Universitat Autònoma de Barcelona
Departament de Ciències de la Computació

Negotiations over Large Agreement Spaces

Author:
Dave de Jonge

Supervisor:
Dr. Carles Sierra

Tutor:
Dr. Jordi González Sabaté

A Dissertation submitted to fulfill the requirements for the degree of PhD in Computer Science.

Bellaterra, April 13, 2015

Elaborated at: Institut d'Investigació en Intel·ligència Artificial
Consejo Superior de Investigaciones Científicas (IIIA-CSIC)

The general who wins a battle makes many calculations in his temple before the battle is fought.

– Sun Tzu, *The Art of War*

Abstract

In this thesis we investigate negotiation algorithms for domains with non-linear utility functions and where the space of possible agreements is so large that the application of exhaustive search is impossible. Furthermore, we explore the relationship between the fields of Automated Negotiations, Game Theory, Electronic Institutions, and Constraint Optimization.

We present three case studies with increasing complexity. Firstly, we introduce an automated negotiator based on Genetic Algorithms, which is applied to a domain where the set of possible agreements is explicitly given as a vector space and, although the utility functions are non-linear, the utility value of any given deal can be calculated quickly by solving a linear equation. Secondly, we introduce a general purpose negotiation algorithm called NB^3, which is based on Branch & Bound. We apply this to a new negotiation test case in which the value of any given deal can only be determined by solving an NP-hard problem. Our third case involves the game of Diplomacy, which is even harder than the previous test cases, because a given deal usually does not entirely fix the agent's possible actions. The utility obtained by an agent thus also depends on the actions it performs after making the deal. Moreover, its utility also depends on the actions chosen by the other agents, so one needs to take Game Theoretical considerations into account. We argue that in this Game Theoretical model there no longer exists a satisfactory definition of a reservation value, unlike the models commonly used in classical bargaining theory.

Furthermore, we argue that negotiations require a mechanism, known as an Electronic Institution, to ensure that agreements are obeyed. One framework for the development of Electronic Institutions is EIDE and we introduce a new extension to EIDE that provides a user interface so that humans can interact within such Electronic Institutions. Moreover, we argue that in the future it should be possible for humans and agents to negotiate which protocols to follow in an Electronic Institution. This could be especially useful for the development of a new kind of social network in which the users can set the rules for their own private communities. Finally, we argue that the EIDE framework is too complicated to be used by average people who do not have the technical skills of a computer scientist. We therefore introduce a new language for the definition of protocols, which is very similar to natural language so that it can be used and understood by anyone.

Resum

En aquesta tesi investiguem algorismes de negociació en dominis amb funcions d'utilitat no linials i en els quals l'espai d'acords possibles és tan gran que l'ús de cerca exhaustiva és inviable. A més, explorem la relació entre les àrees de negociació automàtica, teoria de jocs, institucions electròniques i optimització amb restrictions.

Presentem tres casos d'estudi de complexitat creixent. Primer, proposem un negociador automàtic basat en algorismes genètics i l'apliquem a un domini on el conjunt d'acords possibles es dóna en forma explícita com un espai vectorial i on, encara que les funcions d'utilitat són no linials, el valor d'utilitat de qualsevol acord es pot calcular ràpidament resolent una equació linial. Segon, presentem un algorisme de negociació general anomenat NB^3, basat en la tècnica de Branch & Bound. Apliquem aquest algorisme a un nou cas de prova on el valor d'un acord es pot determinar únicament resolent un problema NP-dur. El tercer cas d'ús és el joc Diplomacy, que és encara més difícil que els casos anteriors ja que un acord no determina completament les accions d'un agent. La utilitat obtinguda per un agent depèn també de les accions triades pels altres agents, de manera que es necessita tenir en consideració aspectes de teoria de jocs. Justifiquem que en aquest model basat en teoria de jocs no existeix una definició satisfactòria del concepte 'valor de reserva', a diferència dels models comunment emprats en la teoria de regateig clàssica.

A més, justifiquem que les negociacions requereixen d'un mecanisme, conegut com a Institució Electrònica, per garantir que els acords siguin respectats. Un entorn per al desenvolupament d'Institucions Electròniques és EIDE i proposem una extensió d'EIDE que proporciona una interfície que permet als humans d'interaccionar en institucions electròniques. També plantegem que en el futur serà possible per als humans i els agents de negociar quins protocols fer servir a una institució electrònica. Això podria ser especialment útil en el desenvvolupament de noves xarxes socials on els usuaris puguin determinar les regles de comportament particulars d'una comunitat privada. Ja que l'entorn EIDE és massa complicat per ser emprat per usuaris normals, sense les capacitats d'un enginyer informàtic, introduïm un nou llenguatge per a la definició de protocols que és similar al llenguatge natural i per tant pot ser usat i entès per qualsevol persona.

Acknowledgments

I would like to thank all my colleagues at the IIIA for five great years working at the institute. Especially I would like to say thanks Carles Sierra for being a great supervisor, to Bruno Rosell and Ismel Brito for their hard work on the implementation of PeerFlow, to Angela Fabregues for her development of the DipGame framework, to Catholijn Jonker and Reyhan Aydogan for a great stay in Delft, as well as to all the other organizers of the ANAC 2014 competition, and to Soledad Valero for letting me stay at her institute in Valencia. Also, I would like to say special thanks to all colleagues I have been working closely with; Matthew Yee-King, Mark d'Inverno, Roberto Confalonieri, Katina Hazelden, Nardine Osman, Lissette Lemus, and Patricia Gutierrez.

Furthermore, I would like to say thanks to the entire world wide Couch Surfing community, especially to those people that have been so kind to host me at their homes, for providing me with company during my travels.

Finally, I would like to thank all friends that I have made in Barcelona and all family and friends that have come to visit me. I would not have been able to do this without their support.

This work was supported by the Agreement Technologies CONSOLIDER project, contract CSD2007-0022 and CHIST-ERA project ACE and EU project 318770 PRAISE.

Contents

List of Figures

Part I

Preliminaries

Chapter 1

Introduction

Multi-Agent Systems is the field of research within Artificial Intelligence that deals with the coordination of systems consisting of more that one agent. An agent is an individual entity capable of autonomous decision making, that makes its decisions with the aim of satisfying its own individual goals. Such goals may be defined in terms of logical sentences which it desires to be satisfied, or in terms of a utility function that maps every possible world state to a real number, or in terms of a (partially) ordered preference relation over the set of possible world states. In this thesis we assume that the goals of the agents are expressed my means of utility functions.

A *multi-agent system* (a MAS) is a set of agents such that for every agent the fulfillment of its goals depends on the actions of other agents in the MAS. Although each agent in a multi-agent system has its own individual goals, it may be the case that every agent in the system has the same goals so that one can expect the agents to cooperate as to bring about their global goal. Therefore, one can make a distinction between two types of multi-agent systems: those in which each agent has its own individual goals, and those in which the agents share a global set of goals. We refer to the first kind as *selfish* systems, and to the second kind as *social* systems.

In this thesis we will investigate how and when agents in a selfish system coordinate their actions. We will show how three subfields of Multi-Agent Systems are related, namely Automated Negotiations, Electronic Institutions, and Game Theory.

1.1 Multi-agent Systems and Game Theory

An essential aspect of multi-agent systems is the fact that the outcome of the actions of one agent may also depend on the actions of other agents. After all, if this would not be the case, then essentially there would be no multi-agent system, but rather just a set of completely independent individual agents. Therefore, when choosing its actions an agent needs to take into account the

actions that have been performed or will be performed by the other agents as well. Game Theory is the mathematical theory that describes such decision problems. Although at first sight Game Theory may seem a theory about playing games, it applies to a much broader class of situations. In fact, it applies to all situations in which one's decisions depend on the decisions of others, with different goals, and hence particularly to any decision problem in a selfish multi-agent system.

In Game Theory it is usually assumed that agents (in this context often referred to as *players*) have an individual utility function that assigns a utility value to each possible outcome of the game. Each agent takes its decisions with the goal of maximizing its own utility. However, if players do not coordinate their actions they may end up in a situation that is bad for everyone. This is exemplified in the well-known Prisoners' Dilemma [Poundstone, 1993].

In the Prisoners Dilemma two prisoners (Alice and Bob) are arrested under suspicion of having committed a crime. Both have the option to either confess the crime or to deny. If one of the two confesses the crime and the other denies, then the one denying is released and the one confessing gets 4 years of imprisonment. However, if both confess, they each get 2 years of imprisonment, and if they both deny they will both get a 3 year sentence. Note that whatever Alice does, Bob is always better off if he denies. The same holds for Alice: whatever Bob does, Alice is always better off if she denies. Therefore, the result is that both will deny, and both will end up in prison for 3 years. Paradoxically however, this is for both a suboptimal outcome because if they had some way to force each other to confess, they would only go to prison for 2 years.

The key here lies in the word *force*. The players prefer the outcome (confess, confess) not because they both like to confess, but rather because they both like the *opponent* to confess. Therefore, although, each player prefers to deny, each one would be happy to give up his or her freedom to deny, if in return he or she could force the opponent to confess. In the Prisoner's Dilemma however, this is not possible because it does not model the option of players exchanging power. In order to overcome this shortcoming, we can modify the game by allowing the players to make binding agreements. This means that we need to add the following two concepts:

1. **Negotiation**: a mechanism to communicate and come to an agreement about the mutual restrictions of the respective players.

2. An **Electronic Institution**: a mechanism that ensures that the agreements are indeed obeyed.

We argue therefore, that these two mechanisms are essential in any real multi-agent system in order prevent undesired outcomes. In this thesis we look at the relationship between these two concepts.

Of course, the Prisoner's Dilemma is an extremely simple example. In a more realistic example the set of possible agreements the players can make could be extremely large. Therefore, a third concept needs to be taken into account:

3. **Search**: an algorithm that analyzes the game in order to find solutions that are beneficial to the players, but that require coordination.

Throughout this thesis we will assume that that negotiations take place over domains with many agents, an intractably large space of possible agreements, complex non-linear utility functions and limited time. Furthermore, the agents are assumed to be selfish and unknown, so one cannot rely on the existence of an impartial mediator.

1.2 Automated Negotiations

As explained above, a MAS requires negotiation between the agents in order for them to achieve desirable outcomes. The research area of Automated Negotiations deals with this problem. Roughly speaking one can say that if a game G without negotiations has no Nash equilibria that are Pareto optimal, then modifying the game to allow for negotiations results in a new game G' that does have a Pareto optimal Nash Equilibrium.

In Automated Negotiations it is assumed that there exists some predefined set of agreements that the agents can make with each other. We call this set the *agreement space*. Agents can propose agreements from this space to each other and can accept or reject proposals made by others. If a proposal is accepted by each of the agents involved, it means that each of these agents has committed itself to execute a number of actions defined by the proposal.

Each agent proposes and accepts agreements with the goal of maximizing its own utility. However, the other agents involved in the deal will only accept to take part in it if doing so also yields increased utility to them. Therefore, a good negotiator needs to find a balance: on the one hand the negotiator should propose deals that yield as much utility as possible to himself, but on the other hand he should also make sure that the proposal is beneficial enough to the others to make them accept it. Nevertheless, we stress the fact that a negotiator is not really interested in optimizing the other agents' utilities. A selfish negotiator is only willing to help the other agents increasing their utilities if that indirectly leads to a higher utility for itself.

1.2.1 Basic Concepts

In the simplest, classical, model of negotiations, there are two negotiators, α_1 and α_2 that can propose deals from a given agreement space, which is a finite set denoted Agr. Both agents have a utility function (denoted f_1 and f_2 respectively) that maps each element of the agreement space to a real number. Each agent only knows its own utility function, and keeps it secret to the other. The negotiation has a given deadline d. If the agents do not come to an agreement before the deadline, they will both receive an amount of utility which is referred to as the *reservation value*. Each agent may have its own reservation value, but often it is simply defined to be zero for every agent. Obviously, an agent only has the

incentive to make an agreement, if that agreement yields a higher utility value than its reservation value.

Note that the existence of a deadline is important, because otherwise each agent could indefinitely insist on the deal that yields the most utility for itself without ever making any concessions. Alternatively, a negotiation scenario may also define a *discount factor* δ between 0 and 1 that causes the utility for the agents to decrease as time passes. In that case each negotiator α_i receives a discounted utility, calculated as $\delta^t \cdot f_i(x)$ where t is the time at which the agreement x was made.

Furthermore, an important aspect of a negotiation scenario is the *protocol* that prescribes how the agents can make proposals and when an agreement is considered binding. The most common protocol is the *alternating offers* protocol, in which agent α_1 makes a proposal, and then agent α_2 may either accept the proposal or make a counter proposal. If α_2 decides to make a counter proposal then again α_1 can either accept that proposal or make a new proposal, etcetera. The negotiations stop as soon as one of the agents accepts the previously proposed deal, or when the deadline passes.

1.2.2 Negotiation Domains

Many variations to the above described model have been studied however. We can distinguish several parameters that define the characteristics of a negotiation scenario, namely:

- The number of agents negotiating.
- The number of agents that can be involved in a deal.
- The complexity of the utility functions.
- The size of the agreement space.
- The agents' knowledge about their opponents' utility functions.
- The agents' knowledge about their own utility functions.

Negotiations in which only two agents are involved are called *bilateral* negotiations, while negotiations with more than two agents are called *multilateral* negotiations. In case of multilateral negotiations one often still assumes that the deals themselves are bilateral. This means the negotiations in fact consist of a number of bilateral negotiations taking place in parallel. However, as we will see later on in this thesis, one can also allow multilateral negotiations with multilateral deals: deals in which more than two agents make commitments.

Furthermore the complexity of the utility functions varies among several studies. In the simplest model each agent simply has a table that maps every possible agreement to its corresponding utility value. A bit more complicated is the case when there is a set of variables (often called *issues*) and the agents negotiate which value must be assigned to each variable [Baarslag et al., 2010].

Each agent has a utility table for each variable, and the total utility of an agreement is then given as a linear combination of the utility values for each variable. More complex models assume non-linear functions that are described in terms of 'constraints' [Marsa-Maestre et al., 2009a, Ito et al., 2008]. Each constraint assigns a value to a subset of the agreement space. These subsets may overlap, and the utility of a deal is given by the sum of the values of all constraints that are satisfied by that deal.

In Chapter 4 we will see a scenario where the utility functions are more complex because the utility of a deal can only be determined by solving an NP-hard combinatorial problem. Finally, this can be made even more complex by introducing Game Theoretical reasoning in order to determine the value of a deal. That is: players make agreements about which moves to make in a certain game, but without entirely fixing their possibilities. Determining the value of such a deal requires determining the values of the game matrix, as well as determining the Nash-equilibrium of the game restricted by the agreements made. An example of such a domain is the game of Diplomacy, which we will also see in Chapter 4. To summarize, we can distinguish the following types of utility functions in the literature, in order of increasing complexity:

1. Given by a table.
2. Linear combination of values given by tables.
3. Non-linear over agreement space, but linear over constraint space.
4. NP-hard problem.
5. NP-hard problem, plus Game Theoretical considerations.

Classical negotiation scenarios have often involved only small agreement spaces, so it was assumed that the utility value for every deal in the agreement space was directly given. However, in this thesis we are only interested in agreement spaces that are so large that calculating every utility value is infeasible (e.g. 10^{100} possible deals). This means that any negotiation strategy needs to be combined with some intelligent search algorithm.

Another parameter that can be adjusted is the amount of knowledge that agents have about their opponents' utility functions. In some models the utility functions of all negotiators are assumed to be generally known [Nash, 1950], while in other studies agents may only know the preference order of their opponents between the several possible deals, but not their utility values. In yet other studies the agents know absolutely nothing about their opponents' utilities or preferences [Baarslag et al., 2010]. We think the most realistic model assumes agents have some knowledge about the opponents' utility functions, but not full knowledge. For example, when a client negotiates with a car salesman about the price of a car, the client probably does not know the minimum price that the salesman is willing to ask, but the client normally does assume a certain window of possible prices that he or she considers realistic. More importantly, the client

does know that the goal of the salesman is to sell the car for the highest price as possible, and with as much extras as possible.

Furthermore, in some cases a negotiator may not even have an expression of its own utility function (e.g. in the ANAC'14 competition, see Section 4.1). Instead, it can apply a sort of 'oracle' that returns the utility of a deal when requested by the agent. In the real world one can imagine such a system if a software agent negotiates on behalf of its human owner. For any given deal the agent may ask its owner how much the owner valuates that deal, and the agent can then use this information to build a model of its owner's preferences which it then uses to estimate the owner's valuation of other deals.

Other parameters that may vary among studies are the types of deadlines and discount factors, and the knowledge of the agents about them. For example, agents can have individual deadlines, or all agents can have the same deadline, or there may not be a deadline at all. Furthermore, they may or may not know the opponent's deadlines and discount factors.

1.2.3 Strategy

Many negotiation strategies have been studied, but most of them are based on the same basic time-based concession strategy. The agent starts proposing the most selfish offer: the one that gives the highest utility to itself, and then, in consecutive rounds make proposals that give more and more utility to the opponent. The main question then is how fast one should concede. A quickly conceding agent is more likely to make an offer that is satisfactory to the opponent before the deadline, and is therefore less likely to fail. However, conceding too quickly may cause the agent to give up more utility then necessary and therefore end up with a less profitable outcome than it would have achieved if it had conceded less quickly. Many variations to this basic strategy can be made depending on the details of the domain. If the opponent's utility function is unknown for example one can try to estimate the opponent's utility of a certain deal by comparing it with proposals previously made by the opponent and applying some similarity measure as proposed in [Faratin et al., 2000]. Also, one can use statistical methods to predict the concession speed of the opponent and use this to find the optimal counter strategy [Williams et al., 2011].

1.2.4 Relation to other Fields

Maximizing a utility function for a set of independent agents is also the goal of Distributed Constraint Optimization Problems (DCOP), but these problems are fundamentally different from negotiation problems, because DCOPs apply to social multi-agent systems. They assume there is only one global function to be optimized and the agents cooperate with the joint goal of finding the solution that maximizes this global utility function [Modi et al., 2005]. Therefore, DCOP algorithms cannot be applied to cases where agents have individual utility functions.

A field closely related to automated negotiations is the field of Cooperative Game Theory [Osborne and Rubinstein, 1994]. In Cooperative Game Theory one assumes that utility is assigned to coalitions of agents and that agents within such a coalition can freely divide the utility between one another. Such a division of utility is called an *allocation* and the set of allocations that keeps the coalition stable is called the *core*. The notion of an allocation in cooperative game theory can be compared to the notion of a deal in automated negotiations, and the notion of a coalition can be compared to a set of agents that together agree on a certain deal. The difference however is that cooperative game theory is mainly concerned with the question of whether the core and other solution concepts exist, while Automated Negotiations focuses more on *how* agents decide to agree on a certain allocation.

We should note however that the term 'cooperative' in Cooperative Game Theory can be a bit misleading, because the agents are still assumed to be purely selfish. The word 'cooperative' means that the agents have the possibility to cooperate, that is, they are able to communicate and form coalitions, but just like in the field of Automated Negotiations, they only do so to satisfy their own selfish goals.

1.3 Search Algorithms

Recently, more attention has been given to negotiations where the size of the agreement space is very large. This means that negotiation algorithms must be combined with an intelligent search algorithm to explore the agreement space efficiently and determine which possible proposals are good enough to propose. The research topic of search algorithms is a vast one, so it would be impossible to give a comprehensive overview. Instead, in this section we will only give a short description of those search algorithms that we will encounter in the rest of this thesis, namely: Branch & Bound, And/Or Search, and Genetic Algorithms.

It is important to note however, that these search algorithms were originally developed for Constraint Optimization Problems, that is: they were developed for domains with a single utility value to maximize. This means that they cannot be applied to negotiations directly, because in negotiations one needs to take the utility of the opponent into account as well as your own utility. The rest of this thesis is therefore largely dedicated to the question of how to combine these search algorithms with negotiation.

1.3.1 Constraint Optimization

Let $\mathcal{X} = \{x_1, x_2, \ldots x_m\}$ be a finite set of logical symbols called variables. For each variable x_i there is a set X_i that we call the *domain* of x_i. A logical expression of the form $x_i = v_{i,j}$, where $v_{i,j}$ is an element of X_i, is called a *variable assignment*. To simplify notation we will here assume that all domains are always equal and consist of the integers 1 to k: $X_i = \{1, 2, \ldots k\}$. In general,

however, domains may consist of different values, and may have a different size for every variable.

Definition 1 *A **solution** is a set of variable assignments, consisting of exactly one variable assignment for each variable.*

For example: $\{x_1 = 2, x_2 = 4, \ldots, x_m = 1\}$. Solutions can be denoted more compactly in vector notation: $x = (x_1, x_2, \ldots, x_m) = (2, 4, \ldots, 1)$. Therefore, we can identify the space of all possible solutions with the Cartesian product of the domains: $X = X_1 \times X_2 \times \ldots X_m$.

Definition 2 *A **partial solution** is a set of variable assignments consisting of zero or one variable assignments for each variable.*

For example: $\{x_2 = 4, x_4 = 3\}$. Partial solutions can be identified with subsets of X. For example, the partial solution $\{x_2 = 4, x_4 = 3\}$ corresponds to the subset $\{(x_1, x_2, \ldots, x_m) \in X \mid x_2 = 4 \wedge x_4 = 3\}$.

Definition 3 *A **constraint** c is a pair (s_c, v_c) where $s_c \subset X$ and $v_c \in \mathbb{R} \cup \{-\infty\}$. We say c is a **soft constraint** if $v_c \in \mathbb{R}$ and c is a **hard constraint** if $v_c = -\infty$.*

Definition 4 *The **characteristic function** f^c of a constraint c is defined as:*

$$f^c(x) = \begin{cases} v_c & \text{if } x \in s_c \\ 0 & \text{if } x \notin s_c \end{cases} \tag{1.1}$$

If $v_c > 0$ then we say that x *satisfies* c iff x is an element of s_c. If $v_c \leq 0$ then we say that x *satisfies* c iff x is *not* an element of s_c.

Definition 5 *An **optimization function** is a function f from X to $\mathbb{R} \cup \{-\infty\}$.*

If the set of variables is finite, and for every variable its domain is a finite set, then any such optimization functions can be expressed in terms of a finite set of constraints C as follows:

$$f(x) = \sum_{c \in C} f^c(x) \tag{1.2}$$

However, it is very important to note that in real-world optimization problems the optimization function may be expressed in an entirely different form, and although it is theoretically possible to cast it into the form of Equations 1.1 and 1.2, this may often turn out to be a practically infeasible task, especially if the solution space is very large.

A constraint optimization problem is a tuple (X, f). An *optimal solution* x^* of a constraint optimization problem is a solution such that for every $x \in X$ we have $f(x^*) \geq f(x)$. The research area of Constraint Optimization is concerned with inventing search algorithms that can find optimal, or near optimal solutions of constraint optimization problems. For a comprehensive overview of the topic of Constraint Satisfaction and Constraint Optimization, we refer to [Rossi et al., 2006].

1.3.2 Exhaustive Search

The most naive Constraint Optimization algorithm is to simply calculate the value $f(x)$ of each possible solution in the solution space X and return the solution for which $f(x)$ is highest. This is known as an *exhaustive search*. However, this is often infeasible since the size of the space X is exponential in the number of variables. If there are m variables and each variable has a domain of size k, then the size of X is k^m. Unless m and k are very small, calculating $f(x)$ for every solution x in X takes too much time.

1.3.3 Branch & Bound

A more efficient approach to a constraint optimization problem is to apply a Branch & Bound algorithm (B&B). A Branch & Bound algorithm iteratively explores partial solutions of X by generating a tree structure in which every node represents a partial solution. Specifically, the root node represents the empty set, and every child of the root node represents the assignment of a different value to x_1. More generally, every node nd with depth d represents a partial solution that assigns values to the variables $x_1, x_2, \ldots, x_d$. If the domain X_{d+1} has size k, then, the algorithm adds k child nodes to nd, each representing a different extension of the partial solution represented by the node nd.

Every time a new node is added, the algorithm calculates two values for that node: the *upper bound* $ub(Y)$ and the *lower bound* $lb(Y)$ (here, Y denotes the subspace of X corresponding to the partial solution represented by the newly added node). These bounds can be defined in various ways, but they should always satisfy the following two inequalities:

$$ub(Y) \geq max\{f(x) \mid x \in Y\}$$

$$lb(Y) \leq min\{f(x) \mid x \in Y\}$$

The trick is now that if we have two nodes nd_1, nd_2 representing two subsets Y_1, Y_2 such that $ub(Y_1) < lb(Y_2)$ then we are sure that the optimal solution will not be in the subset Y_1. Therefore, when exploring the search tree we can *prune* node nd_1, meaning that no more children will be added to it. This can highly reduce the search space since none of the solutions inside Y_1 will ever need to be evaluated.

One interesting aspect of B&B is that it is an anytime algorithm: if the algorithm is stopped before the optimal solution has been found, it may still return solutions that are near optimal. The longer the algorithm runs, the better solutions it finds. Moreover, the algorithm provides bounds on the value of the optimal solution. Therefore one knows how far from optimal a given solution can maximally be.

In case we have an expression of f in the form of Equations 1.1 and 1.2 and all constraints have non-negative values, then such bounds can be defined in a straightforward manner:

$$ub(Y) = \sum_{c \in C_Y} v_c \quad \text{where } C_Y = \{c \in C \mid s_c \cap Y \neq \varnothing\}$$

$$lb(Y) = \sum_{c \in C'_Y} v_c \quad \text{where } C'_Y = \{c \in C \mid s_c \subset Y\}.$$

Unfortunately, there is not always an efficient way to accurately calculate these bounds, so B&B is not always applicable. Moreover, the efficiency of this approach depends highly on many details, such as:

- The order of the variables.
- The way in which the utility function and the constraints are expressed.
- The accuracy of the lower- and upper- bounds.

For more details about B&B algorithms we refer to [Lawler and Wood, 1966, Papadimitriou, 1994].

1.3.4 Exploiting Independence

We will now look at a technique that can be applied in combination with almost any search algorithm to increase its efficiency. It is based on the principle of 'divide and conquer': split a problem into several smaller subproblems and solve each of them independently.

Let us first look at a simple example. We assume there are four variables: $\{x_1, x_2, x_3, x_4\}$ and an optimization function f. Now, if it happens that this function f can be written as the sum of two functions f_1, f_2 as follows:

$$f(x_1, x_2, x_3, x_4) = f_1(x_1, x_2) + f_2(x_3, x_4) \tag{1.3}$$

then, instead of applying a search algorithm (such as B&B or exhaustive search) to the entire search space X, we can split the problem into two separate problems. The first problem is to maximize the function f_1, and the second problem is to maximize the function f_2. It is easy to see that if (x_1^*, x_2^*) maximizes f_1 and (x_3^*, x_4^*) maximizes f_2, then the optimal solution for f must be $(x_1^*, x_2^*, x_3^*, x_4^*)$, so indeed we have solved the entire problem by solving the two subproblems.

Since each of these subproblems only involves 2 variables, finding the optimal solution using exhaustive search requires evaluating k^2 solutions, so to solve both subproblems one only needs to explore $2k^2$ solutions, which is much less then the k^4 possible solutions one would need to explore when applying exhaustive search to the entire problem at once. Similarly, one can expect Branch & Bound or any other search algorithm to be much faster when applied to the two subproblems separately.

The key insight here, is that the optimal values for x_1 and x_2 can be determined without knowing the values of x_3 and x_4, and vice versa. In order to investigate how the various variables depend on one another one can draw a dependency graph. That is: a graph in which each node represents a variable and in which an arc between two nodes represent that their variables depend upon each other. Each connected component of that graph then represents a subproblem that can be solved independently of the others. Let us define this more precisely.

Definition 6 *A function g is called **monotonic** iff:*

$$\forall\, i : x_i \leq x_i' \Rightarrow g(x_1, \ldots x_i, \ldots x_m) \leq g(x_1, \ldots x_i', \ldots x_m)$$

Definition 7 *If a function f can be expressed as:*

$$f(x_1, \ldots x_m) = g(f_1(x_{i_1} ..., x_{i_k}), f_2(x_{j_1}, ..., x_{j_l}), ..., f_n(...)) \tag{1.4}$$

*where g is a monotonic function, then we say two variables x_i and x_j **directly depend** on each other if there is a function f_j in this expression that depends on both variables.*

Note that addition is an example of a monotonic function, so the example of Equation 1.3 is indeed of this form. We see that x_1 and x_2 in this example directly depend on each other, and that x_3 and x_4 directly depend on each other.

Definition 8 *A **Dependency Graph** for a function f is an undirected graph in which each node corresponds to a variable of f, and two nodes are adjacent iff their corresponding variables directly depend on each other.*

Definition 9 *Given a dependency graph of a function f, two of its variables x_i and x_j **indirectly depend** on each other iff there exists a path between x_i and x_j in the dependency graph.*

Definition 10 *Given a dependency graph of a function f, two of its variables x_i and x_j are **independent** iff there does not exist any path between x_i and x_j in the dependency graph.*

Note that if there exist one or more pairs of independent variables, it means that the dependency graph has multiple connected components. Whenever a dependency graph has multiple connected components each connected component represents a subproblem and the entire problem can be solved by solving each subproblem separately. Of course, it is not always practically possible to find an explicit expression in the form of Equation 1.4. However, one may still be able to reason that certain variables must be independent, even without such an explicit expression.

1.3.5 And/Or Tree Search

As we have seen above, if we know that some variables of the optimization function are independent then we can exploit this by dividing the problem into smaller subproblems. We will now show however, that we can take this a step further and apply a similar procedure even if two variables are indirectly dependent, as long as they are not directly dependent.

Again, we begin with an example. Assume we have the following optimization function:

$$f(x_1, x_2, x_3) = f_1(x_1, x_2) + f(x_1, x_3) \tag{1.5}$$

Note that all variables depend indirectly on each other, but that x_2 and x_3 do not depend directly on each other. Now, let us also assume that we already know the optimal value of x_1. We can then find the value for x_2 that maximizes f_1 by exhaustively exploring all values of X_2, and we can separately find the optimal value for x_3 that maximizes f_2 by exhaustively exploring X_3. This means that if we know x_1^* we only need to explore $|X_2| + |X_3| = 2k$ values to find the optimal solution. Of course, the optimal value of x_1 will usually not be given. However, this is not a problem because we can simply repeat the procedure for each possible value of x_1, which would require a total of $|X_1| \cdot (|X_2| + |X_3|) = 2k^2$ evaluations, which is still less then the k^3 solutions we would have to explore when using exhaustive search. The key insight here, is that although x_2 and x_3 are not independent, they *become* independent once we fix a value for x_1.

Definition 11 *If $\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3$ are three disjoint sets of variables, then we say $\mathcal{Y}_2$ and $\mathcal{Y}_3$ are **independent given** $\mathcal{Y}_1$ iff every path from any variable in $\mathcal{Y}_2$ to any variable in $\mathcal{Y}_3$ passes at least one variable in $\mathcal{Y}_1$.*

This definition means that once all variables in $\mathcal{Y}_1$ have been assigned a value, the problem splits into two subproblems, involving $\mathcal{Y}_2$ and $\mathcal{Y}_3$ respectively. As a special case, note that if x_i and x_j do not directly depend on each other, this means that $\{x_i\}$ and $\{x_j\}$ are independent given $\mathcal{X} \setminus \{x_i, x_j\}$.

An And/Or tree search [Dechter and Mateescu, 2007] is a tree search algorithm that exploits the graph structure as illustrated by this example. If we compare And/Or search with ordinary tree search, then ordinary tree search may also be called Or-search, and all nodes in an ordinary search tree may be called Or-nodes, because for each node its corresponding partial solution can be extended by picking exactly one of its child nodes.[1] In And/Or search, on the other hand, there are two types of nodes: And-nodes and Or-nodes. Just like in an ordinary search the Or-nodes represent partial solutions, while And-nodes represent subproblems. If two clusters $\mathcal{Y}_2, \mathcal{Y}_3$ are independent given $\mathcal{Y}_1$, then the tree first explores the variables in $\mathcal{Y}_1$, according to an ordinary tree search, but for each node that represents a full assignment of $\mathcal{Y}_1$, its children will then be And-nodes, one for each of the clusters $\mathcal{Y}_2, \mathcal{Y}_3$. Underneath each And-node representing a cluster $\mathcal{Y}_i$, the search will continue, but in that subtree the nodes will only involve the variables in $\mathcal{Y}_i$. Hence, a subtree under the And-node representing $\mathcal{Y}_i$ can be seen as the search tree that solves the subproblem consisting of the variables of $\mathcal{Y}_i$. This subtree in turn may also be an And/Or-tree, if the cluster $\mathcal{Y}_i$ itself can be partitioned into clusters $\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3$ such that $\mathcal{Z}_2$ and $\mathcal{Z}_3$ are independent given $\mathcal{Y}_1 \cup \mathcal{Z}_1$.

Furthermore, while in an ordinary search tree a solution is represented by a linear branch from the root to a single leaf node, a solution in an And/Or tree is given by a tree itself, that branches at every node for which its children are And-nodes. Therefore, if the children of a node *nd* are And-nodes, the partial solution represented by *nd* is extended by adding *all* children to the solution (hence the name 'And-node'). However, just as in an Or-tree, a full solution still

[1] One could argue that 'XOR-search' would actually be a better name

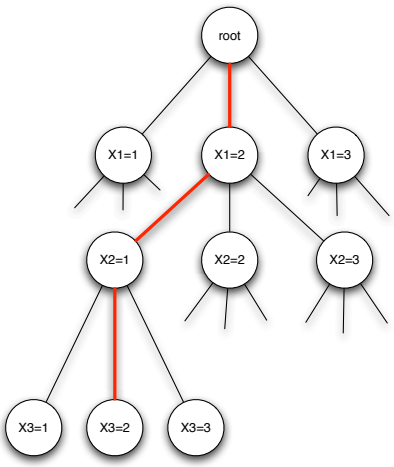


Figure 1.1: An ordinary search tree. The solution $\{x_1 = 2, x_2 = 1, x_3 = 2\}$ is represented by the branch colored red.

contains exactly one Or-node for each variable. Figures 1.1 and 1.2 respectively show an Or-tree and an And/Or tree for the example of Equation 1.5.

Again, we should stress that this technique only works if it is possible to determine a dependency graph, but one may be able to do so if one can reason that some variables are independent even without having an explicit expression of the optimization function in the form of Equation 1.4. Furthermore, it is worth noting that And/Or search automatically exploits total independence between variables. After all, if two variables are independent, we can also say that they are independent *given the empty set*. Therefore, And/Or search should be seen as a refinement of the technique described in Section 1.3.4. Finally, it is important to note that And/Or search can be combined with B&B so one can profit from the advantages of both techniques.

1.3.6 Genetic Algorithms

An entirely different type of optimization algorithm is a so-called Genetic Algorithm (GA). This is a search technique inspired on the principle of natural selection. It does not explore the solution space in a systematic way, as tree search algorithms do, but instead takes random samples. The idea is then that bad samples are discarded, while good samples are combined with each other, yielding even better solutions. Many variants of Genetic Algorithms exist, but they are all based on the same principles. A standard implementation works as follows:

1. **Initial Population:** Randomly pick a number of solutions (say 100) from

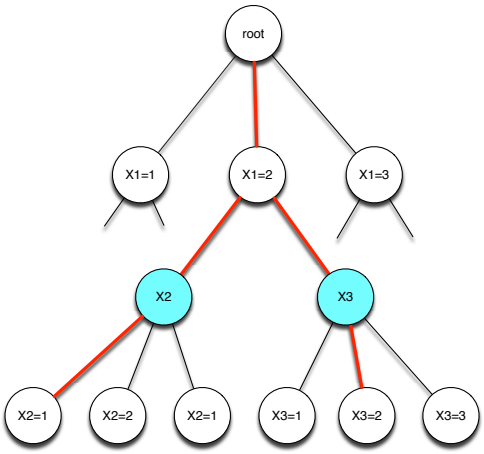


Figure 1.2: An And/Or search tree. The solution $\{x_1 = 2, x_2 = 1, x_3 = 2\}$ is represented by the tree colored red. The solution tree splits every time the children of a node are And-nodes (colored blue).

the solution space. This is the initial population.

2. **Selection:** Pick the solutions with the highest value $f(x)$ out of this the population (e.g. the 10 solutions with highest value). These are called the *survivors*. All others are discarded.

3. **Mutation** For some of these survivors, randomly change some of their assignments. (e.g. the vector (v_1, v_2, v_3) is replaced by (v_1, v_2', v_3))

4. **Cross-over** For each pair of vectors x, y from the survivors create two new vectors x' and y' according to a principle called *cross-over* (explained below).

5. **New population** The survivors and the newly created vectors are put together, this set is then the new population.

Steps 2-5 are repeated until some stopping condition is met.

Cross-over is a procedure that takes two solutions x, y as input and creates two new solutions x', y' by interchanging variable-assignments. A common way to implement this, is as follows:

$$x_i' = x_i \text{ if } i \leq j, \quad x_i' = y_i \text{ otherwise}$$

$$y_i' = y_i \text{ if } i \leq j, \quad y_i' = x_i \text{ otherwise}$$

for some randomly chosen value of j. In this case the solutions x and y are 'cut' into two halves at the j-th position, then interchanged, and 'glued' together

again. In Chapter 5 we use another type of cross-over however, that randomly chooses for each position whether the values should be interchanged or not.

$$x'_i = x_i \text{ and } y'_i = y_i \quad \text{with 50\% probability,}$$

or:

$$x'_i = y_i \text{ and } y'_i = x_i \quad \text{with 50\% probability.}$$

Either way, the idea behind this is the assumption that if x satisfies one constraint and y satisfies another constraint, then cross-over may combine these solutions and yield a new solution that satisfies both constraints.

Let us take a look at another numerical example. Given two constraints and two randomly picked solutions we compare the probability that one of the two solutions will satisfy both constraints, with the probability that one of the solutions will satisfy one constraint and the other solution will satisfy the other constraint, and that cross-over will combine them.

We assume that there are two constraints c_1 and c_2. The subspace of c_1 is defined by $x_1 = 1$, while the subspace of c_2 is defined by $x_2 = 1$. If we pick a random solution x, then the probability p_1 that it will satisfy c_1 is k^{-1}. The same holds for the probability p_2 that x satisfies c_2. Therefore, if we pick s random solutions then the probability that at least one of them will satisfy both constraints is approximately[2]:

$$1 - (1 - p_1 p_2)^n \approx s p_1 p_2 = s k^{-2}$$

The probability that one or more of the s samples satisfy c_1 and that one or more samples satisfy c_2 is approximately[3]:

$$(1 - (1 - p_1)^n) \cdot (1 - (1 - p_2)^s) \approx s p_1 \cdot s p_2 = s^2 k^{-2}$$

Which means that the probability that both of the constraints are satisfied by some sample, but none of the samples satisfies both, is:

$$s^2 k^{-2} - s k^{-2}$$

Now we note that if a sample x satisfies c_1 but not c_2, and a sample y satisfies c_2 but not c_1, then the probability that after cross-over either x' or y' will satisfy both constraints is 0.5. (we can simply define x' to satisfy c_1, so we only need to know the probability that x'_1 will also inherit c_2, which is indeed 0.5). Therefore, if we assume that whenever both constraints are satisfied by some sample in the population both constraints will also be satisfied by some sample in the set of survivors, then we can compare the probabilities of finding a solution that satisfies both constraints with and without cross-over:

$$P \approx \begin{cases} s k^{-2} & \text{without cross-over} \\ s k^{-2} + 0.5 \cdot (s^2 k^{-2} - s k^{-2}) = \frac{1}{2}(s^2 + s) k^{-2} & \text{with cross-over} \end{cases}$$

[2] if $s << k^2$
[3] if $s << k$

We see that the cross-over increases the probability of finding a good solution by a factor of $\frac{1}{2}(1+s)$, with respect to random sampling without cross-over.

An important aspect of GAs is that, unlike And/Or-search or B&B, their application does not require any knowledge about the constraints of the specific problem instance. The only requirement is that you have some representation of the solution space as a vector space, and that you have some function to calculate the value $f(x)$ of any solution x. This can be seen both as an advantage and as a disadvantage. It is an advantage because it means that one can apply GAs even in the absence of instance-specific information. However, it also means that if one does have more specific information, a GA will not be able to exploit that information.

However, we should note that we have looked only at constraints that involve a single variable. The probability of successful cross-over becomes smaller as the number of involved variables becomes larger. To be precise: if we have two constraints, one involving d_1 variables, and one involving d_2 variables, then the probability that either x' or y' will satisfy both constraints is: $0.5^{d_1+d_2-1}$. Therefore this technique will be less successful if the constraints involve many variables. Finally, this technique only works if the several constraints can indeed be combined (i.e. the subspaces corresponding to these constraints have non-empty intersection). For more details on Genetic Algorithms we refer to [Schmitt, 2001, Falkenauer, 1998].

1.3.7 Summary

We have described several algorithms and techniques to determine the solution x in a space X that maximizes a function f. If X is very large an exhaustive search is infeasible, as it would take too much time. If one has a way of determining an upper- bound and a lower- bound for the function f for any partial solution then one can make use of Branch & Bound. If one can determine that certain variables are independent, one can exploit this fact, by dividing the problem into several subproblems and solve each subproblem independently by using any search algorithm. Even more, if one can determine that certain variables do not directly depend on each other, then this can be exploited by applying And/Or-search. And/Or search and B&B can be combined to profit from the advantages of both techniques. Finally, in case one does not have any specific knowledge about the problem instance, one can apply a Genetic Algorithm, which is efficient in case the combination of good partial solutions may yield even better solutions and the constraints involve only a few variables.

1.4 Enforcement of Agreements

When the agents in a MAS make agreements about the respective actions they will perform, they require a mechanism that ensures that these agreements are indeed obeyed. After all, if an agreement is not enforced, it has no formal meaning. In the prisoner's dilemma for example, even if Alice and Bob agree to

both play 'confess', if Alice is not forced to obey this agreement she would still be better off playing 'deny', no matter what Bob does, and likewise for Bob. Therefore, agreements do not have any effect, unless the agents can be forced to obey them.

1.4.1 Regimentation vs. Punishment

The literature distinguishes between two kinds of rule enforcement: *regimentation* and *punishment*. Regimentation means that the agents are placed in an environment in which it is impossible to break the rules. An example of regimentation in real life would be the rule that you cannot make a bank transfer if you have no money in your bank account. This rule is usually impossible to break, for example because the website of the bank is implemented such that the 'submit' button on the page is disabled if you do not have enough money in your account.

In the case of rule enforcement by punishment, on the other hand, agents still have the ability to break the rules, but they are being punished if they do so. This is in fact how most rules in our every day lives are enforced. One can for example jump a red traffic light, but if you do so (and you get caught) you need to pay a fine. Rules that can be broken, but that come with a punishment are also called *norms*.

When we look at this from the perspective of Game Theory, we note that punishment is not possible in a one-shot game, because in a one-shot game, by definition, it is assumed that the players will not interact anymore after the game is over. The Prisoner's Dilemma is an example of a one-shot game. It is implicitly assumed that the actions taken by the players have no consequences to the players after the game. This is of course an unrealistic situation: in real life, if Alice confesses but Bob denies, Alice may later take revenge on Bob. Therefore, a more realistic way of describing real world situations in terms of Game Theory, is by means of games with multiple rounds (possibly even infinite).

An example of punishment in Game Theory arises for example in the iterated Prisoner's Dilemma. This is a game with multiple rounds, in which in each round the players play an instance of the Prisoner's Dilemma. A well-known strategy for this game is the tit-for-tat strategy. Playing this strategy means that a player plays 'confess' in the first round and repeat this every round, except when the opponent has played 'deny' in the previous round. Note that this strategy can indeed be interpreted as a form of punishment: the player that first plays 'deny' is being punished by the other player in the next round, by also playing 'deny'.

The main disadvantage of punishment however, is that it is very hard to implement in an electronic system. First of all, one needs to detect the fact that one has infringed the agreements. Second, one may need to establish for each possible infringement what its corresponding punishment is. This can be a hard task because one needs to be sure that the punishment is severe enough in order to have effect. Moreover, one may need to impose meta-rules; for example, one needs to determine what happens if the punisher fails to impose the punishment, or when it punishes incorrectly.

On the other hand, in human societies regimentation is difficult to implement because it is virtually impossible to literally prevent people from misbehaving. One can normally only do so by punishment. But in electronic societies, where the actions of agents have a purely electronic nature, it is much easier to regiment the behavior of the agents, including agents that are actually human, but operate through a graphic user interface. For this reason, the rest of this thesis will purely focus on norm enforcement by regimentation.

1.4.2 Electronic Institutions

A framework that implements the enforcement of rules is called an *Electronic Institution* (EI). Just like in a human institution, an EI allows participants to come together and interact, but imposes a pre-defined protocol on their interactions. It ensures that certain rules are enforced upon its participants and thus prevents them from misbehaving.

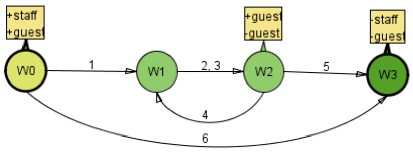
A commonly cited example is that of a fish market, with buyers and sellers that come together with the goal of buying and selling fish. Buyers and sellers make bids according to some auction protocol in order to determine who sells which lot of fish to whom and for which price. It is essential for all participants in such a market place that trade takes place fairly, meaning that every participant has access to the same information, and that everybody is treated on an equal basis. In order to maintain such a fair trading ground and prevent disputes over the rightful winner of an auction, one needs to set rules, and one needs to enforce those rules. Therefore, sellers and buyers have agreed on precise protocols under which the fish auctions need to take place, that guarantee fairness and that leaves no doubt about the winner of the auction. Such rules may state for example that participants are required to register before entering the market or, upon winning an auction, to make their payments directly.

Beyond this fish market example, many other institutions have similar sets of regulations that can be identified. One can think for example of hotels, universities, and governmental agencies. Electronic Institutions are related to negotiations in two ways: firstly they can be used to implement the negotiation protocol, and secondly they can be used to enforce that the agreements the agents made during the negotiations are indeed obeyed.

Electronic Institutions have been subject of extensive research and a large number of such frameworks have been implemented. Examples of such frameworks are ANTE [Cardoso et al., 2013], MANET [Tampitsikas et al., 2012], S-MOISE+ [Hübner et al., 2006], and EIDE [Esteva et al., 2008]. A comparative study of some of those systems has been made in [Fornara et al., 2013]. In the rest of this thesis we will focus on the EIDE framework.

1.4.3 Electronic Institution Development Environment

The Electronic Institution Development Environment (EIDE) has been under development for more than 15 years [Arcos et al., 2005, d'Inverno et al., 2012, Noriega, 1997, Esteva, 2003] which has resulted in a large framework consisting of



1. (request (?x guest) (?y staff) login(?user ?email)))

2. (inform (!y staff) (!x guest) accept()))

3. (failure (!y staff) (!x guest) deny(?code)))

4. (request (?x guest) (!y staff) login(?user ?email)))

5. (inform (!y staff) (all guest) close()))

6. (inform (?y staff) (all guest) close()))

Figure 1.3: Login scene example

tools for implementing, testing, running and visualizing Electronic Institutions. It assumes that any action performed by any agent is represented as a message being sent by that agent to one or more other agents. It is entirely based on regimentation: any message that does not satisfy the protocol is intercepted and will not arrive at its intended recipients. Such intercepted messages will not influence the state of the institution and can therefore be seen as 'not sent'. EIDE assumes a closed-world interpretation: everything is forbidden, unless the protocol explicitly says that something is allowed. In this section we will introduce the basic concepts behind EIDE.

Scenes Just as there are meetings in human institutions in which different people interact, Electronic Institutions have similar structures, known as *scenes*, in which the agents interact. Scenes are essentially group meetings, with well-defined communication protocols that specify the possible dialogs that agents are allowed to have. For example, an electronic fish market may include an auction scene in which buyers make bids to purchase fish, following a certain auction protocol, such as an English auction protocol, a Dutch auction protocol or a Vickery auction protocol. There may be many simultaneous instances of such auctions within a fish market, each referred to as a *scene instance*.

Performative Structure Scenes within an institution are connected in a network that determines how agents can legally move from one scene to another (see Figure 1.4). This network is called the *performative structure*. In a fish market for example, the performative structure may dictate that a buyer must first pass a registration scene before it can enter the auction scene.

Messages In the EIDE framework all possible actions that can be performed by agents are represented as non-divisible utterances that occur at discrete instants of time. These utterances are modeled as messages that conform

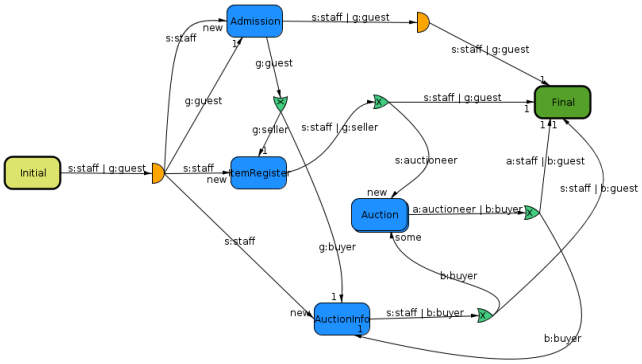


Figure 1.4: Auction house performative structure example

to a certain pattern, and physical actions are represented by appropriate messages of this form. In a fish auction, for example, a buyer commits to buy a box of fish at a certain price by making a bid, while the actual physical action of transferring money from the buyer to the auction house is triggered when the auctioneer declares that the box is sold.

For each message that can be sent, a number of parameters may be specified by the protocol. When making a bid in an auction for example, the maker of the bid should include the price he bids in the message. The Electronic Institutions framework supports several basic parameter types, such as 'Integer', 'String' and 'Boolean'. Apart from these basic types the designer of an institution can define custom types, which are composed of one or more parameters of a basic type.

Scene Protocols The interactions between the agents in a scene in an Electronic Institution have to follow a certain *scene protocol*. The protocol defines which agent can say what and when within the scene. At each moment during the execution of a protocol, the protocol is in a certain *state*, depending on the messages that have been said so far. The current state of the protocol determines what kinds of messages each agent can send.

In an auction for example, the protocol may start in a state in which the auctioneer introduces the next item under auction. Participants are not allowed to make any bid yet in this state. Once the auctioneer announces the start of the auction, the state changes to a bidding state, in which the participants are allowed to make their bids.

A protocol is therefore represented as a directed graph in which the nodes are the states of the protocol. Each edge of the graph is labeled with one or more message patterns (see for example Figure 1.3). A message can only be sent if it satisfies one of the patterns labeling one of the outgoing arcs from the current state.

Roles Scene protocols are not specified in terms of agents, but rather in terms of *roles*. Every agent plays a specific role that determines which actions it can perform at which moments. When entering an Electronic Institution an agent is required to adopt a role, but can change this role in the transitions between the scenes. A major advantage of the use of roles is that it allows the designer of a protocol to set rules for groups of agents without knowing their identities.

Constraints As explained above, the state of the protocol restricts the set of possible actions that can be performed by the agents. Moreover, this set of possible actions can be restricted even further by including constraints in the protocol. Constraints are given as sentences in a first-order logic attached to a message pattern. A message can only be sent if its corresponding constraints are satisfied.

Ontology Any message pattern may contain parameters, which are determined at run time by the sender of the message. These parameters can be of a basic type or of a user-defined type. Each Electronic Institution has an Ontology associated to it that stores the definitions of these user-defined types. It also stores for each message pattern how many parameters it has and which type each of those parameters has.

Transitions When moving from one scene to another an agent always passes a so-called transition. When the agent is at a transition it can choose which role it will play in the scene it is going to, and it can wait for other agents so that they can move to the next scene simultaneously.

The Electronic Institutions Development Environment is implemented in Java and consists of a set of tools to design and run Electronic Institutions. We will now introduce the two main existing components of the EIDE framework: ISLANDER and AMELI. Later on in this thesis we will introduce two new components (GENUINE, Section 8, and SIMPLE, Section 10) that we have added to the EIDE framework.

ISLANDER ISLANDER [Esteva et al., 2002] is a graphical tool that enables users to design an Electronic Institution (see Figure 1.5). It allows the user to visually define the scenes, roles, protocols, message patterns, constraints, ontology and other components of the institution. It then converts the visual representation into xml format (the *EI-specification*) that can be parsed and executed by AMELI.

AMELI AMELI is the core software component that executes Electronic Institutions [Esteva et al., 2004]. Once an Electronic Institution is specified with ISLANDER, the specification file is used as the input for AMELI. A new instance of the Electronic Institution is launched by starting AMELI with that file. When it is running, agents can join the institution by requesting entrance to the institution. AMELI then makes sure that the

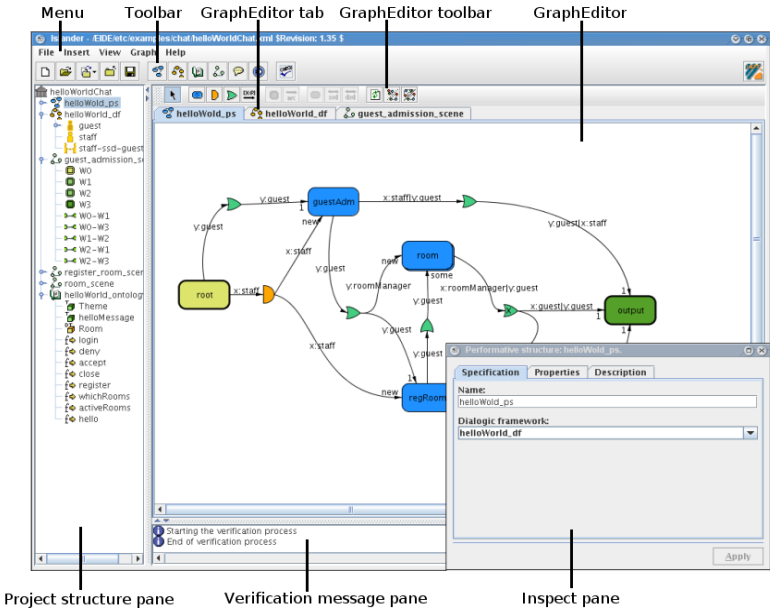


Figure 1.5: A screen shot of the Islander editor

messages exchanged between those agents obey the protocols defined in the specification file.

1.4.4 AMELI

As explained, AMELI is the core software component of EIDE. It enables agents to communicate according to protocols defined in an EI-specification file created with ISLANDER. In essence, AMELI is a communication channel through which all messages between the agents must pass. It can therefore verify for each message whether it satisfies the protocol or not and prevent it from arriving at its recipients if it does not satisfy the protocol. Moreover, it keeps an internal world state then can be regarded as a sort of ground truth for the agents.

Figure 1.6 displays the different components of AMELI. It consists of three layers: the *communication layer*, which enables the agents to exchange messages, a layer consisting of the agents that are participating in the institution, and in between there is the *social layer*, which controls the behavior of these participating agents. The social layer consists of the following agents:

Governors Each agent participating in an Electronic Institution has a special agent from the social layer assigned to it, called its *governor*. The governor of an agent α has control over each message that is being sent by α. Whenever α tries to send a message, this message first passes α's governor, which checks whether the protocol is in the correct state and whether the

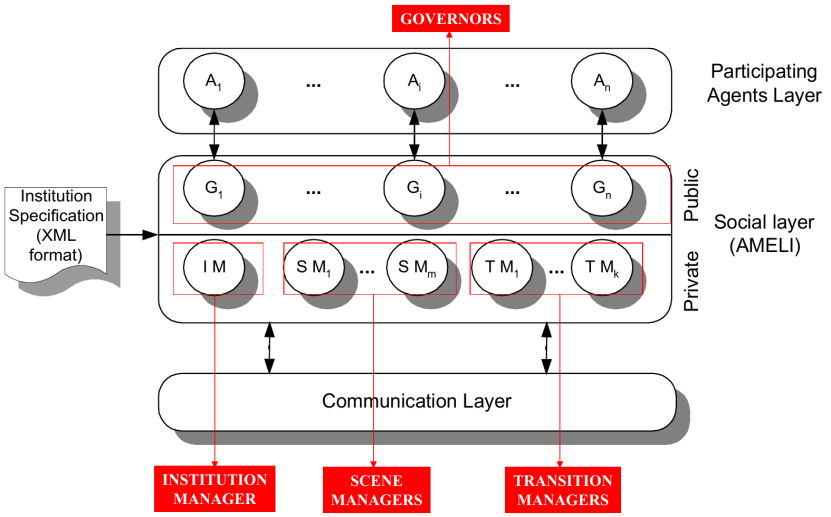


Figure 1.6: The layered structure of AMELI.

corresponding constraints are satisfied. If so, the governor forwards the message to its recipients. If not (for example, because the agent made a bid that is higher than what he can afford), the governor blocks the message.

Scene Managers There is a *scene manager* for each scene instance that is active in the institution. This agent controls and stores the state of the scene instance. Whenever an agent requests to enter or leave the scene instance, the scene manager determines whether this is possible or not.

Transition Managers There is one *transition manager* for each transition in the performative structure. This agents controls and synchronizes the movements of the agents between two scenes in the performative structure.

EInstitution Manager Each instance of an Electronic Institution has one EInstitutionManager. This agent stores and updates the global state of the institution and allows or forbids agents to enter the institution.

1.5 Contributions

In this thesis we make several contributions both to the field of Automated Negotiations and to the field of Electronic Institutions. We will here give an overview.

We introduce a new family of negotiation algorithms, called NB^3 (Negotiation-Based Branch & Bound), that applies a heuristic Branch & Bound search to explore the agreement space and determine which of the possible proposals are

good enough to propose or accept. Our main motivation for this is that many existing papers make strong assumptions about the environment that we consider unrealistic in real-world negotiations. That is, we take the following assumptions into account:

- Utility functions are highly non-linear and calculating them or inverting them is computationally expensive.
- Solutions may involve a large number of agents, possibly including humans.
- The space of solutions is very large, i.e. there is no possibility to exhaustively explore the set of solutions.
- Other agents in the system are unknown.
- Decisions have to be made within a limited time frame.
- There is no impartial mediator that one can rely on.

Although many of these assumptions have been made before in existing work, to the best of our knowledge no algorithms exists that take all of these into account. We model the fact that agents have approximate knowledge about their opponents' utilities by assuming that the *expressions* of the utility functions of the agents are publicly known, but evaluating these expressions to obtain utility *values* is costly in terms of time. Therefore, our agent can only make approximations of the opponents' utility values, and can only do so for a limited set of possible deals.

Furthermore, we introduce another negotiation algorithm, called GANGSTER that applies genetic algorithms. This algorithm has participated in the Annual Negotiating Agents Competition 2014 (ANAC'14) and won the second price. Both algorithms apply a new negotiation strategy that not only determines what proposal to make next, but also takes time into account to determine whether the agent should really make a new proposal or rather continue searching for better proposals. This distinguishes our strategy from existing negotiation strategies. This is achieved by calculating two aspiration functions, rather than one as is common in many existing negotiation algorithms.

We present a new protocol for multilateral negotiations. While most of the previous studies on automated negotiation assume strict negotiation protocols such as the Alternating Offers Protocol [Rosenschein and Zlotkin, 1994] to structure the actions of the agents, we assume an unstructured protocol that imposes almost no restrictions; the agents are allowed to propose any deal whenever they want, and are never required to reply to any proposal. This high degree of freedom introduces another dimension of complexity to the scenario, but makes it closer to real-world negotiations and can therefore be applied, for example, in negotiations with humans.

Furthermore, in order to test the NB^3 algorithm, we have defined a new negotiation domain, which is a variant of the Traveling Salesman Problem in which there are several salesmen that have to negotiate in order to minimize

their individual path lengths. The complexity of the Traveling Salesman Problem makes it a non-trivial task to calculate the utility of a given proposal, or to find a proposal with a given utility value, so traditional negotiation algorithms cannot be applied. We have implemented and tested an instance of NB^3 for this domain, called BraneSal.

We have implemented an agent, called D-Brane, that can play the game of Diplomacy. It applies And/Or search with Branch & Bound in order to determine its moves to make, and applies the NB^3 algorithm to negotiate with its coalition partners. Experiments show that it plays much better than existing agents.

We have implemented a tool called GENUINE that automatically generates a user interface that allows users to enter an Electronic Institution and interact with agents or other human participants following the protocols of the institution. This user interface is generated based on information provided at run time by the institution so that one does not need to program a new interface for every new institution. On the other hand, if a more customized interface is required, it provides an API so that one can simply design a new interface on top of the engine of GENUINE, making the information about the state of the institution directly available to the designer.

And finally, we introduce a very simple new programming language, called SIMPLE, which is very close to natural language and that allows users to define protocols for Electronic Institutions. Although many languages already exist for the specification of protocols or rules, most of those are very difficult to use for the average person. They are designed for computer scientists and require knowledge of mathematics, logic or programming. Our language on the other hand, is designed for the ordinary user: it is very close to natural language, and therefore anyone should be able to directly understand a protocol written in SIMPLE or even write a new protocol.

In short, we have made the following contributions to the field of Multi-Agent Systems:

- A negotiation protocol: the Unstructured Negotiation Protocol.
- A new testbed for negotiations: the Negotiation Salesmen Problem.
- A negotiation algorithm based on Genetic Algorithms: GANGSTER.
- A negotiation algorithm based on Branch & Bound: NB^3
- An implementation of NB^3 for the Negotiating Salesmen Problem: BraneSal.
- A negotiating agent that plays the game of Diplomacy: D-Brane.
- A tool for the creation of user interfaces for Electronic Institutions: GENUINE.
- A language to define protocols for Electronic Institutions: SIMPLE.

1.6 Outline of this Thesis

This thesis is divided into four parts: In Part I we provide all the preliminary information necessary to understand the rest of the thesis. In Part II we describe the negotiation algorithms that we have developed, in order of increasing complexity of their respective domains. In Part III we show how Electronic Institutions can be made more user-friendly so that they can be used as the infrastructure that enables humans to negotiate with agents. Finally, In Part IV we summarize our conclusions and discuss future work.

More precisely, Part I consists of chapters 1 - 4. In chapter 2 we give an overview of the existing work that has been done on negotiations and Electronic Institutions. In chapter 3 we introduce the notation used throughout the thesis and we introduce a novel model that unifies the topics of Game Theory, Electronic Institutions and Negotiations. In Chapter 4 we describe the three domains that we investigate in this thesis: the ANAC domain, the NSP, and Diplomacy.

Part II then consists of Chapters 5 - 7. In Chapter 5 we introduce GANGSTER, our agent that participated in the ANAC'14 competition. In Chapter 6 we describe the NB^3 algorithm and BraneSal, the implementation of NB^3 for the NSP. In Chapter 7 we then introduce D-Brane: our implementation of NB^3 for Diplomacy.

Part III consists of Chapter 8, in which we introduce GENUINE, Chapter 9, in which we discuss how EI and negotiations can be useful for the development of more flexible social network, and Chapter 10 in which we introduce SIMPLE. Finally, Part IV consists of the Conclusions (Section 11) and Future Work (Section 12).

Chapter 2

State of the Art

In this chapter we will give an overview of existing work in the fields of Automated Negotiations, search algorithms and Electronic Institutions.

2.1 Automated Negotations

2.1.1 Game Theoretical Approach

The earliest known work on negotiations is a paper by Nash [Nash, 1950] in which it is shown that under the assumption of certain axioms the outcome of a bilateral negotiation is the solution that maximizes the product of the players' utilities. These axioms however, are purely theoretical and often do not hold in practice. Mainly his assumption of symmetry between the players is normally not met. Negotiators do not have the same knowledge: they often do not know each others' utility functions, for example.

Therefore, many more papers have been published to deal with negotiations under more realistic assumptions. They can be divided into two main branches:the *Game Theoretical Approach* and the *Heuristic Approach*. The game theoretical approach focuses on the game theoretical properties of negotiation, such as the existence of equilibrium strategies, in settings where the axioms of Nash have been adapted or generalized. In [Zhang, 2005] for example, a logical extension to Nash's bargaining theory was made in which the agents' beliefs and desires are encoded as logical propositions. Multilateral versions of Nash's bargaining problem have been studied for example in [Krishna and Serrano, 1996] and [An et al., 2009], while a non-linear generalization has been made in [Fatima et al., 2009]. A general overview of such game theoretical studies is made in [Serrano, 2008].

2.1.2 Heuristic Approach

The Heuristic Approach on the other hand focuses on implementing algorithms that can negotiate under circumstances where no equilibrium results are known,

or where the equilibria cannot be determined in a reasonable amount of time. The disadvantage of this approach is that it is unlikely to ever have any hard guarantees about the success of the approach, but it has the advantage that it is much closer to any real-world application. The results obtained in game theoretical papers can be interesting, but are rarely applicable to real-world negotiations. In this thesis we therefore focus exclusively on the heuristic approach.

One of the seminal papers on heuristic negotiation is [Faratin et al., 1998], in which the authors define a time-based concession strategy. Their strategy is based on so called *aspiration levels*. The aspiration level of an agent is a value that that decreases over time during the negotiations. Whenever the agent is required to make an offer it tries to propose an offer for which its utility is exactly equal to its aspiration level. Therefore, the specific function used by the agent determines its strategy. The slower it decreases, the tougher it negotiates.

Most studies that have been done in this category involve scenarios with only two agents, a small agreement space and linear additive utility functions that are explicitly given or can be calculated without much computational cost. For example in the first four editions of the annually held Automated Negotiating Agent Competition (ANAC 2010-2013) [Baarslag et al., 2010]. Also, most of these studies assume an alternating offers protocol, which is good for automated agents, but not desirable for negotiations with humans, because with humans there is no guarantee that they will indeed follow the protocol.

2.1.3 Non-linear Utility Functions

Negotiations with non-linear utility functions have been studied for example in [Lai et al., 2008]. The negotiations are however bilateral, the agreement space is continuous and it is assumed the agreements at least have a known, closed-form, expression. Also in [Klein et al., 2003] the utility functions are strictly spoken non-linear over the issues, but they are still linearly additive over *pairs* of issues.

Other research on large agreement spaces with non-linear utility has been done in [Marsa-Maestre et al., 2009a, Ito et al., 2008, Marsa-Maestre et al., 2009b]. They introduce constraint based utility functions, which are also used in the Automated Negotiating Agents Competition of 2014. These utility functions are defined as the sum over the values of the constraints that are satisfied (see Section 4.1). This means that given a solution x the utility function $f_i(x)$ of some agent can still be calculated quickly. Although in theory any non-linear function over a finite space can indeed be modeled in this way, in practice it is often not feasible to convert a given representation of a utility function into this form. Therefore, we still consider this model too simplistic for real negotiations. Moreover, the algorithms described in [Ito et al., 2008] and [Marsa-Maestre et al., 2009b] depend largely on a mediator.

Work that comes relatively close to ours is [Robu et al., 2005], in which non-linear utilities are handled using preference-graphs. They focus however on how to simplify the utility by exploiting knowledge about independence between issues. They assume that utility can indeed be simplified in such a way that the

search space is shrunk to a reasonable size and can be explored exhaustively. Moreover, they only consider bilateral negotiations.

2.1.4 Multilateral Negotiations

Most research on multilateral negotiations that we are aware of (apart from the game theoretical papers mentioned above) focuses on developing protocols (e.g. [Endriss, 2006] and [Hemaissia et al., 2007]) or on non-selfish negotiations [Koenig et al., 2006]. We do not know of many papers in which multilateral negotiation algorithms for selfish agents are developed, like in this thesis.

One case in which such an algorithm was proposed, is [Nguyen and Jennings, 2004]. In their study however, a strict separation is made between *buyers* and *sellers*, so a buyer can only come to an agreement with a seller. Our approach is more general, since we do not make this distinction. Indeed, in many real life negotiations one often does not make this distinction either. A retailer, for example, sells its products to consumers, but buys them from a wholesaler, so acts both as buyer and seller. Also, in the stock market people act both as sellers and as buyers. Moreover, Nguyen and Jennings consider the presence of only one buyer, therefore excluding competition between possible buyers, and although multiple sellers are present, they still assume that each agreement is strictly bilateral. Once again utility is linear additive and the alternating offers protocol is assumed.

Also [An et al., 2006] describes multilateral negotiations in which one buyer negotiates with n sellers, but each negotiation thread between the buyer and a seller follows the alternating offers protocol, and they negotiate only about the price of a single item.

2.1.5 Search and Negotiation

The goal of our work is to develop search algorithms that can be applied to negotiations, that is: to find the right deals to propose from a large space of possible proposals. In Chapter 1.3 we already described a number of existing search algorithms. However, applying these algorithms to negotiation is not a straightforward matter, because a negotiator is not simply interested in finding the solution that maximizes its own utility function, but rather aims to find solutions that strike the right balance between its own utility and its opponents' utility. Moreover, this balance usually changes over time: the closer one gets to the deadline of the negotiations, the more one should be willing to concede. In other words: as time passes the negotiator should give more importance to the utility of the opponent(s), and less importance to its own utility. The main topic of this thesis is how we can apply or modify existing search algorithms to deal with Automated Negotiations.

The combination of search and negotiation has been studied before, for example in [Faratin et al., 2000]. There, the agent has a fixed aspiration level for its utility and searches for the deal that satisfies this aspiration level and is closest (with respect to some similarity measure) to the deal previously proposed

by the opponent. This assumes however that there are only two agents involved in the negotiation. Also, their algorithm does not try to model the opponent's preferences and therefore only considers deals that are close to deals previously proposed by the opponent. Moreover, in order to find the next best deals to propose, it assumes that the utility function is linearly additive.

Klein et al. also propose a negotiation scenario with search in [Klein et al., 2003], but time constraints are not taken into account. Another difference between their approach and ours is that their algorithm applies a mediator that must be trusted and that limits the control that the agents have over the search, since they can only accept or reject proposals made by the mediator, while this mediator does all the searching. In our work we are assuming circumstances where other agents cannot be trusted, so the use of a mediator is not an option. In the same article they also propose a variant of their algorithm without a mediator, involving a mutually observable 'die' to steer the search, instead. But this still means that the agents should trust the fact that the die is fair. Moreover, the agents need to follow a strict protocol, so this algorithm is only suitable for negotiation between agents that were designed for this particular protocol.

2.2 Electronic Institutions

2.2.1 Frameworks

Electronic Institutions have been subject of research for a long time and a number of frameworks have been implemented that often consist of tools for implementing, testing, running and visualizing protocols. Examples of such frameworks are ANTE [Cardoso et al., 2013], MANET [Tampitsikas et al., 2012], S-MOISE+ [Hübner et al., 2006], and EIDE [Esteva et al., 2008]. A comparative study of some of those systems has been made in [Fornara et al., 2013].

ANTE [Cardoso et al., 2013] has been implemented as a JADE-based platform, including a set of agents that provide contracting services. It integrates automatic negotiation, trust & reputation and normative environments. Users and agents can specify their needs and indicate the contract types to be created. Norms governing specific contract types are predefined in the normative environment. Although ANTE has been targeting the domain of electronic contracting, it was conceived as a more general framework having a wider range of applications in mind.

The MANET [Tampitsikas et al., 2012] meta-model is based on the assumption that the agent environment is composed of two fundamental building blocks: the physical environment, concerned with agent interaction with physical resources and with the MAS infrastructure, and the social environment, concerned with the social interactions of the agents. In the MANET meta-model it is assumed that the normative system can be composed of three structural components: agents, objects and spaces.

2.2.2 Deontic Logic

A logical system to define norms and rules is called a *deontic* logic. The best known system of deontic logic is called Standard Deontic Logic (SDL) [von Wright, 1951]. Important refinements of this logic are Dyadic Deontic Logic (DDL) [Lewis, 1974] and Defeasible Deontic Logic [Nute, 1997]. Furthermore, an extension of this taking temporal considerations into account was proposed in [Governatori et al., 2005].

In [Makinson and Van Der Torre, 2000] A system to formalize norms using input/output logic was proposed, while in [van der Hoek et al., 2007] the authors provide a model for the formalization of social law by means of Alternating-time Temporal Logic (ATL). In [Kröger, 1987] the author proposes the use of Linear Time Logic (LTL) to express norms. Other important approaches are based on Propositional Dynamic Logic (PDL) [Meyer, 1987], on See-to-it-that logic (STIT) [Belnap and Perloff, 1990] and on Computational Tree Logic (CTL) [Broersen et al., 2004]. Models for the verification of expectations in normative systems are proposed in [Cranefield and Winikoff, 2011] and [Alberti et al., 2006], and in [Sergot and Craven, 2006] the authors introduce the *nC+* language for representing normative systems as state transition systems.

The above mentioned systems however mainly focus on the theoretical properties of norms. Work that is more focused on the actual implementation of norms is for example [Lopez y Lopez et al., 2004] which proposes a model to define norms in the Z language, while in [Artikis et al., 2005] the authors propose the use of Event Calculus for the specification of protocols. A programming language designed to program organizations, called 2OPL, was introduced in [Dastani et al.,]. Other important examples of languages and frameworks for the implementation of norms are described in: [Vázquez-Salceda et al., 2004], [Argente et al., 2008], [Uszok et al., 2008], [García-Camino, 2008], [Kollingbaum, 2005], and [Cranefield, 2005]

Although some of the languages mentioned here are more user-friendly than others, they still all seem to require the user to be a computer scientist or at least to have some knowledge of programming, logic or mathematics. To the best of our knowledge no work has been published on the specification of rules or protocols that aims for truly unexperienced users and tries to stay as close as possible to natural language.

Chapter 3

Formal Model

A multi-agent system is a set of agents that exchange digital information by sending messages to one another. For example, they could be connected over the Internet and sending messages following the TCP/IP protocol. When we refer to 'agents' we may refer to either humans or software agents. However, even if we are talking about humans, we still assume that information is exchanged digitally. For example, a human agent may communicate through a graphical user interface such as a web site. In this chapter we will formally define the concepts that we will use throughout the rest of the thesis.

3.1 Messages and Agents

Let $A = \{a_1, a_2, ...\}$ be a set of constants that we call the *Agent Identifiers*. In practice, agent identifiers could be network addresses for example, or user names, or signatures generated with asymmetric encryption.

Definition 12 *A **message** is a tuple (a_i, J, c, t) where $a_i \in A$ is called the **sender**, $J \subseteq A$ is a nonempty set called the set of **receivers**, $c \in \mathbb{N}$ is a natural number called the **content**, and $t \in \mathbb{N}$ is the **timestamp***

The content represents the actual information that is being sent from the sender to the receivers. Since any form of digital information is in fact always a binary number of finite length, this can indeed by represented as a natural number. In principle, the content of a message can be any stream of bytes, but depending of the domain of application it may be restricted to some domain language. The timestamp represents the time at which the message is sent (see Section 3.3.3 for more details).

The set of all possible messages is denoted M.

$$M = A \times (2^A \setminus \{\varnothing\}) \times \mathbb{N} \times \mathbb{N}$$

we use the notation M_t to denote the set of all possible messages at time t:

$$M_t = A \times (2^A \setminus \{\varnothing\}) \times \mathbb{N} \times \{t\}$$

and we use the notation $M_{i,t}$ to denote the set of all possible messages that can be sent by an agent with identifier a_i at time t:

$$M_{i,t} = \{a_i\} \times (2^A \setminus \{\varnothing\}) \times \mathbb{N} \times \{t\}$$

To simplify some of the definitions below we model the event that an agent is not sending any message at all as a specific kind of message, denoted as $none_{i,t}$.

$$none_{i,t} = (a_i, A, 0, t) \in M_{i,t}$$

So if an agent with identifier a_i does not send any message at time t, this is considered equivalent to the agent with identifier a_i sending the message $none_{i,t}$ to all other agents. We will sometimes simply write 'none' instead of $none_{i,t}$.

Definition 13 *A **message history** h_t at time t is a set of messages, such that for each message $(a_i, J, c, t') \in h_t$ the inequality $t' < t$ holds and such that for all $a_i \in A$ and all $t' \in \mathbb{N}$ the following holds: $|M_{i,t'} \cap h_t| = 1$.*

The second condition in this definition represents the fact that no agent can send more than one message at the same time (strictly speaking it says that every agent must send exactly one message at each instance of time, however, this message can be the 'none' message, so in practice it means that each agent may send 0 or 1 messages).

In order to model the fact that an agent can only have knowledge about those messages that were either sent by that agent or received by that agent, we define the concept of an individual message history.

Definition 14 *Given a message history h_t the **individual message history** h_t^i of an agent with identifier a_i is the subset of h_t for which either a_i is the sender or a_i is one of the receivers.*

$$h_t^i = \{(a_j, J, c, t') \in h_t \mid a_i = a_j \vee a_i \in J\}$$

Let H^i denote the set of all possible individual message histories of an agent with identifier a_i. We will now define the concept of an 'agent' as a deterministic algorithm that determines for each possible individual message history a new message to send in the next instance of time.

Definition 15 *An **agent** α_i with identifier a_i is a function that maps any individual message history h_t^i to a message:*

$$\alpha_i : H^i \to M \text{ such that } \alpha_i(h_t^i) \in M_{i,t}$$

Of course one may argue that agents are not always deterministic, especially when the agent is human. In order to incorporate indeterminism we could alternatively define an agent as a probability distribution over a set of 'agent types' and define an agent type as a deterministic algorithm. However we think that this is not really relevant for the rest of this thesis, because the agents that we

have implemented never have full knowledge about the other agents in the system anyway. Given the fact that one agent's knowledge about another agent is incomplete, it does not really matter whether that other agent is deterministic or not. Therefore, in this thesis we will stick with the definition of an agent as a deterministic algorithm. In the rest of this thesis the notation α_i always refers to an agent with identifier a_i.

Definition 15 only says that an agent *generates* some message for each possible message history. Now we would like to interpret this generated message as the message that the agent will send in the next instance of time. For this reason we define the 'realized message history':

Definition 16 *Given a time stamp $t \in \mathbb{N}$ and a finite set of agents $Ag = \{\alpha_1, \alpha_2, \dots \alpha_n\}$ the **realized message history** $\mathcal{H}_{Ag,t}$ is a message history defined as follows:*

$$\mathcal{H}_{Ag,t} = \begin{cases} \varnothing & \text{if } t = 0 \\ \mathcal{H}_{Ag,t} = \mathcal{H}_{Ag,t-1} \bigcup (\bigcup_{i=1}^{n} \alpha_i(\mathcal{H}^i_{Ag,t-1})) & \text{otherwise} \end{cases}$$

Here, $\mathcal{H}^i_{Ag,t-1}$ is the 'individual' subset of $\mathcal{H}_{Ag,t-1}$ as defined by Definition 14. The interpretation of this recursive definition is at $t = 0$ the realized message history is the empty set, and at each following instance of time t the messages sent by the agents in Ag at time t are added to it.

3.2 Protocols

We will now define the concept of a protocol. Essentially, a protocol defines what the agents are allowed to do and what they are not allowed to do, depending on their previous actions.

Let $\mathcal{E}$ denote any set (finite or countably infinite), which we call the set of world states. Then a state evolution map is defined as follows:

Definition 17 *Let $A_n = \{a_1.a_2, \dots a_n\}$ be a finite set of agent identifiers of size n. Then a **state evolution map** $\mathcal{F}$, given A_n, is a function that maps any world state and any set of n messages containing one message for each sender $a_i \in A_n$ and for which the time stamps are the same, to a new world state:*

$$\mathcal{F} : \mathcal{E} \times \prod_{i=1}^{n} M_{i,t} \to \mathcal{E}$$

The interpretation of this is that, given a world state ϵ_t at time t and a set of messages that are being sent at time t, the state evolution map determines the new world state that results from the agents sending those messages.

Definition 18 *Given a set of agent identifiers A_n of size n, a state evolution map $\mathcal{F}$, an initial world state ϵ_0 and a message history h_t then the message history's **corresponding state** ϵ_{h_t} is defined as:*

$$\epsilon_{h_t} = \begin{cases} \epsilon_0 & \text{if } t = 0 \\ \mathcal{F}(\epsilon_{h_{t-1}}, m_{1,t-1}, m_{2,t-1}, \dots m_{n,t-1})) & \text{otherwise} \end{cases}$$

where each $m_{i,t-1}$ *denotes the unique message in* $h_t \cap M_{i,t-1}$.

This means that if the initial world state is ϵ_0 and the messages sent by the agents are given by h_t then at time t the world will be in a state ϵ_{h_t}.

Definition 19 *Given a state evolution map* $\mathcal{F}$*, an initial world state* ϵ_0 *and a set of agents* Ag *of size* n*, the* ***realized world state*** $\epsilon_{t,Ag}$ *at time* $t > 0$ *is defined as:*

$$\epsilon_{t,Ag} = \mathcal{F}(\epsilon_{t-1}, \alpha_1(\mathcal{H}^i_{Ag,t-1}), \alpha_2(\mathcal{H}^i_{Ag,t-1}), \ldots, \alpha_n(\mathcal{H}^i_{Ag,t-1}))$$

Note that the realized world state is exactly the world state corresponding to the realized message history.

We have seen that, given a state evolution map, world states are entirely determined by the history of all messages that have been sent. Therefore, world states are in fact a redundant concept; every concept defined in terms of world states could have been defined equivalently in terms of message histories. However, we will see it is often more convenient to use world states for many definitions. In a sense we can consider world states as equivalence classes of histories, which contain the relevant information.

One advantage of the introduction of world states is that it allows us to make a distinction between *informative* messages and *active* messages. Active messages are those messages that change the world state when uttered, while informative messages never change the world state.

To formalize this, we use the notation $m_{-i,t}$ to denote a sequence of messages at time t, one for each agent except agent α_i:

$$m_{-i,t} \in \prod_{j=1,j\neq i}^{n} M_{j,t}$$

Furthermore, we use the notation $(m_{i,t}, m_{-i,t})$ to denote a sequence of messages at time t, one for each agent, where agent α_i sends the message $m_{i,t}$ and all other messages are the messages in $m_{-i,t}$.

Definition 20 *Let* $m_{i,t}$ *be a message sent by agent* α_i*. If for any world state* ϵ *and for any sequence of messages* $m_{-i,t}$ *the following holds:*

$$\mathcal{F}(\epsilon, (m_{i,t}, m_{-i,t})) = \mathcal{F}(\epsilon, (none_{i,t}, m_{-i,t}))$$

then $m_{i,t}$ *is an* ***informative message****. Otherwise,* $m_{i,t}$ *is an* ***active message****.*

It is important to note that, although informative messages do not *directly* change the world, they do change the message history, and therefore, according to Definition 15 may influence the future messages of other agents, which in turn may change the world.

Although the distinction between informative messages and active messages may not have important consequences to the formalization of our model, we think that this distinction is very important conceptually. It reflects the way we think and act in the real world, because in the real world we also distinguish

between actions that physically change the world, or actions that are simply a form of communication.

An example of an informative message could be an e-mail sent from one colleague at work to another, discussing the time of a meeting. Such a message does not change anything about the world, but serves to exchange of information between agents about the world, allowing them to coordinate their future actions. An example of an active message on the other hand, could be a bank transfer. Of course, one may still argue that a bank transfer does not really physically change the world, as it only changes some numbers in a database. Therefore, one should keep in mind that this distinction is open to interpretation and depends on the way the world states of thew MAS are defined. However, in our formal model we assume that there is a well-defined mapping that maps any message history to a world state. As as long as we have such a mapping, we can make this distinction. In the rest of the thesis we may sometimes refer to a active messages as *actions*, and we may talk about "performing an action" when we mean to say "sending an active message".

Some active messages may only have effect in certain world states. Therefore, we say an action is infeasible in some world state if it does not have any effect on that world state.

Definition 21 *A message $m_{i,t}$ is called an **infeasible action** in a given world state ϵ if it is an active message, and for any sequence of messages $m_{-i,t}$ we have:*

$$\mathcal{F}(\epsilon, (m_{i,t}, m_{-i,t})) = \mathcal{F}(\epsilon, (none_{i,t}, m_{-i,t})).$$

*The message is called a **feasible action** otherwise.*

Note that in the equation in Definition 21 the world state is fixed, while in the definition of an informative message the equation must hold for *all* world states.

Definition 22 *A permission map $\mathcal{G}$ is a function that assigns a set of **allowed messages** $O_{i,t} \subseteq M_{i,t}$ to each world state ϵ, agent identifier a_i and time stamp t:*

$$\mathcal{G} : \mathcal{E} \times A \times \mathbb{N} \to 2^M \text{ such that } \mathcal{G}(\epsilon, a_i, t) \subseteq M_{i,t}.$$

Definition 23 *A **protocol** Pr (for n agents) is a tuple $(A_n, \mathcal{E}, \epsilon_0, \mathcal{F}, \mathcal{G})$ consisting of a set of agent identifiers A_n of size n, a set of world states $\mathcal{E}$, an initial world state $\epsilon_0 \in \mathcal{E}$, a state evolution map $\mathcal{F}$ and a permission map $\mathcal{G}$.*

Definition 24 *We say a protocol is **regimented** if and only if for every world state $\epsilon \in \mathcal{E}$, every action that is not allowed in ϵ, is also not feasible in ϵ.*

In other words: a regimented protocol is a protocol for which the messages that are not allowed simply will not have any influence on the world state. In this thesis we will always assume all protocols are regimented.

Definition 25 *We say a protocol has **deadline** d iff for all $t \geq d$ we have $\mathcal{G}(\epsilon, a_i, t) = \varnothing$ for all $\epsilon \in \mathcal{E}$, and all $a_i \in A_n$.*

A protocol may or may not have a deadline.

3.3 Games

We will now define the concept of a game. We would like to stress however that the term 'game' should be interpreted in a very broad sense, because it does not only represent those systems that one would call a game in daily life, but rather any kind of multi-agent system.

Definition 26 *A **game** G (for n players) is a tuple $(Pr^G, \bar{f}^G, \bar{d})$ where, Pr^G is a regimented protocol for n agents, $\bar{f}^G = (f_1^G, f_2^G, \ldots f_n^G)$ a finite sequence of utility functions, and $\bar{d} = (d_1, d_2, \ldots d_n)$ a finite sequence of deadlines.*

In this definition each utility function f_i^G can be any function from $\mathcal{E}$ to $\mathbb{R}$, where $\mathcal{E}$ is the set of world states of the protocol. Each agent is assumed to have its own individual deadline $d_i \in \mathbb{N}$. These individual deadlines are independent of the deadline of the protocol. Let $\epsilon_{d_i,Ag}$ denote the realized world state at time d_i, for some set of agents Ag. Then in this thesis we always assume that every agent α_i in Ag is implemented with the goal of maximizing the value $f_i^G(\epsilon_{d_i,Ag})$. That is: each agent aims to bring about a world state that maximizes its individual utility at its individual deadline. Note that the introduction of individual deadlines allow us to properly define the goals of the agents even if the protocol of the game does not have a deadline.

Definition 27 *A **game over** d **rounds** is a game for which $d_i = d$ for all $a_i \in A$. A **one-shot game** is a game over 1 round.*

In the context of Game Theory agents are also called *players*, the set of allowed actions $O_{i,t}$ for agent α_i is called the set of *moves* of agent α_i. Furthermore we define $O_t = \prod_{i=1}^{n} O_{i,t}$. If the time t is irrelevant or clear from the context, we will omit the subscript t and denote these as O_i and O respectively.

Definition 28 *A **joint move** at time t is an element of the set O_t.*

Instead of saying "α_i sends a message o_i" we may also say "α_i makes the move o_i".

Note that in Game Theory it is common to define the utility functions f_i^G directly over the set of joint moves, rather than over world states. Of course this is because the state has been abstracted away. In many real games, such as chess or tic-tac-toe there clearly exists a notion of a state (namely the board configuration). The moves of the players determine how the state changes, and the state in turn determines the utility value. Therefore, in case of a one-shot game we may sometimes also use the notation $f_i^G(o)$ for some joint move o, in place of the notation $f_i^G(\epsilon_o)$ where ϵ_o is the realized world state when joint move o is played.

3.3.1 Incomplete Information

According to the definitions above the realized world state may depend on the entire message history, while agents only know about those messages they have

either sent or received. Therefore, in general agents may not exactly know the state of the world. In practice however, this does not need to be a problem, because the utility function of an agent may have the same value over all world states that could possibly be the true world state. In that case the agent simply does not care about the actual world state, because it is only interested in its own utility value.

3.3.2 External Influence

The definitions above are made on the assumption that the state of the world evolves purely based on the messages exchanged between the agents in the MAS and that the agents base their decisions what to do purely on the history of the messages sent inside the MAS.[1] In other words, they have no contact whatsoever with the 'outside world'. Nothing outside the MAS can influence the evolution of the MAS or the decisions of the agents.

One may argue that this is not a realistic model, because agents may have other inputs rather then just the messages they receive. If an agent is a human acting through a GUI, he or she is clearly also influenced by events other than the events observed in the GUI. Also, if an agent is a software agent, it may be connected to a camera or microphone to obtain external information.

However, this is not really a problem, because if one really needs to incorporate dynamic information about the external world into the model one could do this by introducing an extra agent in the model that represents the outside world. Whenever some event in the outside world happens it would send a message to the other agents to inform them it.

3.3.3 Discrete Time

In the above definitions we have modeled time as being discrete. In some domains it may make more sense to consider time as continuous, and represent instants of time as real numbers rather than natural numbers. Nevertheless, we prefer to represent time by natural numbers, for several reasons. Firstly, In computer science time is often measured as a number of CPU cycles, or as the number of milliseconds that have passed since 1 january 1970 (Unix Time[2]), both cases obviously being natural numbers. Secondly, even of one wants to model time as a continuum, any physical measurement of time will always yield a result consisting of a finite number of digits which is essentially the same as a natural number. Thirdly, in Game Theory the moves of the players usually take place in discrete rounds. Therefore, a notion of discrete time is closer to such descriptions. In fact, in the rest of this thesis we may sometimes refer to 'rounds' rather than to instants of time, meaning essentially the same.

[1] Interestingly, one may argue that in a model such as ours there is in fact no such thing as *the* world. Instead, each agent simply lives in its own world of which the state is determined purely by the messages that the agent has sent and received. After all, messages that were not sent or received by an agent cannot possibly influence that agent.

[2] `http://en.wikipedia.org/wiki/Unix_time`

Finally, if we set the deadlines d_i to be some very large number and take the discrete intervals of time to be very short, then this model approximates continuous time anyway. The fact single rounds are then very short may mean that players do not have enough time to decide which move to make in each round, but that is not a problem since we assume that not making a move is interpreted as making the move 'none'.

3.4 Negotiation

A negotiation protocol does not only define which messages can be sent, but also determines for each history which deals the agents have committed themselves to. These deals are then called the *confirmed deals*.

Definition 29 *A **negotiation protocol** is a tuple $(Pr, Agr, conf)$ where Pr is a protocol, Agr is some set called the **agreement space**, and $conf$ is a function that assigns to each possible world state ϵ a set of **confirmed deals**, which is a subset of the agreement space.*

$$conf : \mathcal{E} \to 2^{Agr}$$

We say a negotiation protocol has deadline d if its underlying protocol Pr has deadline d. As an example, we will now define the Alternating Offers (AO) protocol [Rosenschein and Zlotkin, 1994] in our model.

Example The standard form of AO assumes there are two agents: $A = \{a_1, a_2\}$ and there is some given agreement space Agr. The set of world states can be defined as: $\mathcal{E} = A \times Agr \times Agr$, so each world state is of the form $\epsilon = (a_i, x, y)$ where a_i is the identifier of the agent whose turn it is to make a proposal, x is the last proposed deal and y is the confirmed deal. The initial state is: $\epsilon_0 = (a_1, x_0, x_0)$, where x_0 represents the 'conflict deal' i.e. the element of Agr that represents the case that the agents do not make any deal at all. The permission map is defined as:

$$\begin{aligned}
\mathcal{G}((a_i, x, x_0), a_i, t) &= \{(a_i, \{a_{-i}\}, accept(x), t)\} \cup \{(a_i, \{a_{-i}\}, propose(y), t) \mid y \in Agr\} \\
\mathcal{G}((a_{-i}, x, x_0), a_i, t) &= \{none_{i,t}\} \\
\mathcal{G}((a_i, x, x), a_i, t) &= \mathcal{G}((a_i, x, x), a_{-i}, t) = \{none_{i,t}\} \text{ if } x \neq x_0
\end{aligned}$$

The first line says that the agent whose turn it is can either accept the previously proposed deal x or propose a new deal y. The second line says that the agent whose turn it is not cannot do anything. The third line says that once some deal x has been confirmed no agent can do anything anymore. The state evolution map is defined as:

$$\begin{aligned}
\mathcal{F}((a_1, x, x_0), (a_1, \{a_2\}, propose(y), t), none_{2,t}) &= (a_2, y, x_0) \\
\mathcal{F}((a_1, x, x_0), (a_1, \{a_2\}, accept(x), t), none_{2,t}) &= (a_2, x, x)
\end{aligned}$$

$$\begin{aligned}\mathcal{F}((a_2, x, x_0), none_{1,t}, (a_2, \{a_1\}, propose(y), t)) &= (a_1, y, x_0)\\ \mathcal{F}((a_2, x, x_0), none_{1,t}, (a_2, \{a_1\}, accept(x), t)) &= (a_1, x, x)\end{aligned}$$

and all other cases are infeasible. These lines simply state that a proposal changes the agent whose turn it is and that an acceptance causes the previously proposed deal to become the confirmed deal. Finally, the confirm function simply projects out the confirmed deal:

$$conf(a_i, x, y) = \{y\}$$

(Note that the Alternating Offers Protocol only allows one deal to be confirmed, whereas our model in general allows more than one deal to be confirmed).

Definition 30 *A **deal** x **over a one-shot game** G is a Cartesian product of nonempty subsets S_i of O_i, one for each player: $x = S_1 \times S_2 \times \ldots S_n$.*

In the following we will use the notation $O_i[x]$ instead of S_i to explicitly indicate that it is part of a deal x.

Definition 31 *We define the set of **participating agents** $pa(x)$ of a deal $x = O_1[x] \times O_2[x] \times \ldots O_n[x]$ as the set of all players α_i for which $O_i[x]$ is a proper subset of O_i:*

$$\alpha_i \in pa(x) \Leftrightarrow O_i[x] \neq O_i$$

Definition 32 *The **Agreement Space over** G is then the set of all deals over the one-shot game G and is denoted Agr_G:*

$$Agr_G = \prod_{i=1}^{n} (2^{O_i} \setminus \{\varnothing\})$$

A deal over a game G should be interpreted as an agreement between the participating agents that each of them will only play a move from the subset $O_i[x]$. If an agent is participating in more than one confirmed deal he should obey them all, and is therefore only allowed to play moves from the intersection $\cap_{x \in X} O_i[x]$, where X is the set of confirmed deals. A correctly defined negotiation protocol should therefore guarantee that this set is always nonempty for every agent.

Definition 33 *A set of deals X is called **consistent** iff its intersection is nonempty: $\bigcap_{x \in X} x \neq \varnothing$*

Note that this definition indeed implies that $\bigcap_{x \in X} O_i[x]$ is nonempty for every agent.

Definition 34 *A **consistent negotiation protocol** is a protocol, such that for every $\epsilon \in \mathcal{E}$ the set of confirmed deals $conf(\epsilon)$ is consistent.*

Definition 35 *The **restricted game** $G[x]$ of a one-shot game G to a deal x is the same game as G, but with the restriction that each player α_i is only allowed to make a move from the set $O_i[x]$. That is: the permission map $\mathcal{G}^{G[x]}$ of the protocol $Pr^{G[x]}$ of the one-shot game $G[x]$ is defined as*[3]*:*

$$\mathcal{G}^{G[x]}(\epsilon_0, a_i, 0) = O_i[x]$$

Similarly, for a set of deals X the game $G[X]$ is the game G with the restriction that each player α_i is only allowed to make a move from the set $\bigcap_{x \in X} O_i[x]$.

We are now getting to the most important definition of this chapter: the definition of a negotiation game. The idea behind a negotiation game is that a set of agents is playing some one-shot game G, however, before doing so, they get the opportunity to negotiate which moves they will play in the game G. The agreements made during these negotiations must be obeyed.

Given a one-shot game G and a negotiation protocol N with deadline d, we denote the negotiation game over G as NG. It is a game over $d+1$ rounds, with protocol Pr^{NG} and utility functions $\bar{f}^{NG}$ which are defined below. The first d rounds of this game are referred to as the **negotiation stage**, and the last round is called the **action stage**. During the negotiation stage the players negotiate a set of deals from Agr_G according to the negotiation protocol N. If X_d denotes the set of deals that were confirmed during the negotiation stage: $X_d = conf(\mathcal{H}_{Ag,d})$, then in the the action stage the players play the game $G[X_d]$.

More precisely, the protocol Pr^{NG} is defined as:

$$Pr^{NG} = (A_n, \mathcal{E}^{NG}, \epsilon_0^{NG}, \mathcal{F}^{NG}, \mathcal{G}^{NG}) \tag{3.1}$$

where the set of world states of NG is the product of the world states of N and the world states of G:

$$\mathcal{E}^{NG} = \mathcal{E}^N \times \mathcal{E}^G \tag{3.2}$$

and the initial state is the pair consisting of the initial state of N and the initial state of G:

$$\epsilon_0^{NG} = (\epsilon_0^N, \epsilon_0^G). \tag{3.3}$$

The permission map $\mathcal{G}^{NG}$ is defined such that during the negotiation stage it is in fact the permission map of N, while in the action stage it is equal to the permission map of $G[X_d]$:

$$\mathcal{G}^{NG}(\epsilon^{NG}, a_i, t) = \mathcal{G}^{NG}((\epsilon^N, \epsilon^G), a_i, t) = \begin{cases} \mathcal{G}^N(\epsilon^N, a_i, t) & \text{if } t < d \\ \mathcal{G}^{G[X_d]}(\epsilon^G, a_i, 0) & \text{if } t = d \end{cases} \tag{3.4}$$

Similarly, the state evolution map is defined such that during the negotiation stage the world state evolves according to N, while in the action stage its evolution is determined by the rules of G:

$$\mathcal{F}^{NG}(\epsilon^{NG}, o_t) = \mathcal{F}^{NG}((\epsilon^N, \epsilon^G), o_t) = \begin{cases} (\mathcal{F}^N(\epsilon^N, o_t), \epsilon^G) & \text{if } t < d \\ (\epsilon^N, \mathcal{F}^G(\epsilon^G, o_t)) & \text{if } t = d \end{cases} \tag{3.5}$$

[3]Note that for a one-shot game the permission map only needs to be defined for $\epsilon = \epsilon_0$ and $t = 0$.

for any joint move o_t made in round t and any world state ϵ^{NG}. Finally, the utility functions $\bar{f}_i^{NG} = (f_1^{NG}, f_2^{NG}, \ldots f_n^{NG})$ are equal to the utility functions of G acting on the the world state of G:

$$f_i^{NG}(\epsilon^{NG}) = f_i^{NG}(\epsilon^N, \epsilon^G) = f_i^G(\epsilon^G) \tag{3.6}$$

Definition 36 *Given a one-shot game G and a negotiation protocol N with deadline d, the **negotiation game over** G, denoted as NG, is a game over $d+1$ rounds with protocol Pr^{NG} defined by (3.1) - (3.5) and utility functions $\bar{f}_i^{NG}$ defined by (3.6).*

We would like to stress that a player's set of allowed moves $O_i[X_d]$ in the action stage can only be smaller than its full set of moves O_i if agent α_i itself has agreed with those restrictions by accepting one or more deals. Of course, restricting your set of moves is never beneficial by itself, but when the other players return the favor by also restricting their moves this can be highly beneficial.

Example Let us take a look at the negotiation game NG where G is the Prisoner's Dilemma and the negotiation protocol is the alternating offers protocol. This negotiation game contains a Nash Equilibrium in which, during the negotiation stage, the first player plays the move 'propose(({confess}, {confess}))' and the second player makes the move 'accept(({confess}, {confess}))', while in the action stage both players play the move 'confess'. This outcome dominates the Nash Equilibrium of the pure Prisoner's Dilemma without negotiations, in which both players play 'deny'. This clearly shows how negotiations can improve individual results.

Example Another example is the purchase of a car. We assume there are two agents: the client and the dealer. The sale of a car can be modeled as a one-shot game in which the client can make a move 'pay(x)' for any $x \geq 0, x \in \mathbb{R}$, and the dealer can either make the move 'sell_car' or the move 'keep_car'. Before the sale takes place, they negotiate over the price. If they cannot agree on the price they will respectively play 'pay(0)' and 'keep_car' in the action stage. If they do agree on a deal to sell the car for a price x during the negotiation stage, then this deal will be executed in the action stage by playing 'pay(x)' and 'sell_car' respectively.

Note that in our model, the utility of the negotiators depends on the deals made, in a rather indirect way. The confirmed deals determine the set of allowed moves of the agents in the game G, the moves of that game determine the realized world state ϵ_{d+1} which in turn determines the utility values. In the literature it is much more common to define utility functions directly as functions over the agreement space. This is not in contradiction with our model, as we can simply abstract

the underlying game G away, as well as the world states. However, our model is richer because it allows us to explicitly take the underlying domain into account. It allows us to define negotiation domains in which the utility values are very hard to calculate.

For example, in our model a deal may not completely determine the utility, since it may only fix a subset of moves for a player, so that that player still has the freedom to choose its move from that subset, and thus its utility will also depend on that choice. We think this is more realistic, because the value of a deal may indeed also depend on further actions of the negotiator beyond the deal itself. Moreover, the inclusion of a game means that one should also take the opponent's options into account. More importantly, it allows for domains in which the negotiator needs to apply complex reasoning in order to determine the value of a deal. A negotiator cannot simply get the utility value from a table, but instead needs to apply Game Theoretical reasoning to determine the consequences of a deal for the game G. In many cases it may not even be possible to determine the theoretically correct value of a deal because of bounded rationality.

3.4.1 Reservation value

An important concept in Automated Negotiations is the concept of the reservation value. Normally, the reservation value rv_i for agent α_i is simply the value that is obtained by α_i if it does not make any deal. However, we here argue that in our model there is no straightforward definition of a reservation value. Instead, we can define a new concept that we call the *paranoid reservation value*, but we show that this definition is not always a satisfactory one.

Let O_{-i} denote the set of all sequences of moves containing one move for each agent except agent α_i, in the one-shot game G.

$$O_{-i} = \prod_{j=1,j\neq i}^{n} O_j$$

Definition 37 *The **paranoid reservation value** for the negotiation game NG is defined as:*

$$prv_i = max_{o_i \in O_i}\, min_{o_{-i} \in O_{-i}} \{f_i(o_i, o_{-i})\}$$

In words, this is the value that α_i would obtain without making any deals, under the paranoid assumption that all other agents make those moves that minimize α_i's utility.

Although this value indeed represents the absolute minimum that α_i can expect to get, it can be very unpractical to use for several reasons. First, it is often unrealistic to assume that all other agents co-operate against α_i. Second, even if the other agents do all cooperate with each other, it does not always have to be the case that minimizing α_i's utility is in their interest, because they may not be playing a zero-sum game.

Any deal that yields less utility than the paranoid reservation value may be discarded immediately because α_i would always prefer not to make any deal at

all. However, the opposite does not hold: if a deal x yields more utility than the paranoid reservation value, α_i may still prefer to make no deal at all if it considers the paranoid assumption to be overly pessimistic in the given domain.

Alternatively, one could try to define other variations to the notion of a reservation value. For example, one could try to determine some equilibrium solution in which no coalition of opposing agents can deviate without decreasing the utility of any coalition member. Then the reservation value rv_i could be defined as the utility that α_i would obtain under that solution. The problem with that solution however is that it will often be very difficult to determine such a solution. Furthermore, even if α_i would be able to find it, it may be that the opponents are not able to find it, or do not have the time to properly co-ordinate as to realize that solution. In that case the "reservation value" defined in this way may be either overly optimistic or overly pessimistic. This is dangerous because it may lead α_i to discard deals assuming it can achieve more utility without any deals, as long as the opponents play their equilibrium strategy. If they do not manage to play that solution however, the real utility for α_i may turn out to be lower than expected. Finally, there could exist several such solutions yielding different utility values for α_i, in which case this reservation value is not even properly defined.

3.4.2 Negotiations over Games with Multiple Rounds

The game G does not have to be a one-shot game. We can also define negotiation scenarios in which the underlying game takes place over several rounds. In that case the negotiations during the negotiation stage do not restrict the moves of a single round, but rather restrict the set of *strategies* of each player, over several rounds. This is the model we will use later in the definition of the Negotiating Salesmen Problem (Chapter 4.2).

Alternatively one can also allow the game to alternate between negotiation stages and action stages, where each action stage corresponds to a single round. This is how the game of Diplomacy can be modeled (Chapter 7).

3.4.3 Commitments

In some cases the possible deals between the players may be built up from smaller constituents, that we call *commitments*. In that case, instead of defining a deal as a subset of all possible joint moves, we may alternatively define a deal as a set $\overline{x}$ of commitments. The set x consists of exactly those moves (or strategies) that contain all commitments in $\overline{x}$. For example, in the case of Diplomacy (Chapter 7), each player has a number of units for which the player needs to submit an order. If two players commit themselves two give specific orders to some of their units, then the subset $O[x]$ is defined as the set of moves that contain those orders. The more commitments an agent makes, the less freedom it has, so the larger the set $\overline{x}$, the smaller the set x.

Furthermore, in the case that G is a game over multiple rounds, then a deal may consist of a number of commitments to make a certain moves $o_{i,t_1}, o_{i,t_2}, o_{i,t_3} \ldots$

in a certain rounds $t_1, t_2, t_3, \ldots$. Then $O_i[x]$ consists of all strategies for which indeed all these moves are played in their corresponding rounds.

3.5 The Unstructured Negotiation Protocol

Above, we have shown how the AO protocol can be defined in our model. In this section however, we will define a new negotiation protocol that we call the *Unstructured Negotiation Protocol*. This is the negotiation protocol that we have used used for our experiments in chapters 6 and 7.

Most previous studies in Automated Negotiations have made use of the AO protocol, or something similar. Alternative protocols are proposed for example in [An et al., 2006]: it describes multilateral negotiation with 1 buyer and n sellers. The buyer maintains a separate negotiation thread with each seller. Each of these threads however still follows an AO protocol, so the agents are still restricted. In [Ortner, 2012] bilateral negotiations are modeled in continuous time, without a strict protocol. They assume that the decision of an agent to make a proposal is determined by external factors, which they model as a random variable. Also [Serrano, 2008] describes several alternative protocols for multilateral negotiations.

We consider however none of these protocols satisfactory for two reasons: firstly, because they make a strict distinction between buyers and sellers, which may not always exist in true negotiations (e.g. in the stock market people act both as buyers and as sellers). Secondly, they seem to be designed with the specific goal of making life easier for the designer of the experiment rather than for the agents themselves, ignoring the fact that in real-world negotiations the negotiators are autonomous and may decide not to follow the protocol. Therefore, we have defined a new protocol which is much less strict.

The Unstructured Negotiation Protocol applies to multilateral negotiation scenarios in which a deal may involve any number of agents. When an agent proposes a deal this proposal is sent to all the other agents participating in it. Other agents, which do not receive the proposal, do not know anything about it, until the deal is confirmed (if ever). The agents are committed to the deal once *all agents participating in it* have accepted it. At each moment each agent α_i can propose any deal, or accept any deal earlier proposed to α_i by any other agent α_j. When an agent has proposed or accepted a deal it is still allowed to withdraw this proposal or acceptance again, as long as the deal has not yet been confirmed.

As explained before, time is modeled as a series of discrete time steps. We assume these time steps are very short in order to approximate a model with continuous time. We assume there is some agreement space Agr, which can be any kind of set, a finite set of agent identifiers $A_n = \{a_1, a_2, \ldots a_n\}$, and a function pa that maps every deal $x \in Agr$ to a set of agents $pa(x) \in A_n$ called the set of participating agents. In each time step t, all agents simultaneously send a message of one the following forms:

- $(a_i, J, accept(x), t)$

- $(a_i, J, \mathit{reject}(x), t)$
- $\mathit{none}_{i,t}$

with $x \in \mathit{Agr}$. A deal is **confirmed in round** t_j if j is the smallest number for which all of the following are satisfied:

- For each participating agent $\alpha_i \in \mathit{pa}(x)$ there is some round t_{k_i} with $k_i \leq j$ in which α_i has sent the message $(a_i, J, \mathit{accept}(x), t)$ for some J
- No participating agent of x has at any time sent the message $(a_i, J, \mathit{reject}(x), t)$ at any time t_k with $k \leq j$, with some $J \subseteq \mathit{pa}(p)$
- There is no deal y that is incompatible with x and for which the previous two predicates have been true in in any round t_i with $i < j$.

A deal is **confirmed** if it was confirmed in any round t_j. This means that if at some point all participating agents have agreed with the deal x, and no participating agent has rejected it, it is confirmed, except when some other deal y had already been confirmed earlier on which is incompatible with x. Furthermore, note that after an agent has made a proposal but he changed is his mind, and the proposal is not yet confirmed, then he can reject it in order to prevent it from becoming confirmed. However, once it is confirmed, it stays confirmed, even if one of the participating agents sends the message $(a_i, J, \mathit{reject}(x), t)$.

In most literature on negotiations, one assumes that one agent 'proposes' a deal, and then another agent may or may not 'accept' the deal. We, however, refer to both actions as a 'propose' in order to simplify the formalization. Similarly, if an agent has made a proposal and then changes his mind, it is usually said he 'withdraws' the proposal, while in this paper we do not make a distinction between 'withdraw' and 'reject'. Furthermore, the fact that each player has the option to send the 'none' message simply means that agents are never obliged to propose or reject a deal. At any time they may simply choose to remain silent.

3.5.1 Properties of this Protocol

We now stress a number of important properties of this protocol. It is important to note that all of these properties indeed follow implicitly from the definitions above.

A proposed plan may involve more than two agents. This is different from most previous work in automated negotiations as one usually assumes only bilateral deals, even if there are more than two agents negotiating.

A proposal may be sent to any subset J of agents. However, if an agent α_i that participates in the proposed deal is not contained in J it will never receive the proposal and therefore never be able to accept it. Therefore it does not make sense to send a proposal to J if J does not contain $\mathit{pa}(x)$. Nevertheless we do

not force the agents to include $pa(x)$ in J, because we leave this responsibility to the agents themselves.

Agents can make more than one deal. Negotiations do not stop after a deal has been made, so agents can continue making more deals. However, a new deal cannot be conflicting with any previously made deals (e.g. once you have sold a car, you cannot sell the same car again to another customer).

Agents can change their minds and reject proposals they earlier accepted, as long as they are not committed to them yet. When an agent accepts a deal, this is not considered a binding agreement until all other agents participating in the deal have also accepted it. Therefore, an agent can reject an earlier accepted deal to prevent getting committed to it. Note however that the last definition implies that once an agent is committed to a deal, it stays committed to it, even if it sends a 'reject' message afterwards.

The agents in this protocol do not take turns. An agent can accept or reject any proposal at any time; it does not have to wait for 'its turn'. Moreover, this means that after making a proposal an agent does not have to wait for a counter-proposal, it can already make new proposals even before any agent has replied to the first proposal.

Agents are not obliged to reply to proposals. If an agent does not want to accept a received proposal, it may or may not explicitly reject it. The agent may simply ignore the proposal without ever replying. Therefore, when an agent has made a proposal and waits for a reply, it should decide for itself how long to wait for this reply. If it takes too long, the agent should consider the proposal as rejected, but it is up to itself when to do so.

When an agent makes a new proposal, it does not have to be compatible with any of the proposals it made before. This means that if one of the proposed deals is confirmed, other proposed plans may become unfeasible. It is up to the agents themselves to determine whether standing proposals are still feasible or not.

3.5.2 Relation to the Formal Model

Let us now describe this protocol in terms of our definitions above.

A world state in the Unstructured Negotiation Protocol is a function that maps to each deal in the agreement space a set of agents that have so far accepted that deal:

$$\epsilon : Agr \to 2^{A_n}$$

Initially, no agent has yet accepted any deal, so the initial state maps every deal to the empty set:

$$\forall\, x \in Agr : \epsilon_0(x) = \varnothing$$

The confirmation map is defined as:

$$conf(\epsilon) = \{x \in Agr \mid \epsilon(x) = pa(x)\}$$

This means that the set of confirmed deals consists of those deals for which all participating agents have accepted it.

Let $m_{i,t}$ denote the message that is sent by agent α_i at time t. Furthermore, let $acc(t,x)$ denote the identifiers of those gents that accept deal x at time t That is: $acc(t,x) = \{a_i \in A_n \mid m_{i,t} = (a_i, J, accept(x), t)\}$ for some set of receivers J. Similarly, $rej(t,x)$ denotes the set of identifiers of agents that reject deal x at time t. The state evolution map is then defined as:

$$\mathcal{F}(\epsilon_t, (m_{t,1}, m_{t,2}, \ldots, m_{t,n})) = \epsilon_{t+1}$$

with:

$$\epsilon_{t+1}(x) = \begin{cases} \epsilon_t(x) & \text{if } x \in conf(\epsilon) \\ \epsilon_t(x) \cup acc(t,x) \setminus rej(t,x) & \text{if } x \notin conf(\epsilon) \end{cases}$$

The means that whenever some agent α_i accepts a deal x the identifier a_i is added to the set of agents that have accepted x, and whenever an agent α_i rejects a deal x, its identifier a_i is removed from the set of agents that have accepted x. However, this only works as long as the deal x has not yet been confirmed. Finally, the permission map is defined as:

$$\begin{aligned} \mathcal{G}(\epsilon, a_i, t) = \{(a_i, J, accept(x), t), \mid x \in Agr\} \cup \\ \{(a_i, J, reject(x), t), \mid x \in Agr \setminus conf(\epsilon)\} \cup \{none_{i,t}\} \end{aligned} \quad (3.7)$$

meaning that any agent can always accept any deal, and any agent can always reject any deal, as long as that deal has not been confirmed yet. Furthermore, any agent may always remain silent.

3.5.3 Motivation for this Protocol

The reason that we have chosen this unstructured protocol is that we think that it resembles the way people negotiate in the real world. This protocol may be considered inconvenient for designers of negotiation agents, but this reflects the problems that negotiators also face in the real world. For example, if you make somebody an offer by e-mail, you have no guarantee that the recipient will ever reply to your mail. If he does not reply, you never know for sure whether the receiver is still deliberating over the offer, or is simply ignoring it. An agent implemented for the Unstructured Negotiation Protocol is therefore much more robust against unexpected human behavior.

Also, the possibility of making several proposals that are mutually incompatible is very common in the real world. Think for example of a real estate vendor that offers a house to several potential customers. Obviously, he cannot sell the same house to all of them, so the customer who reacts first, or bids the highest price, wins. For all other costumers the deal then becomes unfeasible.

3.5.4 Notary Agent

A problem that one may encounter with this protocol in practice, is that messages may arrive at different agents in a different order, which may cause agents to disagree on which proposals have been confirmed, and which are considered unfeasible. For example, consider the following scenario:

1. agent α_1 proposes a deal x_1 to agent α_2 and α_3.
2. agent α_1 proposes another deal x_2, which is incompatible with $x1$, to agents α_2 and α_4.
3. agent α_2 accepts both proposals.
4. agent α_3 accepts x_1.
5. agent α_4 accepts x_2.
6. The acceptance of x_1 arrives at agent α_1 before the acceptance of x_2. Therefore, agent α_1 considers x_1 as confirmed, and considers x_2 to be unfeasible.
7. However, the acceptance of x_2 arrives at α_2 before the acceptance of x_1, so agent α_2 considers x_2 as confirmed and x_1 as unfeasible.
8. Agent α_1 and α_2 now have a conflict over which of the two proposals they are committed to.

For this reason, in practice one may need a *Notary agent*, which plays a similar role as a *Scene Manager* in EIDE. A notary agent is an agent that does not take part in the negotiations but listens to all the messages and determines which of the proposals is officially confirmed. It is important to note that the Notary itself does not have any preferences, but simply follows the rules above. The Notary therefore decides which proposals are confirmed, purely based on the order in which he receives the acceptance messages. The notary sends a confirmation message to all participating agents whenever a deal is confirmed.

3.6 Electronic Institutions

We will now compare our definitions with the concepts defined in the EIDE framework (see Section 1.4.3). We show that all our definitions are compatible with the definitions in EIDE.

In EIDE, every agent has a *name* and a *role*. We identify a pair (agent name, role name) with an agent identifier in our model. A role can then be seen as a set of agent identifiers, that is: all pairs (agent name, role name) with the same role name. Note that the set of agent identifiers is very large, since each participant can pick any agent name. However, protocols are defined in terms of roles, and since each agent identifier contains a role name, which can only be picked from a limited list, a protocol indeed defines the allowed messages for every possible agent identifier.

A *scene* in EIDE consists of a set of role names, a protocol, a set of world states and an initial world state. In EIDE protocols are partially defined as graphs that represent a finite state machine, and partially in terms of a set of abstract variables. The 'state' of the scene is therefore defined as the current state of the finite state machine together with the values of all scene variables. This corresponds to our notion of a 'world state'. The state evolution map in EIDE is implemented along the edges of the graph. That is: each edge may be labeled with one or more post-conditions that become true when a certain message is sent. Moreover, such a message causes the state of the finite state machine to change.

In EIDE the state of the scene may not only change whenever an agent sends a message, but may also change whenever an agent enters or leaves the scene. In our model this can be incorporated by defining special messages that represent the fact that an agent is entering or leaving the scene. Also, the fact that the state may change as a consequence of a timeout can be represented in our model, because the state evolution map in our model does (implicitly) depend on time, since each message contains a time stamp.

Note that there is no notion of a utility function in a scene. This is because the scene generally does not assume anything about the goals of the agents in the scene. Of course, a scene is usually designed with an idea of the goals of the agents in mind, but they are not explicitly represented in the scene. Each agent that enters the scene may have its own private utility function. This is a fundamental principle of the EIDE philosophy: one cannot know who will enter the institution and one cannot know what the participants' objectives are.

In our definitions we have allowed all nonempty possible subsets $J \subseteq A$ to be the set of receivers in a message. In EIDE, one can only define specific subsets. That is: J can consist of exactly one agent, or J can consist of all agents with a specific role, or all agents currently in the scene instance. Therefore, in our model the definition of a message is slightly more general than in EIDE.

Furthermore, a message in EIDE does not only have a sender, a set of receivers and content, but also has an 'illocutionary particle', which defines the objectives that the sender is trying to achieve with the message. In our model we do not define such a illocutionary particles. However, in our model one can consider the illocutionary particle as a part of the content, so the combination (particle, content) in EIDE corresponds to the notion of the content in our model.

In EIDE, in order to define a protocol, one needs to define an ontology, which defines what can be put in the content of the messages. In our model we did not explicitly define the notion of an ontology, we simply allowed any content in the messages. However, a protocol may implicitly define an ontology, because there may only be a limited set of messages that are feasible under the protocol. In the alternating offers protocol for example, all feasible messages have content of the form 'propose(x)' or 'accept(x)'. This defines an ontology containing the functions 'propose()' and 'accept()'.

One important, non-trivial fact about scenes in EIDE, is that a scene may not always finish. In theory it could run forever. It is exactly for that reason that

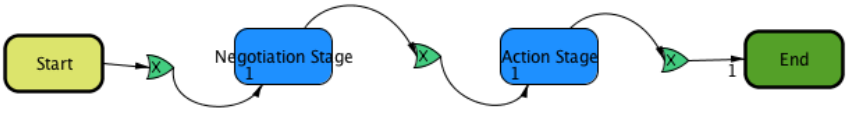


Figure 3.1: The Negotiation Game over a game G can be represented as a scene in which the agents follow a negotiation protocol and a scene in which the game follow the protocol associated with G.

in our model we have introduced *individual* deadlines for each agent, rather than a single deadline for the entire MAS. As soon as an agent's individual deadline has passed he is no longer interested in the further evolution of the scene, so he will try to leave.

One issue that we have ignored in our definitions, is the problem that in practice it can be very difficult to objectively determine the time stamp of a message. After all, the clocks of the several agents may not be synchronized, and messages take time to traverse the network, so the time at which a message is sent is different from the time at which it arrives at its recipients. Of course, one could simply request the agents to add the time stamp to the message when it sends the message, according to its own clock, but there is of course no guarantee that the agent will always do this correctly. This is a problem, because if one cannot objectively determine the time stamp of the message, one cannot even objectively determine the realized world states. However, in the EIDE framework this issue is solved because all messages need to pass a central Scene Manager. Therefore, it is the order in which the messages arrive at the Scene Manager that determines the current state of the scene. The time stamp of a message in our model should be seen as the time at which the message arrives at the Scene Manager.

The notion of a Negotiation Game can be represented very elegantly in EIDE, by defining two scenes, representing the negotiation stage and the action stage respectively. See also Figure 3.1. This allows us to implement the negotiation protocol N and the game G independently from each other.

3.7 Conclusions

In this chapter we have defined the formal model that we use throughout this thesis. We model time as a discrete set of instants, but these instants of time may be very short, so that the model may approximate continuous time. Agents are deterministic algorithms that send messages to one another depending on time and on the messages they have previously received from the other agents.

Moreover, we define the notion of a *world state* which can be seen as an equivalence class of message histories. Messages that may change the world state are also called *active messages*, or *actions*, while messages that do not change any world state are called *informative messages*. An action is called *infeasible* in a world state ϵ if it does not change that particular world state.

A *state evolution map* defines how the world state evolves as a function of the messages that have been sent, and a *permission map* determines which messages the agents are allowed to send and when. A *protocol* then consists of a set of agent identifiers, a set of world state, an initial world state, a state evolution map and a permission map. A protocol is said to be *regimented* if all actions that are not allowed, are infeasible.

We have defined a *game* as a protocol, together with a utility function and a deadline for each agent. Agents in a game are also called *players* and actions in a game are also called *moves*. We have defined a *negotiation protocol* as a protocol, together with a space of possible deals (the *agreement space*) and a *confirmation function* that defines for each world state of the protocol which agreements are considered confirmed, and therefore binding. A deal over a one-shot game G is an agreement between a number of players that each will only play a move from a certain subset of its complete set of moves.

Given a negotiation protocol N and a game G we have defined the *negotiation game* over G, denoted NG, which consists of two stages: the *negotiation stage* and the *action stage*. In the negotiation stage the players negotiate which moves from the game G they will make, and in the action stage they will make their moves, restricted by the deals they have committed themselves to during the negotiation stage. We have argued that under this model of negotiations, there is no clear and satisfactory notion of a reservation value.

We have introduced a new negotiation protocol, called *The Unstructured Negotiation Protocol*, which gives more freedom to the negotiators than the more commonly used Alternating Offers protocol. Specifically, negotiators are not obliged to reply to proposals, and after making a proposal negotiators are not obliged to wait for response from the opponents; they may make a new proposal whenever they want.

Finally, we have shown that many concepts of the EIDE framework are compatible with our formal model.

Chapter 4

Negotiation Problems

In this chapter we will describe three test cases for negotiations with large agreement spaces and non-linear utility functions. We describe these domains in order of increasing complexity. In the next part of the thesis we will describe negotiation algorithms that we have implemented for each of these domains.

4.1 The ANAC domain

The first domain we take a look at is the negotiation scenario that was applied in the Annual Negotiating Agents Competition 2014 (ANAC'14). In the agreement space of this domain, introduced in [Marsa-Maestre et al., 2009a, Ito et al., 2008, Marsa-Maestre et al., 2009b], the utility functions are given as non-linear functions over some abstract vector space. However, they are defined by means of constraints as in Section 1.3. This means that the utility value $f_i(x)$ of a given agent α_i and a given deal x can be calculated quickly by summing the values of all constraints that are satisfied by x.

The description of this domain does not refer to any underlying game or underlying world states. We could still fit this model into our previously described model of negotiation games, by introducing a dummy game G with dummy world states. We will not attempt to do this however, as this is a straightforward exercise and is not necessary for the purpose of this chapter.

We use the notation $[\mu, \nu]$, where μ and ν are integers, to denote the set of integers z such that $\mu \leq z \leq \nu$. We use the symbol α_1 to denote our agent, and α_2 to denote its opponent. Furthermore, we use $\mathcal{H}_{1\to 2}(t)$ to denote the set of deals proposed by α_1 until time t, and $\mathcal{H}_{2\to 1}(t)$ to denote the set of deals proposed by α_2 until time t.

4.1.1 The Agreement Space

The agreement space Agr_m in this domain is represented by an m-dimensional vector space, where m can be any positive integer. For each vector entry there

are 10 possible values.

$$Agr_m = [0, 9]^m$$

For a vector $x \in Agr_m$ we use the notation x_j to denote its j-th entry.

$$x = (x_1, x_2, ...x_m)$$

Both agents have a utility function f_i, which is a non-negative real function defined over the set of agreements:

$$f_i : Agr_m \to \mathbb{R}^+$$

Definition 38 *A **rectangular subspace** s is a subset of Agr_m defined as the Cartesian product of a set of m intervals $[\mu_j, \nu_j] \subseteq [0, 9]$:*

$$s = \prod_{j=1}^{m} [\mu_j, \nu_j]$$

In the ANAC domain, a constraint c is given by a pair (s_c, v_c) where s_c is a rectangular subset of Agr_m and v_c is a positive real number. As in Section 1.3 the characteristic function f^c of a constraint c is defined as:

$$f^c(x) = \begin{cases} v_c & \text{if } x \in s_c \\ 0 & \text{if } x \notin s_c \end{cases}$$

which is defined in terms of a set of constraints C_i:

$$f_i(x) = \sum_{c \in C_i} f^c(x)$$

The values of the constraints in the ANAC domain are normalized such that $f_i(x) \leq 1$ always holds. Both sets of constraints remain hidden to both agents, so an agent cannot directly calculate its own utility values. Instead, each agent α_i has access to an oracle that, for any given deal $x \in Agr_m$ returns its corresponding utility value $f_i(x)$. The agents cannot know anything about their opponents' utility functions (i.e. agent α_1 can request $f_1(x)$ from its oracle, but not $f_2(x)$). Note that, in principle, an agent can request the value of each deal in the agreement space. However, since the agreement spaces are extremely large (consisting of up to 10^{50} deals) this is obviously infeasible, so the agents can only explore a tiny fraction of the agreement space.

4.1.2 Other Parameters of the Competition

In the competition each agent engaged in several negotiation sessions, where each session involved two agents: $Ag = \{\alpha_1, \alpha_2\}$. In each session both agents had the same deadline which was set to 180 seconds. Each session would finish as soon as the agents made an agreement, or when the deadline had passed. The agents had to negotiate according to the alternating offers protocol (see Section 3.4). The agents alternate turns, and in each turn the agent whose turn it is may propose a deal x from the agreement space Agr_m (see Sect. 4.1.1) or accept

the last deal proposed by the other agent. This continues either until a deal is accepted or until a the deadline passes. The dimension m of the agreement space could be as high as 50, meaning that the agreement space could contain up to 10^{50} possible deals.

Each agent may take as much time as it likes before making the next proposal (or accepting the previous proposal). Therefore, the agent's decision is not only *what* deal to propose (or accept) but also *when* to propose. More precisely: when an agent finds a potential deal to propose it should determine whether it will propose that deal or whether it should continue searching for a better deal.

If the deadline passes without any agreement having been made each agent receives a certain utility value known as its *reservation value* rv_i. Otherwise, each agent α_i receives the utility value $f_i(x)$ associated with the deal x they agreed upon.

The utility function as defined above is in fact the *undiscounted utility*. In some instances of the ANAC domain the undiscounted utility of each agent is multiplied with a factor $\delta^{t_{acc}}$ to obtain its final score. The parameter δ is a value between 0 and 1, known as the *discount factor*, which differs per negotiation session. The value t_{acc} is the time at which the proposal x is accepted.

4.1.3 Limitations of the ANAC Domain

In principle every utility function over some vector space can be described in terms of constraints, as above. However, it is important to understand that in practice this is not always the case. In practice, utility functions may be given in an entirely different form, and although they can be converted to the form of this section, this may be a highly non-trivial task, impossible to fulfill in practice.

Take for example the case of a chess game. We can look at the space of all possible board configurations and assign the value 0 to every configuration in which white has a winning strategy, the value 1 to every configuration in which both players can enforce a draw, and the value 2 to every configuration in which black has a winning strategy. This is a well defined function over a finite domain, so in principle this function should be expressible in terms of constraints as above. However, this is obviously impossible in practice to explicitly write down this function in terms of constraints and calculate the value of any given board configuration.

4.2 The Negotiating Salesmen Problem

In this chapter we will define a new negotiation scenario that is much more complex than commonly used negotiation scenarios. We have used it a test case for our for the NB^3 algorithm that we will introduce in the next chapter. We call this problem the *Negotiating Salesmen Problem* (NSP). It resembles the multiple Traveling Salesmen Problem (mTSP) described in [Bektas, 2006], but with the main difference that each agent in the NSP is only interested in minimizing its individual path, while in the mTSP the agents form a social MAS, and intend

to minimize the total length of all the agents' paths together. Therefore, unlike the mTSP, the NSP is a game in which the agents act selfishly.

4.2.1 Definition

The idea of the NSP is that there is a map with a number of cities, and there are a number of agents (the salesmen) that need to visit those cities. All salesmen start at the same city (the home city), and all other cities should be visited by at least one agent. Initially, each city is assigned to one salesman that has to visit it. However, the salesmen are allowed to exchange some of their cities, which may enable them to decrease the distances they need to cover. For example: if a city v is assigned to agent α_1, but α_1 would have a shorter route if it would visit another city v' instead of v, which is assigned to agent α_2, then α_1 may propose to α_2 to exchange v for v'. If α_2 however also prefers to have v' over v then it will not accept this deal. If no other agent wants to accept v either, then α_1 is obliged to travel along city v. However, we impose the restriction that not all cities are allowed to be exchanged. The cities that can be exchanged are referred to as the *interchangeable cities*, while the cities that cannot be exchanged are called the *fixed cities*.

We will first define a family of games called Traveling Salesmen Games (TSG) and then define an instance of the NSP as the negotiation game over an instance of a Traveling Salesmen Game.

Let Gr be a finite, complete, weighted, undirected graph: $Gr = \langle V, w \rangle$ with V the set of vertices (the *cities*) and w the weight-function that assigns a cost to each edge: $w : V \times V \to \mathbb{R}^+$ and that satisfies the triangle inequality:

$$\forall\, a, b, c \in V : \quad w(a, c) \leq w(a, b) + w(b, c)$$

One of the vertices is marked as the *home city*: $v_0 \in V$. Each agent has to start and end its trajectory in this city. We use the symbol $\overline{V}$ to denote the set of *destinations*, that is: all cities except the home city: $\overline{V} = V \setminus \{v_0\}$. The set of destinations is partitioned into two disjoint subsets: F and I, so: $\overline{V} = F \cup I$ and $F \cap I = \varnothing$. They are referred to as the set of *fixed cities* and the set of *interchangeable cities* respectively.

The set of identifiers of the agents (also known as the *salesmen*) is denoted by $A_n = \{a_1, a_2, ...a_n\}$. A world state ϵ is a function that assigns each destination to a salesman: $\epsilon : \overline{V} \to A_n$. The set of cities assigned to agent a_i in world state ϵ is denoted as $\overline{V}_{\epsilon,i}$:

$$\overline{V}_{\epsilon,i} = \{v \in \overline{V} \mid \epsilon(v) = a_i\}$$

The definitions above imply that for each agent its set assigned cities can be further subdivided into: $\overline{V}_{\epsilon,i} = F_{\epsilon,i} \cup I_{\epsilon,i}$ where $F_{\epsilon,i}$ is defined as $\overline{V}_{\epsilon,i} \cap F$ and $I_{\epsilon,i}$ is defined as $\overline{V}_{\epsilon,i} \cap I$.

Given a graph Gr and an initial world state ϵ_0 a Traveling Salesmen Game TSG_{Gr,ϵ_0} is a turn taking game over some fixed number of rounds. Each round

t one player α_i can make a move o from the set $O_{i,t}$ defined as:

$$O_{i,t} = \{o \in M_{i,t} \mid J = A_n, c \in C_{i,t}\} \cup \{none_{i,t}\}$$

$$C_{i,t} = \{(a_i, v, a_j) \in \{a_i\} \times I \times A_n \mid a_i \neq a_j, v \in I_{\epsilon_t,j}\}$$

That is, o is either the 'none' move, or a message of the form

$$o_{i,v,j,t} = (a_i, A_n, c_{i,v,j,t}, t) \text{ with } c_{i,v,j,t} = (a_i, v, a_j)$$

where a_i is not a_j and v is a city currently owned by α_j. This has the interpretation of agent α_i taking the city from α_j The inclusion of the 'none' move means that players have the option, but not the obligation to take a city from another player.

The world state ϵ_t is updated according to $\mathcal{F}(\epsilon_t, o_{i,v,j,t}) = \epsilon_{t+1}$ where:

$$\epsilon_{t+1}(v) = a_j \text{ and } \forall\, z \in \overline{V} \setminus \{v\} : \epsilon_{t+1}(z) = \epsilon_t(z)$$

This is interpreted as follows: in the new world state the city v is assigned to agent α_j while all other cities remain with the same owner as in the previous world state.

The permission map is defined as:

$$\mathcal{G}(\epsilon, a_i, t) = \begin{cases} O_{i,t} & \text{if } t = i \ (mod\ n) \\ none_{i,t} & \text{otherwise} \end{cases}$$

In words: the players take turns. If it is agent α_i's turn it can make a move from $O_{i,t}$, otherwise it cannot make any move.

In order to define the agents' utility functions we first need to introduce some more definitions.

Definition 39 *Given any finite set $S = \{s_1, s_2, \ldots s_k\}$ of size k, and a permutation π of the integers 1 to k we say a* cycle *$T_{S,\pi}$ through S is an ordered sequence of size k consisting of the elements of S:*

$$T_{S,\pi} = (s_{\pi(1)}, s_{\pi(2)}, \ldots s_{\pi(k)})$$

We use the notation $\mathcal{T}_S$ to denote the set of all cycles through S.

Definition 40 *If S is a set of nodes from a weighted graph, with the weights denoted by $w(s_i, s_j)$, then the* ***length*** *$l(T_{S,\pi})$ of a cycle is defined as:*

$$l(T_{S,\pi}) = w(s_{\pi(k)}, s_{\pi(1)}) + \sum_{j=2}^{k} w(s_{\pi(j-1)}, s_{\pi(j)}) \tag{4.1}$$

With these definitions we can now define the cost function l_i for an agent α_i over the set of world states.

Definition 41 *The **cost function** l_i for an agent α_i is defined as:*

$$l_i(\epsilon) = min\{l(T) \mid T \in \mathcal{T}_{\overline{V}_{\epsilon,i} \cup \{v_0\}}\} \tag{4.2}$$

In words, this means that the cost of an agent α_i for a given assignment of cities ϵ is defined as the shortest path through the cities assigned to α_i, plus the home city.

The utility f_i of an agent can now be defined as its difference in cost between the initial world state and a given world state ϵ.

$$f_i(\epsilon) = l_i(\epsilon_0) - l_i(\epsilon) \tag{4.3}$$

Now, we can make two important observations. The first observation is that a TSG is by itself not interesting at all. Players want to minimize their paths, while taking a city from another player only increases a player's path length. The subgame perfect equilibrium consists of each player always playing the none-move.

Lemma 1 *In the subgame perfect equilibrium of any TSG every player always plays the move 'none'.*

Proof

The player whose turn it is in the last round would not want to take any city from any other player, since this could only increase its cost. Knowing this, the player in the second last round would also not want to take any city. By backward induction it follows that the same holds in every round of the game.

The second observation however, is that this equilibrium strategy is often inefficient and can therefore be improved if we allow agents to negotiate and sign binding agreements. Therefore, we define a Negotiation Salesmen Problem as the negotiation game over a TSG:

Definition 42 *Given some negotiation protocol N, an instance of the **Negotiation Salesmen Problem** (NSP) is a negotiation game over a TSG.*

The definition of the NSP as the set of negotiation games over a TSG implies that the agents go through a negotiation stage in which they can commit themselves (following some negotiation protocol) to play non-trivial moves during the action stage, which they would not play if there were no negotiations.

Agents α_1 and α_2 could agree that α_1 will take city v_1 from α_2, if in return α_2 will take city v_2 from agent α_1. Taken together, these two actions can be beneficial to both agents, even though none of these actions is beneficial by itself for the agent performing it.

We would like to stress that when an agent α_1 makes a proposal to another agent α_2 that benefits them both, α_2 may still decide to reject the offer, for

several reasons. Firstly because α_2 may be planning a counter proposal that reduces his individual cost even more, but that would become impossible after accepting α_1's proposal. Since both agents explore the agreement space independently they have a different view of their possibilities and bargaining power. Secondly, agent α_2 may also choose to make a deal with another agent α_3, which is incompatible with the offer made by α_1. Thirdly, agent α_2 may simply want to wait and continue searching for better deals.

4.2.2 The NSP as a Testbed for Automated Negotiations

The NSP is not meant as a realistic model for real traveling agents, but rather as a testbed to test algorithms for general, complex, negotiation scenarios. We will show now that a number of properties of real-world negotiations are also present in the NSP.

In many real world negotiations *the utility of a set of issues is non-additive.* That is: the value of a contract depends on the combination of issues. For example: when booking a holiday you need both a plane ticket to your destination and a hotel booking. The hotel booking is worthless without the plane ticket and vice versa. So the value of having both a plane ticket and a hotel booking is higher than just the sum of their individual values. This non-additivity also occurs in the NSP. For example: if some agent α_i currently owns cities v_1 and v_2 and there are two cities v_3 and v_4 which are both farther away from the home city than v_1 and v_2. Then α_i is not interested in exchanging one of its cities for one of the other two cities. However, it could be that v_3 and v_4 lie very close to each other and therefore it would be profitable to exchange *both* v_1 and v_2 for *both* v_3 and v_4.

The fixed cities in the NSP represent the fact that in real negotiations *different agents have different preferences.* Without fixed cities, every agent would have exactly the same utility profile: the path between cities v_1, v_2 and v_3 is equally long for every agent. However, because each agent also has its own fixed cities, every agent would have to traverse a different path even if they would visit the same interchangeable cities. One agent may prefer to visit v_1,v_2 and v_3 because they are close to his fixed city v_4, while another agent may prefer to visit cities v_5,v_6 and v_7, because they are closer to his fixed city v_8. If one wants to model a negotiation scenario in which the preferences of the other agents are unknown, one can impose the restriction that the position of any fixed city is only known to the agent that owns it.

In real-world negotiations it is often *very hard to assign a precise utility value to a deal.* This is captured in the NSP by the fact that for each possible deal the agent has to solve a traveling salesman problem for each agent involved in it. This is very hard, and often it is better to make only a quick approximation rather than to do an exact calculation. Of course, in the NSP the hardness of calculating utility stems from the fact that it is computationally hard, while for many real world problems it is caused by lack of information, but the point is that in both cases the utility can only be approximated.

Finally, we would like to stress that the size of the agreement space in the NSP

is very large. If the number of agents is n, and the number of interchangeable cities per agent is denoted by m, then there are in total $n \cdot m$ cities. A proposal in the NSP may assign an agent to every city, so there are n^{nm} possible proposals. We will see in Section 6.5 that we have conducted some experiments with the NSP, of which the largest instance involved 20 agents and 10 interchangeable cities per agent, so there were 20^{200} deals in the agreement space.

4.2.3 The NSP as a Package Delivery Problem

Although the NSP is primarily meant as an abstract test bed, we think that it is also a first step towards a model of a real world problem, namely the problem of package delivery. This means that if we manage to write an efficient algorithm for negotiation in the NSP domain (and we do, as we will see in Section 6), with some adaptations it may be applied to a real package delivery scenario.

Package delivery companies often operate in wide areas (such as entire countries or continents) that overlap with the areas of competing companies. Although this is convenient for customers that want to send packages over large distances, it is inefficient, since deliverers (postmen) may need to traverse unnecessary long distances. Efficiency would be improved if each postman could deliver all its packages in a small area. Therefore, it would be profitable for postmen to negotiate with each other about who will deliver which package, in real time. This would allow postmen to exchange packages even when they are already on their way to deliver them.

Alternatively, one could try to divide the packages among the postmen using a DCOP-solver to find solutions that distribute the packages in a fair way. The problem with this however, is that fair solutions are not always feasible, since package delivery is a discrete domain so there might not be any solution that gives the same amount of profit to every party. Moreover, the companies are competitors and may have different opinions about what can be considered 'fair'. This would make the package delivery companies distrust the system and could lead to conflicts.

Let us therefore discuss the adaptations that need to be made to the NSP in order for it to describe real package delivery.

Utility

In the real world the cost of a postman is not simply given by the length of its trajectory, but rather by the financial cost of traversing its trajectory, which depends for example on the amount of gas used and the presence of toll booths. Taking this into account however does not change the essence of the NSP, since we could simply re-interpret the weights of the edges of the graph as the financial cost associated of traversing them. Of course, the true cost might not only be the financial cost, but could also include time, so we could define weights of the graph as a linear combination of time and money.

Furthermore, the postman may not receive the same amount of money for each package it delivers. Again, this is not a problem, since one could sim-

ply assign the value of a package to its destination vertex. This value is then subtracted from the cost of a path that passes this vertex.

Locations

To interpret the NSP as a real world problem, we should interpret the cities as generic locations, rather than real cities. In reality one can identify an infinite number of locations, but this can be overcome by only considering the destinations of packages that are currently in process to be delivered, the current locations of the postmen, and the locations of the post offices. This set of locations is dynamic, as clients might request the delivery of new packages at any moment.

Also, to make the problem more realistic we should drop the assumption that all agents start in the same home city, as real postman are spread out across their area of operation. The initial location of each postman is then simply its current location at the time of negotiations. If the negotiation algorithm is running continuously, it should regularly update the positions of the postmen.

Constraints

Furthermore, one has to take into account that postmen are subject to certain constraints. For example, there is a maximum on the number of packages it can carry at the same time, depending on their weights and sizes. Moreover, postmen are constrained in the amount of time they can (or want to) work.

Non-linearity

Although the NSP is already a non-linear problem, since the length of a trajectory is a non-linear function of the co-ordinates of the visited cities, we encounter yet another form of non-linearity when we are dealing with real-world problems. This is because, given the weight of a certain trajectory, the true utility of this trajectory might be a non-linear function of this weight.

For example: a postman may be willing to work 8 hours a day for 80 Euro, but not be willing to work 12 hours a day for 120 Euro, because the 4 extra hours are much more tiresome then the first 8 hours, so the postman demands a higher salary per hour for these extra hours. Utility is then a non-linear function of time and money.

Utility Learning

Finally, we should take into account that a postman has emotions that determine how much it values time and money. This makes it almost impossible to find an exact utility function that expresses the postman's preferences. We should therefore develop a system in which the user can express its preferences, in a discrete and qualitative way, from which the system can approximate a utility value. A postman could pre-define some of its preferences, but could also refine its preference representation during the negotiations. The system can suggest

several solutions to the user, who may then react by indicating which of them it prefers. The system can use this information to dynamically learn and adapt a representation of the user's preferences. Since this happens in real time, the learned preference profile might even reflect the current emotional state of the user.

4.3 Diplomacy

Although the Negotiating Salesmen Problem is an interesting problem because it involves hard calculations, it is still a rather artificial problem and the calculation of the utility values still happens in a very straightforward manner. Therefore, in this section we will look at an even harder problem, which is the game of Diplomacy. Unlike the NSP, in Diplomacy the utility of one player may be affected by the actions of other players. Moreover, the set of confirmed deals usually does not entirely fix the players' actions; they merely impose restrictions on the players' actions. Therefore, we here encounter the full notion of a Negotiation Game as defined in Chapter 3, where determining the utility value of a deal involves Game Theoretical considerations.

4.3.1 Informal Description of Diplomacy

The full set of rules of Diplomacy is rather complicated, so we only give a simplified description. The differences with the full set of rules are not relevant here anyway.[1]

Diplomacy is a game widely played on the Internet.[2] It is a game for 7 players, with no chance moves and playing well requires good negotiation skills. There is no hidden information and all players make their moves simultaneously. The game is played over multiple rounds and each player begins with 3 or 4 units that are placed on a map of Europe. The map is divided into *provinces*, which can each hold 0 or 1 units. Some of the provinces (34 in total) are called *supply centers*. In each round each player must 'submit an order' for each of his units. Such an order can be either a 'move-to' order, meaning that the player tries to move the unit from its current location to a neighboring province, or a 'support' order, meaning that the unit will not move, but instead will give more strength to a moving unit. A support order is unsuccessful however, if there is another unit moving to the location of the supporting unit. In that case we say the support has been 'cut'. If two or more units are ordered to move to the same province, then only the unit that receives the most successful supports will succeed. A player 'owns' a supply center if the last unit that was located in it belongs to that player. If the owner of a supply center changes, the new owner will receive an extra unit in the next round, and the previous owner will lose one unit. A player is eliminated when he or she loses all units. A player wins

[1] A full description of the rules can be found at: https://www.wizards.com/avalonhill/rules/diplomacy.pdf

[2] http://www.playdiplomacy.com/

the game when he or she owns 18 supply centers (a 'solo victory'), but a game may also end when all surviving players agree to a draw.

The main difference between Diplomacy and other deterministic games like chess and checkers, is that in Diplomacy players are allowed to negotiate with each other and form coalitions. Each round, before the players submit their orders, the players can make agreements about the orders they will submit. Typically, players may agree not to attack each other, or they may agree that one player will use some of his or her units to support a unit of the other player. Although the ultimate goal for each player is to achieve a solo victory, it is very common that the players in a coalition agree to end the game in a draw when all other players outside that coalition are eliminated.

In a real Diplomacy game the agreements made between the players do not have any formal consequences. Players may break their promises so all agreements are based on trust. The notion of trust however, is beyond the scope of this thesis. Therefore, we will here assume that all players always obey their agreements. Also, we will not look at the problem of coalition formation. We assume coalitions are fixed from the beginning of the game and players do not break away from their coalitions.

4.3.2 Formal Description of Diplomacy

We will now define the one-shot game *Dip*. The idea behind this definition is that a single round of Diplomacy can be seen as a negotiation game over *Dip*, as defined in Section 3.4 (note however that a negotiation game itself is also modeled as consisting of many 'rounds' so one should not confuse the rounds of the negotiation game with the rounds of the Diplomacy game).

The game *Dip* is defined on a graph of which the vertices are called **provinces**. The set of provinces is denoted $Prov$ and we use the notation $adj(p, q)$ to state that provinces p and q are adjacent in the graph. The set of **supply centers** SC is a proper subset of $Prov$. Each player α_i has a set of units $Units_i$. The set of all units is denoted: $Units = \bigcup_i Units_i$. Each unit u has a **location** $loc(u) \in Prov$ and a set of possible orders Ord_u:

$$\begin{aligned} Ord_u &= Mto_u \bigcup Sup_u \\ Mto_u &= \{(u, p) \mid (p = loc(u) \vee adj(p, loc(u)))\} \\ Sup_u &= \{(u, u') \mid u' \in Units \wedge u \neq u'\} \end{aligned}$$

The set of moves O_i for a player α_i is the Cartesian product over the sets Ord_u for each of its units.

$$O_i = \times_{u \in Units_i} Ord_u$$

If o is a joint move then $\hat{o}$ denotes the set of all orders submitted by all players.

Definition 43 *A support (u, u') is called **valid**, if the owner of u' submits the order (u', p) for some province p adjacent to the location of u.*

$$val(o, u, u') \Leftrightarrow \exists\, p \in SC : (u', p) \in \hat{o} \wedge adj(p, loc(u))$$

Definition 44 *If a support order (u, u') has been submitted then we say that the unit u is **cut** if another unit tries to move to the location of u:*

$$cut(o, u) \Leftrightarrow (u, u') \in \hat{o} \wedge (u'', p) \in \hat{o} \wedge p = loc(u))$$

Definition 45 *The set of **successful supports** of u is defined as those orders that support u, and that are valid and not cut:*

$$sucsup(o, u) = \{sup(u', u) \in \hat{o} \mid val(o, u', u) \wedge \neg\, cut(o, u')\}$$

Definition 46 *The **force** exerted by unit u on province p is defined as:*

$$s(o, u, p) = \begin{cases} 1 + |sucsup(o, u)| & \text{if } (u, p) \in \hat{o} \\ 0 & \text{otherwise} \end{cases}$$

Definition 47 *We say a player α_i **conquers** a province p if α_i has a unit that exerts more force on p than any other unit:*

$$conq(o, i, p) \Leftrightarrow \exists\, u \in Units_i \;\; \forall\, u' \in Units \setminus \{u\} : s(o, u, p) > s(o, u', p)$$

We define the utility function for a player in the game *Dip* as the number of supply centers he or she conquers:

$$f_i^{Dip}(o) = |\{p \in SC \mid conq(o, i, p)\}|$$

4.4 Conclusions

We have presented three negotiation scenarios involving large agreement spaces and non-linear utility functions. In the first case, the ANAC domain, the negotiations are bilateral and utility functions are expressed as a sum over constraints. This means that for a given deal and agent, the utility value can be calculated quickly. Although in principle any function over a finite domain can be expressed in this way, it is in many cases impossible in practice to explicitly write down a given function in this way.

The second test case described (the NSP) is a new problem that we have introduced in order to test negotiation algorithms in domains where calculating the utility value of a deal is harder, as it involves solving an NP-hard problem. We think that this property makes the scenario much more realistic than other scenarios, because in a real negotiation setting a negotiator would also need time to evaluate each offer that is being made to him or her. In fact, we think that negotiating well depends more on the ability of a negotiator to make a proper evaluation of a proposal, rather than on his or her concession strategy. Furthermore, the NSP allows for multilateral negotiations in which all negotiators are considered equal (i.e. there is no distinction between 'buyers' and 'sellers'). We have explained how the NSP could be adapted in order to make it a more realistic model for real-world package delivery.

Finally, the third test case described is the game of Diplomacy. This game has existed as a board game and is still widely played online. This game involves negotiations that are even more complicated than those in the NSP, because a given deal does not directly determine a utility value, but just restricts the possible moves a player can make. This means that, in order to assign a value to such a deal, one needs to solve a Game Theoretical problem.

Part II

Negotiation Algorithms

Chapter 5

Applying Genetic Algorithms to the ANAC Domain

In this chapter we describe a negotiating agent that applies Genetic Algorithms for the exploration of the agreement space. It is called GANGSTER, which stands for Genetic Algorithm NeGotiator Subject To alternating offERs. It has successfully participated in the Annual Negotiating Agents Competition 2014 (ANAC'14) and finished in second and third place respectively in the two categories of the competition.

5.1 The Competition

In the competition each agent engaged in several negotiation sessions, where each session involved two agents: $Ag = \{\alpha_1, \alpha_2\}$. In each session both agents had the same deadline which was set to 180 seconds. Each session would finish as soon as the agents made an agreement, or when the deadline had passed. The agents had to negotiate according to the alternating offers protocol (see Section 3.4). The agents alternate turns, and in each turn the agent whose turn it is may propose a deal x from the agreement space Agr_m (see Sect. 4.1.1) or accept the last deal proposed by the other agent. This continues either until a deal is accepted or until a the deadline passes. The dimension m of the agreement space could be as high as 50, meaning that the agreement space could contain up to 10^{50} possible deals.

It is important to note that an agent may take as much time as it likes before making the next proposal (or accepting the previous proposal). Therefore, the agent's decision is not only *what* deal to propose (or accept) but also *when* to propose. More precisely: when an agent finds a potential deal to propose it should determine whether it will propose that deal or whether it should continue

searching for a better deal. If the deadline passes without any agreement having been made each agent receives its reservation value rv_i. Otherwise, each agent α_i receives the utility value $f_i(x)$ associated with the deal x they agreed upon.

In this chapter we adopt the convention that α_1 always refers to *our* agent, and α_2 refers to our opponent. Furthermore, whenever we use the term 'utility' without specifying an agent, we mean the utility function f_1 of our agent α_1.

5.2 Overview of the Algorithm

Let us first give a global overview of the algorithm before we go into more detail on each of its steps. Each turn the algorithm takes the following steps:

1. Calculate the *aspiration value* and *max distance* (Alg. 1, lines 1-2).
2. Decide whether to accept the previous offer made by the opponent (Alg. 1, lines 3-11).
3. Apply a *Global* genetic algorithm to sample the agreement space, and store the 10 proposals with highest utility (Alg. 1, lines 12-13).
4. Apply a *Local* genetic algorithm to sample the agreement space, and store the 10 proposals with highest utility (Alg. 1, lines 14-15).
5. Apply the offer strategy to pick the "best" proposal found by the GAs in the current round or any of the previous rounds (Alg. 1, line 16).

Algorithm 1 chooseAction(t, x, z)

Require: η_1, η_2

```
1: η1 ← calculateAspirationValue(t)
2: η2 ← calculateMaxDistance(t)
3: if f1(x) ≥ η1 then
4:    accept(x)
5:    return
6: else
7:    if f1(z) ≥ η1 then
8:       propose(z)
9:       return
10:   end if
11: end if
12: newFound ← globalGeneticAlgorithm()
13: found ← found ∪ newFound
14: newFound ← localGeneticAlgorithm(x)
15: found ← found ∪ newFound
16: proposeBest(found, η1, η2)
```

5.3 Acceptance Strategy

The acceptance strategy of GANGSTER is given in lines 3-11 of Algorithm 1. It depends on a function of time $\eta_1(t) \in \mathbb{R}$ that we call our *aspiration level*. We will explain how this value is calculated in Section 5.8. For now the only important thing to know is that it is a decreasing function of time that represents our agent's willingness to concede.

Let $x \in Agr_m$ denote the last offer proposed by the opponent, and let $z \in Agr_m$ denote the offer with highest utility among all deals made by the opponent in the earlier rounds:

$$\forall\, z' \in \mathcal{H}_{2\to1}(t) \;:\; f_1(z) \geq f_1(z')$$

If the utility $f_1(x) \geq \eta_1(t)$, then our agent immediately accept the proposal made by the opponent. If not, then our agent compares its aspiration level with the highest utility offered by the opponent so far, $f_1(z)$. If $f_1(z) \geq \eta_1(t)$ then α_1 *reproposes* that deal to the opponent. Note that this means that z is a deal that was earlier rejected by our agent (because at that time its aspiration level was higher), but now that the deadline has come closer it has lowered its standards since the risk of failure has become bigger and is now willing to accept it after all. Unfortunately, the alternating offers protocol does not allow an agent to accept a proposal from an earlier round, so it needs to be proposed again. If, on the other hand $f_1(z) < \eta_1(t)$ it means that it does not consider z good enough (yet), so it will apply the search strategy (Sec. 5.4) and offer strategy (Sec. 5.5) to determine a new proposal to make.

Let us now compare this strategy with existing acceptance strategies. In [Baarslag et al., 2013] a study was made of several acceptance strategies. The most common acceptance strategy they identified is named $AC_{prev}(1,0)$. In that strategy the agent α_1 compares the utility of the opponent's offer $f_1(x)$ with the utility $f_1(w)$ of the proposal w that agent α_1 would make if it would not accept x.

Our strategy is a variation of that strategy. However, instead of comparing the utilities of two proposals, our agent compares the utility of the opponent's proposal $f_1(x)$ with its aspiration level $\eta_1(t)$. The reason for applying this strategy is that α_1 can already determine whether or not to accept the opponent's offer x before it has determined its own proposal w. This has two advantages: firstly, this may save some time because determining the next proposal w can be time consuming. Secondly, it may not always be possible to find a deal w for which the utility is higher than the aspiration level. For example, the agent may only be able to find a deal w with utility $f_1(w) = \eta_1 - 0.1$. If it would apply $AC_{prev}(1,0)$ it would not accept the opponent's proposal x, even though x yields a utility value higher than our agent's aspiration level. That would be suboptimal, since the aspiration level is by definition the amount of utility that the agent considers high enough to accept.

Another improvement with respect to the $AC_{prev}(1,0)$ strategy is that we introduce the concept of *reproposing* (Alg. 1, lines 6-11). After all, the fact that

our agent rejected an earlier proposal from the opponent does not have to mean it would never accept it. The great advantage of reproposing, is that we know that the opponent has already proposed it, and therefore it is very likely that he will accept it. Moreover, the fact that it was already proposed to α_1 means that our agent does not have to search for a new proposal, it already has z readily available in memory.

5.4 Search Strategy

We will now explain the Genetic Algorithms (GA) that our agent applies to find good deals to propose. To apply a Genetic Algorithm one needs to model the elements of the search space as vectors (in the context of GAs also called *chromosomes*). Luckily, in the ANAC-domain the possible deals are already given as vectors, as defined in Section 4.1.1, so we do not have to put any effort in this modeling. Our GA consists of the following steps:

1. **Initial population:** Randomly pick 120 vectors from Agr_m. This is the initial population.
2. **Selection:** Pick the 10 vectors with highest utility from the population. These are the *survivors*.
3. **Mutation:** Pick another random vector from Agr_m and add it to the survivors.
4. **Cross-over:** For each pair (v, w) of these 11 survivors, create two new vectors v' and w' (so in total we create 110 new vectors).
5. **New population:** The new population now consists of the 110 new vectors from step 4, plus the 10 survivors from step 2. Go back to step 2, and repeat until convergence, or until we have iterated 10 times.

After a number of iterations the population may contain the same vector more than once. Therefore, when we pick the 10 vectors with highest utility in the selection step, we mean the 10 best *unique* vectors. In other words: we first remove any duplicates from the population and then pick the 10 best vectors. It may happen however that the population has evolved so quickly that no more new unique vectors are created by cross-over. In that case we say it has converged, and the GA is stopped.

Cross-over

A common way for a GA to apply cross-over, is to cut two vectors both in two halves, and then gluing the first half of the first vector to the second half of the second vector and vice versa (as explained in Section 1.3.6). We have however opted for a different kind of cross-over, in which random vector-entries are swapped. Suppose we have two vectors $v, w \in Agr_m$. The cross-over mechanism

will output two vectors v' and w' as follows. It first generates a random vector r of dimension m, where each entry r_i has the value 0 with probability 50% or the value 1 with probability 50%. Then, given the vectors v,w and r, the vectors v' and w' are defined according to:

$$\text{if } r_i = 0 \text{ then } v'_i = v_i \text{ and } w'_i = w_i$$

$$\text{if } r_i = 1 \text{ then } v'_i = w_i \text{ and } w'_i = v_i$$

The reason that we have chosen for this type of cross-over, is that (as far as the participants of the competition know) there is no relation between the variables in the domain. A constraint may for example involve the 3rd and the 7th variable, and there is no reason to assume that consecutive variables are stronger related than non-consecutive variables. This is reflected in our cross-over mechanism by the fact that it is symmetric under any permutation of the variables, whereas the regular cross-over mechanism has a strong bias towards the survival of consecutive sequences of values.

Global Search vs. Local Search

As we see in Algorithm 1, in each turn our agent applies two GAs. The first is called the *global* GA, and the second one is called the *local* GA. The difference is that the global GA picks vectors randomly from anywhere in the agreement space, while the local GA only picks vectors that are close to the last proposal made by the opponent. Specifically: there is a decreasing time-dependent function $\eta_2(t)$ and our agent only picks vectors for which the Manhattan distance to the last proposal made by the opponent is smaller than $\eta_2(t)$. The idea is that on one hand α_1 wants to maximize its own utility f_1 and therefore searches for good deals anywhere in the space, but on the other hand also needs to find proposals that are good for the opponent so it applies a local GA to find deals that are similar to the proposals made by the opponent.

5.5 Offer Strategy

In lines 12-15 of Algorithm 1 we see that the vectors returned by the GAs are added to a set of potential proposals. Next, it is the task of the offer strategy to determine which of those potential proposals should be proposed to the opponent (Alg. 1, line 16).

We use the notation $d(x, y)$ to denote the Manhattan distance between vectors x and y:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

For each vector x in the set of potential proposals our agent determines three properties, called *utility*, *distance*, and *diversity*, to decide which deal is the best to propose. The first of these properties, utility, is the most obvious: the higher our agent's utility $f_1(x)$, the better the deal.

Definition 48 *The **distance** $dist_t(x)$ of a vector $x \in Agr_m$ at time t, is the lowest Manhattan distance between x and any proposal previously made by the opponent:*

$$dist_t(x) = min\{d(x, y) \mid y \in \mathcal{H}_{2\to1}(t)\}$$

The idea is that, since our agent cannot know $f_2(x)$ it uses $dist_t(x)$ as a measure for the opponent's utility instead. If $dist_t(x)$ is low, then there is a high probability that $f_2(x)$ is high. Therefore, our strategy prefers to propose deals with low distance.

Definition 49 *The **diversity** $div_t(x)$ of a potential proposal $x \in Agr_m$ at time t is the shortest Manhattan distance between x and any of the proposals previously made by our agent:*

$$div_t(x) = min\{d(x, y) \mid y \in \mathcal{H}_{1\to2}(t)\}$$

Our offer strategy prefers proposing deals with high diversity because this has two advantages:

- If α_2 rejected proposal v, and the vector w is close to v (i.e. $div_t(w)$ is low), then it is likely that α_2 will also reject w. So our agent should avoid proposing deals that are similar to earlier rejected deals.
- By proposing more diverse offers, α_1 gives the opponent more information about its utility function f_1, making it more easy for α_2 to find proposals that are profitable to α_1.

Let us clarify this a bit more. Imagine that α_1's utility function has one very high peak. That is: there is a small area inside Agr_m where f_1 is very high. Then α_1 would be inclined to only make proposals from that area. However, if the opponent's utility f_2 is very low in that same area this will be an unsuccessful strategy. Now, suppose that there are a number of other areas of Agr_m where f_1 is less high, but still high enough to be proposed. Then by giving priority to deals with high diversity, we make sure that α_1 also makes proposals around those alternative peaks, hence increasing the probability that for some of its proposals the opponent utility f_2 is also high. Secondly, in this way α_1 reveals to α_2 the locations of the alternative peaks, which also makes it easier for α_2 to find deals that are profitable to α_1.

Let us now explain how utility, distance and diversity are used to determine which proposal to make. This procedure depends on two time-dependent functions: the aspiration level $\eta_1(t)$ and the maximum distance: $\eta_2(t)$. Let X denote the set of potential proposals found by the local and global GAs. Then we define the subset $Y \subseteq X$ as:

$$Y = \{y \in X \mid f_1(y) \geq \eta_1(t) \ \wedge \ dist_t(y) \leq \ \eta_2(t)\}$$

The deal that α_1 will propose next is then the element $y^* \in Y$ with highest diversity:

$$y^* = \arg\max\{div_t(y) \mid y \in Y\}$$

We see here that η_1 acts as a minimum amount of utility α_1 requires for itself, while η_2 acts as a minimum amount of utility that α_1 considers necessary to offer to α_2 (recall that low distance represents high opponent utility). Both of these functions are decreasing functions of time. This means that as time passes, α_1 requires less and less utility for itself, while it forces itself to offer more and more utility to α_2. After all, the closer it gets to the deadline, the more desperate the agent will get to make a proposal that gets accepted by the opponent. If there is more than one potential proposal for which the utility is high enough and the distance is low enough then α_1 picks the one with highest diversity.

The maximum distance η_2 is calculated based on how many good proposals have been found by the local GA. The more good proposals found, the more η_2 is decreased. The reason is that if our agent finds many proposals, then this is a sign that η_2 is too high. To be precise: it counts how many proposals were returned by the last local GA, for which $f_1(y) \geq \eta_1(t)$ and $dist_t(y) \leq \eta_2(t)$ both hold. If this number is higher than some parameter called DECREASE_THRESHOLD, then we decrease η_2 with 1, unless η_2 has already reached its lowest value of MINIMAL_MAX_DISTANCE. We have determined the parameters DECREASE_THRESHOLD and MINIMAL_MAX_DISTANCE through experiments with Gangster negotiating against itself, and set them to 4 and 5 respectively. The maximum distance is initialized to have the value $1.5 \cdot m$ where m is the dimension of the agreement space. Note that the higher the dimension of the agreement space, the higher the Manhattan distances can become. Therefore, the initial maximum distance indeed needs to be scaled with the dimension.

5.6 Motivation for Using Manhattan Distance

Let us now explain why we have chosen to use the Manhattan distance as our distance measure in the definitions. The idea is that we use distance to measure the difference between the utility values of two deals. The closer two deals x and y are, the more likely that $f_i(x)$ is close to $f_i(y)$. Indeed, the utility of a deal is determined by the constraints that it satisfies and if two deals are close to each other then they are likely to satisfy the same constraints. The question however, is which distance measure best reflects the similarity in utility values.

Let c be a constraint that is satisfied by x, that is: $x \in s_c$. Now for each variable x_j the constraint defines two integers μ_j and ν_j which are unknown. This means that if we increase or decrease x_j by 1, there is a probability that x will no longer satisfy the constraint as x_j may 'drop out' of the interval $[\mu_j, \nu_j]$. If x_j is in the interval $[\mu_j, \nu_j]$ then we denote the probability that $x_j + 1$ or $x_j - 1$ is not, by p:

$$P(x_j \pm 1 \notin [\mu_j, \nu_j] \mid x_j \in [\mu_j, \nu_j]) = p$$

Since we have no reason to assume that any variable $x_j \in \{x_1, \ldots x_m\}$ is different from any other variable, we can assume that p is equal for each x_j. Then the probability of no longer satisfying the constraint after making l steps is p^l,

independent of the directions of these steps. Specifically: it does not matter whether we take two steps in the $j = 1$ direction or two steps in the $j = 2$ direction, or one step in the $j = 1$ direction and one step in the $j = 2$ direction. In other words, the probability that y satisfies c equals $p^{d(x,y)}$.

Note that this is a direct consequence of the fact that constraints are defined by *rectangular* subspaces. If they had been defined by spherical subspaces for example, then the same reasoning would have lead us to use Euclidean distance.

5.7 Motivation for Using of Genetic Algorithms

Let us now explain why we think the use of Genetic Algorithms was a natural choice for the given domain, instead of other search techniques such as tree search.

Genetic Algorithms work well in domains where it is easy to find good partial solutions, simply by picking random samples, and where the combination of good partial solutions often yields even better solutions. In that case the cross-over mechanism can combine the good features of one vector with the good features of another vector. We show that both these criteria are satisfied in the ANAC domain.

First, we show that it is easy to find good partial solutions. We define the volume of a constraint as the fraction of vectors in the entire agreement space that satisfy the constraint.

Definition 50 *The **volume** $vol(c)$ of a constraint c is defined as:*

$$vol(c) = \frac{|s_c|}{|Agr_m|}$$

Note that when randomly picking a vector from Agr_m the probability that it will satisfy some constraint c is equal to $vol(c)$.

Definition 51 *The **defining dimensions** of a constraint c are the indices $j \in [1, m]$ for which $[\mu_j, \nu_j] \neq [0, 9]$.*

In the example domains provided to the participants before the competition no constraint had more than 4 defining dimensions. This means that the volume of any constraint could never be smaller than 10^{-4}. For most constraints however, the volume was much larger, often in the range between 0.1 and 0.5.

Furthermore, we recall from Section 1.3.6 that cross-over strongly increases the probability of finding solutions satisfying more than 1 constraint, if the value $1+0.5^{2 \cdot d-1} \cdot (s-1)$ is significantly larger than 1, where d is the number of defining dimensions of a constraint. Indeed, in the ANAC domain the value d was never higher than 4 and we chose s to be of size 120, which means that this value $1 + 0.5^{2 \cdot d-1} \cdot (s - 1)$ was around 2, in the worst case.

As explained in Section 1.3.6 the application of GAs does not require any knowledge about the constraints of the specific problem instance, which can be seen both as an advantage and as a disadvantage. In the case of the ANAC competition this is an advantage because indeed we only know the general structure

of the problem instances, as given in Section 4.1.1, but we know nothing about the specific instances.

5.8 Aspiration Level

The calculation of the aspiration level η_1 is a bit more complex than the calculation of η_2. Let u^* denote the highest utility that the opponent has offered to α_1 so far.

$$u^* = max\{f_1(x) \mid x \in \mathcal{H}_{2\to1}(t)\}$$

Then we can assume that α_1 can achieve at least this utility value, because it could simply accept x, or repropose x to the opponent, and assume the opponent would accept it. Now, our strategy is to decrease the aspiration level in such a way that it reaches the value u^* at the deadline. However, we do not want it to decrease linearly, because that might make it too easy for the opponent to guess how much α_1 is willing to concede. Instead, η_1 first decreases slowly, and then decreases faster and faster as time passes.

Let t be the current time (i.e. the time for which we want to calculate $\eta_1(t)$) and let t' be the time at which we last calculated η_1 (so $t' < t$). In order to determine $\eta_1(t)$ we first calculate a 'reference value' g_t. Given this value we calculate $\eta_1(t)$ by linearly interpolating between the values $\eta_1(t')$ at t' and g_t at time 1. To be precise:

$$\eta_1(t) = \tfrac{1-t}{1-t'} \cdot (\eta_1(t') - g_t) + g_t$$

Although this is linear, the overall behavior of $\eta_1(t)$ will be non-linear, because g_t itself also decreases in time. It starts half way between the 1 (the maximum utility) and u^*, that is: at $t = 0$ we have $g_0 = \frac{1}{2} + \frac{1}{2}u^*$, and decreases towards u^*, as follows:

$$g_t = (g_0 - u^*) \cdot (1-t)^{0.3} + u^* = (\tfrac{1}{2} - \tfrac{1}{2}u^*) \cdot (1-t)^{0.3} + u^*$$

In the case that there is a discount factor δ, we need to concede a bit faster, so in that case we aim to reach g_t not at the deadline, but at an earlier time t^*. Since the lower the discount factor the faster we need to concede we have chosen to set $t^* = \delta$. The formulas above then become:

$$\eta_1(t) = \tfrac{\delta-t}{\delta-t'} \cdot (\eta_1(t') - g_t) + g_t$$

$$g_t = (\tfrac{1}{2} - \tfrac{1}{2}u^*) \cdot (\delta - t)^{0.3} + u^*$$

Note that not having a discount factor, is equivalent to saying that $\delta = 1$, so these formulas are a generalization of the previous ones. The value of 0.3 here is a parameter that we have determined by trial-and-error.

5.9 Conclusions

We have introduced an agent, called Gangster, that makes use of Genetic Algorithms in order to negotiate over a domain where the utility functions are given in terms of constraints, which are in turn defined by rectangular subspaces. The trade-off between maximizing the agent's own utility and maximizing the opponent's utility was solved by using an time-based aspiration level for our agent's own utility, and at the same time demanding that the proposed deals were close to earlier proposals made by the opponent. If both criteria are met by more than one deal, then our agent proposes the deal that is most different from earlier proposals made by our agent. Furthermore, we have introduced a new acceptance strategy.

Our agent has participated in the ANAC'14 competition, which consisted of two categories: the individual category, in which the agents where ranked according to the individual utility they obtained, and the social category in which agents were ranked by the social utility, which is the sum of the agent's own utility and its opponent's utility. Gangster ended in third place in the individual category, and in second place in the social category, among more than 20 participants. We conclude that our agent is a good negotiator and that Genetic Algorithms are a good search technique for the given domain.

However, although the negotiation domains were very large, we think that this may have had only very little influence on the success of the negotiators. The reason for this belief is that when testing our GA on the largest test domain it was often able to find deals with $f_1(x) > 0.95$ (the highest possible value was 1) in less than 70 ms., on a standard desktop computer. This is very short in comparison to the total amount of 180 seconds available. An algorithm that is 10 times as slow would still be able to find more than enough good proposals to negotiate successfully. Therefore, we think the success of a participant depended more on its bargaining strategy than on its search algorithm. This is a pity, because the search was supposed to be the distinguishing property of this year's competition with respect to other years. We would therefore be very interested to know what the results would have been if the deadlines had been much shorter.

We think that the reason that the size of the domain was not that influential, is that the definition of the utility functions was too simplistic. In reality, constraints are not always given by rectangular subspaces and we think that constraints often involve more than 4 variables in any realistic domain.

Chapter 6

Applying Branch & Bound to the NSP

In this chapter we will introduce a new family of negotiation algorithms, called NB^3, that applies Branch & Bound to explore the agreement space. Unlike regular B&B however, in each node it stores upper- and lower- bounds for *every agent* rather than just for itself. As in the previous section, it applies a time-based negotiation strategy that considers two utility aspiration levels: one for the agent itself and one for its opponents. We describe an implementation of NB^3 designed for the NSP and present the results of experiments with this implementation.

6.1 Problem Statement

Before introducing the algorithm, we will first state the assumptions we have made, and motivate the approach we have taken.

6.1.1 Assumptions

The goal of the work presented in this chapter is to design an agent that is able to maximize its utility function by negotiating with other agents. We have made the following assumptions:

- Negotiations are multilateral.
- Every agent has a finite set of actions it can perform to change the current world state.
- Each agent has an individual preference relation over world states, defined by a utility function.

- Agents are selfish: each agent wants to take those actions that increase its own utility. The agents have no interest in maximizing other agents' utility functions or reaching a social optimum.
- The definitions of the utility functions are publicly known.
- The utility functions do not have an explicit formula, but are expressed in terms of a hard problem to solve and therefore calculating the utility of a world state or proposal is computationally expensive (such as the NSP as defined in Section 4.2).
- Agents can make binding agreements with each other about the actions each will take. This can improve the efficiency of their actions.
- The number of possible agreements is too large to apply exhaustive search (we have done experiments in which this number was as large as 20^{200}, see Section 6.5).
- The agents negotiate under the Unstructured Negotiation Protocol (see Section 3.5).
- There is a fixed deadline for the negotiations which is equal for all agents and known to all agents.
- The utility functions do not change over time (e.g. there are no discount factors).
- There is no mediator to help the negotiations.

Furthermore, we have made the following assumptions in this chapter, purely to keep the discussion and the notation simple. Our algorithm would work equally well without these assumptions.

- For any agent α_i if a deal in which α_i does not participate gets confirmed this does not influence the utility of α_i (this is rather strong assumption which in the case of Diplomacy does not hold, so this assumption will be dropped in Chapter 7).
- The order of execution of actions is irrelevant for the outcome of those actions.

Finally, we mention some important properties we do not take into account, although we think a realistic algorithm should take them into account. We leave them for future work.

- Non-numerical preferences: when negotiating with real people it is often not possible to express preferences as numerical values. Therefore, it would be better to use preference relations, rather than real-valued cost functions.

- Modeling opponent utility functions: in this thesis we assume that there is always some expression of the opponents' utility functions given. In real-world scenarios such an expression will not be given and should instead be determined with some specialized learning algorithm.

- Modeling opponent strategy: we do not make any attempt to model the concession strategy of the opponents. Our agent just uses a generic, fixed strategy that does not adapt to the opponents' strategies.

6.1.2 Complete Information

Although formally speaking the agents in this model have complete information in the sense that the utility functions are publicly known, we feel it is important to stress that in practice the information they have is far from complete. This is because the agents only know the *definitions* of the utility functions. In order to know the *values* of the utility functions however, they need to perform heavy, time consuming calculations. Given that the domains under consideration are very large, it is absolutely impossible for any agent to know all the utility values of all possible deals for all agents. Therefore, an agent will usually only make *approximations* of the utility values and can only do so for a *very small subset* of the agreement space.

6.1.3 Approach

The approach that we take is purely heuristic. We do not try to find any equilibrium strategies because we do not think calculating an equilibrium strategy in the real world is a feasible thing to do. Also, even in the scenario we treat in this paper we cannot think of any way to find formal game theoretical results without simplifying our assumptions so much that they become unrealistic in real-world applications. Let us state some arguments to support this:

- We do not make common assumptions such as the existence of a discount factor, which are often needed to obtain non-trivial results, because we do not think in real negotiations you would ever explicitly have such a discount factor (or know its value).

- Any result that provides hard mathematical guarantees would probably only refer to the test case under consideration (such as the NSP, Section 4.2), while our goal is to tackle negotiation problems in general.

- The number of possible deals the agents can make is very large: up to 20^{200} in our experiments, and there are no clear symmetries to reduce this considerably. Analyzing all possible options of a player is impossible.

- Since the players cannot calculate the utilities of 20^{200} deals, they need to apply a heuristic exploration of the space of possible deals to determine which ones to calculate. This exploration takes place continuously,

meaning that the knowledge the agents have about the world changes continuously, and since many solution concepts depend on the knowledge of the agents, such results would also change continuously.

- Even if one can determine an optimal strategy that tells a player to propose a deal with a given target utility, there is no guarantee that one can actually find such a deal with that utility value.

- Since each player explores the space of possible deals independently, each player discovers different possible deals. Therefore, a player does not know which proposals the other players have discovered so far, so there is lack of information about the opponents' options.

- Players do not only accept or propose deals, but also need to decide how long to search for good deals before making a proposal. There is no straightforward way to assign utility to such a decision. Of course one could define some kind of utility function for that, but the results would depend on that choice, therefore lose all generality, and therefore not satisfy our goals.

Note that these points indeed hold for all three the negotiation problems defined in Chapter 4. Especially noteworthy is the fact that the game Diplomacy has been played by many players worldwide for more than 50 years. If people had been able to find an optimal negotiation strategy for this game, it would not have been interesting to play it anymore. Therefore, the fact that it is still being played is evidence that so far no optimal negotiation strategy has been discovered for it.

6.2 The NB^3 Algorithm

In this section we present our negotiation algorithm called NB^3, which stands for *Negotiation Based Branch & Bound*. As explained in Section 1.3 Branch & Bound is a search algorithm capable of searching through large spaces efficiently and has reasonable solutions available at any time. When searching for the optimal solution, it is usually unnecessary to examine every possible solution. One can often, after examining only a partial solution, already discard all full solutions that extend this partial solution. Furthermore, as the algorithm is running, it yields solutions that get closer to the optimal solution, so at any time it has a solution available that at least approximates the optimum. The fact that B&B allows one to discard large parts of the search space and that it is an anytime algorithm makes it ideal to apply to our domain.

We next explain the various components of NB^3 assuming that it runs on the agent α_1. The other agents might also run a copy of NB^3, but they may just as well run any other negotiation algorithm, or they could even be human.

6.2.1 The Search Tree

As explained in Section 3.4 we assume there is a negotiation stage followed by an action stage. We assume that the action stage is a game over one or more rounds, and in each round each player has a set of possible actions O_i.

In this chapter we will model a deal as a set of committed actions, that is: $\overline{x} \subseteq \cup_{i=1}^{n} O_i$, where n is the number of agents, and O_i the set of allowed actions for agent α_i. This is interpreted as each of the agents α_i committing itself to execute his part of the deal $\overline{x} \cap O_i$ (recall that a deal x defines for each player a subset of all its possible strategies. In this case that subset would consist of those strategies in which each player α_i performs all actions in $\overline{x} \cap O_i$).

An agent that runs the NB^3 algorithm builds a search tree which is explored according to a best-first strategy. Each arc between a node and its parent is labeled by a certain action from the set of possible actions $\cup_{i=1}^{n} O_i$. Each node can then be interpreted in four equivalent ways:

- Each node nd represents the deal that consists of all the actions that label the arcs in the path from the root to nd; this deal is denoted by $path(nd)$. The root node corresponds to the empty deal (i.e. no commitments at all).
- Equivalently, each node represents a set of deals,[1] denoted $deals(nd)$, consisting of all deals that can be constructed by adding more actions to $path(nd)$. The root node then represents the entire agreement space and the children of a given node form a partition of the set of deals represented by the parent node. So if a node nd has children nd_1, nd_2, nd_3, then $deals(nd) = deals(nd_1) \cup deals(nd_2) \cup deals(nd_3)$.
- A third way of interpreting nodes is to see them as world states. The root node then represents the initial world state ϵ_0 and node nd represents the world state ϵ_{nd} that results from letting $path(nd)$ act on the initial world state: $\epsilon_{nd} = \mathcal{F}(\epsilon_0, path(nd))$.
- Finally, each node represents the set $\mathcal{E}_{nd}$ of all world states that can be reached by the deals in $deals(nd)$:

  $$\epsilon \in \mathcal{E}_{nd} \quad \text{iff} \quad \exists\, \overline{x} \in deals(nd) : \mathcal{F}(\epsilon_0, \overline{x}) = \epsilon$$

  The root node represents the set of all world states that can be reached by letting any deal act on the initial world state ϵ_0. If a node nd has children nd_1, nd_2, nd_3, then $\mathcal{E}_{nd} = \mathcal{E}_{nd_1} \cup \mathcal{E}_{nd_2} \cup \mathcal{E}_{nd_3}$.

To summarize: each node can be identified with a deal $path(nd)$, a set of deals $deals(nd)$, a world state ϵ_{nd}, and a set of world states $\mathcal{E}_{nd}$. The relationship between these objects is given by:

$$deals(nd) = \{\overline{x} \mid path(nd) \subseteq \overline{x}\}$$

[1] As mentioned before, to keep the discussion simple we assume that the order in which actions are taken is irrelevant for the outcome of the state of the world, even though our algorithm would work just as well without this restriction. Therefore, we see a deal as a set of actions, rather than a sequence of actions. So a set of deals is a set of sets of actions.

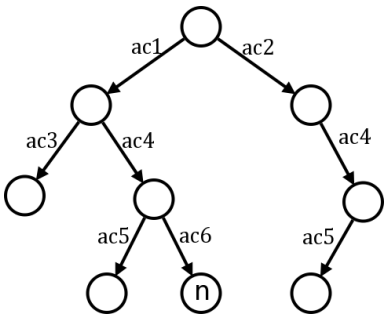


Figure 6.1: The search tree. Node n represents the deal consisting of the actions $ac1$, $ac4$ and $ac6$.

$$\epsilon_{nd} = \mathcal{F}(\epsilon_0, path(nd))$$

$$\mathcal{E}_{nd} = \{\epsilon = \mathcal{F}(\epsilon_0, \overline{x}) \mid \overline{x} \in deals(nd)\}$$

In Figure 6.1 the node marked n represents the deal consisting of actions $ac1$, $ac4$ and $ac6$, so $path(nd) = \{ac1, ac4, ac6\}$. The set $deals(nd)$ consists of all feasible deals that include these three actions. The world state ϵ_{nd} is the world state that would result from letting these actions are executed in the initial world state, and the set of world states $\mathcal{E}_{nd}$ consist of all the world states that can still be realized after these actions have been executed.

6.2.2 Making Decisions

In our environment the agreements among the negotiators have to be made during the search process. This is because an agent cannot wait until it finds the optimal deal before negotiating with other agents, as it might then be too late to sign any agreement with them: they might already have committed themselves to other, incompatible deals. Therefore, a trade-off exists between optimality and availability.

When α_1 receives a proposal from another agent, it has to decide whether to accept it or not, but it may not take this decision immediately. It may prefer to expand the tree a bit more, in order to see if it can find a better alternative to the proposed deal. The more the agent explores the tree before making any agreements, the more likely it is that it will find better deals. But, on the other hand, the less likely it becomes that it will get the other agents to accept those deals. How to solve this trade-off is a key decision when implementing an instance of NB^3.

Another important decision for any implementation of NB^3 is the question which node to split and how to split it. This may depend on the ongoing negotiation thread. For example, when an agent (say agent α_2) rejects a deal proposed

by α_1, this means the actions by α_2 should get less priority in future selections to be made by the algorithm. The idea behind this is that if you are under time pressure and there are several negotiation partners, you would be more inclined to negotiate with those partners that are showing more interest in reaching an agreement with you. Otherwise, you would be wasting your time. Moreover, if someone is not willing to concede, you can put him under pressure by suspending negotiations with him and continue your negotiations with others. In Section 6.3 and 6.4 we show how we have solved these issues for a particular implementation of NB^3.

6.2.3 Bounding

Branch & Bound algorithms require that each node *nd* compute upper- and lower bounds for the utility function in the subspace corresponding to this node. In the case of negotiations, however, an agent should not only take its own utility into account but also the utility functions of its negotiation partners. For this reason, each node does not only compute bounds for the utility of agent α_1, but also for every other agent.

The algorithm, running on agent α_1, is thus assumed to have a model[2] of the utility functions f_i of the other agents, and uses this model to calculate for every node *nd* and every agent $\alpha_i \in Ag$ the following bounds. Given a node *nd* we define $\epsilon_{nd} = \mathcal{F}(\epsilon_t, path(nd))$, and given a deal $\overline{x}$ we define $\epsilon_x = \mathcal{F}(\epsilon_t, \overline{x})$.

- For each node *nd* and agent α_i there is an **upper bound**: $ub_i(nd)$. This is the maximum utility α_i may receive from any deal that extends the deal $path(nd)$.

  $$ub_i(nd) = max_{x \in Agr}\{f_i(\epsilon_x) \mid path(nd) \subseteq x\}$$

- For each node *nd* and agent α_i an **intermediate value**: $e_i(nd)$. The utility agent α_i receives if it only commits to the deal $path(nd)$.

  $$e_i(nd) = f_i(\epsilon_{nd}).$$

- For each node *nd* and agent α_i a **lower bound**: $lb_i(nd)$. The minimum utility that α_i will receive from any deal that extends the deal $path(nd)$.

  $$lb_i(nd) = min_{x \in Agr}\{f_i(\epsilon_x)) \mid path(nd) \subseteq x\}$$

Lemma 2 *The upper bound is decreasing, and the lower bound is increasing. That is: for any node nd and any child nd′ of nd we have:*

$$ub_i(nd) \geq ub_i(nd') \text{ and } lb_i(nd) \leq lb_i(nd')$$

[2]We will not discuss how it could obtain such a model, because there are many ways to do this and depends on the domain. In the case of NSP this is simple, because it is known that each agent wants to minimize its path.

This implies that the upper bound for agent α_i of the root node is the highest utility agent α_i could ever achieve. Below we indicate the root node with nd_0.

Definition 52 *The* global lower bound glb_i *of an agent* α_i *is the lower bound for agent* α_i *in the root node.*

$$glb_i = lb_i(nd_0)$$

The following lemma follows directly from the definition of the reservation value, and the definition of the intermediate value.[3]

Lemma 3 *The reservation value of an agent* α_i *is equal to the intermediate value of the root node:*

$$rv_i = f_i(\epsilon_0) = e_i(nd_0)$$

The bounds defined here cannot always be calculated exactly, for two reasons. First, because α_1 may not have complete knowledge of the world state and of the other agents' utility functions f_i. And second, because the time restrictions often make it impossible for α_1 to compute these quantities exactly in real time, so α_1 may only be able to estimate them. To be clear, in the rest of this paper we will add a superscript index to any quantity if it does not represent the exact value, but only the approximation that the agent with that index makes of this quantity. For example, $ub_2(nd)$ indicates the theoretical value of agent α_2's upper bound of node nd, while $ub_2^1(nd)$ denotes the approximation that α_1 makes about agent α_2's upper bound.

The intermediate value of a node nd is the utility that the agent will receive if the deals that are confirmed during the negotiations consist of exactly the actions in $path(nd)$. So if $e_1(nd) \leq rv_1$ the deal $path(nd)$ is not profitable for α_1. Therefore we say a node nd is *rational* for agent α_1 iff $e_1(nd) \geq rv_1$.

Definition 53 *We say agent* α_1 ***believes*** *a node* nd ***to be rational for agent*** α_i *iff* $e_i^1(nd) > rv_i^1$.

Definition 54 *We say* α_1 ***believes*** *a node* nd *is* ***individually rational*** *iff it believes it is rational for all agents participating in* $path(nd)$.

6.2.4 Searching and Pruning

Since NB^3 performs a best-first search, we need a heuristic h that calculates a value for each node: $h(nd) \in \mathbb{R}^+$ to rank the nodes. We call this heuristic the *expansion heuristic*. Each time after splitting a node the algorithm picks the leaf node with the highest expansion heuristic from the tree to be split next. The value of h depends on the values of the bounds defined above. The precise way

[3] in Section 3.4.1 we argued that, depending on the domain, there may not be any satisfactory definition of the reservation value. Nevertheless, this definition is still correct, but the problem carries over to the fact that a generally satisfactory definition of the intermediate value may not exist; it needs to be determined separately for each domain.

in which h is calculated from the bounds might depend on the domain, but in Section 6.2.5 we give an example of such a function that is domain independent.

The upper bound is used for pruning: it defines the highest utility an agent could possibly receive from any descendant of the node. If $ub_i^1(nd) < rv_i^1$ for some agent α_i participating in $path(nd)$, it means that not only this deal is unprofitable for agent α_i, but also any deal that extends $path(nd)$ would be unprofitable for α_i, so in that case agent α_i would never agree with any deal descending from node nd and therefore this node can be pruned. Of course α_1 only has estimations of the true bounds for the utilities of the other agents, so it is essential that these estimations are good.

Note that for general B&B algorithms a node is pruned if its upper bound is below a global lower bound, which is defined as the highest lower bound among all leaf nodes. This is however not the case in NB^3. The reason for this is that, if we look at the node with the highest lower bound, we cannot be sure that its corresponding world state can actually be realized, since we are never guaranteed that the other participating agents will accept the corresponding deal. Therefore, we use the reservation value rather than the global lower bound to prune nodes.

One should note that the kind of pruning described here does not have to be done explicitly. After all, the heuristic h determines which node to expand next, so as long as we make sure that $h(nd) = 0$ whenever $ub_i^1(nd) < rv_i^1$ for some participating agent α_i, this node will always have lowest priority, which is essentially the same as being pruned.

However, NB^3 also applies another form of pruning which is done explicitly. Whenever agent α_1 gets committed to a deal $\overline{x}$, some actions from the action sets O_i may become infeasible, More precisely, given some deal $\overline{x}$ and some action o_i it may happen that $o_i \notin \mathcal{G}(\epsilon_0, \overline{x}, t)$, where $\mathcal{G}$ is the permission map of the action stage. All such actions that have become incompatible with the actions in $\overline{x}$ become unfeasible so α_1 can prune all nodes that have any of the incompatible actions in their paths to the root.

6.2.5 The Expansion Heuristic

A crucial property of NB^3 is the expansion heuristic h, that determines in which order the nodes are to be split. We will now discuss the default implementation of this heuristic, which is independent of the domain in which the algorithm is used. It can however be improved for specific cases where knowledge about the domain may help guiding the search.

Definition 55 *The set of* ***participating agents*** *of a node nd is denoted as* $pa(nd)$ *and is defined as the set of participating agents of its corresponding deal:* $pa(nd) = pa(path(nd))$.

The goal of NB^3 is to maximize the utility of agent α_1. However, α_1 needs the acceptance of other agents in order to make a deal, so it is not enough for α_1 to only look at its own utility. It is better to say that the search algorithm aims to

find the deal for which the *expectation value* of α_1's utility is maximized. That is: the deal for which the product of α_1's utility and the probability that all other participating agents accept it, is maximal.

Definition 56 *The **node value** $V_1^1(nd)$ of a node nd for agent α_1 is the utility of the node for α_1 times the probability that the corresponding deal gets accepted by all participating agents.*

$$V_1^1(nd) = f_1^1(\epsilon_{nd}) \cdot \prod_{\alpha_i \in pa(nd) \setminus \{\alpha_1\}} P^1(acc_i(e_i^1(nd))) \tag{6.1}$$

Here $P^1(acc_i(u))$ stands for the probability, estimated by α_1, that agent α_i would accept a deal x for which $f_i(x) = u$. Of course it is impossible to calculate this probability exactly, but we will see below how it can be estimated.

NB^3 aims to generate nodes with high node value. Now the question is: which node should be expanded in order to generate descendant nodes with high node value? The expansion heuristic of a node nd is therefore defined as the highest node value that we expect to find among the descendants nd' of nd.

$$h(n) = V_1^1(nd^*) = max\{\, V_1^1(nd') \mid nd' \in desc(nd)\} \tag{6.2}$$

with $n^* = \arg\max_{nd' \in desc(nd)} V_1^1(nd')$ and $desc(nd)$ denoting the set of nodes in the subtree under nd. Of course, when the algorithm is calculating $h(nd)$, the subtree under nd has not been generated yet so it cannot directly calculate the value $V_1^1(nd^*)$, but with some assumptions it can be estimated, as shown below. For this we need one more definition:

Definition 57 *The **offer value** off_i^1 of an agent α_i is the lowest utility that α_i would receive from any of the deals so far proposed or accepted by α_i.*

$$off_i^1 = min\{f_i^1(\epsilon_x)) \mid x \in accept_i^1\}$$

Here $accept_i^1$ denotes the set of all deals (known to α_1) that were proposed or accepted by agent α_i.

The offer value represents the lowest utility value that agent α_i has demanded so far. Now we will show how, for any node nd, NB^3 estimates the probability $P^1(acc_i(e_i^1(nd)))$. It is safe to assume that α_i would never accept a deal for which its utility $e_i^1(nd)$ is lower than its reservation value rv_i^1. Also it is reasonable to assume that this probability increases as the utility e_i^1 for agent α_i increases. Furthermore, since agent α_i has already accepted a deal for which it would receive a utility of off_i^1 one can assume that α_i would accept any proposal x for which $f_i^1(x) \geq off_i^1$. Therefore, this probability is modeled as a linearly increasing function from 0 to 1 between the values rv_i^α and off_i^α:

$$P^\alpha(acc_i(u)) = \begin{cases} 1 & \text{if } u \geq off_i^1 \\ \frac{u - rv_i^1}{off_i^1 - rv_i^1} & \text{if } rv_i^1 < u < off_i^1 \\ 0 & \text{if } u \leq rv_i^1 \end{cases} \tag{6.3}$$

With this formula, we can calculate the probability that an agent will accept a deal $path(nd)$ that yields a utility of $e_i^1(nd)$. However, in order to use (6.2) we need to calculate the probability that nd^* will be accepted, while nd^* has not been generated yet, and therefore we cannot know the values $e_i^1(nd^*)$. Therefore, for nd^* we estimate the probability of acceptance as follows (to simplify notation from now on we denote $e_i^1(nd^*)$ as e^*):

$$P^\alpha(acc_i(e^*)) = \int_0^\infty P^\alpha(acc_i(u)) \cdot P^\alpha(e^* = u)du \tag{6.4}$$

In order to calculate this, the algorithm then needs to estimate a probability distribution $P^1(e^* = u)$. From the definition of the upper bound it follows that e^* can never be higher than the upper bound $ub_i^1(nd)$ of nd. Furthermore, the algorithm assumes that the value e^* will not be lower than $e_i^1(nd)$ (so it makes the optimistic assumption that each node nd always has some descendant node with higher intermediate value). Therefore, the probability distribution $P(e^* = u)$, is modeled as a uniform distribution between $e_i^1(nd)$ and $ub_i^1(nd)$. We can then rewrite (6.4) (we further simplify notation by denoting $e_i^1(nd)$ as e and $ub_i^1(nd)$ as ub) as:

$$P^\alpha(acc_i(e^*)) = \frac{1}{ub - e}\int_e^{ub} P^\alpha(acc_i(u))du \tag{6.5}$$

Finally, we need to estimate the value of $f_1^1(\epsilon_{nd^*})$. We know that nd^* is by definition in the subtree under nd, which means that the deal $path(nd)$ is a subset of the deal $path(nd^*)$, so we can assume that $f_1^1(\epsilon_{nd^*})$ is not very different from $f_1^1(\epsilon_{nd})$. Therefore, we make the simplifying assumption that $f_1^1(\epsilon_{nd^*}) = f_1^1(\epsilon_{nd})$. Now we can calculate the expansion heuristic by combining (6.1) and (6.2):

$$h(nd) = f_1^1(\epsilon_{nd}) \cdot \prod_{\alpha_i \in pa(nd)\setminus\{\alpha_1\}} P^1(acc_i(e^*)) \tag{6.6}$$

which can be calculated explicitly by combining it with (6.3) and (6.5).

One very important remark we would like to state here, is that every time an agent makes a proposal or accepts a proposal its offer value can change, which means the expansion heuristic of every node in the tree changes. If an agent concedes, its offer value increases, and therefore the expansion heuristic of every node in which this agent is participating increases. In other words: if α_2 concedes, it becomes more attractive for α_1 to explore deals in which α_2 is participating. We see thus that not only is the negotiation influenced by the search, but also the other way around: *the search is influenced by the offers that are made by the other agents. Search and negotiation have become intimately intertwined with each other.* This is a unique property of NB^3.

6.2.6 Modeling Preferences of Other Agents

As explained above, the NB^3 algorithm requires a model of the utility functions of the other agents. We do not see this as a limitation because we believe that

knowledge of your opponents' preferences is essential in almost all negotiation scenarios. If a negotiator does not know anything about the preferences of its opponent, it is almost impossible to make any sensible proposal.

Agents may base their opponent models on prior knowledge of the domain and the opponents, on the proposals made by the other agents during the negotiations, on arguments exchanged between the agents, or on any other form of information provided by the other agents (see for example [Sierra and Debenham, 2007]).

In the case of NSP modeling the opponents' utility functions is easy, because we know that each agent is only interested in making its own path as short as possible and the positions of all cities are known. In many other problems it may be much more difficult to know the preferences of the other agents. Especially since agents might hide their preferences or lie about them.

We have intentionally chosen to use a domain in which opponent modeling is trivial, because we want to focus on the other aspects of the negotiation algorithm. We consider modeling the preferences of the opponents a domain specific matter and therefore we will not discuss how to do this for domains other than NSP. Moreover, the problem of modeling opponents has already been studied extensively, for example in [Hindriks and Tykhonov, 2008] and [Williams et al., 2011].

6.3 Negotiation Strategy

In this section we explain the negotiation strategy applied by NB^3 for negotiations under the Unstructured Negotiation Protocol. Although NB^3 applies a model of the opponents' utility functions, it does not apply any model of the opponents' negotiation strategies. We leave that as future work.

6.3.1 Proposing and Accepting

As the search tree is expanding, some nodes that are being generated represent individually rational deals. The agent needs to determine which of them to propose to the other agents.

The question when to accept an offer and what to propose has been solved theoretically, under specific assumptions such as the presence of a discount factor that decreases the utility as time passes and the assumption that the players follow the alternating offers protocol, in [Rubinstein, 1982]. For more general settings such as ours however, there is no known optimal strategy.

In classical bargaining models previously studied it is assumed the negotiators have strictly opposing interests. For example: a car salesman aims to sell the car for the highest possible price, while the client aims to buy it for the lowest possible price. The assumption of strictly opposing interests implies that 'concession' can be defined in two equivalent ways: either as 'demanding less utility from the opponent', or as 'offering more utility to the opponent'. In our situation this is no longer true. In our scenario the search for possible agreements

takes place simultaneously with the negotiations meaning that new solutions are being found during the negotiations that may dominate earlier found solutions. Therefore agents sometimes propose new offers that increase their own utilities as well as their opponents'. Moreover, in most existing work when an agent receives a proposal it only has two options: to reject it or to accept it. In our situation however, there is a third option: to continue searching for better solutions.

These differences motivate us to define a new negotiation strategy, in which the agent takes into account not one, but two aspiration levels. This negotiation strategy is in fact very similar to the one we described earlier in Chapter 5, however, in that case we assumed no knowledge about the opponent's utility whatsoever, so we used a distance measure to represent the opponent's utility. In the current chapter on the other hand we assume there is an explicit model of the opponent's utility, so we directly use that model to compare a deal with an aspiration level for the opponents. In order to keep the explanation simple we first present this negotiation strategy for the case of bilateral negotiations and then generalize it to the multilateral case.

6.3.2 Bilateral Negotiation Strategy

During the negotiations agent α_1 regularly needs to make a choice between making a new proposal, accepting a previous proposal from an opponent, or continue searching. The agent bases its decision on three values: the time t passed since the start of the negotiations, the normalized utility $\bar{u}_1^1$ it receives from a deal and the normalized utility $\bar{u}_2^1$ the opponent α_2 receives from a deal.

Definition 58 *The **normalized utility** of a deal x for agent α_i, is defined as follows:* $\bar{u}_i(x) = \frac{f_i(\epsilon_x) - rv_i}{gub_i - rv_i}$.

To make a decision, agent α compares these utility values with two values, denoted as $\eta_1^1(t)$ (the self-aspiration-level) and $\eta_2^1(t)$ (the opponent-aspiration-level) respectively, which are time-dependent functions. Note that although η_2^1 represents an aspiration level for the utility of α_2, this value exists in the mind of α_1. It is the amount of utility that α_1 considers necessary to offer to α_2.

Definition 59 *A deal x is **more selfish** than deal x' iff $\bar{u}_1^1(x) > \bar{u}_1^1(x')$. For a given time instant t we say x is **selfish enough** iff $\bar{u}_1^1(x) > \eta_1^1(t)$. Given a set of deals X, the deal $x \in X$ that maximizes $\bar{u}_1^1(x)$ is called the **most selfish** deal of X.*

Definition 60 *A deal x is **more altruistic** than deal x' iff $\bar{u}_2^1(x) > \bar{u}_2^1(x')$. For a given time instant t we say x is **altruistic enough** if $\bar{u}_2^1(x) > \eta_2^1(t)$. Given a set of deals X, the deal $x \in X$ that maximizes $\bar{u}_2^1(x)$ is called the **most altruistic** deal of X.*

Notice that 'selfish' and 'altruistic' as defined here are not necessarily each other's opposites. If deal x yields more utility than deal x', for both negotiators, x is more selfish *and* more altruistic than x'.

At given moments t separated by time intervals of fixed length, α_1 decides what to do: to propose a new deal, to accept a previously proposed deal, or to continue searching for better deals (the length of these intervals is a parameter of the algorithm). This decision is taken according to the following 6-step strategy:

1. First α_1 determines the set X of all deals it has found so far and believes to be individually rational.
2. Then, it determines the subset $Y \subset X$ of all deals in X that are altruistic enough.
3. If Y is not empty, α_1 picks the deal $x \in Y$ that is most selfish. If on the other hand Y is empty, then α_1 picks the deal $x \in X$ that is most altruistic.
4. Next, α_1 determines the set Z of all deals that have been proposed to it by other agents. From this set it picks the most selfish deal $x' \in Z$.
5. From the two deals x and x', it then picks the one which is most selfish.
6. Finally, α_1 checks whether the deal chosen in the previous step is selfish enough. If yes, then this deal will be proposed or accepted. If however this deal is not selfish enough, the agent will continue to search for better deals.

To summarize this: α_1 will only accept or propose any deal that is individually rational and selfish enough. It will only propose a new deal x if there is no standing proposal x' proposed to α_1 that is more selfish than x (because then it prefers to accept x'). And from all candidate deals it could propose, it prefers to propose the most selfish deal that is altruistic enough. If however no deal is altruistic enough, it prefers the deal that is most altruistic (also see Algorithm 6 in Section 6.4.5 for a description of this procedure in pseudo-code.)

Since α_1 should start selfish, and concede as time passes, $\eta_1^1(t)$ is a decreasing function, so that less selfish deals are proposed as time advances, and $\eta_2^1(t)$ is an increasing function, so that more altruistic deals are proposed as time advances.

Notice that in order to be proposed or accepted, step 6 requires that the deal is selfish enough, while, because of step 3, it does not need to be altruistic enough. This is because if α_1 has a deal that is probably not good enough for α_2 to be accepted, it does not harm much to try and propose it anyway. On the other hand, if α_1 would propose a deal that yields very little utility for itself, and it gets accepted by α_2, then α_1 is committed to this deal, which might make other, more profitable deals in the future impossible.

For the aspiration levels we have chosen the following expressions:

$$\eta_1^1(t) = 1 - \frac{e^{-\gamma_1 \frac{t}{t_d}} - 1}{e^{-\gamma_1} - 1} \tag{6.7}$$

$$\eta_2^1(t) = \frac{e^{-\gamma_2 \frac{t}{t_d}} - 1}{e^{-\gamma_2} - 1} \tag{6.8}$$

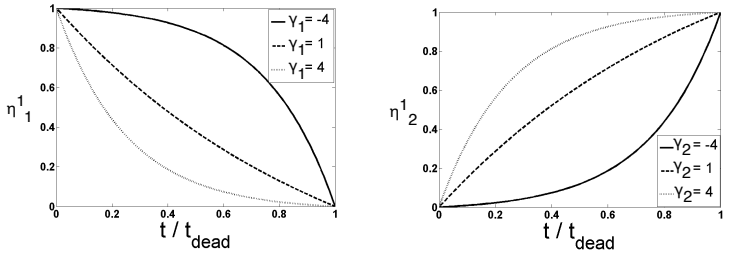


Figure 6.2: The graphs of η_1^1 and η_2^1 for several values of γ_1 and γ_2.

Their graphs are plotted in Figure 6.2. Notice that η_1^1 decreases from 1 to 0 and η_2^1 increases from 0 to 1. The higher the values of γ_1 and γ_2, the faster the agent concedes. Therefore γ_1 and γ_2 are called the *concession degrees*. The strategy of the agent can be adapted by adjusting these two parameters.

The fact that η_1^1 and η_2^1 go to 1 and 0 respectively makes this strategy a very weak one for bilateral negotiations, since it can be easily countered by any opponent. The opponent α_2 would simply not concede, but wait until η_2^1 is so high that α_1 will propose a deal that is highly favorable to α_2. However, one should keep in mind that this strategy is developed for multilateral negotiations. In the multilateral case, agent α_2 does not have the opportunity to wait until α_1 makes a highly altruistic offer, since α_2 has competition from other agents. If α_2 does not concede, α_1 might close deals with some of the other agents, leaving α_2 with nothing.

The algorithm intends to find deals for which the opponent-utility is as close as possible to the opponent-aspiration-level. The self-aspiration level imposes an extra criterion, that determines whether the deal is selfish enough to be proposed, or whether it is better to continue searching instead. This solves the trade-off problem discussed in section 6.2.2.

Finally, note that this strategy never rejects any proposal. Instead, it simply ignores bad proposals. The advantage of this is that our agent never has to worry about the question when to definitely reject a proposal and that it always keeps every proposal as an option to accept in the future. On the other hand, rejecting proposals would have the advantage that our agent could inform the other agents about its preferences which could improve the proposals made by them.

6.3.3 Comparison with Single Aspiration Level

To further justify why we are using *two* aspiration levels, we will now take a look at what would happen if one of the two aspiration-levels would be set to zero, so that there is effectively only one aspiration-level.

In general it can happen that the deal x chosen in step 3 is very unprofitable for agent α_1, even though it is the most selfish one. But step 6 then makes sure

that such a deal is not proposed because it is not selfish enough. Now if η_1^1 would be zero however, every deal would be considered selfish enough so even bad deals would be proposed. In simple negotiations, where the entire set of solutions is known, this would not be a problem because x would simply be the most selfish deal existing, so there would not exist any better solution for α_1.

However, in our situation, because the search for good solutions runs simultaneously with the negotiations, it is only sure that deal x is the most selfish solution *found so far*. It would therefore be better to continue searching than to propose the bad deal. This is exactly the reason why we have η_1^1: it determines whether the deal is selfish enough to propose, or if it is better to continue searching for a better deal before proposing it. The more time available, the better it is to continue searching. If however there is very little time left before the deadline, there is little hope of finding a better deal. Therefore, the selfishness-criterion should become weaker as the deadline approaches, so η_1^1 must be a decreasing function of time.

Now instead suppose that η_2^1 is always zero. Each deal would be considered altruistic enough and the agent would only have the following two options: propose the most selfish deal (if it is selfish enough) or continue searching until it finds a deal that is selfish enough. But in this way α_1 might never concede, because there might exist a lot of very selfish deals. An agent that does not concede at all is generally not a good negotiator, especially in a multilateral environment where the opponents may ignore α_1 and come to agreements with each other, without α_1.

6.3.4 Characterization of Strategies

Our concession strategy is parametric in the two concession degrees. We will now discuss how the various settings of these parameters would affect negotiations. We define four strategies by setting the values of γ_1 and γ_2 either high or low.

Greedy: low γ_1, low γ_2. Only proposes very selfish deals. If it hasn't found any deals that are selfish enough, it prefers to continue searching for them rather than to concede.

Lazy: high γ_1, low γ_2. Proposes very selfish deals, but if it can't find any, it will propose less selfish deals, rather than to search for better solutions.

Picky: low γ_1, high γ_2. This strategy is willing to propose altruistic deals, but only if they are also selfish, otherwise it prefers to continue searching. So it keeps searching until it finds a deal that is both very selfish and very altruistic.

Desperate: high γ_1, high γ_2. Concedes fast, even if it has to give up a lot of utility.

Roughly we can say that the higher the value of γ_1, the less the agent likes to search. The higher the value of γ_2, the more altruistic the deals are that the agent proposes (or is willing to accept). The Greedy strategy should only be played if the agent has little competition. If the agent knows it has a stronger position than its opponents it can use this strategy to exploit them. The Desperate strategy, on the contrary, should only be played in a highly competitive environment. If there is a lot of competition it is better to try to come to an

agreement as soon as possible, before the competition takes away all the good deals.

The other two are more moderate strategies. The Lazy strategy should be played if good deals are scarce. In such an environment it is not likely to find many deals that are better than the current options, so it is better to give up some utility than to continue searching for a better deal. If good deals are abundant, it is better to play the Picky strategy In that case, if the current options are not good enough, instead of giving up utility it is better to keep searching a bit more because it is likely that the agent will find some better deal.

From basic experimentation we have concluded that good values of the concession degrees for the NSP are $\gamma_1 = 2$ and $\gamma_2 = 4$ and have fixed these values for our further experiments in Section 6.5. We leave a more thorough experimentation to determine the best values for these parameters as future work.

6.3.5 Multilateral Negotiations

Things become more complicated when we want to apply our strategy to multilateral negotiations. In order to make the agent capable of multilateral negotiation, we have chosen a simplified model in which the agent treats the set of all opponents as if it were one opponent, and then follows the same concession strategy as above. It defines the opponent-utility of a deal as the product of all the normalized utilities of the other agents participating in the deal. When choosing whether to propose, accept, or wait, it applies exactly the same procedure as in the bilateral case, only the quantity $\bar{u}_2^1$ is replaced by $\bar{u}_{pa}^1$ with:

$$\bar{u}_{pa}^1(x) = \prod_{\alpha_i \in pa(x) \setminus \{\alpha_1\}} \bar{u}_i^1.$$

with the extra condition that $\bar{u}_{pa}^1(x)$ is zero if $\bar{u}_i^1$ is negative for any $\alpha_i \in pa(p)$.

This multilateral concession strategy does not take into account which agents are involved in the proposals α_1 makes. So when one proposal x_1, made by α_1, is not accepted, α_1 will try to make a new proposal x_2 that gives more utility to its participating agents than x_1 did, even though the agents participating in x_2 might be completely different from the ones in x_1. The idea behind this is that α_1 considers all agents as equivalent. After all, we assume that the agents are unknown, so we cannot distinguish between them. In the rest of this chapter we will denote the opponent aspiration-level for multilateral negotiations as η_{pa}^1 instead of η_2^1.

A possible way of improving the multilateral strategy, which we leave for future work, would be to store information about the opponents obtained during the negotiations. We could then make a profile of each opponent and use this to set a different opponent-aspiration-level for each individual opponent. Furthermore we could try alternative definitions of $\bar{u}_{pa}^1$, such as the minimum of the opponent-utilities. Again, we leave this as future work.

6.4 Branesal

Until now we have kept the description of NB^3 as general as possible. In this section however, we describe how we have implemented NB^3 to be applied to the NSP defined in Section 4.2. We call this implementation *Branesal (BRAnch and bound NEgotiating Salesmen ALgorithm).*

6.4.1 Calculating the Bounds

We will now explain how the bounds of the search tree are calculated in the case of the NSP. To calculate these bounds the agent needs to know the shortest path that goes through a given set of cities. It is however much too costly to calculate this length exactly. Therefore, we consider an estimation of the shortest path by calculating the *greedy path* instead.

Definition 61 *The **greedy path** through a set of cities is calculated as the path that goes from the home city v_0 to its nearest neighbor v_1, then from v_1 to v_1's nearest neighbor v_2, etc. until we have visited all cities and come back to v_0.*

So for any world state ϵ the path length $l_i(\epsilon)$ for agent α_i, is estimated by agent α_1 as the length of the greedy path through the set of cities assigned to α_i in the world state ϵ. This estimated value is denoted by $l_i^1(\epsilon)$. Accordingly, the estimated utility of an agent α_i is denoted $f_i^1(\epsilon)$ and is calculated as $f_i^1(\epsilon) = l_i^1(\epsilon_0) - l_i^1(\epsilon)$

This greedy heuristic may not be very accurate, but from experiments it turns out to work quite well in practice. We have tried to use more accurate heuristics, but the extra computations this involved were so costly that it was not worth using them.

In the NSP an 'action' consists of one agent giving one city to another agent. Each arc between two tree nodes is labeled by such an action, and the path from a node back to the root represents the deal consisting of all the actions that form the labels along the path. One can check that the definitions below are consistent with the general definitions of the bounds as given in section 6.2.3. The calculations of the intermediate values and the lower bounds are also given in pseudo-code in Algorithm 7.

Upper Bound

The upper bound $ub_i^1(nd)$ is the difference between $l_i^1(\epsilon_0)$ and the minimum path length that α_i could possibly achieve in any deal descending from node nd. The most optimistic scenario is the case in which α_i is able to give away all its interchangeable cities, except those that it has acquired in $path(nd)$. The idea is that once a city v is acquired from some agent α_j in $path(nd)$ it will not be given away again to some other agent α_k in any deal that descends from nd. This is because our agent would not consider such a deal because it would already consider an equivalent deal, in which v is given directly from α_j to α_k, somewhere else in the tree.

The upper bound of a node nd for an agent α_i is therefore calculated as $l_i^1(\epsilon_0)$ minus the length of the greedy path through the home city, the fixed cities owned by α_i and the cities given to α_i in any of the actions in $path(nd)$.

Intermediate Value

In order to calculate the intermediate value of node nd, we take the set of cities currently assigned to α_i, remove all the cities that are given away by α_i in any of the actions in $path(nd)$, and add all the cities that are acquired by α_i in any of the actions in $path(nd)$. We then calculate the greedy path through this new set.

Lower Bound

Although the specification of the NB^3 algorithm defines a lower bound for every node and every agent, the Branesal implementation does not calculate it. The reason for this is that the expansion heuristic defined in Section 6.2.5 does not use it. Other implementations of NB^3 may however use another implementation of the expansion heuristic.

6.4.2 Splitting

Besides the bounding and the expansion heuristic, a very important design-issue for any B&B algorithm is the question of how to split the nodes. Since a deal is built up from actions, it would be natural to split a node according to the several alternative actions that can be performed. This would mean adding a new node for each action that is compatible with $path(nd)$. However, in many cases this would result in a huge amount of child nodes, making the algorithm very inefficient (if there are m interchangeable cities per agent and n agents, then the total number of actions is in the order of $m \cdot n^2$). Therefore we have chosen an implementation in which an 'action' is split up in smaller components.

An action in the NSP consists of three components: an agent that gives away one of its cities (the donor) the city that is being given away, and the agent that receives the city (the acquirer). So instead of adding a child node for each possible action, we first add a child for each possible donor, then pick the best of these children (i.e. the child node with the highest expansion heuristic) and expand it by adding a child for each of the cities that the donor can give away. Finally, we pick the best of these nodes and add to it a set of children, each one of which corresponds to one of the possible acquirers. So each arc between two nodes is not labeled by an action but by a component of an action, and a path of three consecutive nodes corresponds to one action. In this way, the maximum number of children that could be needed to be generated in each cycle is reduced to only $m + 2n$ (only n of these nodes however correspond to a new deal).

In the rest of this chapter however we will continue as if these three steps were taken in one step. That is: as if each arc between a parent and a child node is labeled with an action, rather than only one component of an action.

6.4.3 Handling Proposals

When a proposal from another agent is received, α_1 adds a new branch to the tree to represent the proposed deal. The bounds and expansion heuristic are then calculated for every new node in this branch, and all these new nodes are put into the open list. In this way α_1 can explore extensions or adaptations to the received proposal.

As soon as a proposal becomes confirmed, the agent will act acts as if the committed actions are immediately executed (although technically we have defined our model such that they are executed in the action stage, after the negotiation stage has finished). Therefore, α_1 has to update its internal representation of the new world state. Many of the branches in the search tree can then be removed from the tree because they represent deals that are incompatible with the new world state.

6.4.4 Data Structures

In order to describe the Branesal algorithm we first describe its most important data structures: *WorldState*, *Commitment*, *Message*, *Node* and *Tree*. These data structures will then be used in the next section to describe the implementation of Branesal in pseudo code.

We assume there is a set of names of cities given, as well as a set of agent identifiers:

$$\begin{array}{l} \mathit{City} = \{v_0, v_1, v_2, \ldots v_{n(m+1)}\} \\ \mathit{AgentID} = \{a_1, a_2, a_3, \ldots a_n\} \end{array}$$

The program contains one *WorldState* object, that represents the current assignment of the cities of the NSP instance. There is a home city, a set of fixed cities and a set of interchangeable cities. The *assign* function represents the assignment of the fixed cities and interchangeable cities to the agents.

WorldState

$$\begin{array}{l} \mathit{homeCity} : \mathit{City} \\ \mathit{fixedCities} : 2^{\mathit{City}} \\ \mathit{intCities} : 2^{\mathit{City}} \\ \mathit{assign} : \mathit{fixedCities} \cup \mathit{intCities} \rightarrow \mathit{AgentID} \end{array}$$

The components of the WorldState structure satisfy the following constraints:

$$\begin{aligned} \forall\, a_i \in \mathit{AgentID} :\ |\{v \in \mathit{fixedCities} \mid \mathit{assign}(v) = a_i\}| = 1 \\ \mathit{fixedCities} \cap \mathit{intCities} = \varnothing \\ \mathit{homeCity} \notin \mathit{fixedCities} \cup \mathit{intCities} \\ \mathit{homeCity} \cup \mathit{fixedCities} \cup \mathit{intCities} = \mathit{City} \end{aligned}$$

The first of these constraints says that each agent has exactly one fixed city assigned to it. The other three constraints together state that the set of fixed cities,

the set of interchangeable cities and the home city together form a partition of the set of all cities.

An *Action* object represents the action of one agent giving one city to one other agent. The agent that gives the city is called the *donor* and the agent that receives the city is called the *acquirer*.

Action

$donor : AgentID$
$city : City$
$acquirer : AgentID$

Agents send messages to each other to propose, accept or reject deals. Note that in Section 3.5 we mentioned that the protocol does not formally include a 'propose' message because a proposal is simply the first 'accept' message in the conversation regarding to a certain deal. It turns out however that the algorithm is easier to implement if it internally does make a distinction between a proposal and an acceptance. Therefore, whenever the algorithm receives an accept message for a new deal, the algorithm internally treats it as a 'propose' message, and whenever the algorithm proposes a new deal, the communication layer of the agent converts it to an 'accept' message to comply with the protocol. If we compare this data structure with our definitions of Chapter 3 then we see that the content of a message in this domain consists of a 'type' and a 'deal'. Furthermore we see that we have left out the time stamp from the data structure, because it is not necessary for the implementation of the algorithm.

Message

$sender : AgentID$
$receivers : 2^{AgentID}$
$type : \{propose, accept, reject\}$
$deal : 2^{Action}$

The tree nodes of the search tree are implemented as the *Node* data structure. Each node contains a reference to its parent node, and is labeled by an *Action*. The set of all labels of all the ancestors of the node, including the node itself, is what we call the *path* of the node and the set of all donors and acquirers of all the actions in the path forms the set of *participating agents* (*pa*). Furthermore, each node contains an intermediate value, an upper bound and an expansion heuristic for each agent. From these values one can calculate the normalized utility $\overline{u}_1^1$ and the opponent-utility $\overline{u}_{pa}^1$. Finally, the node contains the set of participating agents that still need to accept the deal (as we will see later, it is initialized as the set of participating agents *pa* of the node, and each time one of these agents accepts the deal, this agent is removed from the set).

Node

$parent : Node$
$label : Action$
$pa : 2^{AgentID}$
$ub^1 : AgentID \rightarrow \mathbb{R}^+$
$e^1 : AgentID \rightarrow \mathbb{R}^+$
$h : \mathbb{R}$
$\bar{u}^1 : AgentID \rightarrow \mathbb{R}$
$\bar{u}^1_{pa} : \mathbb{R}$
$haveNotAcceptedYet : 2^{AgentID}$

Tree represents the search tree, which maintains a root node, an open list, and for every agent a reservation value and a global upper bound.

Tree

$root : Node$
$rv^1 : AgentID \rightarrow \mathbb{R}^+$
$gub^1 : AgentID \rightarrow \mathbb{R}^+$
$openList : 2^{Node}$

6.4.5 Procedures

We now describe the Branesal algorithm itself, which is given in Algorithm 2. We see that after initialization, it consists of a while loop that repeatedly calls three functions: *expand*, *handleIncomingMessages* and *acceptOrPropose*, which are described below. The algorithm keeps looping until the deadline for the negotiations has passed.

The *expand* method starts by extracting the node with the highest expansion heuristic from the open list and determining for which actions it should add child nodes to it (we do not provide here the implementation of the function that determines what actions to split over, but it works as explained in Section 6.4.2). For each action to split over, we create a new node, label it with the given action, set its participating agents, calculate its bounds (see Algorithm 7 for details on that), calculate its expansion heuristic (in the way explained in Section 6.2.5), add the new node as a child of the original node and add the new node to the open list. Finally, if the new node is individually rational (i.e. for each participating agent the intermediate value is lower than the reservation value) we can add it to the list of deals that are candidates to be proposed. *HandleIncomingMessages* checks whether a message has been received from any of the other agents. If not, the method returns. Otherwise, if the incoming message is a proposal, then a new node nd' is created that corresponds to the proposed deal and is added to the tree. The agent does not decide how to reply to this proposal yet, because this is done later by the *acceptOrPropose()* function (Algorithm 6). If the incoming message is an acceptance of a deal, the agent retrieves the deal from the tree. Note that this deal can indeed be found in the

Algorithm 2 Branesal

Require: startTime, timePassed, expandInterval, lastAcceptOrProposeCall, $t_d : \mathbb{R}$
Require: ϵ : WorldState
Require: theTree : Tree
Require: foundByMe = $\varnothing$
Require: proposedToMe = $\varnothing$
Require: $\eta_1^1 : [0, t_d] \rightarrow [0, 1]$
Require: $\eta_{pa}^1 : [0, t_d] \rightarrow [0, 1]$
Require: $\mathit{off}^1 : \mathit{AgentID} \rightarrow \mathbb{R}^+$
1: initializeTree(ϵ, theTree)
2: startTime ← getCurrentTime()
3: timePassed ← 0
4: lastAcceptOrProposeCall ← 0
5:
6: **while** timePassed < t_d **do**
7: expand(ϵ, theTree, foundByMe)
8: handleIncomingMessages(theTree, off^1)
9: **if** timePassed - lastAcceptOrProposeCall > expandInterval **then**
10: acceptOrPropose(foundByMe, proposedToMe, η_1^1, η_{pa}^1, *timePassed*)
11: lastAcceptOrProposeCall ← timePassed
12: **end if**
13: timePassed ← getCurrentTime() - startTime
14: **end while**

Algorithm 3 initializeTree(ϵ, theTree)

Require: home, fixed, interchangeable, current, minimal : 2^{City}
1: home ← {ϵ.homeCity}
2: **for all** $a_i \in$ AgentID **do**
3: fixed ← $\{v \in \epsilon.\text{fixedCities} |\ \epsilon.\mathit{assign}(v) = a_i\}$
4: interchangeable ← $\{v \in \epsilon.\text{intCities} |\ \epsilon.\mathit{assign}(v) = a_i\}$
5:
6: current ← home ∪ fixed ∪ interchangeable
7: minimal ← home ∪ fixed
8:
9: theTree.root.e_i^1 ← 0
10: theTree.root.ub_i^1 ← greedyPath(current) − greedyPath(minimal)
11:
12: theTree.rv_i^1 ← $root.e_i^1$
13: theTree.gub_i^1 ← $root.ub_i^1$
14:
15: theTree.root.$\bar{u}_i^1$ ← 0
16: **end for**
17: theTree.root.$\bar{u}_{pa}^1$ ← 0
18: theTree.openList ← {theTree.root}

Algorithm 4 expand(ϵ, theTree, foundByMe)

Require: nd : Node
Require: splitActions : 2^{Action}

```
1: //Get the node with the highest expansion heuristic and remove it from the
   open list:
2: nd ← arg max_{nd'}{nd'.h | nd' ∈ openList}
3: theTree.openList ← theTree.openList \{nd}
4:
5: //Get the set of actions to split over.
6: splitActions ← chooseSplitActions(nd)
7:
8: //For each such action: create a new node, calculate its properties and add
   it to the tree.
9: for all action ∈ splitActions do
10:    nd' ← new Node
11:    nd'.label ← action
12:    nd'.pa ← nd.pa ∪ {action.acquirer}
13:    nd'.haveNotAcceptedYet ← nd'.pa
14:    calculateBounds(ε, nd', theTree)
15:    nd'.h ← calculateExpansionHeuristic(nd')
16:    theTree.openList ← theTree.openList ∪{nd'}
17:    nd'.parent ← nd
18:
19:    //If the node is individually rational, then add it to the list of proposals
       we might want to propose.
20:    if ∀ a_i ∈ nd'.pa : rv_i^1 > nd'.e_i^1 then
21:       foundByMe ← foundByMe ∪{nd'}
22:    end if
23: end for
```

tree, because the agent had stored the deal in the tree when the deal was first proposed.

In either case, α_1 checks whether the proposing or accepting agent is willing to receive less utility than before, and if that is indeed the case the offer value of that agent is adapted. This means the expansion heuristic of each node in the open list needs to be recalculated.

Finally, if the deal in the incoming message is accepted by all participating agents, the execute method is called (Algorithm 8), which updates the world state and resets the root of the tree. The *acceptOrPropose* method determines whether α_1 itself should accept or propose a deal, according to the procedure described in Section 6.3.1. The *calculateBounds* function (Algorithm 7) calculates the intermediate values and the upper bounds of each agent. Also, it uses the reservation values and the global upper bounds to calculate the normalized utility of each agent. Finally it calculates the opponent utility by taking the product of all the normalized utilities of the other agents participating in the node's deal.

6.4.6 Complexity

We will now discuss the amount of time and memory that is needed for each new deal generated by the search algorithm.

Time Complexity

The most time consuming part of the algorithm is the calculation of the bounds and the expansion heuristic each time a new node is added. The time complexity of calculating the expansion heuristic is proportional to the number of agents $O(n)$, because for each participating agent we have to calculate the value of $P^1(ac_i(e^*))$.

Calculating the bounds of a node for one agent is quadratic in the number of cities that that agent owns. This is because finding the greedy path involves finding the nearest neighbor for each city owned by a certain agent. The bounds have to be calculated for each agent, so if there are m interchangeable cities per agent, calculating the bounds has a time-cost of $O(nm^2)$. Generating a new node therefore has a time complexity of $O(n + nm^2)$.

Each time a node is split we need to generate $m + 2n$ new children (as explained in Section 6.4.2). Splitting a node therefore has a time cost in the order of $(m + 2n) \cdot (n + nm^2)$, that is: $O(n^2m^2 + nm^3)$. Each time that such a split is made there are n deals generated so we can say that the amount of time needed to explore one possible deal is $O(nm^2 + m^3)$.

Another point regarding the time complexity that we would like to stress, is that each time the agent receives a proposal (or the acceptance of an earlier made proposal), the offer value of the agent that made the proposal (or accepted the proposal) must be adapted. This means that for every node in the open list the expansion heuristic needs to be recalculated. Moreover, the open list has to be reordered (it is implemented as a priority queue so the node with highest expansion heuristic can be retrieved fast).

Algorithm 5 handleIncomingMessages(theTree, off^1)

Require: msg : Message
Require: nd' : Node
Require: a_i : AgentID

```
1: msg ← getMessageFromMessageQueue()
2: a_i ← msg.sender
3:
4: if msg.type = propose then
5:    nd' ← insertProposedDealIntoTree(theTree.root, msg.deal)
6:    proposedToMe ← proposedToMe ∪{nd'}
7:
8:    //If the proposer offers to pay a higher price than he it offered before,
9:    //update its offer value and re-calculate the expansion heuristic for every
      node in the
10:   //tree.
11:   if off_i^1 < nd'.e_i^1 then
12:      off_i^1 ← nd'.e_i^1
13:      for all nd ∈ theTree.openList do
14:         nd.h ← calculateExpansionHeuristic(nd)
15:      end for
16:   end if
17: end if
18:
19: if msg.type = accept then
20:   nd' ← getNodeCorrespondingToDeal(msg.deal)
21:
22:   //If the accepting agent accepts a price to pay higher than it has offered
      before,
23:   //update its offer value and re-calculate the expansion heuristic for every
      node in the
24:   //tree.
25:   if off^1 l_i < n'.e_i^1 then
26:      off_i^1 ← nd'.e_i^1
27:      for all nd ∈ theTree.openList do
28:         nd.h ← calculateExpansionHeuristic(nd)
29:      end for
30:   end if
31:
32:   //The sender of the message has accepted the proposed deal,
33:   //so we can remove it from the list of agents that have not accepted it
      yet.
34:   //If all agents have accepted the deal, it can be executed.
35:   nd'.haveNotAcceptedYet ← nd'.haveNotAcceptedYet \ {msg.sender}
36:   if nd'.haveNotAcceptedYet = ∅ then
37:      execute(nd')
38:   end if
39: end if
```

Algorithm 6 acceptOrPropose(foundByMe, proposedToMe, η_1^1, η_{pa}^1, *timePassed*)

Require: myAspiration, opponentAspiration : $\mathbb{R}$
Require: bestFound : Node
Require: bestProposed : Node

```
myAspiration ← η¹₁(timePassed)
opponentAspiration ← η¹_pa(timePassed)

bestFound ← getMostSelfish(foundByMe)

//Get the most selfish node that is altruistic enough
repeat
   bestProposed ← getMostSelfish(proposedToMe)
until bestProposed.ū¹_pa > opponentAspiration or proposedToMe = ∅

//If no node is altruistic enough, then get the most altruistic one instead
if bestProposed.ū¹_pa ≤ opponentAspiration then
   bestProposed ← getMostAltruistic(proposedToMe)
end if

//If the best deal found by us (or proposed to us) is selfish enough, then
propose it (or accept it).
if bestFound.ū¹₁ >bestProposed.ū¹₁ then
   if bestFound.ū¹₁ > myAspiration then
      propose(bestFound)
      foundByMe ← foundByMe \{bestFound}
   end if
else
   if bestProposed.ū¹₁ > myAspiration then
      accept(bestProposed)
      proposedToMe ← proposedToMe \{bestProposed}

      //If we are accepting a deal that all others already have accepted,
      //then the deal will be executed.
      if bestProposed.haveNotAcceptedYet = {α₁} then
         execute(bestProposed)
      end if
   end if
end if
```

Algorithm 7 calculateBounds(ϵ, nd, theTree)

Require: home, fixed, interchangeable, acquired, donated, current, minimal : 2^{City}
Require: path : 2^{Action}
1: path $\leftarrow$ getPath(nd)
2: home $\leftarrow$ {ϵ.homeCity}
3: **for all** $i \in nd$.pa **do**
4: fixed $\leftarrow$ $\{v \in \epsilon.\text{fixedCities} \mid \epsilon.assign(v) = a_i\}$
5: interchangeable $\leftarrow$ $\{v \in \epsilon.\text{intCities} \mid \epsilon.assign(v) = a_i\}$
6: acquired $\leftarrow$ {ac.city | ac $\in$ path, ac.acquirer $= a_i$}
7: donated $\leftarrow$ {ac.city | ac $\in$ path, ac.donor $= a_i$}
8:
9: initial $\leftarrow$ home $\cup$ fixed $\cup$ interchangeable
10: current $\leftarrow$ home $\cup$ fixed $\cup$ acquired $\cup$ interchangeable $\setminus$ donated
11: minimal $\leftarrow$ home $\cup$ fixed $\cup$ acquired
12:
13: $nd.ub_i^1 \leftarrow$ greedyPath(initial) $-$ greedyPath(minimal)
14: $nd.e_i^1 \leftarrow$ greedyPath(initial) $-$ greedyPath(current)
15:
16: $nd.\bar{u}_i^1 \leftarrow (nd.e_i^1 - theTree.rv_i^1)/(theTree.gub_i^1 - theTree.rv_i^1)$
17: **end for**
18: $nd.\bar{u}_{pa}^1 \leftarrow \prod_{a_i \in pa \setminus \{\alpha_1\}} \bar{u}_i^1$

Algorithm 8 execute(nd, theTree)

Require: path : 2^{Action}
1: //Get the set of actions that form the path from the root to nd
2: path $\leftarrow$ getPath(nd)
3:
4: //Update the world state by letting the actions in *path* act on it.
5: //That is: change the assignment of the cities to the agents
6: **for all** $ac \in nd$.path **do**
7: ϵ.assign $\cup\{v \mapsto ac.acquirer\} \setminus \{v \mapsto ac.donor\}$
8: **end for**
9:
10: //Node nd becomes the new root node
11: //and the rv and glb are set equal to the bounds of this new root.
12: theTree.root $\leftarrow nd$
13: theTree.$rv^\alpha \leftarrow nd.e^\alpha$
14: theTree.$glb^\alpha \leftarrow nd.lb^\alpha$

In order to recalculate the expansion heuristic of a node, we only need to update the value $P^1(ac_i(e^*))$ of the agent for which the offer value has changed. Therefore, this can be done in constant time and thus the recalculation of the expansion heuristics of all of the nodes is done in $O(k)$ time (with k the size of the open list).

Reordering the open list can be done in $O(k\ log(k))$ time. Although k can be a very large number, it turns out from experiments that the time spent on updating the open list is in practice negligible. This is because the number of times that a proposal or acceptance from another agent is received is very small compared to the number of times that a node is expanded.

Space Complexity

Regarding the space complexity it is important to note that in each node we need to store the bounds for each agent, so the memory needed for each node is $O(n)$. Therefore, each time we expand a node, the extra memory we need is $(m + 2n) \cdot n$, that is: $O(n^2 + nm)$. And, since expanding a node yields n new deals, the average amount of memory needed for generating a single new deal is $O(n + m)$.

6.5 Experiments and Results

We have conducted a number of experiments with Branesal and in this section we present their results.

6.5.1 Experimental Setup

For our experiments we have made use of two types of NSP instances that differ in the way they are generated: *random instances* and *simple instances*. For the random instances all cities are represented as points in the two-dimensional plane, with the home city located at the coordinates $(0, 0)$. The x and y coordinates of all other cities are integers randomly chosen from a uniform distribution over the interval $[-100, 100]$. After generating the coordinates of the cities, the cities are randomly divided among the agents, such that every agent owns the same amount of cities. Also, for each agent, one of its cities is randomly chosen to be its fixed city. All other cities (except the home city) are interchangeable. The distance between two cities is given by the Euclidean distance. In all experiments except those in Section 6.5.6 we have used the random instances. The generation of simple instances is explained there.

For each run of the experiments we store for each agent the coordinates of the cities it initially owns and the coordinates of the cities it owns after the negotiations. When the run has finished we find the shortest path through each of these sets of cities, by feeding them into the *Concorde TSP Solver* [4].

[4] `http://www.tsp.gatech.edu/concorde`

Recall that we denote the length of the shortest path through the initial set of cities owned by agent α_i as $l_i(\epsilon_0)$. Similarly, we denote the length of the shortest path through the final set of cities owned by agent α_i as $l_i(\epsilon_f)$ (here ϵ_f denotes the final world state). With this notation we then define our performance measure for the random instances as the percentual cost reduction averaged over all agents:

$$Q = \frac{100}{n} \sum_{i=1}^{n} \frac{l_i(\epsilon_0) - l_i(\epsilon_f)}{l_i(\epsilon_0)} \tag{6.9}$$

This is the result of one run. The results presented in this section are all averaged over 100 runs, each with the same parameters, but with a different instance of the NSP. We should stress however, that the agents do not try to optimize this value, but rather each agent tries to minimize its individual path length. Therefore, *we are not so much interested in the value of Q, but rather in how it changes as the problem instances get more complex.*

Note that in the literature on bilateral negotiations one often uses the product of the agents' utility gains (the Nash Product [Nash, 1950]) rather than the sum to define a measure of performance. The problem with this is that when we are dealing with multilateral negotiations, the set of agents participating in a deal is often a subset of all the agents involved in the negotiations. Therefore it can happen that one agent does not decrease its cost at all, while all other agents do manage to obtain low costs. If we would then take the product of all utility gains, the result would be zero, because of the single agent that did not succeed in its negotiations.

For each data point we have also calculated the *standard error*, as $\frac{\sigma_k}{\sqrt{k}}$ with $k = 100$, where σ_k is the standard deviation of Q over k runs. For each experiment we will mention the highest and lowest standard errors among the data points. All experiments were conducted on an iMac with 3.4 GHz Intel Core i7 processor and 8 GB of memory. The agents were implemented in Java on top of the Jade platform.[5]

6.5.2 Varying Negotiation Length

In order to determine how the results improve with longer negotiations, we have done a number of tests, each with the negotiation length set to a different value. Each of these tests involved 10 agents, all running the Branesal algorithm, with 10 interchangeable cities per agent.

Because in some of the following experiments we vary the number of agents, we always measure the length of negotiations in milliseconds per agent. So if we say that the negotiation length was 500 ms per agent and there were 10 agents, this means that the deadline of the negotiations was set to 5 seconds. We should remark however that, since all agents are running on the same machine, we had no control over the amount of CPU cycles assigned to each agent, since this is

[5] `http://jade.tilab.com`

controlled by the Java Virtual Machine and the operating system. Therefore, we can only be sure that the amount of time that an agent has to run the algorithm is 500 ms *on average*.

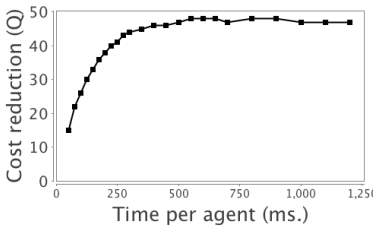


Figure 6.3: Cost reduction as a function of time.

The results are presented in Figure 6.3. We see that *the costs of the agents decrease significantly* and that the results get better as the available time increases. After all, the more time the agents have, the more good deals they will find, and therefore the better the final agreements they will make. The highest value (48%) is reached with 550 ms per agent. It seems this value does not improve with more time. The standard errors of these data points lie between 0.38 and 0.69.

6.5.3 Varying the Number of Agents

To determine how the algorithm scales with the number of agents, we have performed a number of tests with each a different number of agents, all running the Branesal algorithm. In each test 10 interchangeable cities were assigned to each agent and the negotiation length was set to 250 ms per agent. According to the results presented in the previous section, 250 ms is not enough for the agents to reach their maximum score. We have chosen this value however in order to put the agents under pressure, increasing the contrast between the various tests. The results are presented in the left graph of Figure 6.4. Interestingly, it

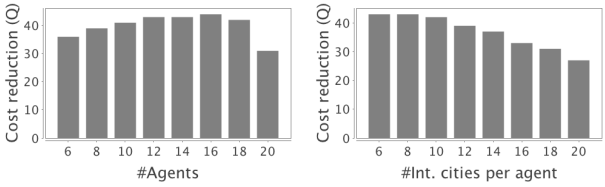


Figure 6.4: Cost reduction as a function of the number of agents and of the number of cities.

seems from this graph that at first, *the results get better as the number of agents*

increases, even though the problem becomes more complex. Apparently, the increased computing power resulting from the larger number of agents and the fact that agents can profit from the deals discovered by other agents outweighs the increased complexity of the problem. Unfortunately this only remains true as long as the number of agents is less than or equal to 16. With more than 16 agents we see that the complexity of the problem becomes more important and the results start to decrease. It is still unclear to us why this turning point takes place at 16 agents. The standard errors of these data points lie between 0.35 and 0.98.

6.5.4 Varying the Number of Cities per Agent

We now look at what happens if we make the agreement space larger (more interchangeable cities per agent), while keeping the number of agents constant. In each test there were 10 agents, all running the Branesal algortithm, with a negotiation length of 250 ms per agent. The results are presented in the right graph of Figure 6.4. As expected, with an increasing number of cities, the results decrease, but we think this decrease is relatively small, as the number of cities more than triples, while the value of Q only drops from 43 to 27. This can be explained by the fact that the expansion heuristic successfully manages to steer the search such that only interesting deals are explored and the unprofitable deals are skipped. In this way the increased size of the problem hardly decreases the efficiency of the algorithm. Therefore, we can conclude from this that *the expansion heuristic successfully manages to limit the number of redundant nodes that are explored*, as it is supposed to do. The standard errors of these data points lie between 0.41 and 1.07.

6.5.5 Comparing with Random Search

In the previous sections all agents have been running the Branesal algorithm. However, the most important question is how it performs when negotiating with agents that run different algorithms. Since there exists however no comparable negotiation algorithm, we have tested it instead against a copy of itself that applies random search.

We let some agents running Branesal (the "smart agents") negotiate with a number of agents running a random search (the "dumb agents"). With "random search" we mean that the agent is running an algorithm that is identical to Branesal, except that the expansion heuristic for each node is replaced with a random number.

We did four tests. Each test involved 10 agents, but for each test the number of dumb agents among those 10 was different. The negotiation length was set to 250 ms per agent. The results are presented in Figure 6.5. For each test we show the average score of the dumb agents (the left graph), the average score of the smart agents (center), and the score averaged over all agents together (right). When we compare the left graph with the middle graph, we can clearly see that, as expected, *the smart agents score significantly better than the dumb agents.* In

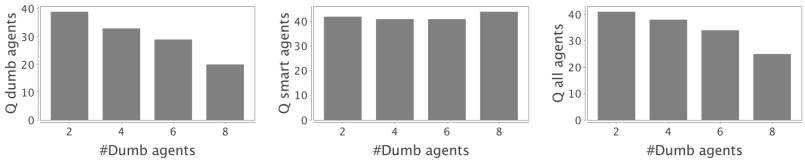


Figure 6.5: Cost reduction of dumb agents (left), smart agents (center), and all agents (right), as a function of the number of dumb agents.

other words: the expansion heuristic is effective. It is also interesting to see that if there are only a few dumb agents the dumb agents still manage to decrease their cost considerably. This can be explained by the fact that, although the deals they discover are bad, they still get offered good proposals from the smart agents. Since the deals found by the smart agents are generally better than the ones found by the dumb agents, the smart agents have more bargaining power, and are thus able to exploit the dumb agents. The standard errors of the results of the dumb agents lie between 0.54 and 1.05, the standard errors of the results of the smart agents lie between 0.47 and 1.30, and the standard errors of the overall results lie between 0.43 and 0.59.

6.5.6 Comparing with the Optimal Solution

In this section we compare the results of our algorithm with the optimal solution. Note however, that the notion of 'optimal solution' can be difficult to define in Automated Negotiations. A common way of defining optimality in games is to use some equilibrium concept such as the Nash Equilibrium [Nash, 1951]. The problem however is that a game often needs very specific properties in order to be able to calculate such an equilibrium (e.g. the presence of a discount factor in bargaining games). Moreover, even if one is able to play a strategy to reach the equilibrium solution, this would only be optimal under the assumption that the opponent also plays that strategy. A negotiator that manages to exploit suboptimal play of its opponents would be even better. Another definition of 'optimal solution' would be the outcome in which your agent achieves the maximum possible utility. However, this definition is unpractical since it is highly unlikely that the opponents of the agent would accept such a solution. For example, when two agents negotiate on how to divide a pie between them, the optimal outcome for agent α_1 would be that α_1 gets all of the pie and α_2 gets nothing. Of course, α_2 would never agree with such a deal, so this definition of optimality is unrealistic. A third way of defining the optimal solution would be to define it as the solution that maximizes the social utility, that is: sum of the utility values of all agents. The problem with this however is that the agents are simply not interested in reaching the social optimum. Every agent would prefer to try to obtain another, more selfish, solution.

So generally speaking, there is no such thing as an 'optimal solution' in

Automated Negotiations. This is a fact that actually occurs in many games. In robot-soccer for example, one can compare one robot-soccer team with another team and see which one is best, but there is no way to compare the team with any kind of theoretically optimal soccer team.

Nevertheless, we have come up with a solution that allows us to define a kind of optimal solution in special cases. We have created a set of special instances of the NSP that have the nice property that there exists one specific solution that is clearly the most reasonable one, because any other solution for which one agent increases its utility would imply a strong decrease in the utility of another agent and would therefore be unrealistic. The idea is that the cities are distributed in clusters around the fixed cities of the agents. The optimal solution is reached whenever each agent has exactly those cities in the cluster around its fixed city. We call these instances *simple instances*.

The cities of these simple instances are again given as 2-dimensional coordinates and their distances are the Euclidean distances. The graphs are however generated in two stages: in the first stage we create n random cities (where n is equal to the number of agents), far away from each other. Each of these cities is assigned to one of the agents as its fixed city (each agent gets exactly one fixed city). For each such fixed city we then randomly generate m cities nearby that city. In this way we have created n clusters of $m+1$ cities each. So after this first stage all the cities of one cluster are assigned to the same agent. We refer to this assignment as the *'optimal assignment'*. This assignment is optimal in the sense that every agent owns a set of cities that lie very close to each other so the agents cannot decrease their path length any further by negotiation.

Then, in the second stage, for each agent we 'swap' some of its cities with cities from other clusters. A swap means that we randomly pick one city assigned to the agent, and one city assigned to an other agent and interchange them. For each agent we make $m/3$ swaps, so after swapping each agent owns at least one city from another cluster, and on average for each agent two thirds of its interchangeable cities are in another cluster (we make $m/3$ swaps for each agent, and each swap involves 2 cities, so in total $2/3 \cdot m \cdot n$ cities change owner). We refer to this new assignment as the *'initial assignment'*, because this will be the assignment of the cities at the start of the negotiations.

The length of the shortest path through the set of cities that are assigned to agent α_i in the optimal assignment, is denoted by l_i^*. The score of a test is calculated as follows:

$$Q_{simple} = \frac{100}{n} \sum_{i=1}^{n} \frac{l_i(\epsilon_0) - l_i(\epsilon_f)}{l_i(\epsilon_0) - l_i^*} \tag{6.10}$$

We have only used NSP instances for which $l_i(\epsilon_0) - l_i^* \geq 10$ for each agent. With these instances we have repeated the experiments of Section 6.5.2 with four different values of m, namely: 6, 9, 12, and 15. We see that *the algorithm is able to reach a score of 80% of the optimal solution.* Furthermore, we note that as the number of cities increases, the algorithm converges more slowly, but still manages to reach 80%. The standard errors of the data points in these four

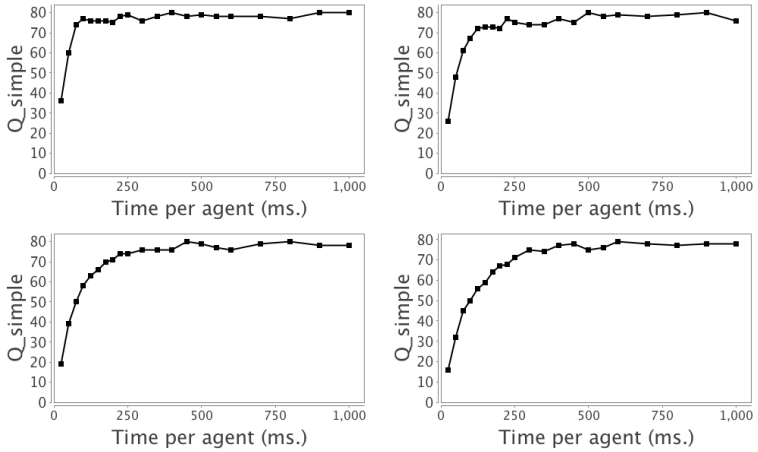


Figure 6.6: Increasing negotiation length, with simple NSP instances. Top left: 6 interchangeable cities per agent, top right: 9 interchangeable cities per agent, bottom left: 12 interchangeable cities per agent, bottom right: 15 interchangeable cities per agent

graphs lie between 0.72 and 1.70.

We expect that non-selfish negotiating agents could reach a higher score than this. Also, a (distributed) constraint optimization algorithm would probably be more successful in increasing the social utility. However, it is important to note that NB^3 is designed to optimize *individual* utility, rather than social utility, so one cannot compare this result with results from non-selfish scenarios (one could make NB^3 a non-selfish algorithm by defining the individual utility functions to be equal to the social utility, but it would then still be less efficient than other non-selfish algorithms, because the agents could be searching through overlapping regions of the agreement space).

Also one should note that the fact that a score of 100% is not *reached*, does not mean that the optimal solution has not been *found* by any of the agents. Even if an agent finds the solution that maximizes social utility it may still try to propose other, more selfish, solutions. It might for example happen that agents α_1 and α_2 come to a deal that yields high utility to both agents, but that is incompatible with the socially optimal solution, especially if the resulting individual utilities for α_1 and α_2 are higher than what they would get in the socially optimal solution. And even if a deal is individually worse than the social optimum, agents still might prefer to come to a quick individually suboptimal solution rather than wait and hope they can find a better solution, since the available time for search is limited and agents fear missing good deals because of competition.

One alternative way of solving the NSP might be to use clustering rather than heuristic search to find good solutions. We have intentionally not tried to

do this, because clustering would *only* be applicable to NSP and not to general negotiation problems. Our goal was not to find the best solution to the NSP, but to use NSP as a testbed for general negotiation algorithms.

Yet another way of solving the NSP would be to apply a centralized approach in which a mediator finds a solution that benefits all agents and is the most fair solution according to some fairness criterion. However, once again, this is not the goal of our work. In real-world situations it is not always possible to find an impartial mediator, to have all agents to agree on the definition of fairness, or to make agents cooperate in finding social solutions.

6.6 Conclusions

In this chapter we have introduced a new family of negotiation algorithms for very large and complex agreement spaces, with multiple selfish agents, non-linear utility functions and a limited amount of time. This family is called NB^3 and applies best-first Branch and Bound to search for good proposals.

Our main motivation for doing so is to bring automated negotiations closer to real-world negotiations. Therefore, we have had to discard a number of assumptions that are usually made in existing literature, as we consider them unrealistic. One of those assumptions is the application of the Alternating Offers protocol. Instead, we have made use of the much more flexible Unstructured Negotiation Protocol.

We have defined a general purpose heuristic to guide the Branch & Bound search of the NB^3 algorithm. Just as in Chapter 5 we have applied a new negotiation strategy that does not use a single aspiration level for the utility, as in most existing work, but that uses two aspiration levels because it considers the utility aspired by our agent and the utility to be conceded to the opponents as two separate quantities. This allows our agent to not only determine what to propose, but also to determine whether it should make a proposal or rather continue searching for better proposals.

We have implemented an instance of NB^3 for the NSP that we call Branesal and we have performed several experiments with it, with extremely large search spaces (of size up to 20^{200}). From these experiments we draw the following conclusions:

- Our agent indeed manages to decrease its costs significantly by negotiation.
- Most of this decrease is obtained within half a second.
- If we increase the complexity of the problem by increasing the number of agents, the results remain stable, up to 18 agents.
- If we increase the complexity of the problem by increasing the number of cities, the results only get slightly worse.
- The heuristic search applied by our algorithm successfully manages to prune redundant nodes.

- The heuristic search applied by our algorithm is significantly better than random search.
- For problem instances that have a clear optimal solution, the algorithm manages to reach a solution which is at 80% of the optimal solution.

For the future, we are planning to fine-tune some of the components of NB^3 such as the expansion heuristic and the negotiation strategy. We plan to experiment with different values of the concession degrees, for example, and with different initial and final values of the aspiration levels, which currently only take the values 0 and 1. Moreover, as explained in Section 6.3, in the current implementation the agent treats the set of opponents as if it were one opponent to which it should concede. We will improve the negotiation strategy by dropping this assumption, so that our agent can treat every single other agent as a different opponent. Furthermore, we have mentioned in Section 6.3.2 that there is a parameter that determines how long the agent continues expanding the search tree before deciding whether to make a new proposal or not (the expandInterval parameter in Algorithm 2). We will investigate the influence of the value of this parameter on the results. In the NSP, the preferences of the agents are expressed explicitly as utility values. We think that for practical applications it is unrealistic to assume that user preferences can be expressed explicitly as numerical utility functions. Therefore, we plan to adapt the algorithm so that it can handle qualitative preference relations instead.

Chapter 7

Applying Branch & Bound to Diplomacy

We now take a look at the last, and most complex of three negotiation domains given in Chapter 4: Diplomacy. Diplomacy has been used as a test-bed for Automated Negotiations before, such as in [S. Kraus, 1989, Kraus, 1995], but these papers mainly focus on the modular structure of their agent rather than on the algorithms they apply. It remains unclear how their agent searches through the large space of possible deals and determines what to propose. An informal online community called DAIDE exists which is dedicated to the development of Diplomacy playing agents.[1] Many agents have been developed by this community but only very few are capable of negotiation. In [Fabregues and Sierra, 2011] a new platform called DipGame was introduced to make the development of Diplomacy agents easier for scientific research. Several agents have been developed on this platform such as in [Fabregues, 2014] and in [Ferreira et al., 2015]. The agent we present in this chapter is also based on the DipGame framework and we will compare it with the agents presented in these two papers.

7.1 Constraint Optimization Games

In Chapter 3 we have explained how a negotiation game NG can be defined over any one-shot game G. We will here however restrict our attention to a specific kind of game, which we call a Constraint Optimization Game (COG). The idea of a COG is that it is a one-shot game that can be decomposed as a number of smaller games that are played simultaneously but that are not independent. The key point of our Diplomacy algorithm is the observation that the game *Dip*, defined in Chapter 4, can be modeled as a COG and therefore that a single round of Diplomacy can be modeled as a Negotiation Game over a COG.

Let MG be a set of m one-shot games: $MG = \{G_1, G_2, ...G_m\}$, each with

[1] http://www.daide.org.uk

the same n players. We call these games the **micro-games**. The set of moves for player α_i in micro-game G_j is denoted as $O_{i,j}$. Then a **Constraint Optimization Game** G_{MG} is a game with n players, such that the set of actions O_i for player α_i in G_{MG} is a subset of the Cartesian product of the moves of each micro-game: $O_i \subseteq O_{i,1} \times O_{i,2} \times ... O_{i,m}$. A joint move o of G_{MG} can then be viewed as a matrix in which each column o_j represents a joint move from the micro-game G_j. The utility function for a player α_i is defined as:

$$f_i^{G_{MG}}(o) = \sum_{j=1}^{m} f_i^{G_j}(o_j)$$

So the game G_{MG} consists of a number of smaller one-shot games that cannot be played independently from each other, because the set of moves O_i of α_i is a *subset* of the product of the sets $O_{i,j}$. In other words: there are hard constraints between the moves of the several micro-games that must be satisfied. Each player α_i can pick for any micro-game G_j any move from $O_{i,j}$, but not all combinations of such moves are allowed. Therefore, the best strategy for the game G_{MG} is not simply the combination of the best strategies of each individual micro-game. For example, player α_1 may have the optimal move $a \in O_{i,1}$ for micro-game G_1 and an optimal move $b \in O_{i,2}$ for micro-game G_2, but the combination of a and b may be illegal, so he is forced to choose a suboptimal move in at least one of these two micro-games.

We see that the concept of a COG combines aspects from Constraint Optimization Problems with aspects from Game Theory. Just like in a COP an agent α_i needs to pick m values for m different variables, such that the combination is consistent and maximizes its utility. However, unlike normal COPs, the utility of an agent does not only depend on the values he himself chooses, but also on those chosen by his opponents, which have different utility functions to maximize, just as in Game theory. Another way to look at it, is to see it as a variation of a Distributed Constraint Optimization Problem in which there is not one single utility function to maximize, but rather each agent aims to maximize its own individual utility function.

7.2 Dip as a COG

We now show that the game *Dip* can be modeled as a COG. A natural way to play *Dip* is to determine for each supply center separately whether, and how, it can be conquered. However, the decision how to attack one supply center may restrict the possibilities to attack another supply center if the same units are involved. This is indeed the essence of a COG.

We model *Dip* as a COG by defining a micro-game Dip_p for each supply center $p \in SC$ in which the actions $O_{i,p}$ for player α_i are the battle plans for p.

Definition 62 *A **battle plan** for a supply center p is a set of orders that consists of one move-to order for a unit to move into p, zero or more orders sup-*

porting that unit, and zero or more move-to orders that intend to cut those units of the opponents that may support a competing move into p.

If one battle plan for supply center p requires unit u to move to p, while another battle plan for supply center p' requires the same unit to move to p' then these two battle plans are incompatible with each other, so, there is a constraint between micro-games Dip_p and $Dip_{p'}$.

The utility function $f_i^{Dip_p}$ is defined as having value 1 for those joint moves in which α_i conquers p and 0 otherwise. Note that the question who manages to conquer the supply center p indeed depends on (and only on) the battle plans that each player chooses for that supply center. Therefore, a round of Diplomacy can be correctly described as a set of micro-games where each micro-game is a battle for a specific supply center.

From personal experience playing the game we know that, in general, it is not difficult to determine for every supply center which battle plans are guaranteed to succeed. The difficulty of Diplomacy is that those battle plans are often incompatible with each other so one needs to determine the best consistent combination of such battle plans.

7.3 D-Brane

We have implemented a Diplomacy-playing agent called D-Brane (Diplomacy BRAnch & bound NEgotiator). It regards each round of Diplomacy as an instance of the negotiation game *NDip* (with the Unstructured Negotiation Protocol of Section 3.5 as the negotiation protocol) and hence tries to maximize the number of supply centers to conquer during that round. We should note however that this is a short-sighted simplification: in reality a good player does not always try to maximize its number of conquered supply centers in the current round, but may sometimes give up some supply centers in the short term in order to conquer more supply centers in the future. Furthermore, it is important to understand that D-Brane is a selfish player: it does help its allies in the game, but only because it expects help from them in return in later rounds.

D-Brane was implemented in Java, using the DipGame framework [Fabregues and Sierra, 2011]. It consists of two main components: *the strategic component* and the *negotiation component*. During the negotiation stage of *NDip* the negotiation component applies the NB^3 algorithm to search through the agreement space, and determines which deals to propose to its coalition partners.

Recall that in *Dip* a move is a vector of orders (one order for each unit). Therefore, a deal x defines for each player a subset of the space of all such vectors. D-Brane uses the model in which a deal is represented by a set $\overline{x}$ of commitments, where each commitment is an order. For example, suppose we have a set of orders $\overline{x} = \{a, b, c\}$ consisting of three orders a, b, and c. Some of these orders maybe for player α_1 and some may be orders for other players. If we use $\overline{x}_i$ to denote those orders in x that are orders for α_i then the subset of moves $O_i[x]$ that α_i can make if x gets confirmed, is the set of all vectors that

contain the orders in $\overline{x}_i$.

$$o \in O_i[x] \Leftrightarrow \overline{x}_i \subset \hat{o}$$

(here $\hat{o}$ is the set that consists of exactly the orders in the vector o).

For each deal x that D-Brane considers, it calls the strategic component to calculate for every player α_i in $pa(x)$ (including itself) the minimum utility it can obtain in the game $Dip[x]$. The negotiating component then uses this value to determine whether to propose that deal to the others. Furthermore, in the action stage of *NDip* the strategic component is called again to determine the best possible moves to make, given the deals that were made during the negotiation stage.

7.3.1 The Strategic Component

Given a deal x and a player α_i the strategic component tries to find the best set of orders for α_i in the game $Dip[x]$ and the minimal utility for α_i resulting from those orders. In theory this could be achieved by determining the set of all consistent combinations of battle plans for each power and then calculate the Nash Equilibrium, but this is computationally too expensive as the number of such combinations can easily be of the order 10^{10}. Our algorithm however manages to quickly make very good approximations by exploiting the fact that $Dip[x]$ can be modeled as a COG. It works as follows:

1. For each $p \in SC$ determine all **invincible plans**: battle plans that cannot be defeated by any opponent battle plan.

2. For all pairs $(p_1, p_2) \in SC \times SC$ determine all **invincible pairs**; consistent pairs of battle plans that cannot be defeated both at the same time by any pair of opponent battle plans.

3. Remove all invincible plans and invincible pairs that are not consistent with x.

4. Find the largest consistent combination of invincible plans and pairs, using an And/Or tree search with Branch & Bound.

5. Return that combination of plans, together with its minimal utility value. This will be the utility value for α_i assigned to the deal x.

The idea behind the use of invincible pairs is that we may have a battle plan a for one supply center which can only be defeated by an opponent's battle plan a', and we may have a battle plan b for another supply center that can only be defeated by an opponent's battle plan b'. The assumption that a and b can be discarded because they can both be defeated is too strong, because it may be the case that a' and b' are incompatible. In that case if we play both a and b we are guaranteed that at least one of them will succeed. In theory, we could continue this reasoning and also determine invincible triples,

invincible quadruples, etcetera, to improve the algorithm even further. However, the number of combinations of plans that need to be checked grows exponentially, so this would make the algorithm very slow.

7.3.2 Generalizing to Other COGs

The algorithm described above was designed by us, specifically for Diplomacy. However, we think it can be generalized easily to other COGs as well. We here give a rough sketch of how to do that. The essence of our algorithm can be described in three steps:

1. Determine a utility value for every move $o_1 \in O_{1,j}$ of player α_1 in every micro-game G_j.

2. Do the same for every pair of moves $(o_1, o_1') \in O_{1,j} \times O_{1,k}$ for every pair of micro-games (G_j, G_k).

3. Find the combination of moves that maximizes the sum of utilities, using And/Or search with B&B.

The idea is that assigning a single value to each move in each micro-game in fact reduces the COG to a standard COP, which can be solved with an And/Or search and B&B. One straight-forward way to assign a value to a move o_1 is to find the vector of opponent moves o_{-1} that minimizes the utility function $f_1(o_1, o_{-1})$. In other words: the utility value obtained in the worst-case scenario that the opponents pick those moves that minimize α_1's utility. If we apply this principle to *Dip*, then this means that every invincible plan receives a value of 1 and all other plans receive a value of 0, which essentially means that all non-invincible plans are discarded, which is indeed what we did.

7.3.3 The Negotiating Component

During the negotiation stage of *NDip* we apply in instance of the NB^3 algorithm to explore the agreement space and find good deals to propose to the coalition partners. As explained, a deal consists of a set of orders. More precisely, D-Brane generates a tree in which each arc between a node nd' and its parent nd is labeled with a battle plan, and the path from the root to that node therefore represents a deal x_{nd} that consists of all orders of all battle plans along the path from the root node to nd. For any node its set of children represents a set of alternative battle plans for the same supply center.

The intermediate value $e_i(nd)$ of a node nd for player α_i is calculated by calling the strategic component. The strategic component will determine the best set of moves for player α_i in the game $Dip[x_{nd}]$ and the number of supply centers conquered by α_i for that set of moves. This number of conquered supply centers is then returned and assigned to the variable $e_i(nd)$.

7.4 Experiments

In this section we compare D-Brane with two other agents recently presented in [Ferreira et al., 2015] and [Fabregues, 2014]. In both papers a negotiating agent was tested by letting it play against a standard non-negotiating bot called the DumbBot. We have repeated some of their experiments with our own agent and compared the results.

When two or more players in Diplomacy are in a coalition, this usually implies two things: it means they will not attack each other and it means that they may support each other when attacking an opponent. We call the first kind of agreement an *implicit agreement* because it is implied by the fact that the players are allies, so no negotiation is needed to establish such agreements. The second kind we call an *explicit agreement* and can only be made by negotiating. The ability of D-Brane to negotiate (i.e. to make explicit agreements) and its ability to obey implicit agreements can both be switched on and off, so it can play in 4 different modes: with negotiations on or off and with implicit agreements on or off. The reason for this is that on one hand it is unrealistic to play without implicit agreements, because any reasonable player would always apply them. On the other hand however, we are testing our agent against the DumbBot which plays entirely individually, so if a coalition of D-Branes beats a number of DumbBots this could be the consequence of implicit agreements rather than negotiations. Furthermore, Note that switching implicit and explicit agreements off essentially means the D-Branes do not form a coalition at all and play entirely individually.

In each of the experiments we performed with negotiations off we played 100 games (playing 100 games without negotiations takes about half an hour). For experiments with negotiations we set the deadline for negotiations to 15 seconds per round. Since such experiments therefore take much more time we only played 50 games in each such experiment. (playing 50 games with negotiations takes about 10 hours, but this depends mainly on the negotiations deadline which we could have made shorter). Each game was stopped after 40 rounds.[2]

In all experiments we assume the D-Branes form a coalition against the DumbBots. All experiments were performed on a single desktop computer with 3.4 GHz Intel Core i7 processor and 8 GB of memory.

7.4.1 D-Brane Compared with DipBlue

In [Ferreira et al., 2015] a negotiating Diplomacy agent called DipBlue was introduced. In their experiments they let two instances of this agent play against 5 instances of the DumbBot. As a measure of success they used the average rank of their two agents over 75 games. That is: after each game the best player gets rank 1, the 2nd best player gets rank 2, etcetera, and the worst player gets rank 7. A player is considered better than another player if it finishes with more supply centers, or if it is eliminated later. If two or more players rank equally

[2] For readers more familiar with Diplomacy: we mean that we stopped each game after the Winter 1920 phase.

Table 7.1: 2xD-Brane vs 5xDumbBot. The numbers indicate the average rank of the D-Branes. For comparison: DipBlue achieves 3.57.

	nego off	nego on
impl. agr. off	2.63 ± 0.10	2.47 ± 0.14
impl. agr. on	2.46 ± 0.11	2.35 ± 0.16

they both received the average of their ranks (e.g. if two players share the second place they both get rank 2.5).

In this way one can objectively determine whether a certain agent plays better than the DumbBot, because a player equally strong to the DumbBot would achieve an average rank of 4. With 2 instances of the player to test the best possible average rank is 1.5 and the worst is 6.5. The best average rank that DipBlue achieves in their experiments is 3.57.

We have done the same experiment, but with 2 instances of D-Brane against 5 DumbBots. We repeated this four times: one time for each playing mode of D-Brane. The results are shown in Table 7.1. It displays the average rank of the D-Branes in the four different modes, with their respective standard errors. We note that in all cases the average rank is around 2.5, even when negotiations were switched off. This means that not only our agent is better than the DumbBot, but also that *even without negotiating D-Brane plays significantly better than DipBlue with negotiations.*

7.4.2 D-Brane Compared with Fabregues' Agent

In [Fabregues, 2014] a nameless agent was presented, which was also compared with the DumbBot. Fabregues did 8 experiments, varying from 0 to 7 instances of her negotiating agent against the DumbBot. In each experiment she played 100 games. As a measure of success she counted the number of victories of her agent. It is important to note however, that her experiments were performed on a super computer and that she used negotiation deadlines of 5 minutes per round, considerably more than our deadlines of 15 seconds per round.

To compare, we did an experiment with 4 D-Brane agents against 3 DumbBots, and counted how often a D-Brane ranked first. Again, we repeated this experiment 4 times, one for each setting of the D-Brane, The results are shown in Table 7.2. For comparison: Fabregues' agent achieves a victory in about 85% of the games. We conclude that *D-Brane scores significantly better than Fabregues' negotiating agent, even if D-Brane was not negotiating, and even though our experiment ran on a single desktop computer, playing 100 games in half an hour, rather than on a super computer.*

Table 7.2: 4xD-Brane vs 3xDumbBot. Displayed are the percentages of victories of the D-Branes. For comparison: Fabregues' negotiator achieves about 85%

	nego off	nego on
impl. agr. off	97%	95%
impl. agr. on	99%	97%

7.4.3 Evalution of Experiments

These experiments make clear that D-Brane plays much better than DumbBot, DipBlue and Fabregues' agent, even when it does not negotiate. We also see however that negotiations only have a small effect on the final results. In the experiment with 4 D-Branes the results even seem to decrease with negotiations, but one should take into account that the number of victories is a more crude measure than the average rank.

After studying the logfiles it seems that whenever there is a good opportunity to make a deal, the D-Branes indeed find it, and make that deal. However, it seems that such opportunities simply do not occur very often, so that although the D-Branes do benefit from such deals, their impact on the final result of the entire game is rather small. The conclusion we draw from this is that only negotiating joint battle plans is not enough to really benefit from negotiations. Indeed, we know from personal experience with the game that a good player not only negotiates battle plans for the current round, but also looks farther ahead and negotiates future actions.

In [Ferreira et al., 2015] and [Fabregues, 2014] negotiations had a stronger positive impact on the respective agents. We think this is because both their agents were implemented by extending the DumbBot with negotiation capabilities, while the DumbBot is not a very good player. It is likely that negotiations have a much bigger impact on bad players, because good players have less room for improvement. Moreover, it is not clear whether their results were due to implicit agreements or explicit agreements, because they do not make this distinction.

7.5 Conclusions

In this chapter we have presented a new Diplomacy-playing agent called D-Brane. We have introduced the concept of a Constraint Optimization Game, which combines aspects of Constraint Optimization with Game Theory, and we have showed that a single round of Diplomacy can be modeled as a Negotiation Game over a COG. We have explained that D-Brane exploits this fact by applying And/Or search with B&B to determine the utility value of any given deal. Furthermore we have explained that D-Brane applies the NB^3 algorithm to explore the agreement space and determine which proposals to make. Finally,

we have presented experiments in which we compare D-Brane with two other negotiating agents and show that D-Brane plays significantly better. However, it also shows that the negotiation algorithm seems to have only a small impact on the results.

We think that D-Brane will be very important for future negotiations research, because it allows researchers to compare their strategies not only with the DumbBot but also with our much stronger agent. We therefore plan to make D-Brane publicly available and we will detach the negotiating component from the strategic component so that other researchers can put their own negotiation algorithms on top of the strategic component of D-Brane.

Part III

User-friendly Electronic Institutions

Chapter 8

Humans Negotiating with Agents

In this chapter we will introduce a new tool that we have developed on top of the EIDE framework and that allows human users to participate in an Electronic Institution, through a web browser.

8.1 Motivation

So far we have defined protocols, world states and world state evolution maps as mathematical objects. We have ignored the question how they are implemented. Although they can be implemented in an ad hoc manner, a general model for implementing protocols is provided by a so-called Electronic Institution (EI). Electronic Institutions are related to negotiations in two ways: firstly they may enforce the negotiation protocol, and secondly they may enforce the obedience of the agreements made during negotiations. This first point can be seen as the implementation of the negotiation stage, while the second can be seen as the implementation of the action stage in a Negotiation Game as defined in Chapter 3.

One of the most comprehensive EI infrastructures is the EIDE platform, that we introduced in Section 1.4.3. However, this framework was originally developed mainly with software agents in mind. Support for human participation existed, but was very user-unfriendly and only suitable for experts with deep knowledge of the EIDE framework. Therefore, we have implemented a new, user-friendly, extension to EIDE to create Graphic User Interfaces (GUI) that allows humans to enter into an EI and interact with other users or software agents. We think that this tool could be very useful for real negotiation platforms that allow people to engage in negotiations with each other or with software agents.

Our tool generates a default user interface automatically from the EI specification, without the need for extra programming. But, on the other hand, if one does require a more case-specific user interface, it still provides an API that

enables any web designer to easily design a custom GUI without the need for much knowledge of Electronic Institutions, or Java programming. Our approach is completely web-based, meaning that the GUI is in fact a website, implemented using standard web-technologies such as HTML5, Javascript and Ajax. In short, we have developed our framework with the following goals:

- To allow people to interact in an EI through a web browser.
- To have a generic GUI that is generated automatically.
- To allow any web designer to easily design a new customized GUI, if wanted.
- To allow testing of an EI under development, before having implemented its agents.

Visual user interfaces for Electronic Institutions have been created before, for example in [Trescak et al., 2013]. In their work the user controls an avatar that walks around in a 3-dimensional virtual world that represents the EI. Although a 3D-virtual world may be more impressive visually, we think that a simple website that runs in a web browser has several advantages:

- Websites are cross-platform.
- No need for heavy hardware: it works on mobile phones and tablets.
- More practical as one does not have to walk around an environment, but simply has all options directly available in the form of buttons or menu-items.
- We think a 2-dimensional environment is more common in the context of social media[1] and therefore users will feel more familiar with it.

The goal of our the work presented in this chapter is to enable human users to participate in Electronic Institutions, negotiating with one another as well as with software agents. One can for example imagine a website where travel agencies and travelers may come together and that acts as a market place so that the clients can negotiate with the agencies about the price and components of their holidays. The EIDE platform ensures that these negotiations can take place under specific protocols and that all commitments are registered. Although such websites can also be made without Electronic Institutions, the application of EI technology could make such websites much more flexible as protocols can be adapted and replaced easily. Another example could be a social network where users set up their own communities and negotiate the rules that apply to those communities. The EI then provides the environment for the negotiation of those rules as well as for the enforcement of those rules that were negotiated.

[1] Although the 3D social game Second Life has been very popular for a while, it has never been nearly as popular as 2D media such as Facebook and Twitter.

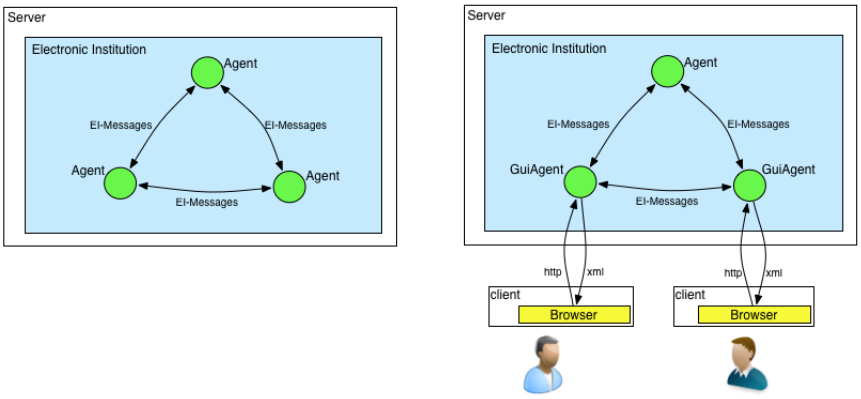


Figure 8.1: Left: a 'classic' EI with only software agents. Right: an EI with one software agent and two users.

8.2 GENUINE

We will now present the tool that we have developed, called GENUINE, which stands for GENerated User INterface for Electronic institutions.

8.2.1 Components

A human user would interact in an EI by clicking buttons in a browser window. To allow these actions to have effect in the EI, we have implemented a software agent that represents the user inside the EI and that executes the actions requested by the user. This agent is called the *GuiAgent*. Its current implementation does not do anything autonomously, but, if necessary, it can be extended with more sophisticated capabilities, such as giving intelligent strategic advice to the user.

When developing the framework we took into account that, on one hand, one may want to have a good-looking GUI that is specifically designed for a given institution. But, on the other hand, one may not want to develop an entirely new GUI for every new institution, or one may want to have a generic GUI available to test a new EI during its development, so that one can postpone the design of its final GUI until the EI is finished. Therefore, our framework allows for both. It generates a GUI automatically from the EI-specification, but at the same time provides an API that enables web designers to easily create a custom GUI for every new EI. GENUINE is used on top of the existing EI-framework and consists of the following components:

- A Java agent called *GuiAgent* that represents the user in the EI.
- A Java component that encodes all relevant information the agent has about the current state of the institution into an xml file.

- A Javascript library called *GenuineConnection* that translates the xml file into a Javascript object called *EiStateInfo*.
- A Javascript library called *GenuineDefaultGUI* that generates a default Graphic User Interface (as html) based on the EiStateInfo object.

8.2.2 How it Works

In order for a user to participate in an institution, there must be an instance of that EI running on some server. To join the institution, the user then needs to open a web browser and navigate to institution's url. The process then continues as follows:

1. A web page including the two Javascript libraries is loaded into the browser.
2. The page sends a login request to the server.
3. Upon receiving this request the server starts a GuiAgent for the user and, depending on the specific institution, other agents necessary to run the institution.
4. When the GuiAgent is instantiated it analyzes the EI-specification to retrieve all static information about the institution.
5. The page starts a polling service that periodically (typically several times per second) requests a status update from the GuiAgent.
6. When the GuiAgent receives a status update request it asks its Governor for the dynamic information about the current status of the institution.
7. The GuiAgent converts both the static and the dynamic information into xml which is sent back to the browser.
8. The GenuineDefaultGUI Javascript library then uses this information to update the user interface (more information about this below).
9. The user can now execute actions in the institution or move between its scenes by clicking buttons on the web page.
10. For each action the user performs, a http-request is sent to the GuiAgent.
11. The GuiAgent uses the information from the http-request to create an EI-message which is sent like any other message in a standard EI.

As explained, the GuiAgent uses two sources of information: static information from the EI-specification stored on the hard disk of the server and dynamic information from the Governor. The static information consists of:

- The names and protocols of the scenes defined in the institution.
- The roles defined in the institution.

Http-Request	Description
/login?name=alice&role=guest	Enter the EI with given name and role.
/sendMessage?name=alice&receiver=bob &msg=bid&amount=1000	Send a message with given parameters.
/gotoScene?name=alice&role=guest &sceneName=Admission	Enter the given scene with the given role.
/exitScene?name=alice	Exit the given scene.
/gotoTransition?name=alice &transitionName=transition1	Go to the given transition.
/request_update?name=alice	Request an update of the status of the EI.

Figure 8.2: The http-requests sent from the browser to the GuiAgent

- The ontology of the institution.

While the dynamic information consists of:

- The current scene and its current state.
- The actions the user can perform in the current state of the scene.
- For each of these actions: the parameters to be filled out by the user.
- Which agents are present in the current scene
- Whether it is allowed to leave the scene and, if yes, to which other scenes the user can move.

8.2.3 Generating the GUI

Every time the browser receives information from the GuiAgent, it updates the GUI. This takes place in two steps, handled by the two Javascript libraries respectively. In the first step the GenuineConnection library converts the received xml into a Javascript object called EiStateInfo, which is composed of smaller objects that represent the static and dynamic information as explained above.

In the second step the EiStateInfo-object is used by the GenuineDefaultGUI library to draw the GUI. This GUI is completely generic, so it looks the same for every institution. If one requires a more fancy user interface tailored to one specific EI, one can write a new library that replaces the GenuineDefaultGUI library.

The fact that these two steps are handled by two different libraries enables developers to reuse the GenuineConnection library when designing a new GUI, so one does not have to worry about how to retrieve the relevant information from the EI. All information will be readily available in the EiStateInfo-object, so one only needs to determine how to display it on the screen.

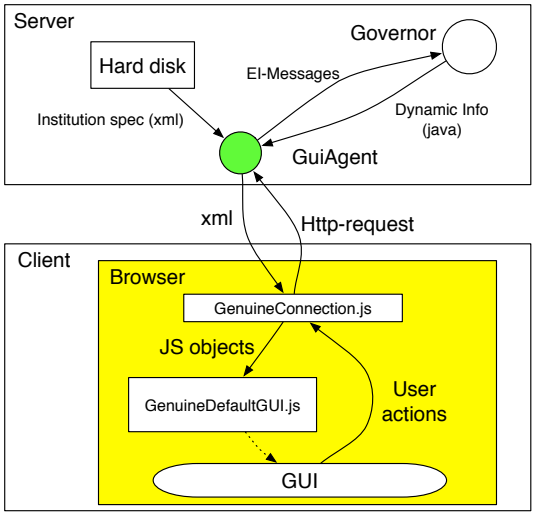


Figure 8.3: The components necessary to generate the GUI. Solid arrows indicate exchange of information. The dashed arrow indicates that the GUI is created by the GenuineDefaultGUI library

8.2.4 The Default User Interface

The default user interface is displayed in Figure 8.4. It is divided in three main sections:

- A menu bar in the top that allows for navigating from scene to scene.
- A screen on the left (the interaction screen) where the user can see the other participants and send messages to them.
- A screen on the right showing all messages in the user's individual message history.

Furthermore, in the top left a personal avatar is displayed, and the user name and role of the participant. We have chosen to make navigation between scenes resemble as much as possible the way a user navigates between menu-items on a regular website. The menu bar contains the name of the current activity in the left, and a list of menu items that one can use to move from one activity to the next. Some of these menu items may be disabled. This happens when the user is not yet allowed to move to those activities. Furthermore, in the right of the menu there is a 'Map' button. Clicking the Map button will make a map appear that gives an overview of the performative structure (see Section 1.4.3).

The interaction screen displays a number of icons. For each participant in the current scene there is a circular icon containing the avatar of that participant. A

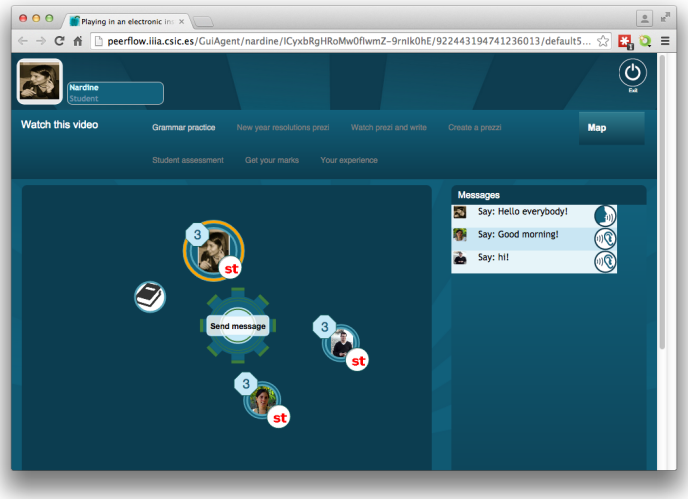


Figure 8.4: The default Gui

number is displayed in the upper left of the avatar which indicates the number of actions undertaken by this participant in the current activity. In the bottom right of each avatar two letters are displayed that represent the role of that participant.

The icon shaped like a cog-wheel represents a message that the user can currently send, according to the protocol. The user can send a message by clicking on its corresponding icon. Doing so, a pop-up screen will appear with a form where the user can enter the content of the message and its receivers. In Chapter 3 we have defined the content of a message to be a natural number, that represents any kind of content. In the EIDE framework however, the content can be any list of parameters of different types. The form therefore shows one input control for each parameter of the content of the message. The type of control depends on the type of the parameter. For example, if the the parameter is of type integer, a numeric input control appears, while if the parameter is of type string, a text box appears. In case the parameter is of a user-defined type, a sub-form appears with several controls, one for each of the variables of the user-defined type.

8.2.5 Shortcomings of the Default GUI

Although an automatically generated user interface can be very useful for testing purposes during the development of an application, we think that it is less useful for a user-friendly end product. The problem with the automatic GUI, is that it is only able to represent the messages that can be send by the user or that have

been received by the user, but it is not capable of automatically representing the current world state.

There are two reasons for this. The first reason, is that world states in EIDE are represented in terms of variables, which can be of certain basic types, such as strings, integers and booleans, or types that are composed of these basic types. It is very difficult to translate such abstract types into something user friendly. For example, one can encode the board configuration of a chess-game into an abstract representation of variables, but given such an abstract representation it is almost impossible to automatically translate this into a graphical picture displaying the chess-game. Such translations can normally only be made by a human programmer that knows that the variables represent a chess-game and that knows how a chess-game can be displayed in a manner understandable to users.

A second problem is that even if we could display the world state, the EIDE framework does not allow this, because EIDE considers the world state as private information that cannot be shared with the participants. The reason for this, is that participants are only allowed to know any information that they have received through messages sent by other participants. Since not every message is sent to every participant, not every participant can have the same information. Note that this is precisely what we discussed in Chapter 3; every agent has an *individual* message history, and can thus not always know the world state ϵ because this is determined by the full message history. In the original idea of EIDE as a platform purely for software agents, this was not a big problem. After all, the programmer of the agent can implement an individual representation of the world state, which depends only on the the individual message history of that agent. However, for humans, it is simply too complicated to understand the current state purely by looking at a list of messages. It is as if a chess-player does not see the board configuration of the game, but only sees the list of moves that have been made by the two players and needs to determine his next move purely based on that. Therefore, we think that any truly user-friendly application would need a customized user interface in which the user's view of the world state is manually implemented and presented in a user-friendly, domain-specific way.

8.2.6 Customizing the GUI

A customized GUI-generator can retrieve all necessary information from the EiStateInfo-object. For example: if the user wants to make a propose a price in a car deal, the GUI-generator would read from the EiStateInfo-object that an Integer parameter must be set to represent the price the user wants to propose. The programmer of the GUI-generator should make sure that whenever a parameter of type Integer is required, the GUI displays an input-control that allows the user to introduce an integer value.

The fact that one can also define user-defined types in an EI adds a lot of flexibility. Suppose for example that one would like a user to record an audio file and send this in a message to an other agent. Electronic Institutions do not

support audio files by default. However, the institution designer may define a new type with the name 'Audio'. Once the user chooses to send a message that includes audio, the EiStateInfo-object will indicate that a parameter of type Audio is needed. A customized GUI-generator could then be programmed such that a microphone is activated whenever this type of parameter is required.

8.3 Conclusions

We have managed to add a new tool, called GENUINE, to the existing EIDE framework that allows humans to interact with each other and with software agents in an Electronic Institution. Specifically, it allows humans to negotiate with each other or with agents, inside the EIDE framework. The tool generates a default GUI automatically, so the creator of an institution can directly use it without having to design anything. However, GENUINE also makes it easy for interface designers to design a customized GUI, tailored to a specific institution. The designer does not have to worry about how the EI technically works or about how to get the necessary information from the EI. We argue that the default GUI is useful for testing purposes, but less useful for a user-friendly end product, because it is only able to represent messages, but not world states. In that case we think having a customized user interface is essential.

Another advantage of having a user interface for Electronic Institutions is that it allows developers to test an institution during its development, without having to program any agents. While an EI is still under construction human users can take the place of the software agents that would later participate in it, for testing purposes. This will allow for faster development.

Chapter 9

Towards the Negotiation of Protocols

In this chapter we argue that the fact that many social networks are nowadays existing next to each other is inefficient and is due to the fact that users are not able to adapt norms and protocols to their own needs. We argue that Electronic Institutions would provide a solution to this problem, because it would allow users to create their own social networks with custom rules. Moreover, it would allow the users to negotiate which protocols and rules should apply to the social network.

Recall that the negotiations determine a set of confirmed deals X_d, and that the action stage involves playing a game G restricted by X_d. The restriction of the game may mean that the underlying protocol of G is still valid, but only with a few parameters changed. However, it can also mean that the protocol itself is entirely changed. We could imagine that in the original game G anything is allowed (so essentially there is no protocol) but that the negotiators agree on a highly restricted protocol for the action stage. In that case we can say, in a sense, that the agents have negotiated which protocol to follow in the action stage.

At this point however, the EIDE framework does not yet allow for the definition of protocols at run-time. It only allows the parameters of a protocol to be adapted at run-time. Therefore, the negotiation of protocols and other ideas presented in this paper should be seen as ongoing work. In the next chapter however, we do show a first step towards this goal, as we define a language that should make it easier for people to define protocols and negotiate their rules.

9.1 Social Networks as Electronic Institutions

In the past few years there has been an enormous increase in the popularity of social networks and many different ones nowadays exist next to each other, such as Facebook, Twitter, LinkedIn and Couch Surfing. Although they are all based

on the same idea: making contacts and sharing information with them, each of these networks applies different protocols and has different interpretations of what it means to be connected to someone. While on Facebook a friend is someone you share your pictures with, a connection on LinkedIn is someone you share your CV with. In this section we make two important observations:

- Social networks are institutions, each with their own rules and protocols.
- The flexibility of Electronic Institutions allows for a more generic type of social network, in which the users can determine their own rules.

Clearly, different kinds of relationships require different rules of behavior, and we claim that this is an important reason why so many social networks exist simultaneously. To illustrate this we will next compare two popular social networks: Facebook and Couch Surfing, regarding to their respective rules and protocols.

9.1.1 Rules and Protocols of Facebook and Couch Surfing

Facebook is mainly designed for friends to share social activities with each other. Due to the informal nature of these activities, like sharing pictures and playing games together, it does not require very strict norms.

Roughly, we can summarize Facebook as follows:

- **Meaning of friendship:** Friends can see each others' pictures.
- **Protocols:** Becoming friends requires only two actions: one person requests the friendship, the other accepts it.
- **Rules:** Users have full control over their profiles: the user can remove anything that anyone else writes on his or her profile.
- **What could go wrong:** The user may by accident share pictures with someone he or she does not like.

Couch Surfing is a social network for travelers[1]. The main idea of this network is that any traveler, instead of booking a hotel, can find somebody who is willing to host him or her for free at home. When planning a trip the user can search for profiles of people at the destination and if the user likes somebody's profile the user can request that person to host him or her.

While Facebook focuses on *online* shared experiences with friends *you already know*, Couch Surfing focuses on meeting *new people*, in *real* life. This means that Couch Surfing requires much stricter policies than Facebook. After all, hosting a complete stranger in ones house, or being hosted by a stranger, can be dangerous (cases are known of women getting raped using Couch Surfing [DailyMail, 2012]). Becoming friends on Couch Surfing therefore requires more effort, and there are several mechanisms to verify the trustworthiness of members that Facebook lacks, discussed for example in [Lauterbach et al., 2009].

Roughly, we can summarize Couch Surfing as follows:

[1] `http://www.CouchSurfing.org`

- **Meaning of friendship:** Friends express trust in one another so that other people know they can safely host or be hosted by either of the two friends.
- **Protocols:** To become friends, one needs to indicate how well one knows the other person, how much one trusts the friend and specify details on how and where you met.
- **Norms:** If somebody posts a negative comment about you on your profile, you cannot remove it.
- **What could go wrong:** Hosts may get robbed by their guests, or worse.

9.1.2 Designing EI-based Social Networks

The fact that the establishment and maintenance of different kinds of social contacts require different protocols, has lead to the creation of many different social networks, even though they often have overlapping communities of users. We now discuss how the application of Electronic Institutions could put an end to this inefficiency by allowing users to set up new sub-communities within a given social network, and invent their own set of rules and protocols for these sub-communities, without having to create an entirely new website.

One problem we have to overcome when implementing a social network as an EI is the fact that Electronic Institutions are based on the assumption that all users that communicate with each other are together in one scene instance. Social networks on the other hand have a much more asynchronous design, in which it is not assumed that all users are online at the same time: when a user shares an image with a friend, the user is in fact uploading it to a database. The friend will not see it until he or she also appears online. At that moment the image is automatically downloaded from the database to the friends' browser, so that he or she can see it.

To overcome this discrepancy we have come up with the following design: each activity a user can do in an EI-based social network takes place in a scene instance where the user and the database are both represented by an agent. So in each scene instance there are exactly two agents: the GuiAgent and a DataBaseAgent. When the user uploads a picture, for example, this is modeled as a message which is being sent from the GuiAgent to the DataBaseAgent. When the other user appears online his or her GuiAgent will also enter a scene together with a DataBaseAgent, and the image will be sent from the DataBaseAgent to the GuiAgent. A second problem we needed to tackle is that the current implementation of Electronic Institutions does not allow for any bulk data (i.e. images, video or audio) to be sent in a message. We therefore have to send the data itself outside the institution. The action of sending this data however, is still represented as a message inside the EI, so that the EI can still verify whether the user is actually allowed to undertake that action. We just need to make sure that when such a message is blocked, this is also prevents

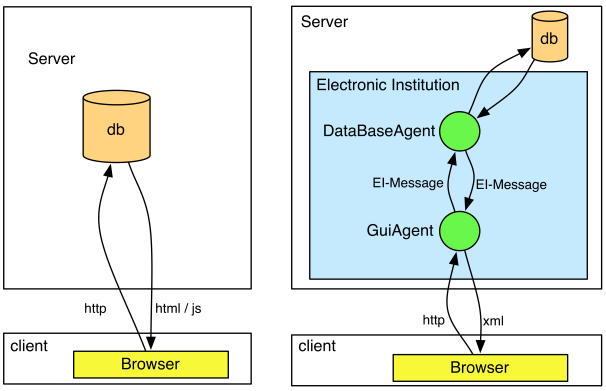


Figure 9.1: Left: a standard website. Right: an EI-based website.

the user from sending the actual data. This can be achieved for example by disabling an upload button.

Finally, one more problem to tackle is the fact that moving from one scene to another scene in an EI is made in three steps: first the agent exits the current scene, then the agent moves to a so-called *transition* and finally the agent moves to the new scene. We think it is very user-unfriendly, since going from one web page to another is usually done with one single menu click. The reason for this 3-step process is that it allows agents to choose to move into more than one scene instance at a time, or to synchronize with other agents before moving into the new scene. Although this is fine for software agents, we think this is overly complex for human users, and not necessary for the application to social networks.

We have solved this by making sure that these three steps are all triggered automatically, one by one, by a single click on a menu-item. The downside of this, however, is that it removes the possibility for the user to make any choices at the transition. Also it could happen that a user chooses to move to a scene instance that is not available. A scene can be unavailable because one needs to wait for other agents to participate in the same scene. However, in the model described above the GuiAgent and its associated DataBaseAgent are always together and never need to participate in any scene instance with any other agents. Therefore, as long as we stick to this model no scene can ever be unavailable.

9.2 Case Studies: MusicCircle and WeBrowse

As a test case we have applied our EI-technology to a social network for online music learning, called *MusicCircle*, which is currently under development. On this website students can learn to play an instrument, with and from their friends,

in a community-driven way. Imagine for example a student who is learning to play the piano. He or she can play a piece and record it, and then upload it to the social network. The student can then ask the other community-members for feedback. These community members may be friends of the student, professional music teachers or even automated music analyzing agents.

MusicCircle will allow users to create their own sub-communities. For example, one can set up a community for guitar players, or for jazz-musicians, or a community consisting only of colleagues from work. We are providing the EI framework underneath this social network. This will enable users to set or change the rules of their communities as they wish.

Some communities may for example have serious users and strict rules, because their members want to study seriously, and want to discuss their progress with other serious students. Other communities may be much more non-committal, consisting of hobbyist who just want to spend some free time playing music as a hobby, without taking it too seriously. A few examples of rules that a community may set, could be the following:

- This group is only for *advanced* guitar-players: to join, you need to be at least at stage 10 of the guitar course.
- Only active users can receive feedback: to be able to receive feedback a user must give feedback to others at least 5 times a week
- Experts will only help serious students: if a user wants to get help from an expert player, the user needs to practice at least 3 times a week.

Another social application where we are applying our technology is the *We-Browse* application [Hazelden et al., 2012], [Yee-King et al., 2013] which enables friends to simultaneously visit a museum online, each from his or her own mobile device. It allows friends, even though they are in different locations, to have a *shared experience* when they visit the museum. The users see the same artifacts on their respective computer screens, and they can see each others' actions, such as zooming in on an object, or adding tags and 'likes'.

Furthermore, the users need to make joint decisions on what to see or do next in the museum. Since people may have different opinions on this matter, protocols are needed to determine how individual decisions and opinions are aggregated into social group decisions that are acceptable to all group members. An underlying EI could enforce these protocols.

9.3 Conclusions

We have argued that many different social networks nowadays exist next to each other, because they all require different rules to be imposed on their users. We think that social networks in the future could be based on EI technology, so that rules can be flexible and allow people to develop their own communities within existing social networks, with their own regulations tailored to the needs of that community. This would require users to negotiate which protocols should be

followed within such a community. As yet, EIDE does not yet allow for the dynamic definition of protocols, so this is ongoing work.

Furthermore, we have shown that if an EI is to represent a social network, we should not model the users in that network as directly sending messages to one another, but rather we should apply a model in which the user sends messages to a database agent, which then at a later stage forwards those messages to the intended recipient.

Chapter 10

Simple Protocol Language

In the previous chapter we have discussed the advantages of people negotiating which protocols to follow. However, the protocols in EIDE are currently defined in a very complicated way. They consist of a finite state machine, plus constraints and consequences on the state transitions of that machine, plus a set of variables defined for the protocol. This makes it very difficult for humans to negotiate such protocols, as there is a very big difference between the intuitive meaning of a protocol and its formal implementation.

Therefore, in this chapter we introduce a new language for the specification of protocols, which is much simpler than the existing tools such as Islander. Our language is very close to natural language so it is easy to negotiate about the rules of a protocol and any such rules that one agrees upon can be directly implemented by the EI, without translating it to some more abstract representation language.

We should note however, that the EIDE framework currently does not allow protocols to be defined at run-time; once the institution is launched, all protocols are fixed. Therefore, the development of this language is only a first step towards the goal of having negotiable protocols.

10.1 Motivation

As explained in the introduction, many frameworks for Electronic Institutions have been developed. However, they all have one important characteristic in common, namely that they have been designed mainly with computer scientists as their target users in mind. They require knowledge of multi-agent systems, programming languages and/or formal logic. For people with no more than average computer skills they are unfortunately too complicated.

We expect however that agent technologies will become more and more common in the near future, creating a demand for simple tools to maintain and organize such systems, and that can be used by ordinary people. We can compare this for example with the evolution of web development. In the early days

of the Internet, developing a web page was considered an advanced task that would only be undertaken by computer experts, and hence web development languages such as HTML, PHP and SQL were invented for professional programmers. However, as web pages became more and more abundant and every shop, social club, or sports team wanted to have its own web page, tools such as DreamWeaver and WordPress were introduced to make the creation of web pages a much simpler task. We strive for a similarly easy tool for the development of multi-agent systems.

A good example of where such a tool would be useful is the organization of online classes, because teachers often want to put restrictions on their students. Teachers may for example require that students can only take a certain exam if they have passed all previous exams. In this way teachers can make sure they do not waste their time correcting exams of students that do not study seriously anyway. Another example could be the process of organizing a conference, where one requires authors to submit before a deadline, or one requires the program chair to appoint at least 3 reviewers to each paper. Also, one can think of a tool that allows users to set up their own social networks, with their own specific community norms, as we suggested in Chapter 9.

Another interesting point of view, is that with this language, we can treat protocols themselves as the subject of negotiation. That is: agents could negotiate the rules of the protocol to follow in some scene of an institution. In this way, the scene protocols in an EI can be can be defined while that EI is already running. We think this is much easier to do with the structured sentences of SIMPLE, than with the graphical representation of Islander that currently used to define protocols in EIDE.

Although many programming languages claim to be similar to natural language [pla, 2014] [hyp, 2014], most of them still aim at real programmers, albeit that they aim for *beginning* programmers. The only exception that we know of, is a language called Inform 7 [Nelson, 2014]. This is a language that in many cases truely reads like natural language. The main difference with our language however is that it is developed for an entirely different domain. Inform 7 is a language to write *Interactive Fiction*: an art form that lies somewhere in between literature and computer games.

We argue that one of the main reasons that Inform 7 can indeed be very close to natural language, is that it is highly adapted to a very specific domain. This restricts the possible things a programmer may want to express and hence keeps the language manageable. We have taken a similar approach: our language is only intended to be used as a language for implementing protocols for multiagent systems, and although it could possibly also be useful for other domains, we restrict our attention purely to this domain.

Therefore, in this chapter we present the first version of a new language to define protocols for multi-agent systems. This language is very close to natural language so that it can be understood directly by anyone without prior knowledge of any programming language. We call this language SIMPLE, which stands for SIMple Protocol LanguagE. Although this language *looks* very simi-

lar to natural language, it has in fact a very strict syntax. Together with this language we also present two tools: an editor that makes it very easy for users to write well-formed sentences, and an interpreter that parses the source file and makes sure that the rules defined in it are indeed enforced upon the agents. The fact that the language comes with an editor is very important, because it enables the users to write correct protocols without having to know the rules of the language by heart. In fact, this editor even makes it impossible to write syntactically incorrect sentences.

We have developed SIMPLE with the following guidelines in mind:

- The language should stay as close as possible to natural language.
- The syntax should remain strict: sentences must be well formed, and every well formed sentence can only have one correct interpretation.
- Given a protocol written in this language anyone should immediately be able to understand what it means, even if he or she has never heard of our language before.
- Users should be able to write a protocol in this language without having to spend any time learning the language.

The only thing we require from the user is that he or she be familiar with the English language. The language as presented here is only the very first version, and we plan to extend it much further in the future.

10.2 Basic Ideas

We assume a multi-agent system in which agents exchange messages according to some protocol, as defined in Chapter 3. This protocol is written in the SIMPLE language. The agents may be autonomous software agents, or may be humans, acting through a graphic user interface. The agents are however not in direct contact with one another. Every message any agent sends first passes a central server (an Electronic Institution) that verifies whether the protocol allows that message to be sent in the current world state. If a message is not allowed then it is blocked by the EI and it will not arrive at its recipients. Moreover, the world state will only be updated if the message is allowed, thus the EI ensures that the protocol is regimented.

We assume that the life-cycle of the MAS is as follows:

1. A user (the *protocol designer*) writes a protocol in SIMPLE using our editor and stores it in a text file.
2. He or she launches an EI on a server, with the location of the text file as a parameter.
3. The interpreter, which is part of the EI, parses the text file.

4. Agents connect to the server through a TCP/IP connection and send messages to one another.

5. Every such message is checked by the interpreter. If it does not satisfy the protocol, it is blocked. If it does satisfy the protocol it is forwarded to its intended recipients and the world state is updated, according to the protocol.

6. The agent that intended to send the message is notified by the server whether the message has been delivered correctly or not.

The text file contains the protocol as a set of sentences that follow the SIMPLE syntax, and are therefore human readable. However, it also stores the protocol in JSON format so that it can be parsed easily by the interpreter.

Just like in the EIDE framework, protocols written in SIMPLE have a closed-world interpretation: every message is considered illegal by default, unless the protocol specifies that it is legal. In order to determine which messages are legal, we use a system based on the notion of 'rights' and 'events', meaning that an agent obtains the right to send a specific message if a certain event has (or has not) taken place. The assignment of such rights is determined by if-then rules in the protocol.

As explained in Chapter 3 messages are of the form (a_i, A, c, t). However, in this case the content c is restricted to only follow one of these two patterns:

- $c =$ ('say', x)
- $c =$ ('tell', y, z)

in which the sending agent can replace x, y and z by any character string (we will see later that the 'tell' message has the interpretation that the value filled in for z will be assigned to a variable of which the name is the string filled in for y). The current version of the language does not yet allow users to specify the receivers of a message, so for now we assume that any message is always sent to all the other agents in the MAS. We plan this to change in future versions of SIMPLE. Also, we expect that future versions will support more types of messages.

The interpreter keeps a world state ϵ_t which consists of a list of **rights** for each agent in the MAS. A right is a tuple of one of the two following forms:

- ('say', v)
- ('tell', w)

Definition 63 *We say that a right ('say', v) **matches** a the message-content $c =$ ('say', x) if and only if x is equal to v, or v is the key word 'anything'. A right ('tell', w) matches the message-content ('tell', y, z) if and only if y equals w. We also say that a right matches a message, if it matches the content of that message.*

For example: if the agent has the right ('tell', 'price') then it matches the the message with content ('tell', 'price', '$100'). A message is considered legal if the agent sending the message has at least one right that matches the message. Whenever the interpreter determines that a message is legal, it stores a copy of that message in the interpreter's message history.

One concept that we have borrowed from EIDE is the concept of a *role*. The rules in the protocol never refer to specific individuals, because we assume that at design time one cannot know which agents are going to join the MAS at run time. Instead, the protocol assigns rights to agents based on the roles they are playing. Every agent that enters the MAS (i.e. connects to the communication server) must choose a specific role to adopt, from a number of roles that are defined in the protocol. An auction protocol for example, could define the roles *buyer* and *auctioneer*. The protocol could then define a rule saying that a buyer can only make a bid after the auctioneer has opened the auction.

10.3 Description of the Language

A protocol is written as a set of sentences that look like natural language, but nevertheless have a strict syntax. Although in this chapter we will often start sentences of the language with a capital, this is not necessary, as the language is in fact entirely case-insensitive. Like in natural language, the end of a sentence is marked with a period. Unlike most other programming languages, variable names are allowed to contain spaces. Another important property of this language, as we will see at the end of this section, is that it is impossible to write inconsistent protocols.

10.3.1 Roles

Definition 64 *A **role definition sentence** is a sentence of the form:*

`This protocol defines the role` r_1 *(*`plural:` r_2*).*

Where the protocol designer can replace r_1 *and* r_2 *by any character string. The string* r_1 *is called the **singular role name** and* r_2 *is called **plural role name**.*

For each role in the protocol there must also be exactly one such role definition sentence. For example:

```
This protocol defines the role buyer (plural:buyers).
```

Definition 65 *A **role constraint sentence** is a sentence of one of the following forms:*

- `There can be any number of` r.
- `There must be at least` x r.
- `There can be at most` x r

- `There must be at least` y `and at most` x r.
- `There must be exactly` x r.

Where x and y can be any positive integer with $y < x$ and r is a plural role name from one of the role definition sentences, except in the case that $x = 1$, in which case r must be a singular role name.

The following two sentences are examples of role constraint sentences:

```
There must be at least 2 buyers.

There must be exactly 1 auctioneer.
```

For each role in the protocol there must be exactly one such role constraint sentence. The interpreter makes sure that these role constraints are not violated. That is, when an agent tries to connect to the EI server with a role for which there are already too many participants, the connection will be refused. If there are not yet enough participants for every role, then every message is considered illegal. In other words: the agents can only start sending messages to one another when there are enough participants for every role.

10.3.2 Conditions and Consequences

The main idea of the language, as explained above, is that rights are assigned to the agents by means of if-then rules. An example of such a rule could be:

```
If the auctioneer has said 'open'
then any buyer can tell his bid price.
```

In order to define precisely which sentences are well formed we first need to introduce a number of terms, namely: *quantifiers*, *identifiers*, *conditions*, and *consequences*.

Definition 66 *A **quantifier** is any of the following keywords: no, any, every, a, an, the, that*

Definition 67 *An **identifier** is a sequence of characters of one of the following forms:*

- q r
- `no one`
- `anyone`
- `everyone`
- `he`

Where q can be any quantifier and r can be any singular role name. Identifiers of the form `no` r *as well as the identifier* `no one` *are called* ***negative*** *identifiers. All other identifiers are called* ***positive*** *identifiers.*

Definition 68 *A* ***past-event condition*** *is a string of characters of one of the following forms:*

- *id* `has said` *'x'*
- *id* `has told` x
- *pid* `has not said` *'x'*
- *pid* `has not told` x

where id can be any identifier and x can be any character string, and pid can be any positive identifier. A past-event condition is called negative if it contains the keyword `not` *or if it contains a negative identifier. A past-event condition is called positive otherwise.*

A past-event conditions is a specific type of condition. Other types of condition are defined later.

The idea behind this is that a positive past-event condition is considered true if there is any message in the message history that matches the condition. For example the condition `any buyer has said 'hello'` is considered true if there exists a message in the message history of the form ('say', 'hello') which was sent by an agent playing the role *buyer*. A negative past-event condition is considered true if there is no message in the message history that matches the condition.

Definition 69 *A* ***right-update consequence*** *is a string of characters of one of the following forms:*

- *pid* `can say` *'x'*
- *pid* `can tell` x

where pid can be any positive identifier and x can be any character string.

A right-update consequence is a specific type of consequence. Other types of consequences are defined later on.

We can now construct sentences ('rules') of the form `If` A `then` B, where A is a conjunction of conditions and B is a conjunction of right-update consequences. We say that a rule is *active* if all its conditions are true. Then the idea is that an agent has the right to send a specific message if and only if there is an active rule with right-update consequence that matches that message.

As we have seen above, identifiers are used inside conditions and consequences to determine to which set of agents these conditions and consequences apply. We would like to remark that the quantifiers `a`, `an`, `any` and `the` all have exactly

the same meaning, so the language contains some redundancy. However, we still consider it very useful to have all of them in the language to help the protocol designer to write more natural sentences. For example, if an auction protocol contains only one auctioneer it makes much more sense to talk about 'the auctioneer' than about 'any auctioneer'.

Also note that we have included the quantifier `that`. This quantifier refers to any agent that was also referred to by the last quantifier earlier in the sentence. For example, suppose that a buyer called Alice says 'hello' and then a buyer called Bob says 'hi'. Then the condition

```
any buyer has said 'hello' and any buyer has said 'hi'
```

is true. However, the condition

```
any buyer has said 'hello' and that buyer has said 'hi'
```

is false, because 'that buyer' refers to the same agent as the one that said 'hello' (which is Alice). This second condition would only be true if the messages ('say' 'hello') and ('say' ,'hi') had been sent by the same agent. Likewise, we have included the identifier `he`, which refers to the same agent as the last identifier that appeared earlier in the sentence. For example:

```
If any buyer has said 'hello' and he has said 'hi'
```

10.3.3 Properties

Recall that, according to our definitions in Chapter 3, the realized world state only depends on the initial world state and the messages that have been sent by the agents. Indeed, so far we have seen that rights are obtained by if-then rules that depend on the messages that have been sent. However, sometimes it is much easier to assign rights in a more indirect way, depending on the values of some variables. These variables have initial values and their values may change depending on the messages that are being sent. Therefore, if the rights of an agent depend on such variables, they indirectly still depend only on the initial world state and the realized message history.

Variables in SIMPLE are called **properties**. A property can be assigned to the protocol, or can be assigned to individual agents. For example, an auction may have a property 'highest bid' and each buyer may have a property 'bid price' to represent the price he or she has bid. If we have for example properties 'the price' and 'account balance' then we can say things like:

```
If the price is lower than account balance
then any buyer can say 'buy'.

If the price is higher than 10
then the auctioneer can say 'sold'.
```

Properties can be added to a protocol by including property initialization sentences.

Definition 70 *A **property initialization sentence** is a sentence of one of the following forms:*

- `Initially`, *x* `is` *v*.
- `Every` *r* `has a` *x*, `which is initially` *v*.
- `Every` *r* `has an` *x*, `which is initially` *v*.

where x can be any character string, v can be any character string, number, or identifier and r can be any singular role name.

For example:

```
Every buyer has an age, which is initially 0.
```

Definition 71 *A **property condition** is a clause of one of the following forms:*

- *x* `is` *v*
- *x* `is not` *v*
- *x* `is higher than` *n*
- *x* `is lower than` *n*

*where x can be any character string, v can be any string, number or identifier, and n can be any number. The string x is called the **property name**, and v and n are called the **value**.*

Note that the current version of SIMPLE supports three types of properties: strings, numbers and identifiers. The type of a property is determined implicitly. That is: if the parser of the protocol is able to interpret the initial value of a property as a number, then the property is considered to be of type number, and likewise for identifiers. In all other cases the property is considered a string.

Definition 72 *A **property-update consequence** is a clause of the form:*

- *x* `becomes` *y*
- *x* `is` *v*
- *x* `is increased by` *n*
- *x* `is decreased by` *n*

where x and y can be any character string, v can be any string, number of identifier, and n can be any number.

Definition 73 *A **current-event condition** is a string of characters of one of the following forms:*

- *id* `says` *'x'*

- *id* `tells` x

where id can be any identifier and x can be any character string.

In order to change the values of properties we can use property-update rules.

Definition 74 *A **property-update rule** is a sentence of the form:*

- `When` x `then` z.

Where x is a current-event condition and z is a conjunction of property-update consequences.

Example of property-update rules are:

```
When any buyer says 'bid!'  then his bid price is increased by
10.
```

```
When the auctioneer says 'sold' then the last bidder becomes the
winner.
```

Note that the clause x `becomes` y means that the value of property y is overwritten with the value of property x. This can be understood as follows: suppose we have a property called `Carol's sister` and a property called `Bob's wife`. Furthermore suppose that `Carol's sister` is initialized to the value `'Alice'`. Then the clause `Carol's sister becomes bob's wife` means that the value `'Alice'` is copied into the property `Bob's wife`. Furthermore, note that when a property is assigned to an agent we use the key word `his` to refer to the agent that owns the property. To be precise: it refers to the last agent that appears earlier in the sentence. So in the above example, `his bid price` refers to the property named `bid price` assigned to the agent that said '`bid!`'.

Finally, note that property-update rules are written in present tense, while right-update rules are written in past tense. This is because they are interpreted in a fundamentally different way, which we will explain in Section 10.4.

Another way that values of properties are updated is when a message with content ('tell', x, y) is sent. In that case the value y is assigned to a property with name x. For example, whenever an agent sends a message with content ('tell', 'the price', 100), the value 100 is automatically assigned to a property with the name 'the price'. The protocol does not need to contain any property-initialization sentence for such a property.

10.3.4 Constraints

Right-update rules can be extended with so called constraints:

```
If the auctioneer has said 'open' then any
buyer can tell his bid price, as long as his
bid price is higher than the current price.
```

the clause `as long as` x `is higher than` y is called a constraint. A constraint is very similar to a property condition, but is written at the end of the sentence, indicated by the keywords `as long as`. A rule containing constraints is considered active if and only if all its conditions and constraints are satisfied.

The difference between constraints and conditions however, is that constraints refer to property values inside the consequences, whereas other conditions may only refer to past events or properties that do not appear inside the consequences. This distinction means that the truth of a condition can be determined independent of a message, and therefore can already be determined before the message is sent, while the truth value of a constraint on a message X can only be determined after the participant has submitted message X, when the interpreter is verifying whether message X is legal. In the example sentence above for instance, the constraint says that the bid price told by the buyer, must be higher than the current price. This can of course only be checked *when* the buyer is telling his bid price, and not before.

We are now ready to give the full definition of a right-update rule.

Definition 75 *A **right-update rule** is a sentence of the form:*

- *id* `can always say` v.
- *id* `can always tell` v.
- `If` x `then` y.
- `If` x `then` y, `as long as` w.

*where id is an identifier, v can be any character string, x and w are conjunctions of past-event conditions and/or property conditions and y is a conjunction of right-update consequences (the conditions in w are also referred to as **constraints**).*

Note that we allow such a rule to have no conditions at all, so that it is always active. In that case the protocol designer needs to include the keyword `always` after the keyword `can`.

Furthermore, we would like to remind the reader that right-update consequences can only have positive identifiers. This is important, because it means that a consequence can only *give* rights to an agent, but not take them away. Nevertheless, we can still make agents lose rights, but we do that by using negative conditions, rather than negative consequences. Take for example the following rule:

```
If the auctioneer has not said 'sold!' then any buyer can say
'bid!'.
```

Here, every buyer initially has the right to say '`bid!`', but loses that right once the auctioneer says '`sold!`', because the condition becomes false.

The big advantage of only allowing positive consequences, is that this makes it impossible to write inconsistent rules. An inconsistency would mean that

there is one rule that specifies that you can do something, while another rule says you can't do that. This is serious problem that one often encounters, for example in law. However, since we only allow positive consequences, this could never happen in our language.

Lemma 4 *A protocol written in SIMPLE is guaranteed to be free of inconsistencies.*

It is easy to prove this: in our language an agent has the right to do something if and only if there is an active rule with a consequence that gives this right to the agent. This can never lead to inconsistencies: either such a rule exists or not.

10.3.5 Summary

A protocol consists of sentences. There are four kinds of sentences:

- Role definition sentences
- Role constraint sentences
- Property initialization sentences
- Rules

There are two kinds of rules:

- Right-update rules
- Property-update rules

Rules consist of conditions, consequences and constraints. There are three kinds of conditions:

- Past-event conditions
- Current-event conditions
- Property conditions

and two kinds of consequences:

- Right-update consequences
- Property-update consequences

A right-update rule consists of:

- 0 or more past-event conditions
- 0 or more property conditions
- 1 or more right-update consequences

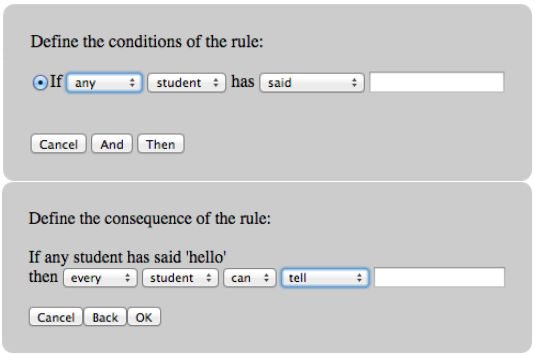


Figure 10.1: Two screen shots of the SIMPLE editor. Users write sentences simply by selecting available options, and they can only write free text whenever the syntax rules indeed allow that. Therefore, it is impossible to write malformed sentences.

- 0 or more constraints

A property-update rule consists of:

- Exactly 1 current-event condition
- 1 or more property-update consequences

10.4 The SIMPLE Interpreter

We will now describe the software component that interprets the protocols and enforces them. Whenever an agent tries to send a message, this message is first analyzed by the interpreter. The interpreter verifies if the agent sending the message indeed has the right to say that message and, if so, updates its internal state and forwards the message to the other agents connected to the server. If the sender of the message does not have the right to send that message he or she is notified that the message has failed. The message will in that case not be forwarded to the other agents and the internal state of the interpreter is not updated. In fact, we consider this message as not sent.

The interpreter keeps a list of all messages that have so far been sent successfully (the realized message history), and a world state that consists of a table that maps the name of each property to the current value of that property. Furthermore it keeps a table that maps the name of each agent in the MAS to the role it is playing, and a table that maps the name of each agent in the MAS to a list of rights for that agent (this table can be seen as an implementation of the permission map $\mathcal{G}$ of the protocol). Every time an agent tries to send a message, the interpreter follows the following procedure:

1. The list of rights of that agent is made empty.
2. For each right-update rule in the protocol, the interpreter verifies if its conditions are true:
   - If the condition is a property condition then it checks whether that property currently has the proper value to make the condition true.
   - If the condition is a past-event condition, the interpreter tries to find a message in the message history that matches the condition. If such a message is indeed found, then the condition is considered true.

   A rule for which all conditions are true is labeled as 'active'.
3. For each right-update consequence in each active rule, the interpreter checks whether the identifier matches the sender of the message and, if yes, adds the right corresponding to this consequence to the sender's list of rights. If this consequence has any constraints assigned to it, they are stored together with the right.
4. After all the rights of the sending agent have been determined the interpreter verifies whether any of them matches the message that the agent is trying to send.
5. Next, if the agent indeed has that right the interpreter checks whether its constraints (if any) are satisfied.
6. If the sending agent has the proper right, and all its constraints are satisfied then the interpreter determines if there are any property-update rules in the protocol for which the condition matches the message. If yes, the properties in the rule's consequences are updated accordingly.
7. Finally, if the agent has the right to send the message and its constraints are satisfied, a copy of the message is added in the message history, and the message is forwarded to all other agents in the MAS.

It is important to note here that property-update rules and right-update rules are treated in a different way. To be precise: to verify whether a past-event condition is true, the interpreter compares the condition with all messages in the message history. Since messages are never removed from the message history this means that whenever a past-event condition becomes true, it remains true forever. For example, when a buyer says 'hello' then the condition `any buyer has said 'hello'` becomes true, and remains true forever. For negative conditions exactly the opposite holds: the condition `no buyer has said 'bye'` is initially true, but as soon as a buyer says 'bye' it becomes false, and will stay false forever.

The current-event conditions on the other hand are only considered true at the moment that the corresponding message is under evaluation of the interpreter. That is, the condition `when a buyer says hello` is considered to be true only while the interpreter is evaluating the message with content ('say',

'hello') sent by some agent playing the role of buyer. As soon as the interpreter handles the next message this condition is considered false again.

The reason for this is that we consider that when you obtain a right, you keep that right for an extended period of time, until one of the negative conditions in the rule becomes false. Updating of a property on the other hand, is a one-time event that only takes place at the moment a certain message is sent.

10.5 Examples

We here provide two examples of protocols. Both have been tested and are correctly executed by the interpreter.

10.5.1 An English Auction Protocol

ROLES:

```
This protocol defines the role buyer
(plural:buyers).

This protocol defines the role auctioneer
(plural:auctioneers).

There must be exactly 1 auctioneer.

There must be at least 2 buyers.
```

PROPERTIES:

```
Initially, the highest bidder is no one.

Initially, the winner is no one.

Initially, the current price is 0.

Every buyer has a bid price which is initially 0.
```

RULES:

```
If the auctioneer has not said 'close'
then he can say 'open'.

If the auctioneer has said 'open'
then the auctioneer can say 'close'.

If the auctioneer has said 'open' and
the auctioneer has not said 'close' then
any buyer can tell his bid price, as long as his bid price is higher
than the current price.
```

```
When a buyer tells his bid price then his bid
price becomes the current price and he becomes
the highest bidder.

When the auctioneer says 'close' then the highest bidder becomes
the winner.
```

10.5.2 A Dutch Auction Protocol

ROLES:

```
This protocol defines the role buyer (plural:buyers).

This protocol defines the role auctioneer (plural:auctioneers).

There must be exactly 1 auctioneer.

There must be at least 2 buyers.
```

PROPERTIES:

```
Initially, the price is 1000.

Initially, the winner is no one.
```

RULES:

```
If no buyer has said 'mine' then the auctioneer can tell the next
price, as long as the next price is lower than the price.

When the auctioneer tells the next price then
the next price becomes the price.

If the auctioneer has told the price and
no buyer has said 'mine' then any buyer can
say 'mine'.

When a buyer says 'mine' then he becomes the winner.
```

10.6 Conclusions

We have introduced an initial version of a new language, called SIMPLE, for the definition of protocols. Its distinguishing property is that its syntax looks very similar to natural language and should therefore be easy enough to be used and understood by average people without specialist knowledge of programming or Electronic Institutions.

We have implemented an editor that should make it easy for people to write protocols in this language and an interpreter that verifies whether messages sent by agents obey the protocol or not. We have given two complete examples of

auction protocols specified in SIMPLE, and we have tested that these examples are executed correctly by the interpreter.

Our goal is to integrate the interpreter and the editor with the EIDE framework so that people can use SIMPLE to specify scene protocols. We think however that the current version of the language is still too limited to be of real practical use. We here list the shortcomings that we consider most important and that we plan to fix in the near future, as well as other improvements that we are considering.

Firstly, we will add the possibility to specify the recipient of a message. Currently every message is sent to all other agents in the MAS, which makes it impossible to send confidential information. This means we will allow to write sentences such as:

```
If the auctioneer has said 'welcome' to a
buyer then that buyer can say 'hello' to the
auctioneer.
```

Secondly, we would like the protocol designer to be able to express that a certain event must have taken place a certain number of times. For example:

```
If a buyer has told his bid price more than
5 times...
```

Thirdly, we would like to add more types of messages. and maybe even allow the protocol designer to define message types. That would make it possible to use certain domain-specific verbs. For example:

```
If the student has finished the assignment...
```

We could even take this a step further and allow the protocol designer to define new data types. Defining new types of objects is typically something that Inform 7 can handle well, so we may draw some inspiration from that language. Furthermore, we will add a system that determines at run time, whenever an agent tries to send an illegal message, which conditions first need to be fulfilled before the agent can indeed legally send that message. In this way the system can explain to the user why he or she made a mistake and will help the user to understand new protocols. In order to make the language more flexible and expressive, we will delve into literature about linguistics and apply some of its principles to our language.

Part IV

Conclusions & Future Work

Chapter 11

Conclusions

In Chapter 3 we have developed a formal model that unifies the subjects of Automated Negotiations, Game Theory, and Electronic Institutions. This model is based around the notions of *agents*, *messages* and *world states*. A *protocol* defines which agent can send which messages in which world state, and determines how the world state evolves as a consequence of the messages sent by the agents. We make a distinction between *active messages* (or *actions*) and *informative messages*. Active messages are sent in order to change the world state, while informative messages are only meant as a means of communication between the agents, in order to coordinate their actions. An action is called *infeasible* in a world state ϵ if it does not change that particular world state. A protocol is said to be *regimented* if all actions that are not allowed, are infeasible.

We have defined a *game* as a protocol, together with a utility function and a deadline for each agent. We have defined a *negotiation protocol* as a protocol, together with a space of possible deals (the *agreement space*) and a *confirmation function* that defines for each world state of the protocol which agreements are considered confirmed, and therefore binding. A deal over a one-shot game G is an agreement between a number of players that each will only play a move from a certain subset of its complete set of moves. Given a negotiation protocol N and a game G we have defined the *negotiation game* over G, which consists of two stages: the *negotiation stage* and the *action stage*. In the negotiation stage the players negotiate which moves from the game G they will make, and in the action stage they will make their moves, restricted by the deals they have committed themselves to during the negotiation stage. We have argued that under this model of negotiations, there is no clear and satisfactory notion of a reservation value.

We have introduced a new negotiation protocol, called *The Unstructured Negotiation Protocol*, in which negotiators are not obliged to reply to proposals, and after making a proposal negotiators are not obliged to wait for response from the opponents; they may make a new proposal whenever they want.

In Chapter 4 we have presented the three negotiation scenarios of investigated in this thesis, each involving large agreement spaces and non-linear utility

functions. In the first case, the ANAC domain, the negotiations are bilateral and utility functions are expressed as a sum over constraints. Although in principle any function over a finite domain can be expressed in this way, it is in many cases impossible in practice to explicitly write down a given function in this way. The second test case described is a new problem, called the NSP, for which calculating the utility value of a deal is harder, as it involves solving an NP-hard problem. We also explain how the NSP could be adapted in order to make it a more realistic model for real-world package delivery. The third test case described is the game of Diplomacy which is a widely played board game that involves negotiations. This test case is even more complex than the other test cases, because a deal only partially restrict the players' moves and because the outcome also depends on the opponents' moves, so calculating the utility value of a deal is a Game Theoretical problem.

In Chapter 5 we have introduced an agent that makes use of Genetic Algorithms in order to negotiate over a domain where the utility functions are given in terms of constraints, which are in turn defined by rectangular subspaces. The trade-off between maximizing the agent's own utility and maximizing the opponent's utility was solved by using an time-based aspiration level for our agent's own utility, and at the same time demanding that the proposed deals were close to earlier proposals made by the opponent. If both criteria are met by more than one deal, then our agent proposes the deal that is most different from earlier proposals made by our agent. Furthermore, we have introduced a new acceptance strategy.

Our agent has participated in the ANAC'14 competition and obtained the second and third place in its two respective categories, among more than 20 participants. We conclude that our agent is a good negotiator and that Genetic Algorithms are a good search technique for the given domain. We think however that the utility functions as given in the competition were too simplistic, because in reality constraints are not always given by rectangular subspaces and often involve more than 4 variables.

In Chapter 6 we have introduced a new family of negotiation algorithms for very large and complex agreement spaces, with multiple selfish agents, non-linear utility functions and a limited amount of time. This family is called NB^3 and applies best-first Branch and Bound to search for good proposals. We have defined a general purpose heuristic to guide the B&B search of the NB^3 algorithm. Furthermore we have defined a new concession strategy that applies two aspiration levels because it considers the utility aspired by our agent and the utility to be conceded to the opponents as two separate quantities. This allows our agent to determine not only what to propose, but also whether it should make a proposal or rather continue searching for better proposals.

We have implemented an instance of NB^3 for the NSP that we call Branesal and we have performed several experiments with it. From these experiments we conclude that, our agent indeed manages to decrease its costs significantly by negotiation, that the results remain stable with increasing size of the problem instances, that redundant nodes in the search tree are successfully pruned, that

our heuristic search performs significantly better than random search, and that in the case of simple problem instances with a clear optimal solution our algorithm manages to approach that solution.

In Chapter 7 we have introduced the concept of a Constraint Optimization Game, which combines aspects of Constraint Optimization with Game Theory, and show that a single round of Diplomacy can be modeled as a Negotiation Game over a COG. Furthermore, we have implemented an agent, called D-Brane, that applies the NB^3 algorithm to explore the agreement space and applies And/Or search with B&B to determine the utility value of any given deal.

We have presented experiments in which we compare D-Brane with two other negotiating agents and show that D-Brane plays significantly better. However, it also shows that the negotiation algorithm seems to have only a small impact on the results. We think that D-Brane will be very important for future negotiations research, because it allows researchers to compare their strategies not only with the DumbBot but also with our much stronger agent.

In Chapter 8 we have presented a new extension to the EIDE framework, called GENUINE, which allows humans to interact with each other and with software agents in an Electronic Institution. The tool generates a default user interface automatically, so the creator of an institution can directly use it without having to design anything. However, it still allows designers to design a custom GUI, tailored to a specific institution, without having to worry about getting the necessary information from the EI. We argue that the default interface is useful for testing purposes, but less useful for a user-friendly end product, because it is only able to represent messages, but not world states.

Furthermore, in Chapter 9 we have argued that many different social networks nowadays exist next to each other, because they all require different rules to be imposed on their users. We think that social networks in the future could be based on EI technology, so that rules can be flexible and allow people to develop their own communities within existing social networks, with their own regulations tailored to the needs of that community. Humans could negotiate the rules of such communities and may even have automated agents to negotiate those rules on their behalves.

Finally, in Chapter 10 we have introduced a new language, called SIMPLE, for the definition of protocols. The syntax of this language looks very similar to natural language and should therefore be easy enough to be used and understood by average people without specialist knowledge of programming or Electronic Institutions. We have implemented an editor that should make it easy for people to write protocols in this language and an interpreter that verifies whether messages sent by agents obey the protocol or not. We have given two complete examples of auction protocols specified in SIMPLE, and we have tested that these examples are executed correctly by the interpreter. Our goal is to integrate the interpreter and the editor with the EIDE framework so that people can use SIMPLE to specify scene protocols. However, the current version of the language still lacks a number of important features that we think are necessary before it can be used in practice.

Chapter 12

Future Work

The NB^3 algorithm is still based on the assumption that we have some expression of the agents' preferences in terms of numerical utility functions. We think that for practical applications this is an unrealistic assumption. Instead, we think it is much more realistic to assume a model in which preferences are modeled in terms of a partial ordering. Therefore, we will adapt NB^3 so that it can handle qualitative preferences that are expressed in terms of logical sentences, such as in the theoretical work done by Dongmo Zhang [Zhang, 2005].

Another aspect of the NB^3 algorithm that deserves more attention is the concession strategy. Currently, the values of the concession degrees need to be determined per domain. We hope to find a strategy that allows the concession degrees of the aspiration levels to be determined at run time. We may also allow the final values of the aspiration levels to be variable (currently they only take the values 0 and 1 respectively). Furthermore, as explained in Section 6.3, in the current implementation the agent treats the set of opponents as if it were one opponent to which it should concede. The negotiation strategy can be improved by dropping this assumption, so that our agent can apply a different concession strategy for every opponent. Also, we have mentioned in Section 6.3.2 that there is a parameter that determines how long the agent continues expanding the search tree before deciding whether to make a new proposal or not (the expandInterval parameter in Algorithm 2). We will investigate the influence of the value of this parameter on the results.

Since we have successfully applied And/Or search to the strategic component of D-Brane, we consider it a logical next step to also apply And/Or search to the negotiation algorithm. That is: we will develop an improved version of NB^3 that applies And/Or search as well as B&B. We expect that this will highly improve the efficiency of NB^3.

We plan to develop yet another test case for negotiation problems. However, this time we will make this domain as realistic as possible. That is: plan to create a virtual travel agency in which the clients can negotiate with the agency about the prices and components of their holidays. By taking such an explicit example we can make sure that nothing essential is abstracted way, and we force

developers of negotiation algorithms to take all practical problems into account. We plan to make this domain available online so that researchers can test and demonstrate their algorithms online. This web site will contain an example negotiator based on the NB^3 algorithm.

We plan to make D-Brane publicly available on the DipGame website and we will detach the negotiating component from the strategic component so that other researchers can put their own negotiation algorithms on top of the strategic component of D-Brane. Furthermore, we will further develop the negotiation component of D-Brane so that it may make more sophisticated deals rather than just battle plans. In this way we hope to improve the impact of negotiations on the results of D-Brane. Also, we plan to endow the DumbBot with our negotiation algorithm, to see if negotiations in that case have a greater impact. Finally, we would like to see if our approach for solving COGs indeed can be generalized. We may for example define games in Game Description Language (GDL) [Genesereth et al., 2005] so that the strategic component can decompose the game into micro-games at runtime.

Regarding our language SIMPLE, we think that it is currently still too limited to be of real practical use. We here list the main shortcomings that we consider most important and that we plan to fix in the near future, as well as other improvements that we are considering. Firstly, we will add the possibility to specify the receivers of a message. Currently every message is sent to all other agents in the MAS, which makes it impossible to send confidential information. This means we will allow to write sentences such as:

> *If the auctioneer has said 'welcome' to a buyer then that buyer can say 'hello' to the auctioneer.*

Secondly, we would like the protocol designer to be able to express that a certain event must have taken place a certain number of times. For example:

> *If a buyer has told his bid price more than 5 times...*

Thirdly, we would like to add more types of messages. and maybe even allow the protocol designer to define message types, which would make it possible to use verbs other than 'to say' or 'to tell' which are more specific for the domain of application. We could even take this a step further and allow the protocol designer to define new data types. Defining new types of objects is typically something that Inform 7 can handle well, so we may draw some inspiration from that language. Furthermore, we will add a system that determines at run time, whenever an agent tries to send an illegal message, which conditions first need to be fulfilled before the agent can indeed legally send that message. In this way the system can explain to the user why he or she made a mistake and will help the user to understand new protocols. In order to make the language more flexible and expressive, we will study the literature about linguistics and apply some of its principles to our language. Finally, we will make sure a demo version of the protocol editor and the interpreter will be available online.

Bibliography

[hyp, 2014] (2014). Hypertalk. `http://en.wikipedia.org/wiki/HyperTalk`.

[pla, 2014] (2014). The osmosian order of plain english programmers. `http://www.osmosian.com/`.

[Alberti et al., 2006] Alberti, M., Gavanelli, M., Lamma, E., Chesani, F., Mello, P., and Torroni, P. (2006). Compliance verification of agent interaction: a logic-based software tool. *Applied Artificial Intelligence*, 20(2-4):133–157.

[An et al., 2009] An, B., Gatti, N., and Lesser, V. (2009). Extending alternating-offers bargaining in one-to-many and many-to-many settings. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '09, pages 423–426, Washington, DC, USA. IEEE Computer Society.

[An et al., 2006] An, B., Sim, K. M., Tang, L., Li, S., and Cheng, D. (2006). Continuous time negotiation mechanism for software agents. *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics*, 36(6):1261–1272.

[Arcos et al., 2005] Arcos, J. L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J. A., and Sierra, C. (2005). Engineering open environments with electronic institutions. *Engineering Applications of Artificial Intelligence*, 18(2):191–204.

[Argente et al., 2008] Argente, E., Criado, N., Botti, V., and Julian, V. (2008). Norms for agent service controlling. *EUMAS-08*, pages 1–15.

[Artikis et al., 2005] Artikis, A., Kamara, L., Pitt, J., and Sergot, M. (2005). A protocol for resource sharing in norm-governed ad hoc networks. In Leite, J. a., Omicini, A., Torroni, P., and Yolum, p., editors, *Declarative Agent Languages and Technologies II*, volume 3476 of *Lecture Notes in Computer Science*, pages 221–238. Springer Berlin Heidelberg.

[Baarslag et al., 2013] Baarslag, T., Hindriks, K., and Jonker, C. (2013). Acceptance conditions in automated negotiation. In *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pages 95–111. Springer Berlin Heidelberg.

[Baarslag et al., 2010] Baarslag, T., Hindriks, K., Jonker, C. M., Kraus, S., and Lin, R. (2010). The first automated negotiating agents competition (ANAC 2010). In Ito, T., Zhang, M., Robu, V., Fatima, S., and Matsuo, T., editors, *New Trends in Agent-based Complex Automated Negotiations, Series of Studies in Computational Intelligence*. Springer-Verlag.

[Bektas, 2006] Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219.

[Belnap and Perloff, 1990] Belnap, N. and Perloff, M. (1990). Seeing to it that: A canonical form for agentives. In Kyburg, Henry E., J., Loui, R. P., and Carlson, G. N., editors, *Knowledge Representation and Defeasible Reasoning*, volume 5 of *Studies in Cognitive Systems*, pages 167–190. Springer Netherlands.

[Broersen et al., 2004] Broersen, J., Dignum, F., Dignum, V., and Meyer, J.-J. C. (2004). Designing a deontic logic of deadlines. In *Deontic Logic in Computer Science*, pages 43–56. Springer Berlin Heidelberg.

[Cardoso et al., 2013] Cardoso, H. L., Urbano, J., Rocha, A. P., Castro, A. J., and Oliveira, E. (2013). Ante: Agreement negotiation in normative and trust-enabled environments. In Ossowski, S., editor, *Agreement Technologies*, volume 8 of *Law, Governance and Technology Series*, pages 549–564. Springer Netherlands.

[Cranefield, 2005] Cranefield, S. (2005). A rule language for modelling and monitoring social expectations in multi-agent systems. Technical report, In: Selected and Revised papers from the AAMAS 2005 Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems. Volume 3913 of Lecture.

[Cranefield and Winikoff, 2011] Cranefield, S. and Winikoff, M. (2011). Verifying social expectations by model checking truncated paths. *Journal of Logic and Computation*, 21(6):1217–1256.

[DailyMail, 2012] DailyMail (2012). http://www.dailymail.co.uk/news/article-1205794/rape-horror-tourist-used-couchsurfing-website-aimed-travellers.html#ixzz29y3wxuck.

[Dastani et al.,] Dastani, M., Tinnemeier, N. A., and Meyer, J.-J. C. A programming language for normative multi-agent systems.

[Dechter and Mateescu, 2007] Dechter, R. and Mateescu, R. (2007). And/or search spaces for graphical models. *Artificial Intelligence*, 171(23):73 – 106.

[d'Inverno et al., 2012] d'Inverno, M., Luck, M., Noriega, P., Rodríguez-Aguilar, J. A., and Sierra, C. (2012). Communicating open systems. *Artificial Intelligence*, 186:38–64.

[Endriss, 2006] Endriss, U. (2006). Monotonic concession protocols for multilateral negotiation. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 392–399, New York, NY, USA. ACM.

[Esteva, 2003] Esteva, M. (2003). *Electronic Institutions: From Specification to Development*. PhD thesis, Technical University of Catalonia.

[Esteva et al., 2002] Esteva, M., de la Cruz, D., and Sierra, C. (2002). Islander: en electronic institutions editor. volume 3, pages 1045–1052, Bologna, Italy. ACM PRESS.

[Esteva et al., 2008] Esteva, M., Rodríguez-Aguilar, J. A., Arcos, J. L., Sierra, C., Noriega, P., Rosell, B., and de la Cruz, D. (2008). Electronic institutions development environment. pages 1657–1658, Estoril, Portugal. International Foundation for Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems.

[Esteva et al., 2004] Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., and Arcos, J. L. (2004). Ameli: An agent-based middleware for electronic institutions. volume I, pages 236–243. ACM, ACM.

[Fabregues, 2014] Fabregues, A. (2014). *Facing the Challenge of Automated Negotiations with Humans*. PhD thesis, Universitat Autònoma de Barcelona.

[Fabregues and Sierra, 2011] Fabregues, A. and Sierra, C. (2011). Dipgame: a challenging negotiation testbed. *Engineering Applications of Artificial Intelligence*.

[Falkenauer, 1998] Falkenauer, E. (1998). *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, Inc., New York, NY, USA.

[Faratin et al., 1998] Faratin, P., Sierra, C., and Jennings, N. R. (1998). Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159 – 182. Multi-Agent Rationality.

[Faratin et al., 2000] Faratin, P., Sierra, C., and Jennings, N. R. (2000). Using similarity criteria to make negotiation trade-offs. In *International Conference on Multi-Agent Systems, ICMAS'00*, pages 119–126.

[Fatima et al., 2009] Fatima, S., Wooldridge, M., and Jennings, N. R. (2009). An analysis of feasible solutions for multi-issue negotiation involving nonlinear utility functions. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1041–1048, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Ferreira et al., 2015] Ferreira, A., Lopes Cardoso, H., and Paulo Reis, L. (2015). Dipblue: A diplomacy agent with strategic and trust reasoning. In *7th International Conference on Agents and Artificial Intelligence (ICAART 2015)*, pages 398–405.

[Fornara et al., 2013] Fornara, N., Cardoso, H. L., Noriega, P., Oliveira, E., Tampitsikas, C., and Schumacher, M. I. (2013). *Modelling Agent Institutions*, chapter 18, pages 277–307. Number 8. Springer-Verlag GmdH.

[García-Camino, 2008] García-Camino, A. (2008). Ignoring, forcing and expecting simultaneous events in electronic institutions. In *Proceedings of the 2007 International Conference on Coordination, Organizations, Institutions, and Norms in Agent Systems III*, COIN'07, pages 15–26, Berlin, Heidelberg. Springer-Verlag.

[Genesereth et al., 2005] Genesereth, M., Love, N., and Pell, B. (2005). General game playing: Overview of the aaai competition. *AI Magazine*, 26(2):62–72.

[Governatori et al., 2005] Governatori, G., Rotolo, A., and Sartor, G. (2005). Temporalised normative positions in defeasible logic. In *Procedings of the 10th International Conference on Artificial Intelligence and Law*, pages 25–34. ACM Press.

[Hazelden et al., 2012] Hazelden, K., Yee-King, M., Amgoud, L., d'Inverno, M., Sierra, C., Osman, N., Confalonieri, R., and de Jonge, D. (2012). Wecurate: Designing for synchronised browsing and social negotiation. Dubrovnik, Croatia.

[Hemaissia et al., 2007] Hemaissia, M., El Fallah Seghrouchni, A., Labreuche, C., and Mattioli, J. (2007). A multilateral multi-issue negotiation protocol. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 155:1–155:8, New York, NY, USA. ACM.

[Hindriks and Tykhonov, 2008] Hindriks, K. and Tykhonov, D. (2008). Opponent modelling in automated multi-issue negotiation using bayesian learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, AAMAS '08, pages 331–338, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Hübner et al., 2006] Hübner, J. F., Sichman, J. S. a., and Boissier, O. (2006). S-moise+: A middleware for developing organised multi-agent systems. In Boissier, O., Padget, J., Dignum, V., Lindemann, G., Matson, E., Ossowski, S., Sichman, J. S. a., and Vázquez-Salceda, J., editors, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *Lecture Notes in Computer Science*, pages 64–77. Springer Berlin Heidelberg.

[Ito et al., 2008] Ito, T., Klein, M., and Hattori, H. (2008). A multi-issue negotiation protocol among agents with nonlinear utility functions. *Multiagent Grid Syst.*, 4:67–83.

[Klein et al., 2003] Klein, M., Faratin, P., Sayama, H., and Bar-Yam, Y. (2003). Protocols for negotiating complex contracts. *Intelligent Systems, IEEE*, 18(6):32 – 38.

[Koenig et al., 2006] Koenig, S., Tovey, C., Lagoudakis, M., Markakis, V., Kempe, D., Keskinocak, P., Kleywegt, A., Meyerson, A., and Jain, S. (2006). The power of sequential single-item auctions for agent coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1625–1629.

[Kollingbaum, 2005] Kollingbaum, M. J. (2005). *Norm-governed practical reasoning agents*. PhD thesis, University of Aberdeen.

[Kraus, 1995] Kraus, S. (1995). Designing and building a negotiating automated agent. *Computational Intelligence*, 11:132–171.

[Krishna and Serrano, 1996] Krishna, V. and Serrano, R. (1996). Multilateral bargaining. *Review of Economic Studies*, 63(1):61–80.

[Kröger, 1987] Kröger, F. (1987). *Temporal Logic of Programs*. Springer-Verlag New York, Inc., New York, NY, USA.

[Lai et al., 2008] Lai, G., Sycara, K., and Li, C. (2008). A decentralized model for automated multi-attribute negotiations with incomplete information and general utility functions. *Multiagent Grid Syst.*, 4:45–65.

[Lauterbach et al., 2009] Lauterbach, D., Truong, H., Shah, T., and Adamic, L. (2009). Surfing a web of trust: Reputation and reciprocity on couchsurfing.com. In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, volume 4, pages 346–353.

[Lawler and Wood, 1966] Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719.

[Lewis, 1974] Lewis, D. (1974). Semantic analyses for dyadic deontic logic. In Stenlund, S., editor, *Logical Theory and Semantic Analysis: Essays Dedicated to Stig Kanger on His Fiftieth Birthday*, pages 1–14. Reidel, Dordrecht.

[Lopez y Lopez et al., 2004] Lopez y Lopez, F., Luck, M., and Puebla, A. (2004). A model of normative multi-agent systems and dynamic relationships. In *Regulated Agent-Based Social Systems. Volume 2934 of Lecture Notes in Artificial Intelligence*, pages 259–280. Springer.

[Makinson and Van Der Torre, 2000] Makinson, D. and Van Der Torre, L. (2000). Input/output logics. *Journal of Philosophical Logic*, 29(4):383–408.

[Marsa-Maestre et al., 2009a] Marsa-Maestre, I., Lopez-Carmona, M. A., Velasco, J. R., and de la Hoz, E. (2009a). Effective bidding and deal identification for negotiations in highly nonlinear scenarios. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1057–1064, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Marsa-Maestre et al., 2009b] Marsa-Maestre, I., Lopez-Carmona, M. A., Velasco, J. R., Ito, T., Klein, M., and Fujita, K. (2009b). Balancing utility and deal probability for auction-based negotiations in highly nonlinear utility spaces. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 214–219, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Meyer, 1987] Meyer, J.-J. C. (1987). A different approach to deontic logic: deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29(1):109–136.

[Modi et al., 2005] Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 161(1-2):149–180.

[Nash, 1950] Nash, J. (1950). The bargaining problem. *"Econometrica"*, "18":155–162.

[Nash, 1951] Nash, J. (1951). Non-cooperative games. *Annals of Mathematics*, 54(2):pp. 286–295.

[Nelson, 2014] Nelson, G. (2014). Natural language, semantic analysis and interactive fiction. `http://inform7.com/learn/documents/WhitePaper.pdf`.

[Nguyen and Jennings, 2004] Nguyen, T. D. and Jennings, N. R. (2004). Coordinating multiple concurrent negotiations. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '04, pages 1064–1071, Washington, DC, USA. IEEE Computer Society.

[Noriega, 1997] Noriega, P. (1997). *Agent Mediated Auctions: The Fishmarket Metaphor*. PhD thesis, Autonomous University of Barcelona.

[Nute, 1997] Nute, D. (1997). *Defeasible Deontic Logic*. Springer.

[Ortner, 2012] Ortner, J. M. (2012). A continuous time model of bilateral bargaining.

[Osborne and Rubinstein, 1994] Osborne, M. and Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press.

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

[Poundstone, 1993] Poundstone, W. (1993). *Prisoner's Dilemma*. Doubleday, New York, NY, USA, 1st edition.

[Robu et al., 2005] Robu, V., Somefun, D. J. A., and Poutré, J. A. L. (2005). Modeling complex multi-issue negotiations using utility graphs. In *Proceedings of AAMAS'05*, pages 280–287.

[Rosenschein and Zlotkin, 1994] Rosenschein, J. S. and Zlotkin, G. (1994). *Rules of Encounter*. The MIT Press, Cambridge, USA.

[Rossi et al., 2006] Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.

[Rubinstein, 1982] Rubinstein, A. (1982). Perfect Equilibrium in a Bargaining Model. *Econometrica*, 50(1):97–109.

[S. Kraus, 1989] S. Kraus, D. Lehman, E. E. (1989). An automated diplomacy player. In Levy, D. and Beal, D., editors, *Heuristic Programming in Artificial Intelligence: The 1st Computer Olympia*, pages 134–153. Ellis Horwood Limited.

[Schmitt, 2001] Schmitt, L. M. (2001). Theory of genetic algorithms. *Theoretical Computer Science*, 259(12):1 – 61.

[Sergot and Craven, 2006] Sergot, M. and Craven, R. (2006). The deontic component of action language nc+. In Goble, L. and Meyer, J.-J. C., editors, *Deontic Logic and Artificial Normative Systems*, volume 4048 of *Lecture Notes in Computer Science*, pages 222–237. Springer Berlin Heidelberg.

[Serrano, 2008] Serrano, R. (2008). bargaining. In Durlauf, S. N. and Blume, L. E., editors, *The New Palgrave Dictionary of Economics*. Palgrave Macmillan, Basingstoke.

[Sierra and Debenham, 2007] Sierra, C. and Debenham, J. (2007). The logic negotiation model. In *Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 1026–1033. ACM.

[Tampitsikas et al., 2012] Tampitsikas, C., Bromuri, S., and Schumacher, M. (2012). Manet: A model for first-class electronic institutions. In Cranefield, S., Vazquez-Salceda, J., van Riemsdijk, B., and Noriega, P., editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems VII*, volume 7254 of *Lecture Notes in Artificial Intelligence*. Springer Verlag.

[Trescak et al., 2013] Trescak, T., Rodriguez, I., Sanchez, M. L., and Almajano, P. (2013). Execution infrastructure for normative virtual environments. *Engineering Applications of Artificial Intelligence*, 26(1):51 – 62.

[Uszok et al., 2008] Uszok, A., Bradshaw, J. M., Lott, J., Breedy, M., Bunch, L., Feltovich, P., Johnson, M., and Jung, H. (2008). New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of kaos. *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:145–152.

[van der Hoek et al., 2007] van der Hoek, W., Roberts, M., and Wooldridge, M. (2007). Social laws in alternating time: effectiveness, feasibility, and synthesis. *Synthese*, 156(1):1–19.

[Vázquez-Salceda et al., 2004] Vázquez-Salceda, J., Aldewereld, H., and Dignum, F. (2004). Implementing norms in multiagent systems. *Multiagent system technologies*, pages 313–327.

[von Wright, 1951] von Wright, G. H. (1951). Deontic logic. *Mind*, 60:1–15.

[Williams et al., 2011] Williams, C. R., Robu, V., Gerding, E. H., and Jennings, N. R. (2011). Using gaussian processes to optimise concession in complex negotiations against unknown opponents. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pages 432–438. AAAI Press.

[Yee-King et al., 2013] Yee-King, M., Confalonieri, R., de Jonge, D., Hazelden, K., Sierra, C., d'Inverno, M., Amgoud, L., and Osman, N. (2013). Multiuser museum interactives for shared cultural experiences: an agent based approach. Saint Paul, Minnesota, USA.

[Zhang, 2005] Zhang, D. (2005). A logical model of nash bargaining solution. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 983–990.