para el problema del viajante de comercio asimétrico (ASTP), que no proporciona particiones del conjunto de soluciones:



Procedimiento de separación para el problema ASTP, que no produce particiones, de Corominas (1996).

El proceso de separación se realiza seleccionando una variable y asignándole valores, por ejemplo:

- asociándole cada uno de sus posibles valores;
- asociando uno de los valores posibles por un lado y negándolo por el otro;
- asociando a cada vértice sucesor una parte del rango de valores de dicha variable; en un problema de programación lineal entera (PLE), se pueden seguir, por ejemplo, las siguientes estrategias de separación:



Procedimientos de separación en un problema PLE, que proporcionan bi-particiones, bi-divisiones y multi-divisiones.

Como se puede comprobar, para un mismo problema es posible definir diversos procedimientos de separación. A continuación se describen procedimientos de separación para diversos problemas de optimización combinatoria (algunos de los cuales ya han sido introducidos en el apartado 3.5.1.3.):

a) Problema de la mochila. En la versión binaria, únicamente hay un objeto de cada tipo y la decisión consiste en colocar o no el objeto: la separación es una partición binaria. En la versión entera, se dispone de un número entero de objetos de cada tipo y la decisión consiste en colocar 0, 1, 2, ..., N_i objetos del tipo i: la separación proporciona $N_i + 1$ conjuntos de soluciones para cada tipo de objeto a colocar.

b) Problemas de programación mixta. Los procedimientos de separación suelen ser de dos tipos: por un lado, se pueden fijar todos los valores del dominio de la variable x_i; y, por otro, se puede restringir el valor de la variable x_i a $\leq \lfloor \bar{x}_i \rfloor$ ó $\geq \lfloor \bar{x}_i \rfloor + 1$, donde $\lfloor \bar{x}_i \rfloor$ es el resultado de resolver el programa lineal resultante de relajar la condición de integridad de las variables.

c) Problemas de secuenciación u ordenación. En problemas de este tipo, como son los problemas de taller mecánico o el problema de líneas de producción expuesto en el apartado 1.1.1., se suelen fijar los elementos que pueden ir en la primera posición, en la segunda, en la tercera, ..., etc.; en general, es una separación en la que el número de sucesores va disminuyendo a medida que aumenta la profundidad de la arborescencia.

d) Problema del viajante de comercio. Habitualmente se utilizan dos procedimientos de separación: por un lado, se considera que se dispone de unos vértices aislados y se deben elegir unos arcos para completar el circuito: se separa seleccionando o no la arista ij; por otro lado, se puede construir un circuito a partir de una ciudad origen (ciudad 0), de la misma forma que en los problemas de secuenciación: desde la ciudad 0 ir a la 1, o a la 2, o a la 3, o a la 4, ...

Una vez definido el procedimiento de separación, se debe concretar la variable con la que se efectúa dicha separación. En el apartado 3.5.1.8. se comentan diversas estrategias para seleccionar la variable de separación: el criterio propuesto por Padberg & Rinaldi en 1991, en función del índice de imposibilidad de obtener una solución factible en el algoritmo de Balas, y un conjunto de otras estrategias expuestas por Corominas et al. (1984), Ibaraki (1988) y Fernández & Companys (1995). Antes de finalizar, conviene realizar tres precisiones: en vez de utilizar criterios por separado, se puede realizar una ponderación entre alguno de ellos; por otro lado, no es obligatorio utilizar siempre el mismo criterio: si no se dispone de ninguna solución factible se pueden utilizar criterios de factibilidad, y una vez se dispone de una, se puede pasar a utilizar criterios de calidad; además, los criterios pueden ser estáticos o dinámicos: la variable de separación se puede seleccionar de una lista establecida al inicio, o de esta misma lista pero actualizándola dinámicamente con la información disponible de las decisiones ya tomadas.

7.4.3. Procedimiento de examen.

Como se ha definido en el apartado 7.2.10., el procedimiento de examen de un vértice se utiliza con el objetivo de aumentar la información y mejorar el conocimiento sobre dicho vértice. Para ello se pueden utilizar de tres tipos de procedimientos:

a) de acotación, con el objetivo de hallar cotas inferiores del valor de las soluciones factibles contenidas en el vértice (apartado 7.2.7.);
b) de reducción, para eliminar soluciones del vértice pero conservando, si éste no es vacío, un mínimo de una solución óptima del subproblema; la reducción puede consistir en fijar el valor de algunas variables o en la disminución de sus rangos de valores (apartado 7.2.8.);

c) de resolución heurístico, con el objetivo de intentar calcular soluciones factibles del subproblema, que constituyen la cota superior de las soluciones que contiene (apartado 7.2.9.).

De todas formas, y como se introduce en el apartado 7.3.3.2., a la vez también se suele mejorar la información disponible sobre otros vértices.

Si únicamente se dispone de procedimientos de acotación, éstos pueden ser aplicados en orden creciente de su dificultad o coste de cálculo, con la esperanza de que los procedimientos menos costosos proporcionen un valor de la cota que permita cerrar rápidamente el vértice. Por otro lado, hay que tener presente que si al aplicar un procedimiento de acotación se puede asegurar la optimalidad de la solución cuyo valor constituye la cota superior del subproblema, dicho procedimiento de acotación desencadena un proceso de reducción que permite cerrar el vértice.

El disponer de varios tipos de procedimientos de examen, puede permite aplicarlos de forma alternativa e iterativa (apartado 7.3.3.2.): se puede comenzar hallando una solución factible del vértice y posteriormente acotar: si se demuestra la optimalidad de dicha solución el vértice es cerrado, sino se puede aplicar una reducción y luego otra vez calcular la cota que, en caso de haber reducido, probablemente será de mayor calidad. Los procedimientos de reducción son normalmente costosos, de esta manera parece razonable comenzar con los procedimientos de acotación (usualmente menos costosos).

En cuanto a los procedimientos de resolución heurísticos, éstos pueden ser de cualquier tipo; así, es posible utilizar algoritmos heurísticos derivados del propio *branch and win*, de procedimientos de exploración de entornos, o incluso desarrollar algunos basados en los resultados obtenidos de los procedimientos de acotación -recuérdese el procedimiento que exponen Johnson et al. (1997), consistente en el redondeo sucesivo de las variables fraccionales obtenidas del programa lineal resultante de la relajación de la integridad de las variables en un problema de programación lineal mixto (apartado 3.5.1.5.)-.

Recordando la formalización del problema que se desea resolver:

$$[\text{MIN}]\ Z = f(X)$$
$$X \in E \subseteq F,$$

donde E es el conjunto finito de soluciones factibles, a continuación se describen un conjunto de procedimientos e ideas generales para hallar cotas:

a) Substituir el problema a resolver por el siguiente esquema:

$$[MIN]\ Z' = F(X)$$
$$X \in E$$
$$F(X) \leq f(X), \qquad \forall X \in E,$$

es decir, se minimiza una función F( ) que es un minorante de la función objetivo original f( ).

En el problema de cubrimiento, se puede optimizar el número de instalaciones que cubren el número total de clientes, N, aunque éstos no sean diferentes; de esta manera, $F(X) \leq f(X)$ y optimizar F(X) es trivial: se seleccionan instalaciones en orden no creciente del número de clientes que cubren y hasta cubrir N clientes.

b) Substituir el problema a resolver por el siguiente problema de optimización:

$$[MIN]\ Z_R = f(X)$$
$$X \in E_R \supset E,$$

de forma que se cumplen las siguientes condiciones: por una lado, $Z_R^* \leq Z^*$, y, por otro, si X_R^* es la solución óptima para Z_R y $X_R^* \in E$, entonces también es la solución óptima para el problema original Z.

A este tipo de sustitución usualmente se le llama relajación, ya que se relajan las restricciones y, por tanto, el conjunto de soluciones factibles. En la práctica, dicha relajación se puede hacer de dos maneras:

- suprimir restricciones de forma que el problema relajado sea más fácil de resolver, como ocurre con los programas lineales resultado de eliminar las condiciones de integridad de las variables (de todas formas, no conviene eliminar muchas restricciones ya que la cota obtenida puede resultar de muy baja calidad);
- agregar restricciones, lo que constituye la relajación subrogada (introducida en el apartado 3.5.1.6.).

c) Utilizar los dos procedimientos anteriores de forma conjunta; de esta manera el problema a resolver se sustituye por el siguiente esquema:

$$[MIN]\ Z_R = F(X)$$
$$X \in E_R \supset E$$
$$F(X) \leq f(X), \qquad \forall X \in E,$$

donde se minimiza una función F que es un minorante de la función objetivo original f y se cumple que $Z_R^* \leq Z^*$; en este caso, no se tiene porque cumplir la condición: X_R^* solución óptima de Z_R y $X_R^* \in E \Rightarrow X_R^*$ solución óptima de Z.

Una aplicación de esta idea proporciona la relajación lagrangiana (comentada en el apartado 3.5.1.6.).

d) El siguiente problema de optimización:

$$[MIN]\ Z = \sum_i f_i(X_i)$$
$$g_i(X_i, X_0) \leq 0 \qquad \forall\ i$$

con la función objetivo separable, puede ser formulado de una manera equivalente de la forma siguiente:

$$[MIN]\ Z = \sum_i f_i(X_i)$$
$$g_i(X_i, X_0^i) \leq 0 \qquad \forall\ i$$
$$X_0^i = X_0^{i-1}$$

Este programa matemático puede ser substituido por el siguiente conjunto de n problemas de optimización combinatoria (i = 1, ..., n):

$$[MIN]\ Z_i = f_i(X_i)$$
$$g_i(X_i, X_0^i) \leq 0$$

De esta manera $\sum_i Z_i^* \leq Z^*$ y constituye una cota inferior del problema original Z. Además, si el valor de X_0^i es el mismo para todos los problemas parciales que resultan, se obtiene la solución óptima del problema original.

e) Substituir el problema a resolver por un problema relajado de éste y calcular una cota del problema relajado. De esta manera se obtiene una cota del problema relajado, que a su vez constituye una cota del problema original.

A modo de ejemplo se podría tener el siguiente programa matemático primal:

$$(P) \quad [MIN]\ Z = c \cdot X \qquad (Z^*)$$
$$A \cdot X \geq b$$
$$X \geq 0$$
$$X \in Z^+,$$

a partir del cual se puede formular un programa primal relajado:

$$\begin{aligned} (RP) \quad [MIN]\ Z_R &= c \cdot X \qquad (Z_R^*) \\ A \cdot X &\geq b \\ X &\geq 0, \end{aligned}$$

de forma que siempre se cumple la relación $Z_R^* \leq Z^*$.

Por otro lado, también se puede plantear el programa dual del problema relajado, de la siguiente manera:

$$\begin{aligned} (DRP) \quad [MAX]\ W &= U \cdot b \qquad (W^*) \\ U \cdot A &\leq c \\ U &\geq 0 \end{aligned}$$

Y si se encuentra una solución $\overline{U}$ factible (es decir que cumple $\overline{U} \cdot A \leq c$ y $\overline{U} \geq 0$), entonces:

$$\overline{U} \cdot b = \overline{W} \leq W^* = Z_R^* \leq Z^*,$$

y por tanto:

$$\overline{W} \leq Z^*$$

Como se ha comentado, para un mismo problema es posible definir diversos procedimientos de acotación. A continuación se describen procedimientos de acotación para diversos problemas de optimización combinatoria:

a) Problema de la mochila. En este problema la dificultad reside en que los objetos son indivisibles; para obtener una cota se puede considerar que los objetos son divisibles, con lo que su resolución resulta trivial.

b) Problemas de programación entera mixta. En este caso se presenta la condición de integridad de las variables; para calcular una cota se puede prescindir de dicha condición y resolver el programa lineal resultante.

c) Problemas de secuenciación u ordenación. Para el problema de taller mecánico *flow-shop* tipo P, existe un conjunto de cotas tradicionales: por máquinas y por piezas. Para el problema de líneas de montaje, en el apartado 1.1.1. se expone un sencillo procedimiento de acotación. Para el problema de secuencias regulares PRV, se puede

relajar la condición de que el número de unidades de tipo i fabricadas hasta el instante t debe ser menor o igual que ese mismo valor para el instante t + 1; como resultado se tienen menos restricciones y el problema es separable para cada instante t, y, de esta manera, fácil de resolver.

d) Problema del viajante de comercio (TSP). La solución del TSP se caracteriza por ser un grafo parcial conexo, con n arcos y sin circuitos parciales ni horquillas (grado del vértice no superior a 2). Para calcular cotas se pueden seguir diversas estrategias:

- Saltarse todas las condiciones excepto la de n aristas: se seleccionan las n aristas de menor coste.
- Relajar la condición de no horquilla: se trata de encontrar un grafo conexo sin ciclos parciales (es el problema del árbol parcial mínimo, a resolver con el algoritmo de Kruskal con N - 1 aristas), al que se le adiciona una arista más (la de menor coste de las restantes).
- Relajar la condición de no circuito parcial: el problema se reduce a un problema de afectación cuya matriz coincide con la del grafo; aunque el uso del algoritmo húngaro se puede considerar una cota "cara", se puede calcular una cota del problema de afectación, y por tanto una cota de la cota, tal como se hace en el algoritmo de Little.
- Mejorar la cota obtenida en el caso anterior, imponiendo restricciones que impidan los circuitos parciales que se han obtenido (si es que se obtienen): se vuelve a calcular la cota, se vuelven a poner restricciones, etc.

7.4.4. Procedimiento de selección del próximo vértice a tratar.

Como se introduce en el apartado 3.2., en las últimas décadas se han diseñado diversas estrategias de búsqueda para dirigir la exploración, que tienen en común el disponer de un procedimiento para seleccionar el próximo vértice a tratar, entre todos los candidatos; normalmente, se evalúan, ordenan y seleccionan los vértices en función del valor proporcionado por una función de evaluación, usualmente denominada indicador.

A pesar de la enorme importancia que tiene la estrategia de selección del próximo vértice a tratar, en algunos casos ésta no es explicitada o se ha expuesto con escasa concreción. Por otro lado, la generalización de una función de evaluación y selección de vértices, es de vital importancia en la definición de un metalgoritmo de procedimientos de búsqueda, *branch and win* en este caso, ya que en función de la concreción de ésta se pueden definir varios de los procedimientos existentes. Además, es necesario distinguir claramente entre la función de guía del indicador y la de poda de la cota: la función del indicador consiste en guiar la exploración, por lo que éste debe ser lo más ajustado posible a un valor de referencia de la solución (debe ser lo más ajustado posible al valor de la

mejor solución alcanzable, sea superior o inferiormente, o, por ejemplo, a la mitad del valor de la mejor solución alcanzable: si se dispone de un indicador con un valor igual a la mitad del valor de la mejor solución alcanzable, se tiene un indicador perfecto que conduce directamente a la solución óptima); por otro lado, la función de la cota consiste en podar los vértices peores que la solución preferible, por lo que, además de ajustar lo máximo posible, nunca debe sobrestimar a la mejor solución alcanzable.

A lo largo del presente texto, se han expuesto tanto funciones globales con diferentes grados de generalidad como funciones particulares -en los procedimientos concretos en las que éstas son utilizadas-. La mayoría de estrategias de selección, únicamente son función del vértice considerado y no tienen en cuenta ni el estado del grafo de la exploración, ni las condiciones de entorno en las que se resuelve el problema. Desde un punto de vista industrial, parece que existen aspectos que no son suficientemente tenidos en cuenta.

A continuación se propone una formalización general de dicha estrategia de selección del próximo vértice a tratar, que todos estos procedimientos de exploración incorporan. Se propone una función general de evaluación y selección, f(n, t, a_t, e), que, mediante la concreción de diversos parámetros, permite obtener fácilmente diferentes estrategias de búsqueda.

Recopilando las diferentes estrategias expuestas en la literatura, e incorporando la idea de que la estrategia de selección es función del vértice considerado, del tiempo t consumido en la computación, del estado de evolución de la arborescencia en el instante t y de las condiciones del entorno en las que se resuelve el problema, se puede definir la función general f(n, t, a_t, e). En este caso se continúa resolviendo un problema de minimización, con lo que se considera que se selecciona para explorar el vértice de menor valor asociado.

La función f(n, t, a_t, e) depende de:

- n, el vértice considerado,
    - g(n), una estimación de $g^*(n)$: una estimación del coste del camino de mínimo coste desde el vértice raíz s hasta el citado vértice n,
    - h(n), una estimación de $h^*(n)$: una estimación del coste del camino de mínimo coste desde n hasta un estado considerado objetivo,
    - c(n), una cota de la mejor solución alcanzable desde el vértice n, y que puede ser igual a: g(n) + h(n) cuando $g(n) = g^*(n)$ y h(n) cumple la condición de admisibilidad, o, para aquellos problemas para los que algunas de estas funciones no son aplicables o no tienen sentido, igual a l(n), definida ésta como una cota global de la mejor solución alcanzable desde n (para la resolución de programas

lineales enteros mediante el algoritmo de Land & Doig, no tiene sentido hablar de g(n) y de h(n) tal y como se han definido anteriormente, ya que el valor asociado a un vértice no se divide en el coste necesario hasta llegar a dicho vértice y en una estimación del coste necesario para alcanzar un vértice objetivo),

- $l_i(n)$, valor del indicador o criterio de selección y/o desempate i, asociado a una característica que se desea incorporar en la función de evaluación, como por ejemplo: posición del vértice en la arborescencia o carácter más o menos completo de la solución parcial que define, accesibilidad de la información que define el vértice (soporte en el que se encuentra almacenada: memoria RAM, disco duro, etc.), disponibilidad de información para efectuar los cálculos relativos al vértice, etc.; otro posible valor para $l_i(n)$ puede ser el obtenido al resolver, mediante un procedimiento heurístico, el problema parcial en el vértice n: una cota superior,
- r(n), una variable aleatoria,

- t, el tiempo de ejecución transcurrido desde el inicio del algoritmo,

- a_t, que representa el estado de la arborescencia en el instante t: parte del árbol generada, cantidad de memoria ya utilizada, etc.,
- $\overline{Z}$, valor de la solución preferible,
- h_t^*, mejor valor de la cota c(n) de los todos los vértices activos en el instante t,
- v_t, número total de vértices generados hasta el momento,
- va_t, número de vértices $\in$ L en el instante t,
- m_t, cantidad de memoria utilizada,
- R_t, factor máximo de ramificación (número de vértices sucesores de un vértice dado) en el instante t,
- F_t, valor umbral de la función de selección en el instante t (profundidad de corte si mide en número de pasos), que, en caso de minimizar, impide que sea explorado un vértice con un valor asociado del indicador superior a F_t,

- e, que simboliza las condiciones de entorno en las que se resuelve el problema: tiempo máximo de cálculo, memoria máxima disponible, etc.,
- T, tiempo máximo de cálculo permitido,
- V, número máximo de vértices que se permite generar,
- VA_t, número máximo de vértices que pueden estar en L de forma simultánea en el instante t,
- M_t, disponibilidad máxima de memoria en el instante t,
- TG_k, tiempo de acceso al soporte k en el que puede estar almacenado un vértice (k: memoria RAM, disco duro, etc.),

- $f(n, t, a_t, e)$, indicador que evalúa la calidad de los vértices candidatos a ser seleccionados en el proceso de búsqueda,

- $\alpha(n, t, a_t, e)$, función de peso asociada a $g(n)$, de valor constante o dinámico en función de n, t, a_t y e,

- $\beta(n, t, a_t, e)$, función de peso asociada a $h(n)$, de valor constante o dinámico en función de n, t, a_t y e,

- $\delta(n, t, a_t, e)$, función de peso asociada a $c(n)$, de valor constante o dinámico en función de n, t, a_t y e,

- $\gamma_i(n, t, a_t, e)$, función de peso asociada a $l_i(n)$, de valor constante o dinámico en función de n, t, a_t y e,

- $\tau(n, t, a_t, e)$, función de peso asociada a $r(n)$, de valor constante o dinámico en función de n, t, a_t y e.

Por tanto, la función general de evaluación y selección del próximo vértice a tratar de *branch and win*, queda como sigue:

$$f(n, t, a_t, e) = \varphi[g(n), h(n), c(n), l_1(n), ..., l_s(n), r(n); t; \overline{Z}, h_t^*, v_t, va_t, m_t, R_t, F_t; T, V, VA_t, M_t, TG_k; \alpha(n, t, a_t, e), \beta(n, t, a_t, e), \delta(n, t, a_t, e), \gamma_1(n, t, a_t, e), ..., \gamma_s(n, t, a_t, e), \tau(n, t, a_t, e)],$$

y en particular podría ser:

$$f(n, t, a_t, e) = \alpha(n, t, a_t, e) \cdot g(n) + \beta(n, t, a_t, e) \cdot h(n) + \delta(n, t, a_t, e) \cdot c(n) + \gamma_1(n, t, a_t, e) \cdot l_1(n) + ... + \gamma_s(n, t, a_t, e) \cdot l_s(n) + \tau(n, t, a_t, e) \cdot r(n).$$

7.4.5. Propiedades que debe cumplir la solución óptima.

Una estrategia utilizada en *branch and win* consiste en el uso de propiedades que debe cumplir toda solución óptima o tales que existe al menos una solución óptima que las cumple, lo cual puede reducir, a veces de forma considerable, el tamaño del árbol de búsqueda a hacer explícito. De esta manera, una solución total o parcial es factible si cumple dichas propiedades, sino se poda o, aún mejor, no se genera en el proceso de separación (apartado 3.5.1.4.).

A continuación se describen propiedades de las soluciones óptimas para dos problemas de optimización combinatoria:

a) Problema de corte de materiales. Dada una solución cualquiera, es posible obtener soluciones de igual calidad permutando la asignación de los patrones de corte entre los diferentes rollos, de esta manera las permutaciones crean muchas soluciones equivalentes; para impedirlas se puede imponer que la solución conserve un cierto orden, por ejemplo lexicográfico.

b) Problema de secuencias regulares. Existen propiedades que permiten no generar ciertas soluciones; por ejemplo, en el problema de Miltemburg, y para las funciones habitualmente utilizadas, las variantes de tasa mayor se secuencian antes que las de menor tasa.

7.4.6. Relaciones de dominancia.

Comprobar la existencia de relaciones de dominancia entre vértices de la arborescencia, persigue el objetivo de reducir la enumeración explícita, al eliminar aquellos vértices que son equivalentes y conservar únicamente el mejor; de esta manera, sólo se examinan problemas parciales diferentes.

El podado por dominancias puede llegar a ser muy potente, pero normalmente es un recurso "caro" y por eso no es habitualmente utilizado. Además, las simetrías, para las cuales las relaciones de dominancia son muy efectivas, pueden ser tratadas en el momento de formular el problema, impidiendo la generación de vértices equivalentes que podrían dar lugar a soluciones simétricas.

A continuación se introducen las relaciones de dominancia para algunos problemas de optimización combinatoria:

a) Problema de la mochila. En este caso, las relaciones de dominancia son la clave para resolver eficientemente este problema mediante programación dinámica; además de una fácil detección de estados equivalentes, en este problema es vital la alta frecuencia con la que se presenta esta circunstancia (en el apartado 3.4. se presenta una aplicación numérica de esta virtud).

b) Problemas de secuenciación u ordenación. En el problema del taller mecánico *flow-shop* tipo P, un vértice del nivel N de la arborescencia domina a otro del mismo nivel, si se han secuenciado en las primeras N posiciones las mismas piezas y los instantes de disponibilidad de las máquinas son, para todas ellas, menores o iguales que los

correspondientes al vértice dominado. Para el problema de líneas de producción, en el apartado 1.1.1. se comenta cómo funcionan las relaciones de dominancia entre vértices.

c) Problema del viajante de comercio. Las relaciones de dominancia para este problema ya han sido definidas en el apartado 7.2.12.

8. *BRANCH AND WIN*: UN METALGORITMO GENERAL.

8.1. Introducción.

Una vez formalizado el metalgoritmo *branch and win* y para mostrar su carácter integrador, a continuación se describen como casos particulares de dicho metalgoritmo, las características más relevantes de la mayoría de los procedimientos de la literatura que se han referenciados en este texto (no se considera dificultosa la especificación como casos particulares de *branch and win*, de aquellos procedimientos referenciados que no son incluidos en el presente apartado).

En el apartado 8.2. se muestran, como casos particulares de la estrategia general de evaluación y selección de *branch and win*, f(n, t, a_t, e), un amplio abanico de las estrategias de selección descritas en esta tesis. A continuación, apartado 8.3., se comentan los procedimientos de búsqueda que, además de la estrategia de evaluación y selección del próximo vértice a tratar, presentan alguna otra característica definitoria. En el apartado 8.4. se comentan los procedimientos heurísticos referenciados. Y, para finalizar, en el apartado 8.5. se introducen algunas ideas generales que pueden resultar interesantes en la resolución eficiente de problemas de optimización combinatoria.

8.2. Estrategias de evaluación y selección: casos particulares de f(n, t, a_t, e).

Una vez expuesta la función general de evaluación y selección f(n, t, a_t, e), no resulta difícil comprobar cómo estrategias de selección descritas en los apartados anteriores se pueden definir como casos particulares de dicho indicador general.

En primer lugar, se muestra cómo se pueden obtener las estrategias generalistas descritas en el apartado 3.2.:

1.- La función $f(n) = g(n) + h(n)$ se obtiene con:

$$f(n, t, a_t, e) = f(n) = \alpha \cdot g(n) + \beta \cdot h(n),$$

y $\alpha = \beta = 1$.

2.- La función formalizada en 1969 por Pohl, $f(n) = (1 - \omega) \cdot g(n) + \omega \cdot h(n)$ con $\omega \in [0, 1]$ (también expuesta en el apartado 3.6.13. como un tipo de algoritmo de potencia heurística), se obtiene con:

$$f(n, t, a_t, e) = f(n, e) = \alpha(e) \cdot g(n) + \beta(e) \cdot h(n),$$

$\alpha(e) = 1 - \beta(e)$ y $\beta(e) \in [0, 1]$.

3.- La función que Pohl utiliza en su procedimiento llamado de pesos dinámicos, $f(n) = g(n) + \omega(n) \cdot h(n)$ (al igual que en el caso anterior, también expuesta en el apartado 3.6.13. como una clase de algoritmo de potencia heurística), se obtiene con:

$$f(n, t, a_t, e) = f(n) = \alpha \cdot g(n) + \beta(n) \cdot h(n),$$

$\alpha = 1$ y $\beta(n) = \omega(n)$.

4.- La función de selección del procedimiento de búsqueda con ancho de banda, que permite diseñar procedimientos ε-admisibles y obtener soluciones ε-óptimas (se trata de procedimientos heurísticos), se obtiene con:

$$f(n, t, a_t, e) = f(n, e) = \alpha \cdot g(n) + \beta \cdot h(n),$$

de forma que $h(n) \leq h^*(n) + \varepsilon(e)$ y tomando $\alpha = \beta = 1$.

5.- Las funciones de selección que deben tener en cuenta jerarquías de criterios quedan definidas como sigue:

$$f(n, t, a_t, e) = f(n, e) = \gamma_1(e) \cdot l_1(n) + \ldots + \gamma_s(e) \cdot l_s(n) + \tau(e) \cdot r(n),$$

donde $l_i(n)$ puede valer, por ejemplo, -1 si se cumple el criterio i y 0 si no se cumple, r(n) uniforme $\in [0, 1]$ y $\tau(e) \neq 0$; además, los coeficientes $\gamma_i(e)$ deben ser suficientemente diferentes para hacer prevalecer el criterio de mayor importancia sobre todos los demás, y así sucesivamente.

6.- Por otro lado, Pearl (1984) comenta que lo prometedor que puede llegar a ser un vértice se puede estimar de diversas maneras:

a) señalar la dificultad de resolución del subproblema representado por el vértice,
b) estimar la calidad del conjunto de soluciones codificadas por el vértice,
c) considerar el aumento de información obtenido al separar un vértice dado y la importancia de esta información en la estrategia de guía sobre todos los demás.

Estas tres formas de estimar la calidad de un vértice, o por qué no, estos tres componentes de su función de selección, se pueden incorporar a $f(n, t, a_t, e)$ de la manera siguiente:

- diseñando funciones $l_i(n)$ especificas para el caso a) (en un problema de programación lineal entera, mediante el número de variables que todavía no adoptan un valor entero) y c) (en el mismo problema de programación lineal entera, mediante el número de variables que antes no eran enteras y que después de dicha separación sí lo son),
- y con $c(n)$ o $l(n)$ para el caso b), siendo $l(n)$, en este caso, el valor obtenido al resolver mediante un procedimiento heurístico el problema parcial en el vértice n.

A continuación se comprueba cómo se pueden obtener, como casos de la función general $f(n, t, a_t, e)$, las estrategias particulares descritas en los procedimientos referenciados. Como en todos los casos anteriores, se selecciona para tratar el vértice de menor valor de dicha función.

1.- Procedimiento del Museo Británico (apartado 3.3.1.): idéntica función $f(n, t, a_t, e)$ que en la búsqueda en profundidad o en anchura.

2.- Búsqueda aleatoria (apartado 3.3.2.):

$$f(n, t, a_t, e) = f(n, e) = \tau(e) \cdot r(n),$$

con $r(n)$ uniforme $\in [0, 1]$ y $\tau(e) \neq 0$.

3.- Búsqueda en profundidad (apartado 3.3.3.):

$$f(n, t, a_t, e) = f(n) = \alpha \cdot g(n),$$

con $\alpha = -1$ y siendo $g(n)$ la distancia en número de pasos desde el vértice inicial hasta el vértice n.

4.- Búsqueda en anchura (apartado 3.3.4.):

$$f(n, t, a_t, e) = f(n) = \alpha \cdot g(n),$$

con $\alpha = 1$ y siendo $g(n)$ la distancia en número de pasos desde el vértice raíz hasta el vértice n.

5.- Búsqueda primero el de menor coste (apartado 3.3.5.):

$$f(n, t, a_t, e) = f(n) = \alpha \cdot g(n),$$

con $\alpha = 1$ y siendo g(n) el coste mínimo del camino desde el vértice inicial hasta el vértice n.

6.- Búsqueda primero el de mejor cota (apartado 3.3.6.):

$$f(n, t, a_t, e) = f(n) = \delta \cdot c(n),$$

con $\delta = 1$; de esta manera, en este caso también se incluyen aquellos problemas en los que las funciones g(n) y h(n) no son aplicables o no tienen sentido.

7.- Búsqueda primero el mejor (apartado 3.3.7.): en este caso la función de selección es la función general $f(n, t, a_t, e)$, ya que en principio puede adoptar cualquier forma; de todas maneras, las más usualmente utilizadas son del tipo:

$$f(n, t, a_t, e) = f(n) = \alpha \cdot g(n) + \beta \cdot h(n) = g(n) + h(n),$$

es decir, con $\alpha = \beta = 1$.

8.- Búsqueda primero el mejor* (apartado 3.3.8.): idéntica función de selección $f(n, t, a_t, e)$ que en la búsqueda primero el mejor, ya que la única variación con ésta reside en la condición de finalización del procedimiento.

9.- Búsqueda primero los mejores (apartado 3.3.9.):

$$f'(n, t, a_t, e) = -l_1(n, a_t),$$

con $l_1(n, a_t)$ igual a 1 si el vértice n se encontraba, en la ultima sesión de actualización de los valores $f'(n, t, a_t, e)$, entre los p vértices (siendo p el número de alternativas que se exploran en paralelo) de mejor valor de una función $f(n, t, a_t, e)$ igual a la de la búsqueda primero el mejor; y 0 en caso contrario. El proceso de actualización de los valores $f'(n, t, a_t, e)$ se realiza cada p vértices tratados.

10.- *Parallel depth first search* (apartado 3.3.10.):

$$f'(n, t, a_t, e) = -K_1 \cdot l_1(n, a_t) + f(n, t, a_t, e),$$

con $l_1(n, a_t)$ igual a 1 si el vértice n se encuentra entre los p vértices más profundos (y 0 en caso contrario); K_1 es un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor $l_1(n, a_t)$ sobre cualquier $f(n, t, a_t, e)$; y con la función $f(n, t, a_t, e)$ igual a la de la búsqueda primero el mejor.

11.- *Floating down search* (apartado 3.3.11.):

$$f'(n, t, a_t, e) = -K_1 \cdot l_1(n, a_t) - K_2 \cdot l_2(n) + f(n, t, a_t, e),$$

donde: $l_1(n, a_t)$ vale 1 si el vértice n ha sido generado en la última separación y no es un vértice terminal, y 0 en caso contrario; $l_2(n)$ vale 1 si el vértice n presenta igual o mejor indicador $f(n, t, a_t, e)$ que sus padres, y 0 en caso contrario; K_1 es un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor $l_1(n, a_t)$ sobre cualquier $K_2 \cdot l_2(n)$ y $f(n, t, a_t, e)$; K_2 es también un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor $l_2(n)$ sobre cualquier $f(n, t, a_t, e)$; y $f(n, t, a_t, e)$ es una función general de evaluación y selección.

12.- *Jump backtracking* (apartado 3.3.12.):

$$f'(n, t, a_t, e) = -K_1 \cdot l_1(n, a_t) + f(n, t, a_t, e),$$

siendo $l_1(n, a_t)$ igual a 1 si el vértice n ha sido generado en la última separación y no es un vértice terminal (y 0 en caso contrario); K_1 es un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor $l_1(n, a_t)$ sobre cualquier $f(n, t, a_t, e)$; y con la función $f(n, t, a_t, e)$ igual a la de la búsqueda primero el mejor.

13.- Estrategia primero el de mejor cota local (apartado 3.5.7.):

$$f(n, t, a_t, e) = -K_1 \cdot l_1(n, a_t) + \delta \cdot c(n),$$

siendo $l_1(n, a_t)$ igual a 1 si el vértice n se encuentra almacenado en la memoria RAM (y 0 en caso contrario); K_1 es un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor $l_1(n, a_t)$ sobre cualquier $\delta \cdot c(n)$; y con $\delta = 1$.

14.- Estrategia A (apartado 3.6.8.):

$$f(n, t, a_t, e) = f(n) = \alpha \cdot g(n) + \beta \cdot h(n),$$

con $\alpha = \beta = 1$ y donde $g(n) = g^*(n)$.

15.- Estrategia A* (apartado 3.6.8.):

$$f(n, t, a_t, e) = f(n) = \delta \cdot c(n),$$

con $\delta = 1$ y siendo $c(n) = g(n) + h(n)$.

16.- Profundización iterativa (apartado 3.6.2):

$$f(n, t, a_t, e) = f(n, a_t) = \alpha(n, a_t) \cdot g(n),$$

donde $\alpha(n, a_t)$ es igual a -1 si una función de evaluación $f'(n, t, a_t, e)$ cumple $f'(n, t, a_t, e) \leq F_t$ (y 0 en caso contrario), y siendo $g(n)$ la distancia en número de pasos desde el vértice inicial hasta el vértice n.

16.1.- Profundización iterativa primero en profundidad (apartado 3.6.3.): idéntica función de evaluación $f(n, t, a_t, e)$ que en la profundización iterativa, pero calculando la función $f'(n, t, a_t, e)$ para el vértice n, de igual forma que en el procedimiento primero en profundidad (el coste asociado es su profundidad).

16.2.- Profundización iterativa primero el de menor coste (apartado 3.6.4.): idéntica función de evaluación $f(n, t, a_t, e)$ que en la profundización iterativa, pero calculando la función $f'(n, t, a_t, e)$ para el vértice n, de igual forma que en el procedimiento primero el de menor coste (el coste asociado es $g(n)$).

16.3.- Profundización iterativa A* (apartado 3.6.9.): idéntica función de evaluación $f(n, t, a_t, e)$ que en la profundización iterativa, pero calculando la función $f'(n, t, a_t, e)$ para el vértice n, de igual forma que en el procedimiento A* (el coste asociado es $c(n) = g(n) + h(n)$).

17.- Algoritmo MREC (apartado 3.6.11.):

$$f(n, t, a_t, e) = k(n, t, a_t, e) \cdot f_1(n, t, a_t, e) + (1 - k(n, t, a_t, e)) \cdot f_2(n, t, a_t, e),$$

con $k(n, t, a_t, e) = 1$ si $m_t \leq \varepsilon(e) + M_t$ y 0 en caso contrario; $f_1(n, t, a_t, e)$ la misma función de evaluación que en la estrategia A*; y $f_2(n, t, a_t, e)$ idéntica función de selección que en la profundización iterativa A*.

18.- Algoritmo Z y Z* (apartado 3.6.7.): idéntica función de evaluación y selección que en los procedimientos de búsqueda primero el mejor y búsqueda primero el mejor*, respectivamente, pero con la particularidad de que la función de evaluación se calcula recursivamente como una función del coste hasta el vértice anterior más los costes de los vértices sucesores.

19.- Algoritmo A* ponderado (apartado 3.6.14.): idéntica función $f(n, t, a_t, e)$ que la utilizada por Pohl en su procedimiento llamado de pesos dinámicos, $f(n) = g(n) + \omega(n) \cdot h(n)$, de forma que $\omega(n) = \beta(n) / \alpha(n)$ y $g(n) + h(n) = c(n)$.

20.- Profundización iterativa A* ponderada (apartado 3.6.15.): idéntica función de evaluación $f(n, t, a_t, e)$ que en la profundización iterativa, pero calculando la función $f'(n, t, a_t, e)$ para el vértice n, de igual forma que en el procedimiento A* ponderado.

21.- Estrategia B (apartado 3.6.16.):

$$f'(n, t, a_t, e) = \alpha(n, t) \cdot g(n) + k(n, t) \cdot f(n, t, a_t, e),$$

donde: $\alpha(n, t)$ es igual a 1 si la función $f(n, t, a_t, e)$, calculada de igual forma que en el procedimiento A*, cumple $f(n, t, a_t, e) < F_t$ en curso, y 0 en caso contrario; $k(n, t) = K \cdot (1 - \alpha(n, t))$; K es un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor $f(n, t, a_t, e)$ sobre cualquier $g(n)$; $g(n)$ es una estimación del coste del camino de mínimo coste desde el vértice inicial hasta el vértice n; y $f(n, t, a_t, e)$ se calcula de igual forma que en el procedimiento A*.

22.- Búsqueda en profundidad-m (apartado 3.6.17.):

$$f'(n, t, a_t, e) = -K_1 \cdot l_1(n, a_t) + f(n, t, a_t, e),$$

con $l_1(n, a_t)$ igual a 1 si el vértice n se encuentra entre los m mejores vértices de entre los m + k vértices de la iteración anterior según $f(n, t, a_t, e)$ (siendo k los vértices sucesores del último vértice separado), y 0 en caso contrario; K_1 es un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor $l_1(n, a_t)$ sobre cualquier $f(n, t, a_t, e)$; y con la función $f(n, t, a_t, e)$ igual a la de la búsqueda primero el mejor.

23.- Búsqueda en anchura limitada (apartado 3.6.18.):

$$f'(n, t, a_t, e) = -K_1 \cdot l_1(n, a_t) + f(n, t, a_t, e),$$

con $l_1(n, a_t,)$ igual a 1 si los hijos del vértice n caben en el nivel siguiente y se puedan guardar (y 0 en caso contrario); K_1 es un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor $l_1(n, a_t)$ sobre cualquier $f(n, t, a_t, e)$; y con la función $f(n, t, a_t, e)$ igual a la de la búsqueda primero el mejor.

24.- Procedimientos de búsqueda *hard barried method* tipo a (apartado 3.6.19.):

$$f(n, t, a_t, e) = (k(n, t, a_t, e) + \varepsilon(e)) \cdot f_1(n, t, a_t, e) + (1 - k(n, t, a_t, e)) \cdot f_2(n, t, a_t, e),$$

con $k(n, t, a_t, e) = 1$ si $m_t < M_t$ y 0 en caso contrario; $f_1(n, t, a_t, e)$ la misma función de evaluación que en la búsqueda primero el mejor; $f_2(n, t, a_t, e)$ idéntica función de selección que en la búsqueda en profundidad; y $\varepsilon(e)$ que únicamente permite desempatar entre vértices de un mismo nivel en una búsqueda en profundidad.

8.3. Procedimientos enumerativos de búsqueda: casos particulares de *branch and win*.

En este apartado se comentan como casos particulares del metalgoritmo *branch and win*, algunos procedimientos que no habiendo sido ya especificados en el apartado anterior, además de la estrategia de evaluación y selección del próximo vértice a tratar, presentan alguna otra característica definitoria.

1.- Algoritmo de Land & Doig para programación lineal entera (apartado 3.5.1.1.). Procedimiento de separación (asociando a cada vértice sucesor una parte del rango de valores de una variable) y acotación (al prescindir de la condición de integridad de las variables), que utiliza la búsqueda primero el de mejor cota como función de selección.

2.- Algoritmo de Balas para la resolución de programas lineales binarios puros (apartado 3.5.1.1.). Procedimiento que realiza una búsqueda en profundidad con *backtracking*.

3.- Técnicas de búsqueda basadas en programación dinámica determinista finita (apartado 3.4.). Procedimientos cuya característica principal consiste en utilizar, como base y rasgo diferencial, las relaciones de dominancia entre vértices.

4.- Algoritmos *branch and bound* (apartado 3.5.1.). Procedimientos de búsqueda que utilizan la separación (*branch*) y la acotación (*bound*) en la exploración del espacio de soluciones; habitualmente, en primer lugar calculan una solución factible inicial mediante procedimientos heurísticos.

5.- *Branch and search* (apartado 3.5.2.). Procedimiento de separación y acotación, con idéntica estrategia de búsqueda que en la búsqueda en profundidad.

6.- *Branch and relax* (apartado 3.5.3.). Procedimiento *branch and bound.*

7.- *Branch and reduce* (apartado 3.5.4.). *Branch and bound* que incorpora procedimientos de reducción basados en técnicas de reducción de rangos; por otro lado, también utiliza procedimientos de resolución heurísticos en los subproblemas y, como estrategia de selección del próximo vértice a tratar, la búsqueda primero el de mejor cota.

8.- *Branch and prune* (apartado 3.5.5.). Como el procedimiento anterior, *branch and bound* que incorpora técnicas de reducción de intervalos.

9.- *Depth-first / Breadth-first / Best-first / ... branch and bound* (apartado 3.5.6.). Procedimientos *branch and bound* con estrategias particulares de evaluación y selección del próximo vértice a tratar: búsqueda en profundidad, búsqueda en anchura, búsqueda primero el mejor, ...

10.- *Branch and cut* (apartado 3.5.8.). Procedimiento *branch and bound* que incorpora, como rasgo diferencial, la generación y adición de planos de corte en el programa lineal resultante de relajar la condición de integridad de las variables enteras, definiendo así un complejo e iterativo procedimiento de acotación, en el que el objetivo consiste en obtener cotas ajustadas. Como cualquier procedimiento derivado de *branch and win*, además puede incorporar diversos refinamientos: procedimientos de reducción, de resolución heurísticos, de exploración de entornos, etc.

11.- *Branch and price* (apartado 3.5.9.). Procedimientos *branch and bound* en los que, como elemento distintivo, los programas lineales resultantes de relajar la condición de integridad de las variables enteras son resueltos mediante generación de columnas.

12.- *Branch and cut and price* (apartado 3.5.10.). Procedimientos *branch and bound* que utilizan como relajación la programación lineal, la cual es complementada con la generación combinada de planos de corte y de columnas.

13.- *Backtracking* dirigido por la dependencia (apartado 3.6.1.). Procedimiento de separación con *backtracking* que incorpora, como elemento característico, técnicas de reducción que a partir del vértice examinado aumentan de forma especial el conocimiento y la información disponible sobre otros vértices.

14.- Búsqueda recursiva primero el mejor (apartado 3.6.5.). Procedimiento de profundización iterativa, con un valor umbral de corte, F_t, particular para cada uno de los sucesores del vértice raíz.

15.- Ensanchamiento iterativo (apartado 3.6.6.). Procedimiento iterativo de separación (cuando finaliza la búsqueda puede volver a comenzar con un nuevo valor de R_t), que incorpora un procedimiento de separación limitado por el factor R_t y, como estrategia de selección del próximo vértice a tratar, la búsqueda en profundidad.

16.- Profundización iterativa A* con control de las reexpansiones (apartado 3.6.10.): Profundización iterativa A*, pero con una actualización más eficiente del valor umbral de corte F_t.

17.- Algoritmo MA* (apartado 3.6.12.): Procedimiento con idéntica función de selección que la estrategia A*, pero que presenta unos procedimientos especiales -en la línea de la formulación general propuesta por Corominas & Companys (1977)-: en la separación únicamente se genera el mejor de los vértices sucesores todavía no generados y, en función de la memoria disponible, se permite reabrir vértices anteriormente cerrados.

18.- Procedimientos de búsqueda *soft barried method* (apartado 3.6.19.). Búsqueda en profundidad-m con control dinámico del parámetro m.

19.- Procedimientos de búsqueda híbridos (apartado 3.7.). Procedimientos que utilizan varias técnicas y/o estrategias de búsqueda de forma alternativa y/o simultánea, tanto exactas como heurísticas; de esta manera, una vez especificados de *branch and win* los procedimientos "puros", estas técnicas quedan automáticamente definidas. Cabe destacar el uso de procedimientos de programación dinámica en *branch and bound* o viceversa -con limitación en el número de vértices almacenados en cada nivel (apartado 3.7.4.) o sin ella (apartado 3.7.3.)- y la incorporación de técnicas de exploración de entornos en procedimientos *branch and bound* (apartado 3.7.5.).

20.- Procedimientos de búsqueda basados en técnicas de reducción (apartado 3.8.). Procedimientos de separación, y de manera no muy frecuente también de acotación, que incorporan como característica más relevante, el uso de procedimientos de reducción en

todos y cada uno de los vértices de la arborescencia. Estos procedimientos de reducción pueden ser de cualquier naturaleza: técnicas de preproceso y/o de consistencia.

21.- Algoritmos de resolución de problemas de satisfacción de restricciones (apartado 3.8.2.). Procedimientos de separación en los que se sustituye la condición de optimalidad por un criterio de factibilidad; la resolución se sustenta de manera casi exclusiva, en el tratamiento sistemático de las restricciones mediante técnicas de reducción, de forma que únicamente se utiliza la búsqueda si no se puede realizar más reducción.

22.- *Backjumping* y *backmarking* (apartado 3.8.2.1.). Procedimientos de separación con *backtracking*, que utilizan técnicas de reducción para detectar inconsistencias que permitan eludir partes de la arborescencia.

23.- Procedimientos de propagación de restricciones (apartado 3.8.2.3.). Esquema que integra un algoritmo de consistencia dentro de un algoritmo enumerativo de búsqueda.

8.4. Procedimientos enumerativos heurísticos: casos particulares de *branch and win*.

Se puede considerar que los procedimientos heurísticos de búsqueda que utilizan la exploración del grafo de estados derivan directamente de procedimientos exactos, en los que se permite descartar una serie de vértices que no cumple una condición especial de permanencia o para los que la búsqueda finaliza antes de poder asegurar la optimalidad de la solución preferible. Por tanto, se relaja la condición de optimalidad de la solución o soluciones a obtener.

De esta manera, no es difícil especificar como casos particulares de *branch and win*, al prescindir de algunas de las condiciones que las definen y realizar la búsqueda en un área restringida, una serie de heurísticas que derivan de procedimientos exactos que se pueden considerar descendientes del metalgoritmo propuesto. Como ya se ha comentado, todos estos procedimientos se obtendrían de *branch and win* principalmente en dos caminos:

a) Descartando una serie de vértices que no cumple una condición especial de permanencia:

- no estar entre los de mejor indicador, o ser el mejor de todos ellos, cuando la memoria se agota o cuando se alcanza un número máximo de vértices abiertos (método del M-corte -apartado 3.9.1.-) o cuando se selecciona el próximo vértice a

tratar (estrategias de escalada -apartado 3.9.4.1.-) o en cada nivel de la arborescencia (programación dinámica restringida -apartado 3.9.3.- y búsqueda en haz de amplitud n -apartado 3.9.4.2.-),
- estar a una distancia del valor de la solución preferible igual o menor al error máximo permitido: el valor de la cota del vértice más el valor del error máximo permitido, es mayor o igual que $\bar{Z}$ (método de la ε-asignación -apartado 3.9.1.-),
- tratar las relaciones de dominancia de forma aproximada (como propone Ibaraki (1988) en el apartado 3.9.1.),
- etc.

b) Y/O finalizando la búsqueda antes de poder asegurar la optimalidad de la solución preferible:

- por llegar al tiempo máximo de ejecución,
- por alcanzar la capacidad máxima de almacenaje,
- por exceder el número máximo de vértices a generar (método del T-corte -apartado 3.9.1.-),
- etc.

El procedimiento *fix and relax* (apartado 3.9.4.3.) es un procedimiento *branch and win* heurístico particular, en el que la relajación de la optimalidad de la solución se obtiene gracias a un elaborado procedimiento de examen de los vértices. Este procedimiento consta de dos etapas: primero una fase de acotación en la que no se relajan todas las condiciones de integridad de las variables y una segunda fase de reducción que fija el valor de las variables no relajadas al valor obtenido en el proceso de acotación. Como se puede comprobar, el proceso de acotación es un procedimiento de búsqueda enumerativo derivado de *branch and win*.

Por último, dos comentarios: por un lado, también es posible obtener procedimientos heurísticos híbridos al considerar varias de las técnicas anteriores; y, por otro, hay que tener presente que también existe un conjunto de procedimientos heurísticos que no derivan de ningún procedimiento enumerativo de búsqueda y, por este motivo, no pueden ser considerados como casos particulares de *branch and win*.

8.5. Nuevas estrategias de búsqueda basadas en *branch and win*.

Una vez comprobado el carácter integrador y unificador de *branch and win*, inmediatamente es posible diseñar tantos nuevos procedimientos particulares de búsqueda como se desee, al combinar las diferentes técnicas expuestas y los parámetros

de la función general de evaluación y selección del próximo vértice a tratar. Además, éstos pueden ser tanto exactos como heurísticos.

A continuación se exponen algunas ideas de carácter muy general, que parece interesante probar para aquellos problemas de optimización combinatoria que todavía no son resueltos eficientemente. Cabe destacar que algunas de estas ideas ya son o están comenzando a ser incorporadas en procedimientos de búsqueda enumerativos:

- Estudiar la influencia de invertir tiempo en la búsqueda de una solución inicial de calidad con la que inicializar *branch and win*: probar diferentes procedimientos heurísticos y diversos valores de los parámetros de control de los procedimientos de exploración de entornos, que las heurísticas pueden incorporar.

- Introducción del concepto de dinamismo en los diferentes procedimientos que forman *branch and win*, y, especialmente, en el procedimiento de evaluación y selección del próximo vértice a tratar.

- Incorporación de aspectos no considerados que pueden mejorar el conocimiento del entorno y de los vértices, sobre todo para resolver problemas combinatorios de tamaño industrial (tiempo de cálculo permitido, tiempo restante, cantidad de memoria utilizada, etc.).

- Prueba de diferentes órdenes de aplicación de los procedimientos que forman parte de los procesos de examen de los vértices (acotación, reducción, resolución heurística, dominancias, etc.).

- Para los procedimientos de acotación, reducción, resolución heurísticos, exploración de entornos y las propiedades de las soluciones óptimas, trabajar con varias opciones, con el objetivo de disponer de una sucesión de procedimientos de cada tipo ordenados de menor a mayor dificultad de cálculo o comprobación, pero que, normalmente, proporcionen mejores resultados (cotas más ajustadas, reducciones más potentes, mejores soluciones factibles, etc.): probar su aplicación "en cascada".

- En los procedimientos de *branch and win* que sea posible según su definición (apartado 7.2.), probar el hecho de transmitir propiedades de vértices sucesores a precedentes y viceversa, con el objetivo de mejorar la información disponible sobre los otros vértices.

- Utilización de procedimientos de reducción y de resolución heurísticos, en los vértices intermedios de la arborescencia (en todos o sólo en aquellos vértices que resulten más prometedores que los demás).

- Diseño de procedimientos de resolución heurísticos que incorporen técnicas de exploración de entornos.

- Uso de procedimientos de exploración de entornos cuando se mejora el valor de la solución preferible, o siempre que se obtiene una nueva solución factible (al seleccionar para tratar un vértice terminal no vacío o al aplicar procedimientos de resolución heurísticos en vértices intermedios).

- Diseño de nuevos procedimientos de búsqueda, tanto exactos como heurísticos, en función de la especificación y combinación de los elementos que componen *branch and win*.

- Aumento del desarrollo de aplicaciones en arquitecturas paralelas buscando, si es posible, aceleraciones anómalas: aplicaciones *adhoc* que aprovechen el paralelismo intrínseco que poseen algunos procedimientos de búsqueda y/o uso de las librerías *branch and bound* en paralelo disponibles.

9. APLICACIÓN DE *BRANCH AND WIN* EN PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA. RESULTADOS Y ANÁLISIS.

9.1. Introducción.

En esta fase de la tesis se puede comprobar que se ha logrado estructurar, ordenar y formalizar, el panorama confuso expuesto en el apartado 1.1. referente a los procedimientos de exploración enumerativos en árboles y grafos de búsqueda. De esta manera, es posible considerar que los objetivos planteados en el apartado 1.2. se han cumplido en su totalidad:

- Se ha realizado una amplia recopilación, análisis y crítica, de diferentes procedimientos, estrategias de resolución y formulaciones generales expuestas en la literatura.

- Se ha diseñado y formalizado *branch and win*: un metalgoritmo de exploración de grafos cuyas principales características son las siguientes: es general, integrador, realista (incorpora aspectos importantes en la resolución de problemas industriales), utiliza terminología clara, dinámico (en función de la evolución de la arborescencia y/o del entorno) y derivador de otros procedimientos y estrategias de búsqueda.

- Se han presentado diversas ideas de carácter general, cuya aplicación permite diseñar y proponer nuevos procedimientos híbridos de búsqueda.

En el presente apartado, se pretende probar algunas de estas ideas de carácter general en dos conocidos problemas de optimización combinatoria: el *flow-shop* tipo P y el cubrimiento; pero nunca con el objetivo principal de solucionar eficientemente dichos problemas, sino con el propósito de entrever si puede resultar interesante utilizar el conjunto de estrategias probadas en la resolución de problemas de optimización combinatoria. De esta manera, se realiza una breve experiencia computacional que permite intuir la conveniencia de las estrategias probadas; en ningún momento se pretende probar su adecuación absoluta para la resolución de los problemas de *flow-shop* tipo P (apartado 9.2.) y de cubrimiento (apartado 9.3.).

9.2. El problema de *flow-shop* tipo P.

A continuación se describe el problema de *flow-shop* tipo P (apartado 9.2.1.) y se presentan las estrategias ensayadas, así como los resultados obtenidos (apartado 9.2.2.).

9.2.1. El problema de taller mecánico *flow-shop* tipo P.

El problema conocido como de taller mecánico es un problema de *scheduling* (problemas introducidos en el apartado 2.2.), que parte de disponer de un conjunto de n piezas (lotes, trabajos u órdenes) que deben procesarse en un grupo de m máquinas (secciones o puestos de trabajo). Para ser fabricada, cada pieza debe someterse a una serie de operaciones prefijadas (en una ruta establecida de partida), de forma que cada operación está asignada a una máquina concreta y tiene una duración conocida. Una vez realizada una ordenación de la secuencia de piezas a tratar, se puede obtener el valor de un criterio que permite medir la calidad de dicha solución frente a otras; las medidas habituales utilizan la media o el máximo de los instantes de salida, del retraso, del tiempo de permanencia, etc.

De esta manera, resolver un problema de taller mecánico se resume en establecer un programa: secuenciar una serie de operaciones en las máquinas hallando al mismo tiempo la temporización de las mismas, de forma que se optimice un cierto índice de eficacia.

Los problemas de taller mecánico han sido estudiados desde hace décadas y en estos momentos se dispone de una abundantemente literatura sobre el tema en la que se presenta, entre otras cuestiones, las hipótesis de partida a considerar, la nomenclatura, su complejidad, varias clasificaciones y diversos procedimientos de resolución tanto exactos como heurísticos -véase por ejemplo, Companys (1989b), Blazewicz et al. (1996), Companys & Corominas (1996) y Hoogeveen et al. (1997)-.

El problema de *flow-shop* tipo P se corresponde con un problema de taller mecánico, en el que desde el instante inicial tanto el número de piezas como el de máquinas está prefijado y todas, piezas y máquinas, están disponibles, las rutas de las piezas son las mismas para todas ellas, y, además, el orden de "visita" de las n piezas a las m máquinas es el mismo para todas las máquinas.

Para la resolución del problema de *flow-shop* tipo P existen tanto procedimientos exactos como heurísticos.

Un procedimiento exacto tradicional de resolución de problemas de *flow-shop* tipo P, es el método Lomnicki: una búsqueda dirigida en el espacio de estados, guiada por una estrategia de búsqueda primero el de mejor cota[26]. Por otro lado cabe destacar el Lomnicki pendular, un nuevo -y más eficaz- procedimiento de búsqueda basado en el método Lomnicki, presentado en Companys (1995).

[26] En numerosos textos se puede encontrar tanto la descripción detallada del procedimiento, como la forma de calcular las cotas. Véase por ejemplo Companys & Corominas (1996).

Los procedimientos heurísticos de resolución del problema de *flow-shop* tipo P son muy abundantes en la literatura; en la presente tesis se han utilizado cuatro de ellos, que conducen a resultados de calidad sin necesitar tiempos de cálculo importantes: heurística de Palmer, trapecios, Gupta y Teixidó[27].

9.2.2. Estrategias ensayadas en el problema *flow-shop* tipo P.

Para la resolución exacta del problema *flow-shop* tipo P se utiliza el procedimiento de Lomnicki, con el objetivo de minimizar el tiempo de permanencia en el taller de la pieza que lo abandona en ultima posición, y, como ya se ha comentado, suponiendo que tanto las piezas como las máquinas están disponibles desde el instante inicial. Con tal fin, se ha utilizado como base un programa de software ya existente, modificándolo convenientemente para adaptarlo a la experiencia computacional a realizar[28]; de esta manera, el procedimiento de búsqueda básico (tomado como punto de partida para realizar los análisis posteriores) consiste en una ramificación y acotación, que parte de una solución preferible inicial igual a la mejor de cuatro heurísticas (especificadas en el apartado 9.2.2.2.), y que selecciona el próximo vértice a tratar según una estrategia primero el de mejor cota y, en caso de empate, según una estrategia primero en profundidad.

En este apartado, en primer lugar se introducen las condiciones de partida del experimento: los juegos de ejemplares que se han utilizado, el ordenador usado y las condiciones de aborto de la búsqueda (apartado 9.2.2.1.); y, posteriormente, se presentan las estrategias que han sido ensayadas, así como las indicaciones que se pueden intuir al analizar los resultados obtenidos (apartados 9.2.2.2. a 9.2.2.5.).

Cabe tener presente que, tras los primeros ensayos realizados, se ha decidido no probar el efecto de considerar varias estrategias de forma simultánea. Esta decisión es debida a que la mayoría de ejemplares utilizados se pueden clasificar, en cuanto a su dificultad de resolución exacta, en dos grandes grupos: o muy fáciles de resolver (y solucionados de forma casi inmediata por la mayoría de estrategias probadas), o, por el contrario, muy difíciles de resolver antes de exceder el tiempo de cálculo permitido y/o la capacidad de la memoria existente (y, además, no resueltos en la mayoría de ocasiones). De esta manera, las indicaciones que se exponen se basan, sobretodo, en la resolución de los escasos ejemplares intermedios.

Por otro lado también es necesario comentar que en un porcentaje no despreciable de los problemas no resueltos de forma óptima, la distancia entre la solución preferible y la

[27] Para una descripción detallada de dichas heurísticas se recomienda Companys & Corominas (1996).
[28] El código del programa informático original procede del Laboratorio de Organización Industrial de la ETS de Ingenieros Industriales de Barcelona (UPC).

mejor cota es tan sólo de una unidad, lo que puede llevar a pensar que en muchos casos ya se dispone de la solución óptima pero ésta no ha sido probada.

9.2.2.1. Condiciones de partida.

Los ejemplares utilizados en la experiencia computacional realizada, generados aleatoriamente con unos tiempos de proceso distribuidos uniformemente entre 1 y 20 unidades de tiempo, son los siguientes: 200 problemas de 15 piezas y 3 máquinas, 100 problemas de 15 piezas y 4 máquinas, y 20 problemas de 15 piezas y 6 máquinas. Algunos de estos ejemplares son resueltos de forma inmediata al ejecutar el procedimiento de búsqueda básico y, por consiguiente y como se ha comprobado, con todas las estrategias adicionales ensayadas; de esta manera, todos los análisis efectuados se refieren a los problemas restantes: 113 problemas de 15 piezas y 3 máquinas, 79 problemas de 15 piezas y 4 máquinas, y los mismos 20 problemas de 15 piezas y 6 máquinas.

Por otro lado, la experiencia computacional ha sido realizada en un ordenador personal Pentium II a 233 Mhz.

Las condiciones de finalización de la búsqueda para cada ejemplar, si no se ha demostrado la optimalidad de la solución preferible, han sido las siguientes: consumir un tiempo de cálculo de 1.000 segundos (limite de tiempo) o alcanzar la cifra de 32.000 vértices generados pero no descartados (limite de capacidad de memoria).

9.2.2.2. Calidad de la solución preferible inicial.

La primer prueba tiene como objetivo el comprobar si el hecho de invertir tiempo en intentar obtener una mejor solución inicial preferible de partida, repercute positivamente en la búsqueda global. Para ello, se han ensayado las siguientes estrategias:

a) Considerar como solución inicial, la mejor de las soluciones obtenidas con las heurísticas de Palmer, trapecios, Gupta y Teixidó.
b) La solución inicial se corresponde con la mejor de las soluciones obtenidas con las heurísticas de Palmer, trapecios, Gupta y Teixidó, a las que se les aplica un procedimiento de optimización local 2-intercambio entre piezas consecutivas de la secuencia.
c) Tomar como solución inicial la mejor de las soluciones obtenidas con las heurísticas de Palmer, trapecios, Gupta y Teixidó, a las que se les aplica un procedimiento de optimización local 2-intercambio entre cualquier pareja de piezas de la secuencia.

d) Partiendo de la solución obtenida con la heurística de trapecios (habitualmente la que proporciona mejores resultados), se obtienen 99 nuevas soluciones con un procedimiento GRASP[29] con k = 4, mejoradas con un procedimiento de optimización local 2-intercambio entre cualquier pareja de piezas de la secuencia.
d') Estrategia anterior en la que se obtienen 999 soluciones mediante GRASP (siempre que el tiempo de cálculo no sea superior a 1000 segundos) y que ha sido aplicada únicamente a los ejemplares no resueltos con la estrategia d.

Las tablas siguientes muestran los resultados obtenidos al resolver los juegos de ejemplares generados:

3 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	75	0	22	113	66.4	24.01	17.45	180
Estrategia b	77	0	30	113	68.1	2.31	1.74	23
Estrategia c	78	0	34	113	69.0	2.92	1.43	7
Estrategia d	79	0	34	113	69.9	44.52	42.42	5
Estrategia d'	79	0	34	112	69.9	---	412.53	---

Resultados de los 113 ejemplares de 3 máquinas.

4 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	43	0	8	78	54.4	24.77	37.14	329
Estrategia b	48	0	10	78	60.8	21.88	26.70	26
Estrategia c	48	0	16	79	60.8	4.06	3.78	10
Estrategia d	51	0	25	79	64.6	67.61	68.03	2
Estrategia d'	51	0	27	76	64.6	---	695.07	---

Resultados de los 79 ejemplares de 4 máquinas.

6 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	9	0	0	16	45	111.00	115.25	1326
Estrategia b	9	0	2	17	45	103.89	41.05	811
Estrategia c	10	0	2	18	50	75.90	34.15	485
Estrategia d	12	0	6	20	60	233.25	182.55	367
Estrategia d'	12	0	8	15	60	---	1000	---

Resultados de los 20 ejemplares de 6 máquinas.

[29] En González (1996) se describe detalladamente los procedimientos GRASP, así como algunas aplicaciones.

En la primera columna se cuantifica el número de ejemplares para los que se prueba la optimalidad de la solución preferible; en la columna siguiente, el número de casos en los que se conoce que se ha obtenido como solución preferible la solución óptima, pero no se ha podido probar; a continuación, el número de ejemplares no resueltos de forma óptima, en los que se alcanza la mejor solución preferible de las estrategias ensayadas en el presente apartado; la cuarta columna muestra el número de veces que se alcanza la mejor cota de las estrategias ensayadas en el apartado; seguidamente se expone el % de ejemplares para los que se prueba la optimalidad de la solución preferible; la sexta y séptima columna exponen, respectivamente, el tiempo medio en segundos que se tarda en resolver los ejemplares para los que se ha probado la optimalidad de la solución preferible, y el tiempo medio, también en segundos, utilizado hasta alcanzar la solución preferible en todos los ejemplares (los 113, 79 y 20, de cada tipo); para finalizar, la última columna muestra el número medio del máximo número de vértices generados pero no explorados que coexisten de forma simultánea, obtenidos al resolver los ejemplares para los que se prueba la optimalidad de la solución alcanzada.

En general, los resultados obtenidos muestran que a medida que se utilizan estrategias más elaboradas se resuelven más ejemplares de forma óptima. Además, el tiempo medio necesario para resolver óptimamente los ejemplares se va reduciendo, excepto cuando se utiliza el procedimiento GRASP en el gran parte de la totalidad del tiempo de búsqueda se utiliza en esta fase. El número máximo de vértices generados pero no cerrados que coexisten de forma simultánea también decrece, lo que indica una menor necesidad de capacidad de memoria. En cuanto a los ejemplares para los que no se comprueba la optimalidad de la solución preferible, la calidad de éstas aumenta al invertir tiempo en mejorar la calidad de la solución inicial.

Una vez analizados los resultados alcanzados, se pueden entrever las siguientes lecciones respecto a la idea de invertir tiempo en intentar obtener una mejor solución inicial preferible de partida, en vistas a una repercusión positiva en la búsqueda gracias a una poda más potente:

a) Parece aconsejable el hecho de completar los procedimientos heurísticos puros con fases finales de optimización local, muy breves en tiempos de cálculo pero muy efectivas en las mejoras que consiguen.

b) Los diferentes procedimientos utilizados en la generación de soluciones vecinas pueden influir en la efectividad de las técnicas de optimización local; de esta manera se recomienda, de nuevo basándose en los breves tiempos de cálculo necesarios, no limitarse a un único tipo de generación de soluciones vecinas.

c) Los procedimientos de exploración de entornos (como el procedimiento GRASP ensayado), también pueden resultar interesantes: son capaces de proporcionar mejores soluciones de partida, aunque gracias a una mayor inversión en el tiempo de proceso; de esta manera, los parámetros de control deben ser cuidadosamente calibrados.

Como conclusión final, se recomienda el invertir tiempo en intentar obtener una mejor solución inicial preferible de partida: se pueden solucionar óptimamente más ejemplares, en menos tiempo, con una menor ocupación de memoria, y, en caso de no probar la optimalidad de la solución preferible, está puede ser de mejor calidad.

9.2.2.3. Optimización local en vértices terminales no vacíos.

Una nueva idea a ensayar consiste en ejecutar en los vértices terminales no vacíos, y que por tanto proporcionan una solución factible, procedimientos de optimización local. Esta acción puede tener un doble objetivo: por un lado, posibilita mejorar el valor de la solución preferible $\overline{Z}$, y, por otro, puede permitir obtener más información acerca de otros vértices (mejorar las cotas superiores de los vértices antecesores, generados y no descartados, de los vértices terminales no vacíos que se van obteniendo en el proceso de optimización local).

En el ensayo efectuado: únicamente se considera el primer objetivo -mejorar el valor de $\overline{Z}$-; se incorpora un procedimiento de optimización local 2-intercambio entre cualquier pareja de piezas de la secuencia (el mismo que se ha utilizado en las estrategias c y d del apartado anterior); y, además, la optimización local se realiza en todos los vértices terminales no vacíos, sean de mejor, igual o peor calidad que la solución preferible.

En este momento cabe recordar que para comparar los resultados obtenidos en esta prueba y todas las que se realizan en los apartados siguientes, únicamente se utiliza la estrategia básica de partida.

Las tablas siguientes muestran los resultados obtenidos al resolver los juegos de ejemplares generados, sin y con optimización local en los vértices terminales no vacíos:

3 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Sin opt. local	75	0	25	113	66.4	24.01	17.45	180
Con opt. local	79	0	34	112	69.9	8.33	13.52	90

Resultados de los 113 ejemplares de 3 máquinas.

4 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Sin opt. local	43	0	16	78	54.4	24.77	37.14	329
Con opt. local	51	0	28	79	64.6	34.24[30]	47.95[31]	361[32]

Resultados de los 79 ejemplares de 4 máquinas.

6 Máquinas	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor Cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Sin opt. local	9	0	6	20	45	111.00	115.25	1326
Con opt. local	11	0	9	19	55	173.55[33]	118.10[34]	1832[35]

Resultados de los 20 ejemplares de 6 máquinas.

Las columnas de las tablas anteriores conservan el mismo significado que en las tablas de resultados del apartado 9.2.2.2.

Los resultados anteriores muestran que el utilizar procedimientos de optimización local en los vértices terminales no vacíos permite obtener mejores resultados globales: se resuelven de forma óptima más ejemplares, en menos tiempo promedio y con menor necesidad de capacidad de memoria; además, en los ejemplares en los que no se prueba la optimalidad de la solución preferible, éstas son de mejor o igual calidad que las soluciones obtenidas sin utilizar optimización local.

De esta manera, se aconseja el hecho de probar la introducción de procedimientos de optimización local en los vértices terminales no vacíos, ya que son procedimientos muy breves en tiempos de cálculo pero parece que efectivos en las mejoras que proporcionan.

9.2.2.4. Función de evaluación y selección dinámica.

La función de evaluación y selección del próximo vértice a tratar que incorpora el procedimiento, permite trabajar con un indicador dinámico. La función de evaluación es la siguiente:

$$f(n,a_t,e) = K \cdot \left(\frac{\alpha(e)\cdot c(n) + (1-\alpha(e))\cdot l_1(n)}{\overline{Z}(a_t)} + \beta(e)\cdot \frac{(Npiezas-1)-l_2(n)}{(Npiezas-1)} \right) + \gamma \cdot g(n)$$

[30] Sin considerar los ocho nuevos ejemplares resueltos, el tiempo promedio es 21.79.
[31] Sin considerar los ocho nuevos ejemplares resueltos, el tiempo promedio es 42.03.
[32] Sin considerar los ocho nuevos ejemplares resueltos, este valor es 295.
[33] Sin considerar los dos nuevos ejemplares resueltos, el tiempo promedio es 105.89.
[34] Sin considerar los dos nuevos ejemplares resueltos, el tiempo promedio es 78.22.
[35] Sin considerar los dos nuevos ejemplares resueltos, este valor es 1009.

donde:

- c(n) es una cota inferior del vértice n,
- $l_1(n)$ es el valor de una cota superior del vértice n calculada con la heurística de trapecios dinámicos, que, por las características del problema *flow-shop* tipo P, se concreta en la heurística de trapecios[36],
- $\overline{Z}(a_t)$ es el valor de la solución preferible,
- $l_2(n)$ proporciona la profundidad del vértice n,
- $\alpha(e)$ y $\beta(e)$ son parámetros de control,
- K es un valor constante lo más pequeño posible, pero lo suficientemente grande como para hacer prevalecer el valor al que multiplica respecto al segundo sumando, una estrategia de búsqueda en profundidad que se utiliza para desempatar,
- g(n) es la distancia en número de pasos desde el vértice raíz hasta el vértice n,
- $\gamma = -1$.

Seleccionando el vértice de menor valor de la función $f(n, a_t, e)$, se han ensayado los siguientes valores de los parámetros de control:

a) $\alpha(e) = 1$ y $\beta(e) = 0$.
b) $\alpha(e) = 0$ y $\beta(e) = 0$.
c) $\alpha(e) = 0.5$ y $\beta(e) = 0$.
d) $\alpha(e) = 1$ y $\beta(e) = 0.5$.
e) $\alpha(e) = 1$ y $\beta(e) = 1$.
f) $\alpha(e) = 1$ y $\beta(e) = 10$.

Las tablas siguientes muestran los resultados obtenidos al resolver los juegos de ejemplares:

3 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	75	0	29	113	66.4	24.01	17.45	180
Estrategia b	77	0	32	112	68.1	2.86	1.65	42
Estrategia c	76	0	33	112	67.3	2.55	12.18	38
Estrategia d	75	0	34	112	66.4	17.59	17.40	63
Estrategia e	75	0	34	112	66.4	17.93	17.73	63
Estrategia f	75	0	34	112	66.4	17.91	17.77	63

Resultados de los 113 ejemplares de 3 máquinas.

[36] En Companys & Corominas (1996) se describe de forma detallada la heurística de trapecios dinámicos.

4 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	43	0	10	79	54.4	24.77	37.14	329
Estrategia b	46	0	17	71	58.2	8.98	23.57	52
Estrategia c	47	0	24	73	59.5	6.72	52.68	51
Estrategia d	43	0	16	73	54.4	41.77	90.08	58
Estrategia e	43	0	16	73	54.4	41.49	191.30	58
Estrategia f	43	0	16	73	54.4	41.12	189.77	58

Resultados de los 79 ejemplares de 4 máquinas.

6 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	9	0	1	19	45	111.00	115.25	1326
Estrategia b	11	0	5	16	55	195.45	129.55	581
Estrategia c	11	0	7	16	55	135.00	60.50	416
Estrategia d	10	0	5	16	50	402.40	318.65	75
Estrategia e	9	0	5	15	45	364.67	351.95	68
Estrategia f	9	0	5	15	45	369.67	354.15	67

Resultados de los 20 ejemplares de 6 máquinas.

Estas tablas exponen la misma información que las tablas de resultados de los apartados anteriores.

Los resultados obtenidos muestran que el uso de la cota superior en el indicador que guía la búsqueda puede ser beneficioso, en general: se prueba la optimalidad de la solución preferible en más ejemplares; los tiempos medios de resolución son menores en los ejemplares de dimensiones pequeñas (3 y 4 máquinas), pero del mismo orden de magnitud en los ejemplares de 6 máquinas; y, además, es necesaria una menor capacidad de memoria. Por otro lado, en los ejemplares en los que no se prueba la optimalidad de la solución preferible, estas soluciones son de mejor o igual calidad que las obtenidas sin incluir la cota superior en dicho indicador.

El hecho de dar importancia a los vértices más profundos, en general no resuelve más ejemplares de forma óptima, ni en un menor tiempo promedio de cálculo; en cambio, claramente sí que se necesita una menor capacidad de memoria (resultado previsible en un procedimiento que incluye un componente de búsqueda en profundidad). En cuanto a los ejemplares para los que no se prueba su resolución exacta, las soluciones preferibles son normalmente de igual o mejor calidad que las obtenidas utilizando la profundidad únicamente para desempatar.

Como conclusión, parece recomendable incluir en el indicador de guía de la búsqueda, además de la cota inferior, algún otro componente: cota superior, profundidad (en este problema parece que únicamente para desempatar), etc.

9.2.2.5. Solución heurística en los vértices, con o sin optimización local.

La última idea probada consiste en ejecutar un procedimiento de resolución heurístico, en todos o únicamente en algunos vértices de la arborescencia. Esta posibilidad puede presentar un doble objetivo: por un lado, posibilitar mejorar el valor de la solución preferible $\overline{Z}$, y, por otro, facilitar un valor susceptible de ser utilizado en la función de evaluación y selección del próximo vértice a tratar.

En el ensayo efectuado: únicamente se considera el primer objetivo -mejorar el valor de $\overline{Z}$-; el procedimiento de resolución heurístico incorporado consiste, como en el apartado anterior, en la heurística de trapecios dinámicos; y, por otro lado, su aplicación no es inmediata en todos los vértices generados: se ha implementado la siguiente función dinámica de probabilidad de utilizar dicho procedimiento:

$$Probabilidad(n,a_t,e) = 1 - \left(\lambda(e) \cdot \frac{c(n)}{\overline{Z}(a_t)} \right)$$

donde:

- c(n) es una cota inferior del vértice n,
- $\overline{Z}$ (a_t) es el valor de la solución preferible,
- λ(e) es un parámetro de control.

En el caso de obtener una nueva solución factible proveniente de utilizar un procedimiento de resolución heurístico, también se ha ensayado la posibilidad de ejecutar un proceso de optimización local. En este caso, esta acción también presenta un doble objetivo: mejorar el valor de la solución preferible $\overline{Z}$ y/o permitir obtener más información acerca de otros vértices (mejorar sus cotas superiores).

Las condiciones de aplicación de esta iniciativa son idénticas a las del apartado 9.2.2.3.: únicamente se considera mejorar el valor de $\overline{Z}$, se incorpora un procedimiento de optimización local 2-intercambio entre cualquier pareja de piezas de la secuencia, y la optimización local se realiza en todas las soluciones obtenidas al ejecutar el procedimiento de resolución heurístico.

En concreto, las estrategias que se han ensayado en el presente apartado son las siguientes:

a) Probabilidad(n,a_t,e) = 0 (y, por tanto, sin optimización local).
b) λ(e) = 0.9 y sin optimización local.
c) Probabilidad(n,a_t,e) = 1 y sin optimización local.
d) λ(e) = 0.9 y con optimización local.
e) Probabilidad(n,a_t,e) = 1 y con optimización local.

Las tablas siguientes muestran los resultados obtenidos al resolver los juegos de ejemplares:

3 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor Cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	75	0	22	113	66.4	24.01	17.45	180
Estrategia b	75	0	25	113	66.4	12.63	11.93	101
Estrategia c	77	0	25	113	68.1	8.13	8.78	44
Estrategia d	79	0	34	112	69.9	3.41	2.19	14
Estrategia e	78	0	34	112	69.0	1.44	1.02	2

Resultados de los 113 ejemplares de 3 máquinas.

4 Máquinas	Z* probado	Z* no probado	Mejor $\overline{Z}$	Mejor Cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	43	0	8	79	54.4	24.77	37.14	329
Estrategia b	44	0	14	79	55.7	34.02[37]	42.47[38]	89[39]
Estrategia c	46	0	16	79	58.2	54.22[40]	45.41[41]	127[42]
Estrategia d	51	0	28	78	64.6	13.71	37.91	14
Estrategia e	51	0	27	77	64.6	5.24	29.41	4

Resultados de los 79 ejemplares de 4 máquinas.

[37] Sin considerar el nuevo ejemplar resuelto, el tiempo promedio es 13.47.
[38] Sin considerar el nuevo ejemplar resuelto, el tiempo promedio es 31.24.
[39] Sin considerar el nuevo ejemplar resuelto, este valor es 71.
[40] Sin considerar los tres nuevos ejemplares resueltos, el tiempo promedio es 14.30.
[41] Sin considerar los tres nuevos ejemplares resueltos, el tiempo promedio es 22.49.
[42] Sin considerar los tres nuevos ejemplares resueltos, este valor es 51.

6 Máquinas	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor Cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	9	0	0	17	45	111.00	115.25	1326
Estrategia b	10	0	1	20	50	183.70[43]	166.75[44]	1183[45]
Estrategia c	10	0	1	20	50	170.30[46]	115.50[47]	804[48]
Estrategia d	11	1	7	16	55	108.64	77.90	112
Estrategia e	9	2	7	14	45	119.78	235.05	16

Resultados de los 20 ejemplares de 6 máquinas.

Las columnas de las tablas anteriores conservan el mismo significado que el expuesto para las tablas de resultados del apartado 9.2.2.2.

Analizando los resultados anteriores, parece aconsejable el hecho de probar la ejecución de procedimientos de resolución heurísticos en todos o algunos vértices de la arborescencia (aunque no es determinante, parece aconsejable una menor ejecución del procedimiento a medida que aumentan las dimensiones del ejemplar); de esta manera se intuye que se pueden obtener mejores resultados globales: más ejemplares resueltos de forma óptima, en menores o similares tiempos de ejecución y con menor necesidad de capacidad de memoria. Además, en los ejemplares en los que no se prueba la optimalidad de la solución preferible, éstas pueden ser de mejor o igual calidad que las soluciones obtenidas sin ejecutar estos procedimientos de resolución heurísticos.

Por otro lado estos resultados pueden ser mejorados, si se incorporan procedimientos de optimización local en la solución obtenida del procedimiento de resolución heurístico.

9.3. El problema de cubrimiento.

En el presente apartado se describe brevemente el problema de cubrimiento (apartado 9.3.1.) y se presentan los resultados obtenidos con las estrategias de búsqueda ensayadas (apartado 9.3.2.).

9.3.1. Introducción al problema de cubrimiento.

Como ya se ha comentado en el apartado 2.2., el problema de cubrimiento (*set-covering problem*) se puede formular como el siguiente problema de programación entera:

[43] Sin considerar el nuevo ejemplar resuelto, el tiempo promedio es 100.11.
[44] Sin considerar el nuevo ejemplar resuelto, el tiempo promedio es 126.32.
[45] Sin considerar el nuevo ejemplar resuelto, este valor es 606.
[46] Sin considerar el nuevo ejemplar resuelto, el tiempo promedio es 104.67.
[47] Sin considerar el nuevo ejemplar resuelto, el tiempo promedio es 81.53.
[48] Sin considerar el nuevo ejemplar resuelto, este valor es 485.

$$[\text{MIN}]\ Z = c \cdot X$$
$$A \cdot X \geq 1$$
$$X \in \{0, 1\},$$

en el que los componentes de la matriz A son ceros o unos. La idea consiste en encontrar, a mínimo coste, un conjunto de variables X iguales a uno, tales que los unos del vector de términos independientes estén "cubiertos" por al menos uno de los unos de las columnas seleccionadas (de esta manera, la cobertura múltiple está permitida).

Este problema admite diversas variantes y en esta tesis se ha trabajado con una de ellas: el problema de cubrimiento con idénticos coeficientes de las variables en la función objetivo (y en particular iguales a 1); de esta forma, el objetivo consiste en minimizar el número de variables iguales a uno. De una manera menos abstracta, el problema de cubrimiento estudiado se puede exponer como sigue: dado un conjunto de clientes a cubrir y una serie de posibles ubicaciones a fijar para cubrir dichos clientes, donde cada ubicación cubre a unos clientes determinados, el problema consiste en cubrir todos los clientes por al menos una ubicación, de forma que el número total de ubicaciones fijadas sea mínimo.

El problema de cubrimiento, un importante problema de optimización combinatoria -"*Set covering problems are, with little doubt, one of the most important classes of combinatorial optimization problems.*" (Ceria et al. 1997, p. 415)-, ha sido estudiado desde hace décadas y en estos momentos se dispone de una abundantemente literatura sobre el tema en la que se presenta la nomenclatura, varias formas de calcular cotas, etc., y diversos procedimientos de resolución tanto exactos como heurísticos -véase por ejemplo, Balas & Padberg (1972), Balas & Ho (1980), Rushmeier & Nemhauser (1993), Afif et al. (1995) y Ceria et al. (1997)-.

Para la resolución exacta del problema de cubrimiento planteado se ha diseñado un procedimiento de ramificación y acotación, implantado en un programa de software *adhoc*, que presenta, en su versión básica tomada como punto de partida para realizar los análisis posteriores, las siguientes características:

- Antes de comenzar la búsqueda, se realiza una fase inicial de preproceso en la que se intenta reducir las dimensiones del problema a resolver. El preproceso comprueba, de forma iterativa, la existencia de clientes que sólo pueden ser cubiertos por una ubicación (lo que implica fijar dicha ubicación), las relaciones de dominancia entre ubicaciones (una ubicación i domina a otra j si cubre los mismos clientes y más, lo que permite eliminar la ubicación dominada o cualquiera de las dos si cubren los mismo clientes) y, también, la existencia de dominancias entre clientes (un cliente i domina a otro j si está cubierto por un subconjunto de las ubicaciones que cubren a j, lo que permite eliminar

al cliente dominado o a cualquiera de los dos si están cubiertos por las mismas ubicaciones).

- La solución preferible inicial se obtiene con el procedimiento heurístico descrito en Johnson (1974), que consiste en seleccionar dinámicamente la ubicación que cubre más clientes todavía no cubiertos.

- La función de evaluación y selección del próximo vértice a tratar consiste en una estrategia de búsqueda primero el de mejor cota, que utiliza para desempatar el número -mínimo en este caso- de clientes que faltan por cubrir.

- Con el objetivo de podar los vértices que no contienen ninguna solución mejor que la preferible, se han programado los dos siguientes procedimientos de acotación:

El problema a resolver se puede formular como el siguiente programa lineal binario:

$$[\text{MIN}]\ Z = \sum_i X_i$$
$$\sum_i a_{ij} \cdot X_i \geq 1 \qquad \forall j$$
$$X_i \in \{0, 1\},$$

donde a_{ij} son los elementos de la matriz A y los subíndices i/j se utilizan para referirse a las ubicaciones y los clientes, respectivamente.

a) La primera forma de relajar el problema consiste en sumar todas las restricciones (relajación subrogada introducida en el apartado 3.5.1.6.):

$$[\text{MIN}]\ Z_R = \sum_i X_i$$
$$\sum_j \sum_i a_{ij} \cdot X_i \geq \sum_j 1 \ \rightarrow\ \sum_i (\sum_j a_{ij}) \cdot X_i \geq n \ \rightarrow\ \sum_i n_i \cdot X_i \geq n$$
$$X_i \in \{0, 1\},$$

donde n es el número total de clientes a cubrir y n_i es el número de clientes que cubre la ubicación i.

Como ya es conocido, la resolución óptima de este problema es inmediata: se ordenan las ubicaciones en orden no creciente de n_i y se van seleccionando ubicaciones hasta cubrir n clientes (aunque éstos no sean diferentes).

b) El problema a resolver también se puede relajar de la siguiente manera:

$$[MIN]\ Z_R = \sum_i X_i$$
$$\sum_i a_{ij} \cdot X_i \geq 1 \qquad \forall j$$
$$X_i \leq 1 \qquad \forall i$$

Y se puede plantear el programa dual del programa relajado:

$$[MAX]\ W_R = \sum_j U_j + \sum_i V_i$$
$$\sum_j a_{ij} \cdot U_j + V_i \leq 1 \qquad \forall i$$
$$U_j \geq 0$$
$$V_i \leq 0$$

Como el objetivo consiste en encontrar una solución factible, se puede fijar $V_i = 0$ (además, no parece aventurado suponer que existe una solución óptima con todos los valores V_i nulos); el resultado final es el siguiente programa matemático:

$$[MAX]\ W_R = \sum_j U_j$$
$$\sum_j a_{ij} \cdot U_j \leq 1 \qquad \forall i$$
$$U_j \geq 0$$

Si además se considera que $U_j \in \{0, 1\}$, el problema resultante se puede "ver" como un grafo en el que los vértices son los clientes y existe una arista entre los clientes i y j, si ambos aparecen juntos en alguna restricción; de esta manera, el problema se convierte en el problema del conjunto interiormente estable de cardinal máximo. Para solucionarlo se ha implementado una heurística que consiste en seleccionar, de forma dinámica y hasta finalizar la lista de vértices posibles, aquél de grado menor.

- Se ha implementado una separación binaria consistente en fijar o no la ubicación seleccionada, que se corresponde con la que cubre más clientes entre los todavía no cubiertos.

9.3.2. Estrategias ensayadas en el problema de cubrimiento.

A continuación se introducen las condiciones de partida del experimento -los juegos de ejemplares que se han utilizado, el ordenador usado y las condiciones de aborto de la búsqueda (apartado 9.3.2.1.)- y se presentan las indicaciones que se pueden intuir al analizar los resultados obtenidos de las estrategias ensayadas (apartado 9.3.2.2.).

9.3.2.1. Condiciones de partida.

Los ejemplares utilizados en la experiencia computacional realizada han sido generados aleatoriamente siguiendo el proceso siguiente: dentro de un área cuadrada, se generan, según una ley uniforme, las coordenadas de un conjunto de poblaciones que deben ser servidas; para ello se deben fijar una serie de centros de distribución, de forma que la distancia entre toda población y al menos un almacén no sea superior a una valor dado; los centros de distribución se deben ubicar en las mismas coordenadas que las poblaciones; y, por último, para trabajar con conjuntos de ejemplares de características similares, se adoptan distancias de forma que se conserve el número medio de clientes servidos desde cada ubicación.

Concretamente, se han generado 100 ejemplares de cada una de las siguientes combinaciones de valores (en las columnas se expone el número de poblaciones, P, y en las filas el número medio de poblaciones que quedan cubiertas desde cada centro de distribución, N):

N\P	100	200	300	400
5	√	√	√	√
30	√	√	√	---

Tipos de ejemplares generados para el problema de cubrimiento.

En la mayoría de casos, y para todas las estrategias de búsqueda ensayadas, algunos ejemplares son resueltos de forma inmediata en la etapa inicial de preproceso; de esta manera, todos los análisis efectuados se refieren a los problemas restantes:

N\P	100	200	300	400
5	6	17	35	36
30	9	88	100	---

Ejemplares restantes de cada tipo para el problema de cubrimiento.

El ordenador utilizado ha sido el mismo que para la experiencia computacional realizada con el problema de *flow-shop*: Pentium II a 233 Mhz.

Para cada ejemplar las condiciones de finalización de la búsqueda, si no se ha demostrado la optimalidad de la solución preferible, han sido las siguientes:

- límite de tiempo: consumir un tiempo de cálculo de 1.000 segundos;
- límite en la capacidad de memoria: alcanzar la cifra de 10.000 vértices generados pero no descartados; para los ejemplares de P/N = 300/30 y 400/5, este valor ha sido de 6.750 vértices, debido a problemas de capacidad de memoria.

9.3.2.2. Estrategias ensayadas.

Con el problema de cubrimiento se han ensayado dos ideas: por un lado, se ha probado la efectividad de realizar preproceso en todos y cada uno de los vértices de la arborescencia generados; y, por otro, se ha comprobado si el hecho de utilizar varias cotas en cascada repercute positivamente en la búsqueda global, tal y como se ha comentado en el apartado 7.4.3. "... éstos (los procedimientos de acotación) pueden ser aplicados en orden creciente de su dificultad o coste de cálculo, con la esperanza de que los procedimientos menos costosos proporcionen un valor de la cota que permita cerrar rápidamente el vértice ...". En concreto se han implementado las siguientes estrategias:

a) Estrategia básica: utilizar como técnica de acotación el primero de los procedimientos de relajación definidos en el apartado 9.3.1. (cota1); no utilizar preproceso en los vértices intermedios.
b) Utilizar el segundo de los procedimientos de acotación definidos en el apartado 9.3.1. (cota2); no utilizar preproceso en los vértices intermedios.
c) Utilizar los dos procedimientos de acotación: cota = max (cota1, cota2); no utilizar preproceso en los vértices intermedios.
d) Utilizar los procedimientos de acotación en cascada: calcular cota1 y, si no se puede podar el vértice, calcular cota2 (usualmente de mejor calidad); no utilizar preproceso en los vértices intermedios.
e) Utilizar la cota1 y el preproceso en los vértices intermedios.
f) Utilizar los procedimientos de acotación en cascada (estrategia d), con preproceso en los vértices intermedios.

Como en el problema de *flow-shop*, en este caso tampoco se ha probado el efecto de considerar varias estrategias de forma simultánea; por este motivo no se han ensayado las técnicas resultantes de incorporar en las estrategias b y c, preproceso en los vértices intermedios.

Las tablas siguientes muestran los resultados obtenidos al resolver los juegos de ejemplares generados:

P = 100 N = 5	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	6	0	0	6	100	0.83	0.67	2
Estrategia b	6	0	0	6	100	1	0.83	2
Estrategia c	6	0	0	6	100	0.83	0.83	1
Estrategia d	6	0	0	6	100	0.83	0.83	1
Estrategia e	6	0	0	6	100	1	1	0
Estrategia f	6	0	0	6	100	1	0.83	0

Resultados de los 6 ejemplares de 100 poblaciones y N = 5.

P = 200 N = 5	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor Cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	17	0	0	17	100	5.24	5.18	16
Estrategia b	17	0	0	17	100	4.82	4.82	4
Estrategia c	17	0	0	17	100	4.76	4.76	4
Estrategia d	17	0	0	17	100	4.88	4.88	4
Estrategia e	17	0	0	17	100	4.94	4.88	1
Estrategia f	17	0	0	17	100	4.59	4.59	1

Resultados de los 17 ejemplares de 200 poblaciones y N = 5.

P = 300 N = 5	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	35	0	0	35	100	30.14	30.09	157
Estrategia b	35	0	0	35	100	16.31	16.20	6
Estrategia c	35	0	0	35	100	16.31	16.26	5
Estrategia d	35	0	0	35	100	16.29	16.14	5
Estrategia e	35	0	0	35	100	16.37	16.29	1
Estrategia f	35	0	0	35	100	16.20	16.17	1

Resultados de los 35 ejemplares de 300 poblaciones y N = 5.

P = 400 N = 5	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	36	0	0	36	100	39.22	37.75	23
Estrategia b	36	0	0	36	100	38.06	37.81	9
Estrategia c	36	0	0	36	100	37.86	37.58	8
Estrategia d	36	0	0	36	100	39.08	38.78	8
Estrategia e	36	0	0	36	100	38.03	37.69	2
Estrategia f	36	0	0	36	100	37.86	37.67	2

Resultados de los 36 ejemplares de 400 poblaciones y N = 5.

P = 100 N = 30	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	9	0	0	9	100	0.67	0.56	4
Estrategia b	9	0	0	9	100	0.56	0.33	3
Estrategia c	9	0	0	9	100	0.89	0.78	3
Estrategia d	9	0	0	9	100	0.67	0.67	3
Estrategia e	9	0	0	9	100	0.67	0.67	0
Estrategia f	9	0	0	9	100	0.33	0.33	0

Resultados de los 9 ejemplares de 100 poblaciones y N = 30.

P = 200 N = 30	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a	83	4	0	83	94.3	72.11	61.56	1001
Estrategia b	88	0	0	88	100	14.50	10.83	16
Estrategia c	88	0	0	88	100	14.81	11.13	16
Estrategia d	88	0	0	88	100	14.57	10.99	16
Estrategia e	88	0	0	88	100	18.67	10.75	9
Estrategia f	88	0	0	88	100	10.48	7.18	4

Resultados de los 88 ejemplares de 200 poblaciones y N = 30.

P = 300 N = 30	Z* Probado	Z* no probado	Mejor $\overline{Z}$	Mejor cota	% optim.	T. medio de Z* (s)	T. medio de $\overline{Z}$ (s)	N° medio max V.A.
Estrategia a[49]	1	1	1	1	1	21.00	14.56	141
Estrategia b	80	0	1	95	80	357.29	286.86	1115
Estrategia c	80	0	1	95	80	364.2	292.15	1098
Estrategia d	81	0	1	96	81	367.20	298.19	1157
Estrategia e	28	55	12	28	28	508.61	239.58	276
Estrategia f	81	7	10	93	81	417.69	164.84	54

Resultados de los 100 ejemplares de 300 poblaciones y N = 30.

Las columnas de las tablas anteriores conservan el mismo significado que el expuesto para las tablas de resultados del problema de *flow-shop* (apartado 9.2.2.).

El análisis de los resultados obtenidos permite entrever, que el hecho de utilizar procedimientos de preproceso en todos los vértices generados puede resultar muy positivo si la cota utilizada es de baja calidad: se resuelven de forma óptima más ejemplares, en menos tiempo promedio y con menor necesidad de capacidad de

[49] En este caso, los valores medios de los tiempos de cálculo y el número máximo de vértices generados pero no explorados que coexisten de forma simultánea, carecen de sentido práctico: se refieren al único ejemplar para el que se ha probado la optimalidad de la solución preferible.

memoria; e incluso puede resultar interesante con cotas de mayor calidad: se resuelve de forma óptima el máximo número de ejemplares, parece que en mayor tiempo promedio -de todas formas sólo del orden de un 12% más-, utilizando una notable menor necesidad de capacidad de memoria, y, además, el tiempo promedio para obtener la solución preferible es también bastante menor. Para los ejemplares en los que no se prueba la optimalidad de la solución preferible, utilizar procedimientos de preproceso en los vértices intermedios permite alcanzar, en la mayoría de ocasiones, o la solución óptima o la mejor solución preferible.

De esta manera y debido a los buenos resultados que pueden ofrecer, se aconseja probar la introducción de procedimientos de preproceso en vértices intermedios de la arborescencia.

En cuanto al hecho de utilizar las cotas en cascada, los resultados obtenidos son del mismo nivel de calidad que los alcanzados sin utilizar esta estrategia; esta situación puede ser debida a la baja calidad de la cota1, que hace mínimo el número de vértices que pueden ser podados directamente gracias a dicho valor. En los ejemplares en los que no se prueba la optimalidad de la solución preferible, los resultados alcanzados son, también, de la misma calidad.

Como conclusión, parece recomendable no perder de vista esta idea si la poda que se obtiene con la heurística más rápida en cuanto a tiempo de cálculo es realmente efectiva.

El programa de software desarrollado ha sido utilizado en la resolución de un problema industrial que se presenta en el campo de la ingeniería electrónica. Dado un circuito electrónico compuesto por N componentes y un conjunto de M pruebas posibles de verificación, cada una de las cuales es capaz de verificar el funcionamiento de varios componentes (relación que se puede expresar en una matriz de cubrimiento de ceros y unos), el problema consiste en minimizar el número de pruebas a realizar para verificar toda la configuración del circuito. Como se puede comprobar, el problema descrito es el problema de cubrimiento resuelto en este apartado. Únicamente se ha resuelto un ejemplar de 80 componentes, 109 pruebas de verificación y una matriz de cubrimiento con una densidad del 61,3%, y se ha alcanzado la solución óptima en unos 3 segundos de cálculo.

10. CONCLUSIONES FINALES Y POSIBLES EXTENSIONES.

A continuación se destaca la aportación original de la tesis; además se presentan las conclusiones finales alcanzadas, así como un conjunto de posibles extensiones y derivaciones de la misma.

La originalidad de la tesis reside en el diseño de *branch and win*: un metalgoritmo para resolver problemas combinatorios mediante la exploración de grafos OR, que incluye, en una terminología unificadora y como casos particulares del mismo, las técnicas existentes hasta el momento tanto en el área de la investigación operativa como en el campo de la inteligencia artificial.

En cuanto a las conclusiones alcanzadas, cabe destacar las siguientes:

a) Actualmente, en el conjunto de procedimientos de resolución de problemas de optimización combinatoria mediante técnicas enumerativas, existe un panorama algo confuso y no suficientemente estructurado: unos autores definen y utilizan unos elementos o procedimientos de una forma y otros de otra que, aun semejante, no es igual; además, las formulaciones generales de este tipo de procedimientos que se han expuesto en al literatura presentan un conjunto de carencias que, en mayor o menor medida, afectan a todas ellas. Como consecuencia, parecía necesario desarrollar una formulación general que englobase a todos estos procedimientos.

b) Se ha realizado una amplia recopilación, estudio y análisis, de los procedimientos y estrategias de resolución de problemas de optimización combinatoria existentes, así como de las formulaciones generales de dichas técnicas.

c) Respecto a las funciones de selección y a las estrategias de exploración que se proponen en los procedimientos utilizados habitualmente, se puede plantear desde un punto de vista industrial, que hay aspectos que no son considerados suficientemente.

d) Se ha propuesto y formalizado *branch and win*, un metalgoritmo general de exploración de grafos que presenta las siguientes características:

1) El metalgoritmo propuesto es suficientemente general e integrador, como para incorporar, como casos particulares del mismo, a todas las técnicas enumerativas de búsqueda descritas en la literatura referenciada, proveniente tanto del área de la investigación operativa como del campo de la inteligencia artificial.

2) *Branch and win* es suficientemente sencillo en cuanto a la terminología y formalización utilizada, como para presentar una fácil e inmediata aplicabilidad práctica.

3) Los diversos procedimientos referenciados se obtienen al combinar un escaso número de elementos comunes, cinco o seis, cuya especificación permite obtenerlos.

4) Se introduce el concepto de dinamismo en los diferentes procedimientos que forman parte del metalgoritmo: acotación, reducción, resolución heurística, exploración de entornos, propiedades de las soluciones óptimas, función de evaluación, etc.

5) Se propone una nueva función de evaluación y selección, f(n, t, a_t, e), que presenta tres características a destacar:

- es una función general que, particularizando los valores de sus parámetros, permite obtener todas las demás funciones y estrategias de selección,

- incluye muchos nuevos elementos no considerados tradicionalmente pero de gran importancia (dónde está el vértice, de cuánta memoria se dispone, de cuánto tiempo se dispone para obtener una solución factible, etc.), que en función de su concreción permiten obtener un amplio abanico de nuevas estrategias de búsqueda, que posibilitan tener presentes aspectos más realistas desde el punto de vista práctico,

- presenta un importante carácter dinámico, que permite cambiar la estrategia de selección en función de diversos parámetros: si se precisa una solución a una hora determinada, cuando falte escaso tiempo se puede cambiar la estrategia de selección dando mayor importancia a encontrar una solución factible o una buena solución (si no se ha obtenido ninguna), más que la solución óptima; en función de la memoria disponible también se puede cambiar la estrategia de búsqueda evitando que ésta se desborde; etc.

6) Se permite el uso de procedimientos de reducción y de resolución heurísticos (con o sin optimización local), en los vértices intermedios de la arborescencia.

7) Se incorporan, como parte integrante de *branch and win*, nuevas técnicas de reducción procedentes del campo de la inteligencia artificial: técnicas de consistencia, etc.

8) *Branch and win* permite diseñar multitud de nuevos procedimientos híbridos, resultado de la combinación de los ya existentes y de los nuevos elementos integrados en el metalgoritmo.

e) El paralelismo, como una nueva herramienta del área informática, es considerado únicamente en la fase de programación de *branch and win* y no en su formulación. Esto es debido a que es un tema todavía en fase de continua evolución y estudio y, además, se considera que no ha de formar parte de la formulación en sí misma.

g) Se ha realizado una implementación informática del metalgoritmo propuesto para el problema de *flow shop* y el de cubrimiento, que ha permitido validar la generalidad de *branch and win*, así como obtener una serie de recomendaciones sobre la conveniencia de, en el momento de solucionar un problema de optimización combinatoria, probar la adecuación de un conjunto de estrategias y procedimientos: invertir tiempo en la búsqueda de una solución inicial de calidad, utilizar funciones de evaluación y selección dinámicas, utilizar "en cascada" diversos procedimientos del mismo tipo (en este caso de acotación), uso de procedimientos de reducción y de resolución heurísticos en los vértices intermedios de la arborescencia, incorporación de procedimientos de exploración de entornos al obtener una nueva solución factible, etc.

En cuanto a las posibles extensiones y derivaciones de esta tesis, existe un gran campo de investigación abierto en la dirección de probar, para diferentes problemas de optimización combinatoria, nuevos procedimientos híbridos, que se pueden obtener al combinar y utilizar de diversas maneras los elementos que forman *branch and win*. Esta investigación se concretaría en ensayar el impacto de los diferentes elementos que forman el metalgoritmo propuesto, con el objetivo de encontrar reglas generales para la resolución efectiva de diferentes problemas de optimización combinatoria o de familias de problemas de características similares. Las acciones a emprender consistirían en el diseño de procedimientos híbridos derivados de *branch and win*, así como en su aplicación de forma extensa a diferentes problemas de optimización combinatoria.

11. REFERENCIAS BIBLIOGRÁFICAS.

AFIF, M.; HIFI, M.; PASCHOS, V.T.; ZISSIMOPOULOS, V. (1995): A new efficient heuristic for the minimum set covering problem. *Journal of the Operational Research Society*, 46, 1260-1268.

AGIN, N. (1966): Optimum seeking with branch and bound. *Management Science*, 13 (4), 176-185.

AHUJA, R.K.; MAGNANTI, T.L.; ORLIN, J.B. (1989): Network Flows. *Handbooks in Operations Research and Management Science. Volume 1 OPTIMIZATION.* Elsevier Science.

ALMEIDA, F.; RODRÍGUEZ, C.; RODA, J.L.; MORALES, D.; GARCÍA, F. (1995): Algoritmos de ramificación y acotación paralela sobre entornos distribuidos. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 305-306. Sevilla.

ALMIÑANA, M.; PASTOR, J.T. (1995): Un algoritmo branch-and-bound para el problema de localización total. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 385-386. Sevilla.

ALONSO, A.J.; ESCUDERO, L.F. (1995): Análisis comparativo de dos formulaciones equivalentes para ciertos problemas combinatorios. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 307-308. Sevilla.

ALONSO, A.J.; ESCUDERO, L.F. (1997): Control del tráfico aéreo: formulaciones equivalentes. *Actas XXIII Congreso Nacional de Estadística e Investigación Operativa*, 48.11-48.12. Valencia.

ALONSO, A.J.; ESCUDERO, L.F. (1998): 0-1 equivalent model representations for air traffic management. *Top*, 6 (1), 19-36.

ÁLVAREZ, R.; TAMARIT, J.M.; VALLS, V. (1988): Estudio computacional de algunos nuevos algoritmos heurísticos para el problema de planificación de proyectos con limitación de recursos. *Trabajos de Investigación Operativa*, 3 (1), 55-70.

BACCHUS, F.; GROVE, A. (1995): On the Forward Checking Algorithm. *Lecture Notes in Computer Science*, 976, 292-309.

BALAS, E.; PADBERG, M.W. (1972): On the set covering problem. *Operations Research*, 20, 1152-1161.

BALAS, E.; HO, A. (1980): Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. *Mathematical Programming*, 12, 37-60.

BALAS, E.; CERIA, S.; CORNUÉJOLS, G. (1993): A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58, 295-324.

BALAS, E.; CERIA, S.; CORNUÉJOLS, G. (1996): Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42 (9), 1229-1246.

BALCÁZAR, J.L. (1993): *Programación Metódica*. McGraw-Hill. Madrid.

BARNHART, C.; JOHNSON, E.L.; NEMHAUSER, G.L.; SAVELSBERGH, M.W.P.; VANCE, P.H. (1998): Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46 (3), 316-329.

BARR, A.; FEIGENBAUM, E.A. (1981): *The Handbook of Artificial Intelligence (Volume 1)*. Ed. A. Barr & E.A. Feigenbaum. Kaufmann. California.

BARTÁK, R. (1998): On-line guide to Constraint Programming. *http://kti.ms.mff.cuni.cz/~bartak/constraints/index.html*.

BAUTISTA, J.; COMPANYS, R.; COROMINAS, A. (1992): Introducción al BDP. *D.I.T. 92/04*. DOE. ETSEIB. UPC. Barcelona.

BAUTISTA, J.; COMPANYS, R.; COROMINAS, A. (1994): Modelos y algoritmos para la determinación de secuencias regulares en líneas de montaje mixtas con restricciones en la elaboración de productos. *D.I.T. 94/21*. DOE. ETSEIB. UPC. Barcelona.

BEALE, S. (1997): Using Branch-and-Bound with Constraint Satisfaction in Optimization Problems. *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference*, 209-214. Menlo Park.

BISDORFF, R.; LAURENT, S. (1995): Industrial linear optimization problems solved by constraint logic programming. *European Journal of Operational Research*, 84, 82-95.

BISIANI, R. (1990): Beam Search. *Encyclopedia of Artificial Intelligence*. Ed. S.C. Shapiro. John Wiley & Sons. Canadá.

BJORNDAL, M.H.; CAPRARA, A.; COWLING, P.I.; DELLA CROCE, F.; LOURENÇO, H.; MALUCELLI, F.; ORMAN, A.J.; PISINGER, D.; REGO, C.; SALAZAR, J.J. (1995): Some thoughts on combinatorial optimisation. *European Journal of Operational Research*, 83, 253-270.

BLAZEWICZ, J.; ECKER, K.H.; PESCH, E.; SCHMIDT, G.; WEGLARZ, J. (1996): *Scheduling Computer and Manufacturing Processes*. Springer. Berlin.

BONDY, J.A. (1995): Basic graph theory: Paths and circuits. *Handbook of combinatorics, vol. 1*. Ed. R.L. Graham, M. Grötschel & L. Lovász. Elsevier Science. The Netherlands.

BOSCH, L. (1993): Resolució del problema del tall de materials unidimensional mitjançant la programació dinàmica acotada. *PFC*. DOE. ETSEIB. UPC. Barcelona.

BRUIN, A.; RINNOOY KAN, A.H.G.; TRIENEKENS, H.W.J.M. (1988): A simulation tool for the performance evaluation of parallel branch and bound algorithms. *Mathematical Programming*, 42, 245-271.

BRUNAT, J.M. (1996): *Combinatòria i teoria de grafs*. Edicions UPC. Aula Teòrica 46. Barcelona.

BRUYNOOGHE, M.; VENKEN, R. (1990): Backtracking. *Encyclopedia of Artificial Intelligence*. Ed. S.C. Shapiro. John Wiley & Sons. Canadá.

CANNON, T.L.; HOFFMAN, K.L. (1990): Large-scale 0-1 programming on distributed workstations. *Annals of Operations Research*, 22, 181-217.

CAPRARA, A.; FISCHETTI, M. (1997): Branch-and-cut algorithms. *Annotated bibliographies in combinatorial optimization*. Ed. M. Dell'Amico, F. Maffioli & S. Martello. Wiley.

CERIA, S.; NOBILI, P.; SASSANO, A. (1997): Set Covering Problem. *Annotated bibliographies in combinatorial optimization*. Ed. M. Dell'Amico, F. Maffioli & S. Martello. Wiley.

CHAKRABARTI, P.P.; GHOSE, S.; ACHARYA, A.; DE SARKAR, S.C. (1989): Heuristic Search in Restricted Memory. *Artificial Intelligence*, 41, 197-221.

COMPANYS, R. (1982): *Programación Dinámica*. Ed. CPDA-ETSEIB. Barcelona.

COMPANYS, R. (1989a): *El emperador de todas las cosas*. Ed. R. Companys. UPC. Barcelona.

COMPANYS, R. (1989b): *Planificación y programación de la producción*. Marcombo. Barcelona.

COMPANYS, R. (1995): Algoritmo de Lomnicki pendular. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 435-436. Sevilla.

COMPANYS, R.; COROMINAS, A. (1996): *Organización de la producción II. Dirección de operaciones 4*. Edicions UPC. Barcelona.

CORMEN, T.H.; LEISERSON, C.E.; RIVEST, R.L. (1990): *Introduction to Algorithms*. The MIT Press. Massachusetts.

COROMINAS, A. (1974): Modelos y algoritmos para la selección de inversiones. *Notas de trabajo de COPIOSI*, 3, 18-35. ETSEIB. Barcelona.

COROMINAS, A.; COMPANYS, R. (1977): Procedimiento generalizado de branch and bound. *Qüestiió*, 1 (1), 49-62.

COROMINAS, A.; ROSELLÓ, X.; COMPANYS, R.; BERENGUER, X. (1984): *Optimización Combinatoria*. Ed. CPDA-ETSEIB. Barcelona.

COROMINAS, A. (1996): Mètodes i Tècniques Quantitatives. *Notas de curso de doctorado*. Programa Administració i Direcció d'Empreses. ETSEIB. UPC. Barcelona.

CORRÊA, R. (1995): A Parallel Formulation for General Branch-and-Bound Algorithms. *Lecture Notes in Computer Science*, 980, 395-409.

CORTÉS, U.; BÉJAR, J.; MORENO, A. (1993): *Inteligencia artificial*. Edicions UPC. Col.lecció POLITEXT. Barcelona.

COVES, A. (1994): Equilibrado de líneas de montaje y producción. *D.I.T. 94/17*. DOE. ETSEIB. UPC. Barcelona.

CROWDER, H.; JOHNSON, E.L.; PADBERG, M. (1983): Solving Large-Scale Zero-One Linear Programming Problems. *Operations Research*, 31 (5), 803-834.

CRUZ, F.R.B.; MATEUS, G.R. (1997): Parallel Implementations of Branch-and-Bound Algorithms Applied to a Network Design Problem. *Working paper*.

DARBY-DOWMAN, K.; LITTLE, J.; MITRA, G.; ZAFFALON, M. (1997): Constraint Logic Programming and Integer Programming Approaches and Their Collaboration in Solving an Assignment Scheduling Problem. *Constraints, An International Journal*, 1 (3), 245-264.

DASGUPTA, P.; CHAKRABARTI, P.P.; DE SARKAR, S.C. (1994): Agent searching in a tree and the optimality of iterative deepening. *Artificial Intelligence*, 71, 195-208.

DE MIGUEL, P.; ANGULO, J.M. (1991): Fundamentos e introducción al paralelismo. *Arquitectura de computadores*, 467-519. Paraninfo.

DIETRICH, B.L.; ESCUDERO, L.F. (1990): Coefficient Reduction for Knapsack-Like Constraints in 0-1 Program with Variable Upper Bounds. *Operations Research Letters*, 2, 9-14.

DILLENBERGER, C.; ESCUDERO, L.F.; WOLLENSAK, A.; ZHANG, W. (1994): On practical resource allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research*, 75, 275-286.

DILLENBURG, J.F.; NELSON, P.C. (1994): Perimeter search. *Artificial Intelligence*, 65, 165-178.

DINCBAS, M.; SIMONIS, H.; HENTENRYCK, P.V. (1988): Solving the Car-Sequencing Problem in Constraint Logic Programming. *Internal Report ECRC TR-LP-32*. European Computer-Industry Research Centre. Germany.

DJERDJOUR, M. (1997): An enumerative algorithm framework for a class of nonlinear integer programming problems. *European Journal of Operational Research*, 101, 104-121.

DYER, M.E.; RIHA, W.O.; WALKER, J. (1995): A hybrid dynamic programming/branch-and-bound algorithm for the multiple-choice knapsack problem. *Journal of Computational and Applied Mathematics*, 58, 43-54.

ECKSTEIN, J. (1994): Parallel branch-and-bound algorithms for the general mixed integer programming on the CM-5. *Siam Journal Optimization*, 4 (4), 794-814.

ECKSTEIN, J. (1996): Parallel computing. *Encyclopedia of Operations Research and Management Science*. Ed. S.I. Gass & C.M. Harris. Kluwer. Massachusetts.

ESCUDERO, L.F.; GARÍN, A.; PÉREZ, G. (1995a): Utilización de sistemas de ciclos solapados para la obtención de condiciones de redundancia e infactibilidad en problemas enteros mixtos 0-1. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 335-336. Sevilla.

ESCUDERO, L.F.; GÓMEZ, E.; SALMERÓN, J. (1995b): Sistema Integrado de Selección y Periodificación de Inversiones. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 585-586. Sevilla.

EVEN, S. (1979): *Graph Algorithms*. Computer Science Press. Maryland.

FERNÁNDEZ, R.; COMPANYS, R. (1995): Determinación del tiempo de ciclo en un proceso de n-etapas con un robot transportador y restricciones de ventana. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 445-446. Sevilla.

FISCHETTI, M.; SALAZAR, J.J.; TOTH, P. (1995): A Branch-and-Cut Algorithm for the Orienteering Problem. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 345-346. Sevilla.

FLYNN, M.J. (1966): Very high-speed computing systems. *Proc. IEEE 54*, 1901-1909.

FRANCO, M.; LÓPEZ, S. (1995): Aplicació de la BDP al Problema del Taller Mecànic (n/m/P/Fmax). Experimentació. Comparació amb altres heurístiques i algorismes. Aplicació a cas d'empresa. *PFC*. DOE. ETSEIB. UPC. Barcelona.

FREUDER, E.C.; WALLACE, R.J. (1992): Partial constraint satisfaction. *Artificial Intelligence*, 58, 21-70.

FRÜHWIRTH, T.; HEROLD, A.; KÜCHENHOFF, V.; LE PROVOST, T.; LIM, P.; MONFROY, E.; WALLACE, M. (1992): Constraint Logic Programming – An Informal Introduction. *Internal Report ECRC-92-6i*. European Computer-Industry Research Centre. Germany.

FUENTES, O.; RAO, R.P.N.; WIE, M.V. (1997): Hierarchical Learning of Reactive Behaviors in an Autonomous Mobile Robot. *Computación y Sistemas*, 1 (2), 71-75.

GANGONELLS, J.; GÓMEZ, F. (1995): Aplicació de la programació dinàmica acotada (BDP) a la programació de projectes amb limitació de recursos i comparació amb procediments heurístics de millora de solucions, intercanvis genètics i mètodes exactes. *PFC*. DOE. ETSEIB. UPC. Barcelona.

GAREY, M.R.; JOHNSON, D.S. (1979): *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman. San Francisco.

GARFINKEL, R.S.; NEMHAUSER, G.L. (1972): *Integer Programming*. John Wiley & Sons.

GASCHNIG, J. (1977): A general backtrack algorithm that eliminates most redundant checks. *Proceedings IJCAI-77*, 457. Cambridge, MA.

GASCHNIG, J. (1978): Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems. *Proceedings Second National Conference of the Canadian Society for Computational Studies of Intelligence*, 268-277. Toronto, Ont.

GASS, S.I.; HARRIS, C.M. (1996): A* algorithm. Branch and bound. Column generation. Combinatorial explosion. Gomory cut. Mixed-integer programming problem (MIP). Optimization. Polynomially bounded (-time) algorithm (polynomial algorithm). Set-covering problem. Set-partitioning problem. Strongly polynomial-time algorithm. Suboptimization. Worst-case analysis. *Encyclopedia of Operations Research and Management Science*. Ed. S.I. Gass & C.M. Harris. Kluwer. Massachusetts.

GENDRON, B.; CRAINIC, T.G. (1994): Parallel Branch-and-Bound Algorithms: Survey and Synthesis. *Operations Research*, 42 (6), 1042-1066.

GINSBERG, M. (1993): *Essentials of Artificial Intelligence*. Kaufmann. California.

GONZÁLEZ, J.L. (1996): GRASP. *Optimización Heurística y Redes Neuronales en Dirección de Operaciones e Ingeniería*. Coord. A. Díaz. Paraninfo. Madrid.

GRANVILLIERS, L. (1998): A Symbolic-Numerical Branch and Prune Algorithm for Solving Non-linear Polynomial Systems. *Journal of Universal Computer Science*, 4 (2).

GREENBERG, H.J. (1996): Artificial intelligence. *Encyclopedia of Operations Research and Management Science*. Ed. S.I. Gass & C.M. Harris. Kluwer. Massachusetts.

GRÖTSCHEL, M.; LOVÁSZ, L. (1995): Combinatorial Optimization. *Handbook of combinatorics, vol. 2*. Ed. R.L. Graham, M. Grötschel & L. Lovász. Elsevier Science. The Netherlands.

GUILLÉN, P.; JOVANÍ, M.; VALLS, V. (1995): El problema de transbordo con costes fijos. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 361-362. Sevilla.

HALL, L. (1996): Computational complexity. *Encyclopedia of Operations Research and Management Science*. Ed. by S.I. Gass & C.M. Harris. Kluwer. Massachusetts.

HARTNELL, T. (1986): *Inteligencia Artificial: conceptos y programas*. Anaya Multimedia. Madrid.

HELMAN, P. (1989): A Common Schema for Dynamic Programming and Branch and Bound Algorithms. *Journal of the Association for Computing Machinery*, 36 (1), 97-128.

HENTENRYCK, P.V.; SIMONIS, H.; DINCBAS, M. (1992): Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58, 113-159.

HENTENRYCK, P.V. (1995): Constraint Solving for Combinatorial Search Problems: A Tutorial. *Lecture Notes in Computer Science*, 976, 564-587.

HENTENRYCK, P.V.; MCALLESTER, D.; KAPUR, A.D. (1997): Solving polynomial systems using a branch and prune approach. *Siam Journal Numer. Anal*, 34 (2), 797-827.

HERVÉ, P.H. (1968): Les procédures arborescentes d'optimisation. *Rev. Francaise d'Informatique et de Recherche Opérationelle*, 3.

HILLIER, F.S.; LIEBERMAN, G.J. (1991): *Introducción a la Investigación de Operaciones*. McGraw-Hill. México.

HOFFMAN, K.L.; PADBERG, M. (1991): Improving LP-Representations of Zero-One Linear Programs for Branch-and-Cut. *ORSA Journal on Computing*, 3 (2), 121-134.

HOFFMAN, K.L.; PADBERG, M. (1993): Solving Airline Crew Scheduling Problems by Branch-and-Cut. *Management Science*, 39 (6), 657-682.

HOFFMAN, K.L.; PADBERG, M. (1996): Combinatorial and integer optimization. *Encyclopedia of Operations Research and Management Science*. Ed. by S.I. Gass & C.M. Harris. Kluwer. Massachusetts.

HOOGEVEEN, J.A.; LENSTRA, J.K.; VAN DE VELDE, S.L. (1997): Sequencing and Scheduling. *Annotated bibliographies in combinatorial optimization.* Ed. M. Dell'Amico, F. Maffioli & S. Martello. Wiley.

HOOKER, J.N.; OSORIO, M.A. (1996): Mixed Logical/Linear Programming. *Working paper.*

HRIBAR, M.; DASKIN, M.S. (1997): A dynamic programming heuristic fir the *P*-median problem. *European Journal of Operational Research*, 101, 499-508.

IBARAKI, T. (1976a): Theoretical and Simulation Studies on Computational Performance of Branch-and-Bound Algorithms. *Survey of Mathematical Programming*, 505-524. Publishing House of Hungarian Academy of Sciences. PREKOPA.

IBARAKI, T. (1976b): Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms. *International Journal of Computer and Information Sciences*, 5 (4), 315-344.

IBARAKI, T. (1988): Enumerative approaches to combinatorial optimization. Part I: *Annals of Operations Research*, 10 (1-4). Part II: *Annals of Operations Research*, 11 (1-4).

JAFFAR, J.; MAHER, M.J. (1994): Constraint logic programming: a survey. *Journal of Logic Programming*, 19 & 20, 503-582.

JOHNSON, D.S. (1974): Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9, 256-278.

JOHNSON, E.L. (1989): Modeling and Strong Linear Programs for Mixed Integer Programming. *NATO ASI Series, F51. Algorithms and Model Formulations in Mathematical Programming.* Wallace. Berlin.

JOHNSON, E.L.; NEMHAUSER, G.L.; SAVELSBERGH, M.W.P. (1997): Perspectives on Discrete Optimization. *Plenaries and Tutorials. EURO XV / INFORMS XXXIV Joint International Meeting*, 33-55. Ed. J. Barceló.

JÜNGER, M.; REINELT, G.; THIENEL, S. (1994): Practical Problem Solving with Cutting Plane Algorithms in Combinatorial Optimization. *Preprint 94 - 24.* Universität Heidelberg.

JÜNGER, M.; THIENEL, S. (1998): The ABACUS System for Branch-and-Cut-and-Price Algorithms in Integer Programming and Combinatorial Optimization. *Technical Report 98322*. Universität zu Köln.

KARP, R.M.; ZHANG, Y. (1993): Randomized parallel algorithms for back-track search and branch-and-bound computation. *Journal of the ACM*, 40 (3), 765-789.

KAUFMANN, A.; CRUON, R. (1967): *La programación dinámica.* Compañía Editorial Continental. México.

KINDERVATER, G.A.P.; LENSTRA, J.K. (1986): An introduction to parallelism in combinatorial optimization. *Discrete Applied Mathematics*, 14, 135-156.

KLEER, J. DE (1990): Backtracking, Dependency Directed. *Encyclopedia of Artificial Intelligence*. Ed. S.C. Shapiro. John Wiley & Sons. Canadá.

KOHLER, W.H.; STEIGLITZ, K. (1974): Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems. *Journal of the Association for Computing Machinery*, 21 (1), 140-156.

KORF, R.E. (1985): Depth-First Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27, 97-109.

KORF, R.E. (1990): Search. *Encyclopedia of Artificial Intelligence*. Ed. S.C. Shapiro. John Wiley & Sons. Canadá.

KORF, R.E. (1993): Linear-space best-first search. *Artificial Intelligence*, 62, 41-78.

KUBIAK, W.; BLAZEWICZ, J.; FORMANOWICZ, P.; SCHMIDT, G. (1997): Flow shops with limited machine availability. *Working paper.*

KUMAR, V.; KANAL, L.N. (1983a): A General Branch and Bound Formulation for Understanding and Synthesizing And/Or Tree Search Procedures. *Artificial Intelligence*, 21, 179-198.

KUMAR, V.; KANAL, L. (1983b): The composite decision process: a unifying formulation for heuristic search, dynamic programming and branch and bound procedures. *Proceedings of the Third National Conference on Artificial Intelligence*, 220-224. Washinton.

KUMAR, V. (1990): Search, Branch-and-Bound. *Encyclopedia of Artificial Intelligence*. Ed. S.C. Shapiro. John Wiley & Sons. Canadá.

KUMAR, V. (1992): Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, 13 (1), 32-44.

KUMAR, V.; GRAMA, A.; GUPTA, A.; KARYPIS, G. (1994): *Introduction to parallel computing. Design and analisis of algorithms*. Benjamin/Cummings Publishing.

KUSIAK, A.; BOE, W.J.; CHENG, C. (1993): Designing Cellular Manufacturing Systems: Branch-and-Bound and A* Approaches. *Industrial Engineering Research & Development*, 25 (4), 46-56.

LAWLER, E.L.; WOOD, D.E. (1966): Branch and bound methods: a survey. *Operations Research*, 14 (4), 699-719.

LAWLER, E.L. (1996): Combinatorics. *Encyclopedia of Operations Research and Management Science*. Ed. S.I. Gass & C.M. Harris. Kluwer. Massachusetts.

LE CUN, B.; ROUCAIROL, C.; BENAÏCHOUCHE, M.; CUNG, V.D.; DOWAJI, S.; MAUTOR, T. (1995): BOB: a Unified Platform for Implementing Branch-and-Bound like Algorithms. *Rapport de Recherche nº 95/16*.

LENSTRA, J.K.; RINNOOY KAN, A.H.G. (1978): Technical Notes: On the Expected Performance of Branch-and-Bound Algorithms. *Operations Research*, 26 (2), 347-349.

LE PROVOST, T.; WALLACE, M. (1991): Domain Independent Propagation. *Internal Report ECRC-91-1i*. European Computer-Industry Research Centre. Germany.

LE PROVOST, T.; WALLACE, M. (1992): Generalised Constraint Propagation Over the CLP Scheme. *Technical Report ECRC-92-1*. European Computer-Industry Research Centre. Germany.

LIAO, C.J. (1994): A new node selection strategy in the branch-and-bound procedure. *Computers & Operations Research*, 21 (10), 1095-1101.

LITTLE, J. (1993): A searching technique! Constraint Handling languages offer a robust method of solving large combinatorial problems. *OR Insight*, 6 (4), 24-31.

LITTLE, J.; DARBY-DOWMAN, K. (1995): The Significance of Constraint Logic Programming to Operational Research. *Working paper.* Department of Mathematics and Statistics, Brunel University.

LUSA, A.; MATEO, M.; PASTOR, R. (1997): Manual d'usuari de PVM (Parallel Virtual Machine). *D.I.T. 97/28.* DOE. ETSEIB. UPC. Barcelona.

MACKWORTH, A.K. (1992): The logic of constraint satisfaction. *Artificial Intelligence*, 58, 3-20.

MAHANTI, A.; DANIELS, C.J. (1993): A SIMD approach to parallel heuristic search. *Artificial Intelligence*, 60, 243-282.

MALANDRAKI, C.; DIAL, R.B. (1996): A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, 90, 45-55.

MANS, B.; MAUTOR, T.; ROUCAIROL, C. (1995): A parallel depth first search branch and bound algorithm for the quadratic assignment problem. *European Journal of Operational Research*, 81, 617-628.

MANZINI, G. (1995): BIDA*: an improved perimeter search algorithm. *Artificial Intelligence*, 75, 347-360.

MARÍN, A. (1997): Sobre cotas inferiores asociadas a la formulación de doble subíndice del problema de localización en dos etapas sin capacidades. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 34.7-34.8. Valencia.

MARSTEN, R.E.; MORIN, T.L. (1978): A hybrid approach to discrete mathematical programming. *Mathematical Programming*, 14, 21-40.

MAYOR, J.A.; RUIZ, P. (1995): Un esquema general de búsqueda local en programación entera. Evaluación computacional. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 369-370. Sevilla.

MEHROTRA, A.; TRICK, M. (1997): Solving Location and Clustering Problems Using Branch-and-Price. *Eighth Annual Meeting of the Production and Operations Management Society.* Florida.

MÉRO, L. (1984): A Heuristic Search Algortihm with Modifiable Estimate. *Artificial Intelligence*, 23, 13-27.

MEYER, R.R. (1989): Modelling for parallel optimization. *NATO ASI Series, F51. Algorithms and Model Formulations in Mathematical Programming*. Wallace. Berlin.

MÍNGUEZ, G.; ROSELLÓ, J. (1997): Estudi i implementació de l'algoritme Lomnicki Pendular per a processadors en paral·lel. *PFC*. DOE. ETSEIB. UPC. Barcelona.

MISHKOFF, H.C. (1988): *A fondo: Inteligencia artificial*. Anaya Multimedia. Madrid.

MITTEN, L. (1970): Branch-and-bound methods: General formulation and properties. *Operations Research*, 18 (1), 24-34. Errata en Operations Research, 19, 550 (1971).

MIZE, J.H.; WHITE, C.R.; BROOKS, G.H. (1973): *Planificación y control de operaciones*. Prentice-Hall International. Madrid.

MOMPÍN, J.; y varios autores más. (1987): *Inteligencia Artificial. Conceptos, técnicas y aplicaciones*. Boixareu. Barcelona.

MORIN, T.L.; MARSTEN, R.E. (1976): Branch-and-Bound Strategies for Dynamic Programming. *Operations Research*, 24 (4), 611-627.

MORIN, T.L. (1978): Computational advances in dynamic programming. *Dynamic Programming and its applications*. Ed. M.L. Puterman. Academic Press. New York.

MÜLLER-MERBACH, H. (1997): Methodology for the Design of Algorithms. *Plenaries and Tutorials. EURO XV / INFORMS XXXIV Joint International Meeting*, 105-122. Ed. J. Barceló.

NAGAR, A.; HERAGU, S.S.; HADDOCK, J. (1996): A combined branch-and-bound and genetic algorithm based approach for a flowshop scheduling problem. *Annals of Operations Research*, 63, 397-414.

NAU, D.S.; KUMAR, V.; KANAL, L. (1984): General Branch and Bound, and Its Relation to A* and AO*. *Artificial Intelligence*, 23, 29-58.

NEMHAUSER, G.L.; WOLSEY, L.A. (1988): *Integer and Combinatorial Optimization*. John Wiley & Sons. Canada.

NEMHAUSER, G.; WOLSEY, L. (1989): Integer Programming. *Handbooks in Operations Research and Management Science. Volume 1 OPTIMIZATION*. Elsevier Science.

NILSSON, N.J. (1971): *Problem-solving methods in artificial intelligence.* McGraw-Hill. USA.

NILSSON, N.J. (1980): *Principles of Artificial Intelligence.* Kaufmann. California.

PADBERG, M.W.; VAN ROY, T.J.; WOLSEY, L.A. (1985): Valid Linear Inequalities for Fixed Charge Problems. *Operations Research*, 33 (4), 842-861.

PAPADIMITRIOU, C.; STEIGLITZ, K. (1982): *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall.

PASTOR, R. (1994): Resolución del problema del equilibrado de líneas de montaje con incompatibilidades: aplicación a una empresa japonesa de montaje de motos. *PFC.* DOE. ETSEIB. UPC. Barcelona.

PASTOR, R.; COROMINAS, A. (1998): Assembly line balancing with incompatibilities and workstation load time windows. *Working paper IOC-DT-A-1998-01.* IOC. UPC. Barcelona.

PEARL, J. (1983): Knowledge versus Search: A Quantitative Analysis Using A*. *Artificial Intelligence*, 20, 1-13.

PEARL, J. (1984): *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley. California.

PLANS, J.; COROMINAS, A. (1998): Modelos de PLM para problemas de líneas de montaje y resolución mediante fix and relax. *Actas XXIV Congreso Nacional de Estadística e Investigación Operativa*, 625-626. Almería.

PORTMANN, M.C.; VIGNIER, A.; DARDILHAC, D.; DEZALAY, D. (1998): Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, 107 (2), 389-400.

POWLEY, C.; FERGUSON, C.; KORF, R.E. (1993): Depth-first heuristic search on a SIMD machine. *Artificial Intelligence*, 60, 199-242.

PRAWDA, J. (1981): *Métodos y modelos de Investigación de Operaciones. Vol. I modelos determinísticos.* LIMUSA. México.

PRESTWICH, S.; MUDAMBI, S. (1995): Improved Branch and Bound in Constraint Logic Programming. *Lecture Notes in Computer Science*, 976, 533-548.

RAFIQUZZAMAN, M.; CHANDRA, R. (1990): Del diseño lógico al proceso paralelo. *Arquitectura de ordenadores*, 375-422. ANAYA multimedia.

RAGSDALE, C.T.; SHAPIRO, G.W. (1996): Incumbent solutions in branch-and-bound algorithms: setting the record straight. *Computers & Operations Research*, 23 (5), 419-424.

RAO, V.N.; KUMAR, V. (1987): Parallel depth-first search, Part I: implementation. *International Journal of Parallel Programming*, 16 (6), 479-499.

RAPHAEL, B. (1990): A*. *Encyclopedia of Artificial Intelligence*. Ed. S.C. Shapiro. John Wiley & Sons. Canadá.

REGIN, J.C. (1998): Non binary constraints. *The 13th Biennial European Conference on Artificial Intelligence*. Workshop W15.

RICH, E. (1990): Artificial Intelligence. *Encyclopedia of Artificial Intelligence*. Ed. S.C. Shapiro. John Wiley & Sons. Canadá.

RICH, E.; KNIGHT, K. (1994): *Inteligencia Artificial*. McGraw-Hill. Madrid.

RIERA, J.; ALCAIDE, D.; SICILA, J. (1995): Algoritmos aproximados para el problema P||Cmax. *Actas XXII Congreso Nacional de Estadística e Investigación Operativa*, 511-512. Sevilla.

ROCA, G. (1992): Resolución del problema del viajante de comercio utilizando la programación dinámica acotada. *PFC*. DOE. ETSEIB. UPC. Barcelona.

ROUCAIROL, C. (1996): Parallel processing for difficult combinatorial optimization problems. *European Journal of Operational Research*, 92 (3), 573-590.

ROY, B. (1969): Procedure d'exploration par séparation et évaluation (P.S.E.P. et P.S.E.S.). *Rev. Francaise d'Informatique et de Recherche Opérationelle*, 6, 61-90.

RUSHMEIER, R.A.; NEMHAUSER, G.L. (1993): Experiments with parallel branch-and-bound algorithms for the set covering problem. *Operations Research Letters*, 13 (5), 277-286.

RYOO, H.S.; SAHINIDIS, N.V. (1996): A Branch-and-Reduce Approach to Global Optimization. *Journal of Global Optimization*, 8(2), 107-139.

SALKIN, H.M. (1975): *Integer Programming*. Addison-Wesley.

SALTZMAN, M.J. (1995): Trends in Computing Technology. *OR/MS Today. Online Edition*. http://lionhrtpub.com/orms/orms-12-95/trends.html.

SARKAR, U.K.; CHAKRABARTI, P.P.; GHOSE, S.; DE SARKAR, S.C. (1991): Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artificial Intelligence*, 50 (2), 207-221.

SAVELSBERGH, M. (1994): Preprocessing and Probing Techniques for Mixed Integer Programming Problems. *ORSA Journal on Computing*, 6 (4), 445-454.

SAVELSBERGH, M. (1997): A Branch-and-Price Algorithm for the Generalized Assignment Problem. *Operations Research*, 45 (6), 831-841.

SCHAERF, A. (1998): Local search techniques for scheduling problems. *The 13th Biennial European Conference on Artificial Intelligence*. Tutorial T3.

SCHOLL, P.C. (1986): *Algorítmica y representación de datos: tomo 2*. Masson. Barcelona.

SCHOLL, A. (1995): *Balancing and sequencing of assembly lines*. Physica-Verlag Heidelberg. Germany.

SEN, A.K.; BAGCHI, A. (1989): Fast Recursive Formulations for Best-First Search That Allow Controlled Use of Memory. *Proceedings IJCAI-89*, 297-302. Detroit.

SHAPIRO, S.C. (1990): Best First Search. *Encyclopedia of Artificial Intelligence*. Ed. S.C. Shapiro. John Wiley & Sons. Canadá.

SHIH, L.C.; ENKAWA, T.; ITOH, K. (1992): An AI-search technique-based layout planning method. *International Journal of Production Research*, 30 (12), 2839-2855.

SHIRAI, Y.; TSUJII, J. (1987): *Inteligencia Artificial. Conceptos, técnicas y aplicaciones*. Ariel. Barcelona.

SIERKSMA, G. (1996): *Linear and integer programming: theory and practice*. Marcel Dekker. New York.

SIRER, S. (1992): Resolució del problema d'equilibrament de línies de producció o muntatge mitjançant la programació dinàmica acotada. *PFC.* DOE. ETSEIB. UPC. Barcelona.

TADEI, R.; GUPTA, J.N.D.; DELLA CROCE, F.; CORTESI, M. (1998): Minimising makespan in the two-machine flow-shop with release times. *Journal of the Operational Research Society*, 49, 77-85.

TAHA, H.A. (1991): *Investigación de Operaciones.* Ra-ma & Alfaomega. México.

TROYA, J.M.; ORTEGA, M. (1989): A study of parallel branch-and-bound algorithms with best-bound-first search. *Parallel Computing*, 11, 121-126.

VALERO, M. (1995): Enseñanza a través de satélite de ingeniería de telecomunicación. *Apuntes de curso.* UPC.

VILA, A.; COVES, A. (1998): Introducción a la propagación de restricciones. *Working paper.* UPC.

WAGNER, H.M. (1975): *Principles of operations research with applications to managerial decisions.* Prentice-Hall. London.

WANG, P.; COLEMAN, N. (1996): Bin-packing. *Encyclopedia of Operations Research and Management Science.* Ed. S.I. Gass & C.M. Harris. Kluwer. Massachusetts.

WASHBURN, A.R. (1998): Branch and Bound Methods for a Search Problem. *Naval Research Logistics*, 45, 243-257.

WHITE, C.C. (1996): Dynamic programming. *Encyclopedia of Operations Research and Management Science.* Ed. by S.I. Gass & C.M. Harris. Kluwer. Massachusetts.

WINSTON, P.H. (1994a): *Inteligencia Artificial.* Addison-Wesley. Delaware, E.U.A.

WINSTON, W.L. (1994b): *Investigación de Operaciones. Aplicaciones y algoritmos.* Iberoamericana. México.

XU, C.; TSCHÖKE, S.; MONIEN, B. (1995): Performance Evaluation of Load Distribution Strategies in Parallel Branch and Bound Computations. *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing. SPDP'95*, 402-405.

YAN, H. (1998): An ILP model with logical constraints for mail sorting facility selection. *Journal of the Operational Research Society*, 49, 273-277.

ZENIOS, S.A. (1994): Parallel and Supercomputing in the Practice of the Management Science. *INTERFACES*, 24, 122-140.

ZHANG, W.; KORF, R.E. (1995): Performance of linear-space search algorithms. *Artificial Intelligence*, 79, 241-292.