Negotiation for Strategic Video Games

Einar Nour Afiouni
Leif Julian Øvrelid

Master of Science in Computer Science
Submission date: June 2013
Supervisor: Pinar Öztürk, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

The field of multi-agent systems and the use of game theoretic interactions is growing ever more popular in the academic artificial intelligence communities. We think that this kind of AI will be very useful in improving the realism and effectiveness of simulated opponents in video games. Academic AI has not been used a lot in the video game industry as it is often computationally expensive. Video games require fast response times and use most of their computing time for graphics and physics calculations, making advanced AI techniques less feasible in this domain. This project examines the use of multi-agent negotiation techniques to expand the AI in a strategic video game. We have implemented such a negotiation system in the video game Civilization IV from Firaxis Games, focusing on making an efficient negotiating AI capable of handling a multi-issue negotiation domain with limited information of its opponent's preferences. In order to overcome the difficulty of limited information, our system incorporates modeling of the opponent's preferences. In this project, we examine the effectiveness of two such modeling techniques, frequency modeling and Bayesian modeling, evaluating their performance and usability in the video game domain.

Sammendrag

I dette prosjektet undersøker vi mulighetene for å bruke spillteoretiske konsepter og multiagent systemer i moderne videospill med sanntidskrav. Vi har implementert et forhandlingssystem for det strategiske videospillet Civilization IV. Vi har evaluert flere ulike forhandlingsteknikker med et fokus på bruk av modelleringsteknikker som finner motstanderens preferanser for å forbedre forhandlingsresultatene.

Preface

This project was conducted at the department of Computer & Information Science at NTNU as a master thesis project during the spring of 2013. We would like to thank our advisor Pinar Öztürk for helping us and giving us useful pointers to what we should focus on in our project. We would also like to thank Firaxis Games for making the Civilization IV Software Development Kit free to use so that we can develop modifications for the AI in this game. The finished modification will not be our property as it is a mod for a commercial video game owned by Firaxis Games and 2K Games, and we will distribute the modification free of charge.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The Video game industry is a massive and fast growing industry, often driving modern computer science forward with its need for the most advanced technology to provide an ever more immersive world for the players. When it comes to artificial intelligence, however, games are lagging behind the current state of the art in academic AI. As games often try to immerse the players in a virtual world, they require low computation times to provide fast response to the players' actions. This makes them soft real-time systems, where fast responses are not strictly necessary, but are required to make the user experience good. As academic AI often explores techniques that require long computation times to make difficult decisions, these techniques are difficult to utilize in a video game domain. In the modern video-game industry, however, the focus is shifting more and more towards the use of artificial intelligence to improve the realism and immersion of games, and in order to do this, it needs to look at the vast field of academic AI for inspiration. In this project, we aim to bring the use of multi-issue interactions and game theory to real-time video games. We do this by expanding the artificial intelligence system in the video game Civilization IV using a game theoretical approach. We focus on the use of multi-agent negotiation systems to improve how the artificial intelligence handles diplomacy, both against other AI's and against real players. Civilization IV is a so called grand strategy game for up to eighteen players, where each player controls one nation and struggles for world domination through war and diplomacy. For a more detailed explanation of Civilization IV, see section 2.

1.1 Motivations

The standard opponent AI in Civilization IV is rather simple and limited when it comes to diplomacy. The system does not allow for counter offers, only allowing accepting and declining offers. In addition, it does not reason about its opponents, and when negotiating with other AI controlled opponents, all the offers they make are automatically accepted as long as they are possible. A more advanced system for negotiation, where the agents are allowed to make counter offers and utilize modern negotiation techniques could vastly improve the player's experience in such games. The Civilization IV negotiation domain is a complex one however, where the negotiations involve multiple issues at the same time, and there is no information about the intentions of the opponents. This is a very difficult domain, and achieving good negotiation results in such a domain is the focus of a lot of research in negotiation theory. Multi-issue negotiations such as in our system make it very difficult to find optimal deals in polygonal time, as the space of possible offers to search through grows exponentially in the number of negotiation issues. This makes it unfeasible to search through all the possible offers to find the best one. Therefore, some optimization and simplification is necessary in order to find deals that are close to optimal in polynomial time. The lack of information about the opponent's preferences is another complicating factor, because making a reasonable concession is difficult when you do not know what the opponent wants. Such difficulties can often be resolved by utilizing opponent modeling techniques from the machine learning field to find the opponent's preferences. In this project we attempt to utilize such techniques and adapt them to the AI in Civilization IV, making it a more challenging and interesting opponent or ally. In order to reason about what to negotiate about and about the preferences of the opponent, we can conceive the problem as a constaint satisfaction problem in the AI field where the preferences of the agent and its opponents can be represented as constraints, making the process of searching for offers a process of constraint satisfaction.

1.2 Task Specification and Scope

The principal idea behind our project is to utilize negotiation theory to improve the decision making in the negotiations in the strategic video game Civilization IV. Using a more advanced negotiation system where counter offers are allowed makes it easier to find a better deal than what can be found if each agent only sends offers suggesting things the agent wants for itself, and the opponent can only accept or decline. A more advanced system, however, needs to allow trade-offs in order to find deals that are Pareto optimal or maximize social welfare. Pareto optimality and social welfare are important properties for negotiation results, describing how good the results actually are (see section 3 for an explanation of these properties). It is therefore a goal of this project to find whether a system guaranteeing Pareto optimality or maximization of social welfare can be made in the Civilization IV domain. The negotiation domain in Civilization IV is a complex one, where the agents or players can negotiate for resources and treaties all at the same time, without having to reveal any information to the other players. This means we are dealing with a multi-issue negotiation with limited information. An important part of the project will be to find a negotiation protocol/strategy that is fast and efficient so that the agents can come to an agreement within a short time frame (Civilization IV is a turn based game and each AI player typically uses about a second per turn). Making the negotiation fast enough to keep the response time short is an important task, as negotiations might be a time consuming process. Due to this, we will prioritize lowering computational complexity over improving negotiation outcomes, although both are important.

1.3 Research Questions

RQ1 Evaluate how well a multi-issue negotiation system with limited information will work in a complex, soft real-time system such as the video game Civilization IV.

RQ2 Is it possible to achieve Pareto optimal, or social welfare maximizing negotiation results in the domain of Civilization IV?

RQ3 Will opponent modeling improve the negotiation results in this domain? Which type of opponent modeling leads to better results? The quality of the results will be evaluated according to the success rate of a series of negotiations and the average utility of a series of negotiation as detailed in section 6.

1.4 Contributions

We have developed a new negotiation system for Civilization IV that uses opponent modeling and constraint based negotiation to achieve better results from negotiations than the ones obtained by the original AI. We have implemented two different opponent modeling techniques, one based on frequency modeling and one based on Bayesian learning, and have compared these techniques with each other, evaluating their usefulness in our domain. There have been a few other attempts at making negotiation based AI's for strategic games similar to Civilization IV, but these have, as far as we are aware of, been limited to turn based board games, where the computation times are less important. In commercial video games, which traditionally leave very little processing time to the AI systems, we are not aware of any systems that utilize these kind of techniques. Our system constitutes an important contribution in showing that multi-agent interactions are both feasible and useful in the field of real time video games.

1.5 Research Method

Our research started with a conceptual design of a negotiation system we have made in our previous semester. This design was implemented in Civilization IV. We also modified Civilization IV such that we can gather data to examine the results of our approach. To answer our research question, we implemented two types of opponent modeling which we compared to each other. In addition, we implemented agents capable of negotiating with full information about their opponents and agents negotiating with no opponent modeling and no information. These agents are used for comparison, to see how much opponent modeling can improve the results of the negotiation. In order to test the system, we have created a set of negotiation scenarios, tailored to test how our system copes with different and often difficult situations. The different opponent modeling types have been run against each other on these scenarios while logging the results, that is the correctness of the opponent models and the time use of the negotiations. The results of our proposed solution can be found in chapter 5. In addition to these tests, we have also analyzed our system manually to find whether it can provide Pareto optimal or social welfare maximizing results. This analysis can be found in sections 5.4.4 and 5.4.5.

1.6 Report outline

Our report starts by describing the game, Civilization IV, that we are modifying in chapter 2, before introducing useful terms and background information in chapter 3. Chapter 4 then examines other relevant scientific work, before we present the system we have developed in chapter 5. We then define the experiments we have performed to verify our research questions in chapter 6, before we provide the results and discussion in chapter 7. In chapter 8, we sum up what we have found out, and present the conclusions we have drawn. Chapter 9 gives pointers to what we intend to work more on in the future, and research directions future work in this field could focus on.

Chapter 2

Introduction to Civilization IV

This chapter introduces the video game Civilization IV. We give an overview of the AI in the game, explaining it in terms of game theoretic concepts such as coalitions and negotiations.

2.1 What Is Civilization IV

Civilization IV (from here on CivIV) is a large scale, turn based strategy game in a genre often called grand strategy or 4X (short for Explore, Expand, Exploit and Exterminate). It is played on a large map in which the players control an empire from its small beginnings as a single city, towards world domination. The general game-play consists of building cities, building improvements and units in your cities, researching technologies, exploring the world map with your units, and most importantly, conducting diplomacy and warfare with your opponents. By default, there are five different scenarios where a player wins the game: by conquering all other civilizations, controlling a majority of the world's land an population, increasing the culture ratings of three different cities to a "Legendary" level, sending a rocket ship to the Alpha Centauri star system or by being declared "World Leader" through election by the United Nations. If the game's clock runs out before any of the latter goals are achieved by anyone, the civilization with the highest score wins the game.

2.2 The Original AI Of The Game

The diplomatic part of the CivIV AI is quite simple, and is largely based on keeping track of the *relationships* between all the players. These *relationships*, represented by a number, are the primary source of all diplomatic decisions, such as whether to start a war, make a treaty or trade and so on. The decision for whether to go to war, is calculated based on whether you have a military force with which you think you can handle a war, how many wars you are currently involved in, whether you have an opponent with *relationship* value below a certain threshold, and a random value. The random value is there to make the the AI less predictable so that players will not be able to know when an opponent will decide to start a war. The AI always chooses the player with which it has the worst relationship when it wants to go to war, regardless of strategic importance.

During negotiations, there are a large number of different treaties and resources that can be offered. These are often traded among players as a means of getting something you have, such as more gold or a specific technology, but can also be given as gifts to improve your relations, or be used to pay for a treaty such as allowing open borders, or making peace. Often players who have good relations with you will ask for resources without payment, and when they enter wars, they will generally ask you to join them using this trading system, although they will generally not offer compensation for such treaties. The following is an overview of the trade-able items in CivIV, most of these items and treaties are trade-able only when you have good relations.

2.2.1 Trade-able Items

- Gold
- Treaty to give a certain amount of gold per turn
- World map
- Open borders treaty

- Technologies
- Resources
- Cities
- Making peace with a player (creates a ten round peace treaty in which you cannot declare war on that player)
- Declaring war on a player
- Stop trading with a player
- Adopt culture
- Convert to religion

2.2.2 Relations vs Repeated Games

The use of relations as a basis for negotiations, makes it possible to view the diplomacy in CivIV as a series of repeated games where the agents remember the behavior of players in previous games (previous negotiations), and react according to this. In this method, the behavior is reduced to a number, a *relation value*, removing (forgetting) a lot of important details in his behavior. The abstraction of a relation value in stead of an extensive history of player behavior, makes this tractable in a real time system such as CivIV. In such a scenario, it could be useful to reason about the consequences actions have on relations and to plan what actions to take based on history, or previous experiences with this player. No such reasoning is being done in the current player AI. Instead, a simple reactive behavior based on current relation values is employed, where retaliations happen when relation values get too low, and cooperation occurs when they are high.

2.2.3 Actions or Situations Affecting Relations Between Two Players

Given two players, P1 and P2, here are two lists with actions and situation that affect relations between the players in a negative and positive way, respectively.

Negative effects

- P1 and P2 having shared borders
- P1 trading with enemies of P2
- P1 having different types of culture or religion than P2
- P1 being at war with P2
- P1 being at war with friends of P2
- P1 declining trades from P2
- P1 declining to help when being asked by P2

Positive effects

- P1 accepting trades from P2
- P1 having treaties together with P2
- P1 keeping peace over time (the longer the peace, the bigger the plus) with P2
- P1 and P2 having common enemies
- P1 having the same type of culture or religion as P2
- P1 Helping when being asked for by P2
- P1 giving gifts to P2

2.2.4 Current Use of Negotiations

As previously mentioned, there are a wide range of possible items and treaties to negotiate over, and the relation value affects what is trade-able and how nice the players treat each other. Negotiations consist of one party approaching the other with a proposal, and the other party can then either accept, deny or make a counter proposal. In normal scenarios, the AI can not make counter proposals, but players are allowed to do this, both towards the AI and towards each other. Negotiation between the AIs is hence currently limited to single round negotiations where the recipient of an offer will either accept or decline an offer. It is, however, possible to suggest a deal and ask the AI what it would like to get for accepting this deal. Regular counter offers will however never be made by the original AI.

2.3 The AI Code

The following section gives a more detailed overview of how the AI code in CivIV works. The most important classes and functions will be explained, with emphasis on the parts of the code that are most relevant for our project.

2.3.1 Overview of the code

The AI code in Civilization is a large and complex system, with many dependencies, but we have separated out the most important parts and present them in figure 2.1. This gives an overview of how the different parts are connected and what they do. In section 2.3.2 the functionality of each class from figure 2.1 are described.

2.3.2 Important Classes

CvGameAI

This is responsible for calling the update functions of both the CvTeamAI and the CvPlayerAI classes. This is the function that recalculates all decisions. Most of the functionality that is called in this function is given in the other classes. Other than that it has a few functions used for decision making but does not do any decision making on its own. It inherits from CvGame which contains the game loop as well as a large part of the game logic.

CvPlayerAI

Most of the AI decisions are made in this class. It is responsible for handling diplomacy between the players The decision making for diplomacy is very simple. It also keeps track of all cities and groups of units and is responsible for calling the individual AIs for these.

CvTeamAI

The CvTeamAI class handles decision making that concerns the entire team (if you are on a team), such as war decisions. All players are, per definition, on a team. This team can either be an actual team, or just that one player. Each team has one CvTeamAI for the decisions that concern all players on the team. This part of the AI is separated from the CvPlayerAI to make it possible to just take these decisions once when you have several players on a team. If no players are on a team, each player will have their own CvTeamAI that makes these decisions.

CvCityAI

The city AI takes decisions about what to build in a city, what type of specialists to hire in the city and other city management decisions. There is one CityAI for each city. The decision making is triggered by the PlayerAI which keeps track of all its cities.

CvUnitAI

Controls one unit. A unit is a move-able property of a player, for instance a soldier or a ship. Higher level control such as where to move is not done here. The higher levels of the AI gives the units or the selection groups missions to perform.

CvSelectionGroupAI

This class keeps track of a group of units that are selected together. It provides functions for reasoning about their state that are used for decision making on unit movement and control. It is also responsible for calling the AI of each individual unit each turn.

CvDeal

This class implements all trades and diplomacy acts when they are decided on.

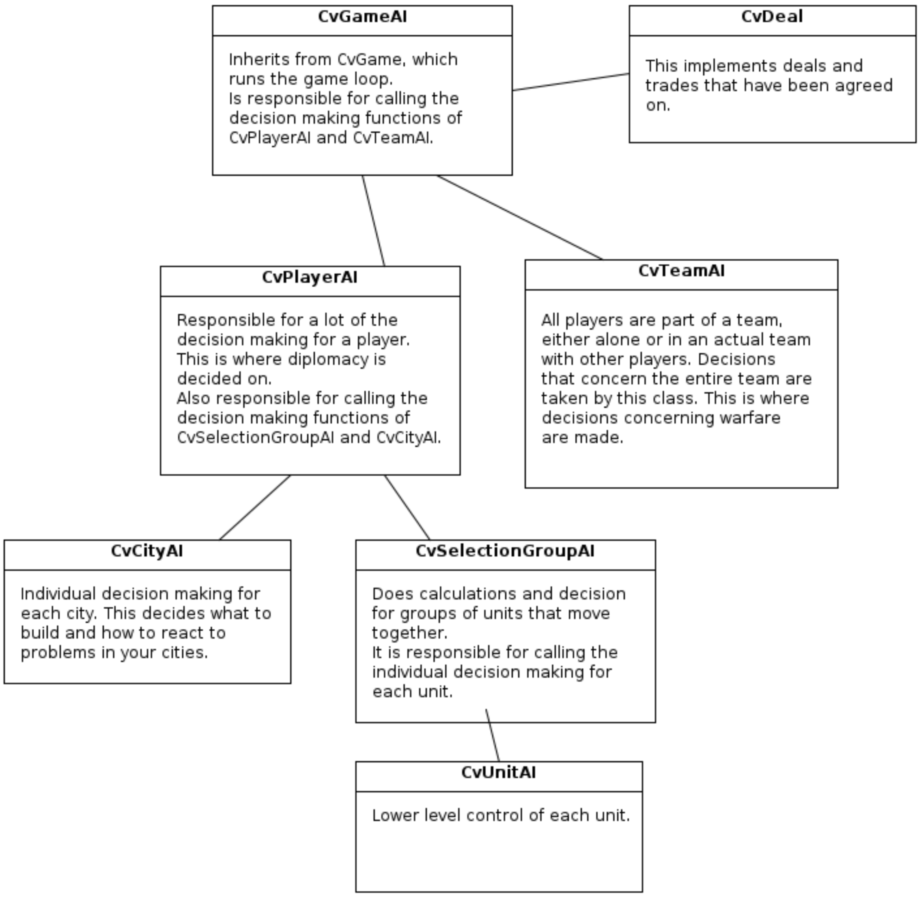

CvGameAI
Inherits from CvGame, which runs the game loop.
Is responsible for calling the decision making functions of CvPlayerAI and CvTeamAI.
CvDeal
This implements deals and trades that have been agreed on.
CvPlayerAI
Responsible for a lot of the decision making for a player.
This is where diplomacy is decided on.
Also responsible for calling the decision making functions of CvSelectionGroupAI and CvCityAI.
CvTeamAI
All players are part of a team, either alone or in an actual team with other players. Decisions that concern the entire team are taken by this class. This is where decisions concerning warfare are made.
CvCityAI
Individual decision making for each city. This decides what to build and how to react to problems in your cities.
CvSelectionGroupAI
Does calculations and decision for groups of units that move together.
It is responsible for calling the individual decision making for each unit.
CvUnitAI
Lower level control of each unit.

Figure 2.1: Overview of the current AI in Civilization IV. Only the most relevant parts are included.

Chapter 3

Background Theory

In this chapter, we introduce the theoretical background to our report such as agent systems, multi-agent systems and game theory, focusing on the parts of these fields that are relevant to our system. For instance, Pareto optimality and social welfare, which are used as a way to check if our results are good or not, Bayes' rule which we use for our Bayesian opponent modeling and Constraint Satisfaction Problem which is how we represent the preferences of the AIs.

3.1 Agent Systems

An agent is a computer system that is capable of acting autonomously given its environment in order to achieve its delegated objectives [1]. In most cases, the agent will not have complete control over the environment, only partial control, in the sense that it influences the environment through its actions. Therefore an action performed in two seemingly identical environments may have two entirely different effects and may even not accomplish the desired effect.

Agents use sensors to observe and gather information about their environments. These can be cameras, proximity sensors, motion detectors, microphones or temperature sensors, amongst others. In a simulated environment it might have full information, or it might need to use sensors for finding information in this environment as well. Agents may also use a knowledge database or learn from previous actions to achieve their goals. Agents vary enormously regarding their level of intelligence. For example a thermostat that automatically controls the temperature in a room based on the current temperature can be described as a simple agent. On the other side of the spectrum, a human being acting on the behalf of a company can also be seen as an agent.

A reactive agent is an agent that does not plan or reason about what to do based on its history. Its decision making is only dependent on its present state. This type of agent responds directly to its environment. The original AI in Civilization IV can act similar to a reactive agent.

3.2 Multi-agent Systems

A multi-agent system, or MAS, is a system where multiple intelligent agents interact within an environment. They are often used to solve problems that are normally too difficult to solve by a single agent.

Agents working in an environment may have different spheres of influence, meaning that they will have influence over different parts of the environment [2][1]. These spheres may coincide and thus leading to dependencies between agents. Agents may also have different type of relationship; coworker-coworker or coworker-boss.

Given the previous analogy where a human being could be seen as an agent, a multi-agent system could be considered as a company where many individual people, with their own area of expertise, work together, delegate, negotiate and exchange messages and subtasks.

3.3 Game Theory

Game theory studies strategical decision making under uncertainly, mostly because of incomplete information. More specifically, the ways in which strategic interaction occurs among agents with preferences over the outcomes of a game. It is mainly used in political science, economics, psychology, logic and biology. It was first studied by Neumann and Morgenstern and in 1944 they released a book on the subject [3]. It was then extensively developed on during the 1950s and has since been recognized as an important tool in many fields, such as computer science and logic. Among others, it has been used extensively in multi-agent systems.

3.3.1 Prisoner's Dilemma

A fundamental game in Game Theory is the *Prisoner's dilemma.* It is a 2×2 game in which two men are collectively charged for a crime and held in separate cells. They are not allowed to make any kind of contact with each other and are told that if only one of them confesses, the confessor gets to go free while the other will be jailed for three years. However, if both confess, then they both be jailed for two years, and if neither confesses, they will both be jailed for one year.

Given these two options: to confess (referred to as D for defecting) or not confessing (referred to as C for cooperating), and the two prisoners i and j, we can make a pay-off matrix to visualize the outcome. This can be viewed in table 3.1.

	i defects (D)	i cooperates (C)
j defects (D)	2_j / 2_i	5_j / 0_i
j cooperates (C)	0_j / 5_i	3_j / 3_i

Table 3.1: Prisoner's dilemma pay-off matrix.

Since the prisoners are not allowed to communicate or make any agreements, none of the prisoners can be sure of what the other will do, and we can conduct what the best response for a prisoner will be:

- If prisoner j chooses to cooperate, then prisoner i's best response is to defect
- If prisoner j chooses to defect, then prisoner i's best response is to defect

In other words, the best response for prisoner i is to defect on all strategies of player j. Defecting is a dominant strategy. The scenario is symmetric, meaning that both prisoners will reason the same and therefore both will choose to defect. If one prisoner assumes that the other will cooperate, then the rational response is to defect, even though both would get a pay-off of 3 if both cooperated, instead of getting a pay-off of 2. The fact that both prisoners could do better here if both cooperated is why this is referred to as a dilemma [1].

An *iterated Prisoner's dilemma* is the same as a Prisoner's dilemma, only the game is played for an infinite or finite number of times, where the prisoners get to know what the other chose on the previous round. What has changed now is that if a prisoner chooses to defect, the other prisoner can choose to punish the first prisoner by also choosing defect. Punishment is not possible in a one-shot prisoner's dilemma. If the prisoner chooses to start by cooperating and receives the sucker's pay-off, this loss of utility is either lost in the infinite number of plays or just a small percentage of the overall utility gained in a finite game. If the game is played for an infinite number of times, the rational decision changes from defecting to cooperating. Knowing what other prisoner chose on the previous round, encourages the first prisoner to choose to cooperate, and vice versa. However, if the game is played for k number of times, cooperation will be the rational decision for the all rounds except the k^{th} round where defection will be the rational decision.

Learning from the opponents past behavior like this is similar to what we intend to do in our system with the opponent modeling. Looking on what the opponent did on previous rounds, we can model what the preferences for the opponent might be and how to act accordingly.

3.3.2 Solution Concepts

In game theory, a solution concept is a formal rule used for predicting how a game will be played and describe which strategies will be adopted by players and, therefore, the result of the game. The most common types of solution concepts are equilibrium.

One of the most known solution concept for many problems in game theory is the Nash equilibrium [4]. This is a set of actions (often denoted as an action profile a^*) for a game which has the property that if all players other than player i choose an action from this profile. Player i will do as least as good when choosing action a_i as when choosing any other action. This means that in a two player game, if one player chooses an action that is in a Nash equilibrium, the other player can do no better than to choose the corresponding action in the equilibrium. In repeated games, this type of solution models a steady state where no player wants to deviate from this state.

To refer back to the *Prisoner's dilemma*, the outcome of (D,D) is a Nash equilibria since if we assume that i will play D, j can do no better than to play D, and if j will play D, i can do no better than to play D. This is also the only Nash equilibria in the game.

3.3.3 Negotiations

A negotiation is a means to reach an agreement in the presence of conflicting goals and preferences [1]. In multi-agent systems, negotiations can be used for dividing resources (goods or services) between agents, or allocation of tasks to agents. Negotiations allow agents to be autonomous in that they can negotiate with other agents freely based on their own preferences and goals without any external intervention.

There are several strategies for negotiation, two of them being *Boulware* and *Conceder*. The two strategies can be seen in figure 3.1 shown in a seller-buyer environment . In (a) we see the sellers point of view. Using a Boulware strategy, the seller decreases the price it is willing to accept at a very small rate in the beginning, and then more and more as the deadline approaches. Conversely, in the conceder strategy, the seller starts by decreasing the price rapidly in the beginning and then less and less as the deadline approaches. The buyers point of view can be seen in (b). In Boulware, the buyer slowly increases the price it is willing to pay in the beginning, but does concedes faster as the deadline approaches. In conceder, the buyer starts by increasing the price fast, and then concedes slower as the deadline approaches.

The Monotonic Concession Protocol

The *monotonic concession protocol* is a negotiation protocol that proceeds in a series of rounds [5, pp.40-41]. On the first round, both agent simultaneously propose a deal from the negotiation set. If either

1. $u_1(\delta_2) >= u_1(\delta_1)$
2. $u_2(\delta_1) >= u_2(\delta_2)$

In other words, if one agent finds that the deal proposed by the other agent is at least as good, or better, than his own deal. If both agents' offers are viable

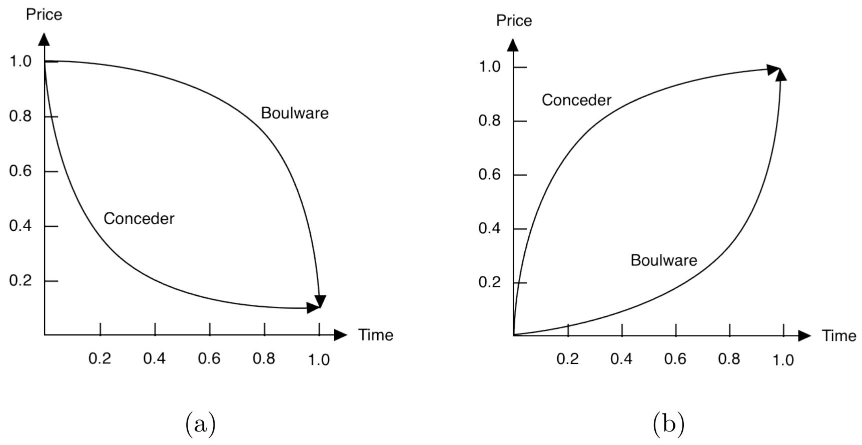


Figure 3.1: Boulware and conceder negotiation strategies for a seller (a) and a buyer (b).

proposals, then one is chosen by random. If not, the proposal that exceeds or matches the other's proposal is chosen. However, if no agreement is reached, the negotiation proceeds to the next round. If t is the current round of negotiation, then no agent is allowed to make a proposal that is less preferred by the other agent in round $t - 1$. If neither of the agents makes a concession in a round $t > 0$, then the negotiation is terminated with a conflict deal.

Using monotonic concession protocol, negotiation is guaranteed to end, either with or without an agreement, after a finite number of rounds. It does, however, not guarantee that an agreement will be reached quickly. The number of rounds are exponential to the number of tasks to be allocated [1].

The Zeuthen Strategy

The *Zeuthen strategy* is a protocol that is measures an agent's willingness to risk conflict. In other words, if there is little difference between its current proposal and the conflict deal, the agent will be more willing to risk conflict. Conversely, if the difference is high, the agent has more to lose from a conflict deal and is therefore less willing to risk conflict, leading it to concede. The Zeuthen strategy can be used as a protocol of how agents should behave when using the *monotonic concession protocol.*

The protocol states that the first proposal of any agent should be the agent's most preferred deal. The agent's willingness to risk conflict thereafter measured the following way [5, pp.43]

$$risk_i^t = \frac{\text{utility } i \text{ loses by conceding and accepting } j\text{'s offer}}{\text{utility } i \text{ loses by not conceding and causing conflict}} \tag{3.1}$$

where i and j are the two agents negotiating and t is the current round of negotiation. The numerator in equation 3.1 is the difference between utility of i and j's current proposals in regards to agent i, while the denominator is the utility of agent i's current proposal. While an agreement is unmet, the risk value will be somewhere between 0 and 1. A higher value means that agent i has less to lose from a conflict deal and is more willing to risk conflict, while a low value results in that agent i has more to lose from a conflict deal and be less willing to risk conflict. This can be shown more formally as equation 3.2.

$$risk_i^t = \begin{cases} 1 & \text{if } u_i(\delta_i^t) = 0 \\ \frac{u_i(\delta_i^t) - u_i(\delta_j^t)}{u_i(\delta_i^t)} & \text{otherwise} \end{cases} \tag{3.2}$$

The reason for the risk being 1 if $u_i(\delta_i^t) = 0$ is because in this case, the utility to i of the current proposal is the same as from the conflict deal and the agent is therefore completely willing to risk conflict.

The Zeuthen strategy proposes that the agent to concede on round t is the one with the smaller value of risk. If both agents have the same value of risk, then they might flip a coin to choose who gets to concede, or one might chose to not to concede and benefit from the other. However, if both does this, then a conflict will arise, and no deal is made. The next question is then how much should the agent concede. The answer to this is *just enough* to change the balance of risk. If the agent concedes too little, the next round the balance of risk will indicate that it still has most to lose from conflict. However, if it concedes too much, then it wastes some of its utility. Therefore, the agent should make the smallest concession necessary to change the balance of risk.

3.3.4 Pareto Optimal

An often desirable property of a negotiation outcome is Pareto optimality. An outcome is Pareto optimal if there exists no other outcome that could improve one player's utility without making somebody else worse off [1]. Conversely, an outcome that is not Pareto optimal is said to be inefficient in that it wasted somebody's utility; there would then exist another outcome that makes somebody better off without anybody else having objected to it. This makes Pareto optimality an important property for negotiation outcomes, as being Pareto optimal implies that no utility is being wasted, and the negotiation outcome is therefore efficient. Pareto optimality is not enough to guarantee a good solution, however, as many unfair and undesirable outcomes can be Pareto optimal. Looking back at the *Prisoner's dilemma*, the only outcome that is *not* Pareto optimal is (D,D) since both prisoners can do better by playing (C,C), and both (C,D) or (D,C) are Pareto optimal since no prisoner can do better without the other doing worse. In this example, it can be argued that the outcomes (C,D) and (D,C) are undesirable, as they give very bad results for one of the players, and give a lower total utility compared to (C,C), yet they are Pareto optimal.

We are interested in guaranteeing Pareto optimality in our negotiation system as this would prevent our system from wasting utility. Guaranteeing Pareto optimality would ensure that none of the players involved could have done better in without the other objecting. However, in a multi-issue negotiation domain, proving that a solution is Pareto optimal can be very computationally complex and is therefore often not feasible.

3.3.5 Social Welfare

Social welfare is another important property of an outcome. It is the total sum of utility in a system of agents. It gives the utility that was achieved by the entire system combined. Social welfare can be written more formally as:

$$sw(\omega) = \sum_{i \in Ag} u_i(\omega)$$

where $sw(\omega)$ denotes the sum of the utilities of each agent for outcome ω.

An outcome that maximizes social welfare should be considered good in many situation where the total utility of the system is important. In multi-agent systems with cooperating agents, one often cares more about the total utility given by the system and not the individual utility of each agent, and in such scenarios social welfare is an important property of the results. Again using the *Prisoner's dilemma* as an example, we can see that the outcome that maximizes social welfare is (C,C).

In our system, we want to ensure that all negotiation outcomes maximize social welfare. It should however, not be a priority of an agent to maximize social welfare, as Civilization IV is a competitive game, where the agents are trying to maximize their own utility regardless of their opponent's results. In order to create successful deals however, suggesting social welfare maximizing offers increases the chances of the offer being accepted. Finding such social welfare maximizing offers can be difficult however, especially in situations with limited information about the opponent's preferences. Without this information, it is impossible to know whether an outcome is actually maximizing social welfare or not.

3.4 Constraint Satisfaction Problems

Constraint satisfaction problems, or CSPs, are mathematical problems in which a set of constraints must be satisfied in order to solve the problem and is how we intent to represent the preferences of the agents in our system. In negotiations, this can be used to represent the preferences of an agent. An example of a constraint could be a limit on the total value one is willing to pay during a negotiation. For more information about constraint satisfaction problems, see [6].

3.5 Machine Learning

Machine learning refers to systems that improve their performance at a given task by learning from experience, or data [7], and is relevant because we use machine learning to learn the opponents preferences. There are many types to machine learning using different approaches. Artificial neural networks, or ANN, is an approach that is inspired by biological neural networks. Genetic programming is another approach that uses evolutionary algorithms inspired by biological evolution. Bayesian networks represents probabilistic relationships between hypotheses and their conditional independences.

3.5.1 Bayesian Learning

Bayesian learning is a form of statistical machine learning that calculates the probability of hypotheses, given data, and makes prediction on that basis. Predictions are made by using all the hypotheses, weighted by their probabilities, rather than just using a single ”best” hypothesis [6]. The probability of each hypothesis is obtained by Bayes’ rule, which is explained in section 3.6.

$$\Pr(h_i|d) = \Pr(d|h_i)\Pr(h_i)$$

where d is observed data in a dataset D, for each hypothesis $i = 1, 2, ..., k$.

3.6 Bayes’ Rule

Bayes’ rule is a mathematical equation that lets us update our beliefs about a hypothesis A in the light of new evidence B, written as $\Pr(A|B)$ [8]. While $\Pr(A|B)$ might not be easily computed, we can find the probability by multiplying our previous belief $\Pr(A)$ with the likelihood that B will occur if A is true, $\Pr(B|A)$, which might be easier to compute. This gives us the equation

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)} \tag{3.3}$$

The denominator $\Pr(B)$ is a normalization constant that can be computed by

$$\Pr(B) = \sum_{i=1}^{k} \Pr(A_i)\Pr(B|A_i)$$

giving us the equation

$$\Pr(A_r|B) = \frac{\Pr(B|A_r)\Pr(A_r)}{\sum_{i=1}^{k}\Pr(A_i)\Pr(B|A_i)} \quad \text{for } r = 1, 2, ..., k. \tag{3.4}$$

Chapter 4

Related Work

Our research focuses on efficient multi-issue negotiations in a complex domain with limited information about the opponents and their preferences or strategies. This is a difficult subject that has been studied extensively from many different angles before. Important subjects for this research are negotiation strategies and protocols, efficiency properties such as Pareto optimality and how to achieve such properties, decision making with limited information, learning based on experience, modeling of learned information and how to combine these fields to make an efficient solution that can be used in a soft real-time system.

Negotiation has been a topic in the field of artificial intelligence since its earliest years. In 1979, Smith introduced a simple form of negotiation among agents with the Contract Net [9]. It allowed for one agent to announce the availability of a task and award them to other bidding agents. Malone et al. took this a bit further and improved it with a more sophisticated economic model [10].

According to Beer et al. [11] the three main topics being studied in the multi agent negotiation subject area are the negotiation protocols, the negotiation objects and the agents' reasoning models. Negotiation protocols are the rules for how agents communicate during negotiations, defining how, when and what each agent may communicate to another agent. Closely related to this is the aspect of communication languages, such as KQML and FIPA ACL. Many systems such as the ones discussed in [12] and [13] define their own language specifically tailored to their problem, while others such as [14] use general languages with specific ontologies. In the domain of complex strategic games with negotiations, it is important to use a communication language that allows for expressing different types of arrangements and arguments. [15] claims that in the domain of the Diplomacy game, which is a board game with many similarities to Civilization, there is a need for a very complex agent communication language because it needs to be able to express threats, persuasion and complex agreements. An AI system for this game made by Kraus and Lehmann [12] defined its own agent communication language based on observation of real players playing the game, making sure that all kinds of deals and argumentation the players made was possible in the AI system. An important property of this language was that it was also possible (and not too difficult) for real players to read and write messages in this language. In order to do this, they used a low level ACL used by the agents and a higher level language more similar to natural language and translated between these as needed.

The second main topic, negotiation objects, describes what objects are under negotiation. Important issues in this field include whether objects are divisible or not, and how many objects may be under negotiation. Part of the negotiation may also be finding what objects should be negotiated about. Systems where more than one object is under negotiation are called multi-issue negotiation systems. According to [16] there are two types of bargaining frameworks for such systems, simultaneous negotiation or issue-by-issue negotiation. In simultaneous negotiation, all objects are considered at the same time, while in issue-by-issue negotiation, an ordering for the negotiation objects is decided first and then each item is negotiated for sequentially. For simultaneous negotiation, either the package deal procedure (PDP) or the simultaneous procedure (SP) [17] is often used. In PDP, all negotiation objects are negotiated together. For example if you wanted to negotiate for the division of two cakes using PDP, you would decide what partition you want from each cake and suggest both partitions in one offer. This can be computationally quite complex when there are many negotiation objects and many different preferences to consider. In SP each object is negotiated independently, similarly to issue-by-issue negotiation, but in parallel.

One difficult factor in such multi-issue negotiations is the calculation of utility of a deal, as the actual utility may often be non-linear. What this means is that the utility is not just the sum of the utilities of each item in the deal, but is affected by what item combinations are in the deal. An example of this would be that a deal that included a desktop computer and a monitor, would have a higher utility than the sum of two deals, one with just a desktop computer and one with just a monitor. In our system utilities are complex and non-linear, as they are based on what plans the player currently has, and combinations of items may often be more worth than the sum of their parts. This makes it difficult to find a Pareto optimal deal, especially in a setting of incomplete information about the other player's preferences. In [17] it is shown that finding Nash equilibria for PDP based negotiations is computationally hard, making it difficult to produce Pareto optimal outcomes, but also that there are solutions for finding approximations to these equilibria in polynomial time by approximating linear utilities. In the same work, it is also shown that in some cases SP is better at producing Pareto optimal outcomes and at increasing social welfare than PDP.

Multi-issue negotiations are often used for situations where one wants to decide how to split a set of resources. Many articles in the multi-issue negotiation literature, such as [18] and the already mentioned [16] focus on such negotiations where a set of objects is divided between the negotiators. These objects can be either divisible such as in [16], and example of this is a cake, which can be divided between the agents, or non-divisible such as [18]. Our system on the other hand will focus on a trading situation where each player has a set of indivisible objects they own that they may offer in a deal in return for objects the other player owns. In such a scenario, several objects may be subject to negotiation, and the objects included in a deal may change during the negotiation. In most multi-issue negotiations, such as the already mentioned [18], the items that are up for negotiation are not changeable, and the negotiation focuses on dividing a set of objects between the players, rather than making a trade. An example of a system similar more similar to ours is [13], where there is a seller agent with a set of items and a buyer agent which decides what kind of items it wants. This does, however, differ from our trading scenario in that here the buyer agent just has to choose the most favorable set of items it could get from the seller, as opposed to the situation where the buyer also needs to find what items it should give the seller in return, and what trade-offs need to be made in order to make the deal happen.

The third main topic of negotiation research, the reasoning models, describes how the agents participating in the negotiation decide what actions to take. This is closely related to the negotiation protocol and the negotiation objects that need to be reasoned about. Reasoning for negotiation is often based on negotiation strategies defining how to behave in different scenarios. There are many well-known types of negotiation strategies such as the Zeuthen strategy

[5] and the Boulware [19] and conceder [20] strategies. These define when the agent should concede or make a trade off, and/or how much it should concede at any given time. In many cases, such as [21] one even mixes several such strategies with a linear weighting in order to make better strategies.

Another approach to finding a good strategy is to find the Nash equilibrium strategies for a negotiation. If you find a strategy pair that forms a Nash equilibrium, no player would benefit from deviating from this strategy. Finding such equilibria in multi-issue negotiation, however, is an NP-hard problem [18]. Approximations may be made, though, and [18] shows how to calculate an approximate equilibrium in $O(nm/\epsilon^2)$ where ϵ is the error of the calculation. This gives an algorithm that can be used to find a strategy that will form an approximate Nash equilibrium with itself for a specific negotiation scenario. The strategy that is found will not necessarily be good for negotiations in general, but will be tailored for this specific scenario.

Sometimes, however, especially in situations with incomplete information or non-linear utilities such as our own system, it can be very difficult to know what an offer is worth for the other player, and alternative strategies that can cope with this uncertainty are needed. According to [22], lack of information about the other players preferences may lead to non-monotonic offers when using normal negotiation strategies or linearly mixed strategies such as in [21]. This can lead to problems such as delayed negotiations, failing to reach agreements in time, non optimal outcomes and high sensitivity to changes in how the agents weight the different strategies if they are using linearly mixed strategies.

It has been shown by [23] that efficient negotiation requires knowledge about both the negotiation domain and the opponent's preferences. In real world negotiation scenarios, however, the participants are often not willing to share their information or preferences with each other. This unwillingness can come from the competitive environment of negotiations, where your opponent may use such information to increase his own gains. In many situations, it is also unpractical for the negotiators to share their information, especially if they do not have a formal specification of these preferences. An example of this is our system, where the negotiators are either agents or real players. The agents have a set of preferences they may disclose to their opponents, but the players do not. This leads to negotiations with limited information about the opponents and their preferences. There have been many different solutions proposed to this problem. Some systems such as [24] use a mediator that gets more information and suggests deals to the negotiators. Using a mediator, they try to avoid the negotiator's unwillingness to disclose information to each other by including a third party that handles the information. This is however, not always possible, as the negotiators may not be willing to give information to anyone, including a third party. Mediators do not help in domains like ours either, as it still requires the players to disclose information that they do not necessarily have, or want to disclose. Another possible solution to the problem of limited information, is opponent modeling. Solutions such as [25] and [26] use the bidding history of their opponents to infer information about their opponent's preferences, and use this information to make better offers.

While opponent modeling does improve the outcome of a negotiation, the role of time should be taken into consideration when dealing with real-time negotiation because of the time/exploration trade-off. Meaning, a computationally complex model may produce better predictions, but may result in less bids being explored in real-time because of the time constraints.

Baarslag et al. did a comparison of a selection of state-of-the-art online opponent modeling [27]. They evaluated two different types of models; frequency models and Bayesian models, testing several different implementations of each modeling technique. What they found out is that the more competitive an agent is, the more beneficial an opponent model is. Also, the bigger the size or distribution of the bid is, the higher the gain when using a model. They also found that even thought he time/exploration trade-off is important, the best performing models did not suffer from it and most of them resulted in a significant improvement compared to not using a model. Interestingly, the frequency model outperformed the Bayesian, not only because they are faster, but also because the effect remains in a round-based setting, which may suggest that frequency models combine the best of both worlds. Despite this, frequency models get little attention in literature compared to Bayesian.

One of the frequency modeling techniques tested in [27] is described in [28]. The technique evaluates the weights and utility values of each negotiation issue by counting how often they are present in the offers suggested. This is a very simple calculation, which means it will be good for real time negotiations. It does not need any a priori information, as Bayesian learning does, but it does assume that opponents restrict their bid to a utility range, and that they prefer to explore different solutions rather than offering the same bid over and over. This approach is found to be prone to underestimating the weights it is trying to model; However, since the relative values of the weights, not the actual ones, are important for the results, it gets as good, or better results than comparable solutions such as Bayesian learning [27]. Due to these good results, we will implement a version of frequency modeling for our system and compare it to a more complex modeling algorithm. The simplicity of the calculation is beneficial for a real-time system such as Civilization, but might also lead to subtleties in the model being lost.

Another example of opponent modeling in multi-issue negotiations is given in [25]. This system makes a partial ordering of all trade items and estimates this ordering using a heuristic based on the opponent's bidding history. Attributes about which reliable guesses can be made based on previous bids, and attributes about which no information is available are separated. When no information is available, all items are given the same weights. [25] found that the opponent model can be sufficiently estimated during the first three rounds of negotiation, which makes this a fast solution. Estimating the opponent's preferences as early as possible is important not only to make the negotiations faster, but also to avoid skipping possible solutions that could have been found early if more information were available.

The opponent's preferences over negotiation outcomes is not the only information that may be necessary to model during the negotiation. In some domains such as the one described in [29], the opponent's strategy, deadline for the negotiation and attitude towards time are unknown and relevant to what your optimal strategy is. The domain described here is a single-issue negotiation domain, where the preferences over the single issue are known. This makes it unnecessary to model the opponent's preferences, but as the strategy, deadline and attitude towards time is relevant to your optimal strategy, this information needs to be modeled in order to negotiate efficiently. [29] uses a probabilistic distribution over the possible values of the deadlines and the attitude towards time. This information is utilized to choose your own optimal bidding strategy, without knowing what the opponent's strategy is.

There are many different approaches to opponent modeling. One of the most popular ones is Bayesian learning, which can estimate the probability of a set of hypotheses based on information such as the opponent's bidding history. In this case a hypothesis is a belief about the opponent's preferences, such as what weight the opponent puts on a specific issue. Using Bayesian learning you could make a set of hypotheses about what weights the opponent has, and estimate their probability, choosing the most likely weight distribution as your opponent model. There are many examples such as [30] and [31] that use this method, but it has one major flaw, which is that it requires a priori knowledge about the probability distribution of the opponent's weights.

In the method used by [30], a series of assumptions about the rationality of the opposing agent are made. These assumptions allow them to minimize the hypotheses space, making it possible to evaluate a set of hypotheses for the preference weights on issues as well as the form of the evaluation function. Bayes' rule is then used to evaluate and update the probabilities of the hypotheses, based on the estimation of what utility the opponent should be asking for now. In order to do this, they need an initial probability distribution for the hypotheses. This can be given by a priori knowledge, or by using a uniform distribution.

A very different method for learning was used by [26]. They used a hybrid soft-computing approach, where possible opponent models are represented as a combination of fuzzy evaluation functions for all the negotiation issues, and a genetic algorithm is used to select the best model. This method avoids the problem of needing a priori knowledge about probability distributions. A genetic algorithm will usually be very slow and unreliable, but by using the fuzzy representation of possible evaluation functions, the search space is reduced significantly, making it possible to evaluate it in its entirety. [26] show that this approach is capable of effective, on-line opponent modeling.

Many methods for opponent modeling, including the ones used in [30] and [26] make the assumption that the opponent is a conceding agent, and use a specific concession rate to guess the utility of the opponent's bid. Using this information allows them to assess their opponent's preferences, but this can be problematic, as in many real life negotiations, the opponent's concession rate or strategy is unknown. One approach that tries to avoid this problem is [32], which only looks at the difference between the last two bids, guessing on the importance of the changes using Kernel Density Estimation. This approach also has the advantage that it requires no a priori knowledge about probability distributions. It is also shown to be a low complexity solution, making it feasible for real time systems. The method is shown to be very stable in the face of many different strategies such as the Boulware and the conceder strategies, but it still assumes that the opponent is making rational concessions, which may not always be the case, especially when negotiating with humans.

Negotiating with humans is a difficult field, because it usually involves little information about your opponent's preferences or strategies, and more importantly, humans may not necessarily be acting rationally or following one strategy throughout the negotiation. Oshrat et al. has tried to tackle this difficult domain by making a general opponent modeling system that learns from all encounters, and extrapolates knowledge for use in later encounters with different negotiators [33]. It stores the bidding histories of its opponents and categorizes them into different negotiating types, such as Boulware or conceder strategies. When negotiating with new opponents, it tries to find its opponent's type, and uses the knowledge learned from other opponents of the same type to estimate what that opponent will bid or what offers it should be willing to accept. Kernel based Density Estimation is used to estimate the probabilities of a bid being accepted or not. They found that using this generic opponent modeling yields a better result than specific opponent modeling and achieves a higher utility value than human players and other state-of-the-art automated agents in one-shot negotiations. It also offers low computational complexity which is always a good thing for real-time negotiation. This approach does, however, require a significant training set and pre-calculation before the negotiations to work efficiently.

Another difficulty that arises in multi-issue negotiations is interdependency between the issues. In many real world scenarios, the utility of a deal is not just dependent on the utility of each of the issues composing the deal, but of what combinations of issues are in the deal. As an example of this, a car with fuel is in many situations worth a lot more than the combined values of a car and fuel received separately. Such interdependencies may be multi-dimensional, making it very difficult to compute an optimal deal, even when you have full information.

An example of systems with interdependent issues is [34], which tries to use a mediated single text negotiation in a multi-issue negotiation with interdependent issues and limited knowledge about the opponent's preferences. In a mediated negotiation, there is a mediator that suggests offers that both agents can either accept or decline. Only when both agents accept an offer, the negotiation is finished with this offer as the outcome. The mediation allows them to use hill climbing and simulated annealing to obtain good solutions in the difficult domain of interdependent issues. The mediation also avoids the problem of not knowing the preferences of your opponent, as the mediator uses both player's preferences as they are slowly revealed during the annealing or hill climbing, this does not require the negotiators to disclose their information explicitly. Hill climbing agents only accept offers where they are better off than the last offer, thereby searching for local maxima. Simulated annealing on the other hand sometimes randomly accepts offers that are not better than the last offer, making them able to skip local maxima and find the optimal solution in complex domains, while running the risk of ending up with sub-par negotiation outcomes. Choosing annealing improves social welfare, as it allows you to search through local maxima. Hill climbing does not achieve this, but usually gets higher utilities against an annealing opponent, because they are less willing to concede. [34] proves that this turns into a prisoners dilemma, where annealing would increase the social welfare, but hill climbing is the dominant action. They try to solve this problem by letting the negotiators vote either strongly or weakly for or against an offer, and limiting the amount of strong votes. The solution searches towards a mutually acceptable deal that may or may not be the optimal solution, where no player needs to think about its opponent's preferences. A major problem in this kind of system is the slow pace. Using hill climbing or annealing to find near optimal deals can take hundreds of negotiation rounds, which is difficult to implement in a video game with strict real time demands, and unfeasible in a negotiation with human opponents, as they may not be willing to participate in such prolonged negotiations.

Negotiations are of course not the only field where limited information occurs, making learning or information retrieval necessary to achieve better decision making. When looking into ways to make better decisions in low information negotiations, we also need to find how this is done in other low information domains, and how learning or information retrieval can be done in these domains. This problem has been studied extensively in the field of machine learning, where agents aggregate, use, and possibly interpret, information based on experience, and this information is used to make decisions. In many cases, methods such as decision trees [35], Bayesian networks [36], or k-nearest neighbors [37] are used to make decisions based on aggregated knowledge, such as probabilities inferred from previous experience. In other domains, such as in case-based reasoning [38], the information is not aggregated to make probabilities, but old solutions are adapted and re-used in new situations. The field of machine learning is a vast one, but we will present a few cases that are relevant to our own work.

An important issue in machine learning, is how to represent the learned information. There are many possibilities for this, and they are often domain dependent. When modeling preferences for the opponent in a negotiation for instance, using using an ordering of all items based on how much the opponent prefers them is a possibility. This representation was used by [25] for their opponent modeling system. In many systems, such as [30] and [26], the importance of each issue is represented as a weight. In [26], these weights are simplified to fuzzy descriptions such as low, medium and high, to simplify the learning approach. A weight can be a combination of several such descriptors, making it possible to infer a crisp value for the weight.

In many machine learning approaches such as Bayesian learning, the information we try to learn is the probability of a set of hypotheses. In the opponent modeling system described in [30] a set of hypotheses representing weights on issues in a negotiation are created, and the probabilities of these hypotheses are learned through a Bayesian classifier. The system can then use the hypothesis with the highest probability as its current opponent model.

Chapter 5

Our Solution

This chapter gives an overview of our system, what influenced the design and development, and how it works. Our negotiation system for Civilization IV uses constraints as a representation of player preferences, and solves constraint satisfaction problems in order to generate offers. It also models the preferences of the opponents as constraints and uses these when generating its own offers. The following section will explain the negotiation domain and the difficulties imposed by it, and how we intend to solve these difficulties.

5.1 Negotiation in Civilization

In order for a player to win a game of Civilization IV, it is necessary that he maintains his relations with the other players, get treaties and negotiate for resources. See section 2 for a more detailed overview of the Civilization IV negotiation domain and what can be negotiated about in this system. A more advanced negotiation system than the currently existing one in Civilization IV will make it possible for players to cooperate more extensively, whether they are computer controlled or real players, than what is possible in the original system. This will make it possible to achieve much more, such as gaining allies, treaties, resources and cities, through pure negotiation. If the AI is going to cope with more advanced negotiation possibilities however, it needs more advanced reasoning about the negotiation, more akin to the mechanisms used in multi-agent negotiations.

5.1.1 Negotiation Domain

In the Civilization IV negotiation domain there are three main challenges that need to be overcome in order to negotiate effectively:

- Multiple issues in the negotiation makes it difficult to find an optimal offer.
- Limited knowledge of the opponent's preferences makes it difficult to concede effectively.
- Real time constraints limit the possibilities for complex calculation.

The negotiation domain in Civilization IV is a multi-issue negotiation domain, which poses a significant challenge to the negotiating agents. Multi-issue negotiations is a domain into which extensive research has been done, due to it being closer to real-life negotiations than most multi-agent negotiation domains, as well as it being a more difficult problem, where achieving Pareto optimal or social welfare maximizing results are much harder to achieve. The reason for this difficulty is the exponentially large search space of possible deals that the negotiation system needs to handle. For each value set on one issue in the negotiation, all possible values could be set on each of the other issues, giving us m^n possible different offers where m is the number of possible valuations of an issue and n is the number of issues. The possible values of m is in most cases constrained to all positive integers, or even to all real values. In order to find the optimal concession at a given time, you would in theory need to search trough all these possible offers, or you might miss a potentially lucrative deal. In many cases, there are also interdependencies between issues, making some issues more valuable only if some other issue is included. This makes the search even more difficult.

Our system is also a little different from the domains described in most multi-issue negotiation research. In most bilateral multi-issue negotiation systems, there is a buyer and a seller, and a set of items that will all be included in the deal. The issue that is really being negotiated about is the price the buyer pays for each of the negotiation issues. In the Civilization IV negotiation domain, however, there are a number of issues that may or may not be included in the deal, and both players are giving and receiving items. This could be looked at as a simplification of the normal multi-issue negotiation, as each issue, has only two possible valuations, included or not included in the deal, making the m^n possible offers a less daunting number. It is however, still to large, and in addition, it makes efficient concession making in the face of limited information even harder, as you only know what items the opponent is asking for, not how he values each issue, as you would know in a normal multi-issue negotiation. Modeling your opponents in this domain is therefore much harder than in most multi-issue domains. How our system overcomes this will be described in later sections of this chapter.

The added difficulty of limited information is another important point that is addressed by many researchers. As mentioned in section 4, it was shown by [23] that efficient negotiation requires knowledge about both the negotiation domain and the opponent's preferences. The negotiating agent needs to have sufficient knowledge about the preferences that shape the behavior of the opponent. This knowledge can be used to decide what to offer the opponent, leading to more acceptable offers, and thereby increasing the chance of a successful negotiation outcome. In most real life negotiations however, the negotiators do not know what the opponent really wants, and most people are unwilling to reveal such information, as it could potentially be abused by the opponent. In many cases the negotiators do not have any formalized preferences, and asking them to formalize these preferences to effectivize the negotiation would seem impractical to the negotiators. This is the case in our system when we are negotiating with real players. Due to this, we cannot know anything about what the player wants, making efficient concession making very difficult, thereby making Pareto optimality and maximizing social welfare harder to reach. In order to overcome this problem, our system uses opponent modeling to estimate what the other player wants. This gives us information about what the player wants without needing to ask the player, albeit with less accuracy. This information can then be used to make more efficient concessions during negotiation. Our opponent modeling approach is described in 5.4.

In the Civilization domain, the time spent negotiating is irrelevant to the negotiation outcome, as it is a turn-based game and all negotiations occur during one turn in the game. Time is therefore not moving during the negotiation. This makes the negotiations slightly simpler, as the agents do not need to worry about time discounting or deadlines. In spite of this, time is still a very important factor for us, as this is a real-time game where the players expect the

AI to act instantaneously. Waiting for the opponent to decide what to do in a negotiation would degrade the user experience, and is not acceptable in such a game. Due to this, it is important to limit the computational complexity of our decision making. Many automated negotiation protocols, and many opponent modeling techniques require a lot of computation, and often hundreds or even thousands of negotiation rounds. In a user friendly video game, this can not be required, and an important point for our negotiation protocol and our opponent modeling technique is therefore to limit computational complexity, and use learning algorithms that converge fast enough to be used in negotiations that only take a few offers to reach an outcome. It is also important to limit the number of rounds a negotiation can take, preventing the game from spending too much time on negotiation. In order to do this, we have decided to limit the number of rounds allowed in a negotiation.

5.2 Solution Architecture

The negotiation system needs to deal with multiple issues, and limited knowledge while working under time constraints. In order to handle the limited information, it is necessary to reason about what the other player wants based on its behavior in the earlier stages of the negotiation. In addition, it is necessary for the rest of the AI system to be able to send its preferences to its own negotiation system whenever it is entering a negotiation in order to achieve its goals in the negotiation. To allow for this we have decided to define the AI's preferences as constraints, forming a constraint satisfaction problem (CSP) for each negotiation. The system will then try to satisfy its own constraints, and to the extent it is necessary, the opponent's constraints. It will model its opponent's constraints in order to know how to satisfy them using a simple opponent modeling technique. As it is important not to use too much time negotiating, our system has a strict limit on the number of

5.2.1 Agent Description

Our negotiation system is incorporated into the original AI of CIvilization IV, expanding its abilities to negotiate. Our negotiation system is therefore a subset of the AI in Civilization IV, and it uses functionality from the original AI to make decisions such as when to negotiate and what its preferences are. The following lists give an overview of the responsibilities of the agent, what knowledge it has and what reasoning it needs to do, focusing on the negotiating part of the agent and not the original AI.

Agent responsibilities:

- Deciding when to negotiate.
- Deciding what its preferences for the negotiation are.
- Modeling the preferences of its opponents
- Sending offers or counter offers.
- Receiving and evaluating offers and sending the appropriate response.

Agent knowledge:

- Own preferences in the form of a constraint set.
- Opponent model for all opponents, also in the form of constraint sets.
- Own previous offer in the current negotiation.
- The opponent's previous offer in the current negotiation.
- Minimum satisfaction limit.
- Current concession limit.
- What trade-able items the opponent has.

Agent reasoning:

- Uses the current AI system to reason about what its preferences should be and when to negotiate.
- Uses one of our two opponent modeling techniques to reason about the opponent's preferences.
- Evaluates offers based on constraint satisfaction and its current concession limit. Constraint satisfaction is explained in detail in section 5.3.1.

- Creates offers based on constraint satisfaction using a local search algorithm. This is also described in more detail in section 5.3.1.

Agents can only be in one negotiation at a time, which simplifies the reasoning. They do, however, keep a separate opponent model for each of their opponents at all times, updating it whenever they negotiate. This model may be empty if no negotiations have occurred yet or if the modeled constraints have been satisfied during a negotiation and are therefore no longer relevant. For more details on the constraint satisfaction and opponent modeling see sections 5.3.1 and 5.4.

Figure 5.1 shows an overview of this architecture and how it is incorporated into the AI of the game. There is a negotiator object that handles the sending and receiving of offers. In order to create offers, it gets both its own preferences and the opponent's preferences in the form of constraint sets from the constraint generator and the opponent modeler. The constraint generator evaluates the agent's current situation choosing what it wants or does not want in the negotiation. This uses functionality that already exists in the original AI to evaluate what items it wants. The constraints are then sent to the offer creator to make the offer, and this will be sent to the current opponent. When an offer is received, it is sent to the offer evaluator together with the agent's preferences and this module decides whether to accept decline or create a counter offer. The following sections will further explain the negotiation protocol, the constraint based representation of preferences and the opponent modeling used to find the opponent's constraints.

5.3 Negotiation Protocol

The negotiation protocol details how our negotiating agents will communicate with each other during negotiations. This includes what kind of messages are allowed and in what circumstances are they allowed, what kind of offers are allowed, what changes are allowed in a counter offer, etc. This section will detail our protocol. The list in section 5.3 gives an overview of all the different negotiation suggestions that can be made. This list is quite similar to the list in section 2, but has been expanded slightly to fit our solution. The list in section 5.3 gives an overview of the possible actions to take in negotiations. Our protocol is an alternating offers protocol as illustrated by figure 5.2 (see [39] for details about this type of protocol) with no discounting, but a strict

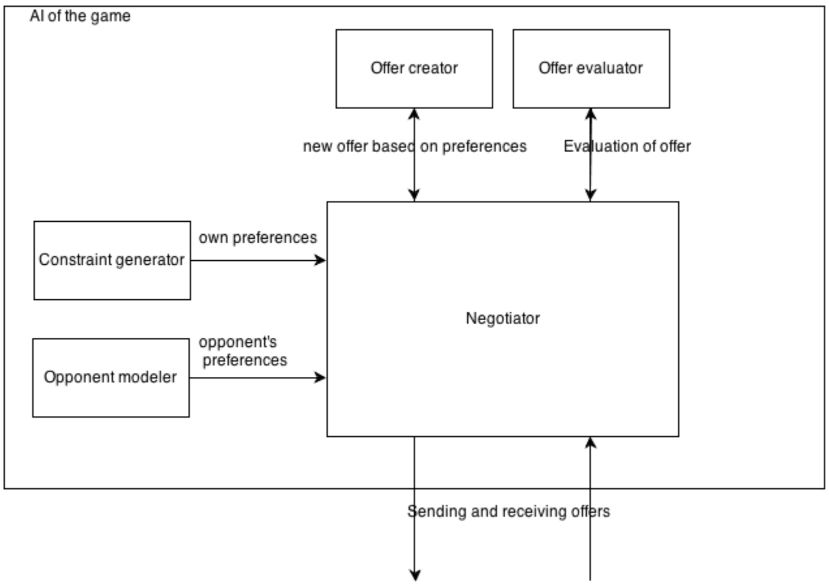


Figure 5.1: Illustration of our solution's architecture

limit on the number of counter offers allowed. This limit is necessary to avoid that negotiations take too much time. A negotiation must be explicitly started with one player, and one can only negotiate with one player at a time. It is a multi-issue negotiation, where the players are allowed to make offers and counter offers containing what items or treaties they will give and what they will receive. There are no limits to what kind of offers they are allowed to make at any time. An example of a legal offer is shown in figure 5.3.

An example of how negotiation works can be seen in figure 5.4. In this figure, P1 gets a set of constraints from its constraints generator and tells its negotiator to negotiate with P2. The negotiator creates an offer that satisfies these constraints and sends this to P2. P2's negotiator gets a set of constraints from its constraints generator, evaluates the offer, finds it unacceptable and creates a counter offer that it sends back to P1. This figure is a simple model of our negotiation system's architecture, and a more detailed explanation of the different parts of the Negotiator can be seen in section 5.2.

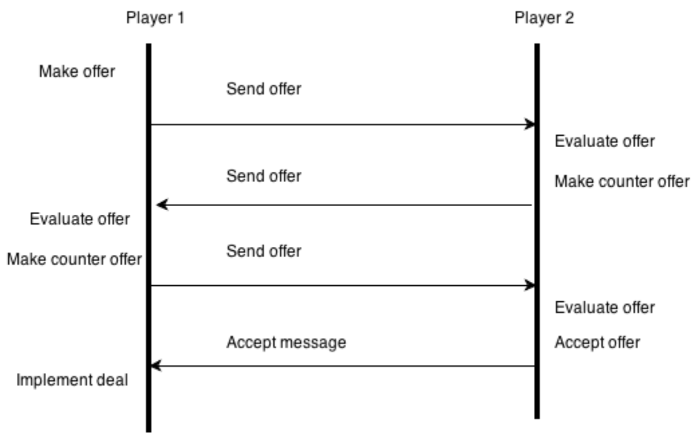
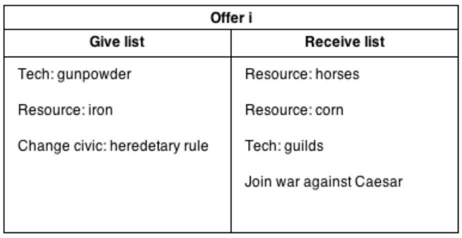


Figure 5.2: Illustration of the alternating offers protocol



Offer i	
Give list	**Receive list**
Tech: gunpowder	Resource: horses
Resource: iron	Resource: corn
Change civic: heredetary rule	Tech: guilds
	Join war against Caesar

Figure 5.3: An example of a possible offer

Possible Negotiation Issues

In Civilization IV, there is a limited set of negotiation issues that can be traded during a negotiation. This is a comprehensive list of all these issues. An offer may contain any number of these issues as long as the agent is capable of trading the issue.

- Join the other player's war
- Suggest the other player joins join our war
- Trade resource

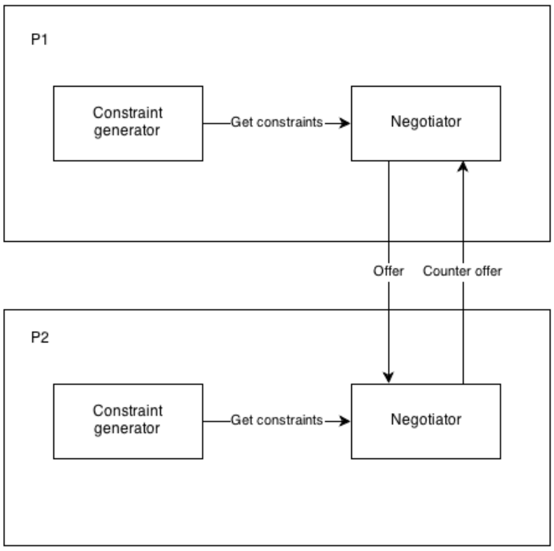


Figure 5.4: Illustration of our negotiation protocol at a high level.

- Trade gold
- Trade gold per turn
- Trade unit
- Trade city
- Trade world map
- Trade open borders treaty
- Trade technology
- Make peace treaty
- Make permanent alliance
- Make defensive pact
- Adopt civic

- Adopt religion
- Declare war on a player
- End war on a player
- End trade with a player
- Attack specific city
- End treaty

Possible Negotiation Actions

When negotiating, the agent may perform a limited set of actions. This is a list of all possible actions that our negotiating agents are capable of.

- start negotiation
- end negotiation
- suggest offer or counter offer
  - precondition: in negotiation already
- Accept
  - precondition: in negotiation already
- Decline
  - precondition: in negotiation already

5.3.1 Constraint Satisfaction for Generating and Evaluating Offers

In our negotiation system, the agents use a set of prioritized constraints to represent their preferences in a structured manner, making it easier to reason about preferences and providing a way for the AI to communicate its goals to the negotiator sub-system. This approach is inspired by the prioritized constraint satisfaction problem (PCSP) negotiation system described in [13], where a buyer/seller situation is modeled as a PCSP, and offers are created by solving the PCSP. In our system, the original Civilization IV AI will be responsible for deciding when to negotiate and what to negotiate for, and we have modified it to create a set of constraints and a prioritization of these constraints, as well as a minimum satisfaction degree τ that defines the lowest satisfaction degree that an agent is willing to accept. For the purposes of this project, the τ value has been set statically by us, but in a finished system, this should be chosen dynamically by the AI based on its situation. The prioritization of the constraints is done by weighting them by a constant $0 \leq \rho \leq 1$.The part of the AI code that creates and prioritizes the constraints is called the constraint generator and is shown in figure 5.1 from section 5.2.1. Using this approach, the offer creator from the same figure can use constraint satisfaction when searching for a suitable offer.

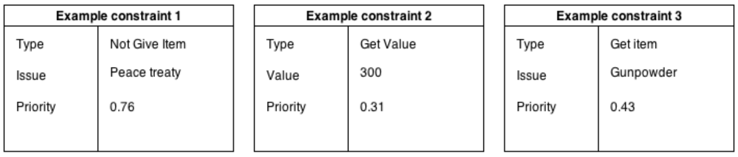
5.3.2 Constraint Satisfaction Problems

A constraint in our system is defined as a 3-tuple $(Type, Issue, \rho)$, where ρ is the priority of the constraint. We have six possible *types* of constraints, these are:

- GET_ITEM
- NOT_GET_ITEM
- GIVE_ITEM
- NOT_GIVE_ITEM
- GET_VALUE
- GIVE_VALUE

An issue can be either an *item* or a *value*, depending on the type of negotiation. Figure 5.5 shows three examples of possible constraints.

The GET_ITEM constraint means that an agent wants a specific item, while the NOT_GET_ITEM constraint means the agent does not want this item. The GIVE_ITEM and NOT_GIVE_ITEM constraints are equivalent, but for giving items rather than receiving them. In all of these constraints, the $Issue$ variable is the item in question. The GET_VALUE constraint is a lower limit for the



Example constraint 1	
Type	Not Give Item
Issue	Peace treaty
Priority	0.76

Example constraint 2	
Type	Get Value
Value	300
Priority	0.31

Example constraint 3	
Type	Get item
Issue	Gunpowder
Priority	0.43

Figure 5.5: Examples of three possible constraints

value of the items the agent receives. All items in the game have a certain value, so that they can be compared to each other. The GET_VALUE constraint is therefore a limit on the sum of this value for all the constraints the agent receives. The GIVE_VALUE constraint is similar, but is an upper limit for what the agent is willing to give rather than a lower limit for receiving. These last two constraint types have values as their $Issue$ variable. All the constraint types can be viewed as logic statements that are either true or false. We define the satisfaction of a constraint c_1 for a given offer o_1 as shown in equation 5.1 where the constraint c_1 is a statement of the type: GET_ITEM iron. This statement is then true if the agent gets the item iron in offer o_1 and false if it does not.

$$Sat(c_1, o_1) = \begin{cases} 1 & \text{if statement is true} \\ 0 & \text{if statement is false} \end{cases} \tag{5.1}$$

Due to the nature of these constraints, we need one satisfaction function for each of the different constraints. These functions are shown in equations 5.2 to 5.7. To abbreviate the equations, we use the expressions *get list* and *give list* to describe the offer. An offer contains two lists, a list of items the agent receives and a list of items the agent gives away. In all the equations, the *get lists* and *give lists* are these lists from the offer o_1. We also use the name *constraintItem* to talk about the item a constraint is about. This is the Issue variable in the constraint 3-tuple described above.

$$Sat_{getItem}(c_1, o_1) = \begin{cases} 1 & \text{if constraintItem is included in get list} \\ 0 & \text{if constraintItem is not included in get list} \end{cases} \tag{5.2}$$

$$Sat_{notGetItem}(c_1, o_1) = \begin{cases} 1 & \text{if constraintItem is not included in get list} \\ 0 & \text{if constraintItem is included in get list} \end{cases} \tag{5.3}$$

$$Sat_{giveItem}(c_1, o_1) = \begin{cases} 1 & \text{if constraintItem is included in give list} \\ 0 & \text{if constraintItem is not included in give list} \end{cases} \tag{5.4}$$

$$Sat_{notGiveItem}(c_1, o_1) = \begin{cases} 1 & \text{if constraintItem is not included in give list} \\ 0 & \text{if constraintItem is included in give list} \end{cases} \tag{5.5}$$

$$Sat_{giveValue}(c_1, o_1) = \begin{cases} 1 & \text{if sum of values of items in give list is below limit} \\ 0 & \text{if sum of values of items in give list is above limit} \end{cases} \tag{5.6}$$

$$Sat_{getValue}(c_1, o_1) = \begin{cases} 1 & \text{if sum of values of items in get list is above limit} \\ 0 & \text{if sum of values of items in get list is below limit} \end{cases} \tag{5.7}$$

When an agent enters negotiation, a set of constraints such as the ones described above is created. This is done by the agent deliberating about what items he needs and what items the opponent has. A constraint set is simply a list of constraints, describing the agent's preferences. As each of these constraints have a priority, some of them may be more important to the agent than others. The complete list of constraints can then be used to calculate the satisfaction degree of an offer. Equation 5.8 shows this calculation, where cs_1 is the set of constraints, m is the number of constraints in the set, c_i is constraint number i from the set and o_1 is the offer. The $Sat(c_i, o_1)$ function in equation 5.8 will be one of the satisfaction functions shown in the equations above depending on what kind of constraint c_i is.

$$Sat_{complete}(cs_1, o_1) = \sum_{i=1}^{m} Sat(c_i, o_1) \tag{5.8}$$

5.3.3 Concession Strategy

Our agents always start every negotiation with offers that maximize their own satisfaction. When their offers are not accepted, the agents try to make a better offer for their opponent while trying to concede as little as possible on their own satisfaction degree. To do this, the agent searches for a bid that minimizes its own concession while simultaneously increasing the utility of its opponent. The agent also uses a concession rate, which gives the maximum loss of satisfaction an agent will accept, either from the bid it creates itself or from the opponent's bid. This concession strategy is quite similar to the strategy used in the well known monotonic concession protocol [5, pp.40–41]. Just like in this protocol, our agents always try to make an offer that gives a higher utility to their opponent than what their last offer did, thereby conceding monotonically. Due to the problem of limited knowledge however, we can not guarantee monotonic concession. When we do not know the opponent's preferences we might give offers we think are better for the opponent, but this may be wrong, making the concession strategy non-monotonic.

```
Data: previousOffer, ownConstraints, opponentConstraints,
      opponentPreviousSatisfaction
Result: newOffer
if newOpponentOffer is first offer then
    foreach constraint in ownConstraints do
        satisfy constraint;
    end
else
    while Sat(opponentConstraints, newOffer) <=
    opponentPreviousSatisfaction do
        find opponent constraint that is the least negative to you;
        satisfy this constraint;
    end
end
```

Algorithm 1: Local search algorithm for creating offers.

The actual making of offers is done through a simple local search by the offer creator module from figure 5.1 using the constraints of the agent as the input. The searching algorithm is shown in algorithm 1. The *Sat*() function in the algorithm is given in equation 5.8 in the previous section. It calculates the satisfaction using *opponentConstraints*, which is the modeled constraints of the opponent. The searching algorithm goes through the agent's set of constraints, trying to satisfy them one by one to make an offer that satisfies all of its own constraints. This can be used both to modify a given offer, or to create a new one. When conceding, the algorithm goes through all the constraints of its opponent, finding the one that is the least negative to its own satisfaction and satisfies this constraint. This ensures the monotonic concession as long as the constraints of the opponent are actually correct. A weakness of this approach is the fact that if no good model exists, it will not be able to concede to the other player's demands. Due to this, we have focused on providing a good opponent modeling approach, which is described in section 5.4.

5.4 Opponent Modeling

The purpose of the opponent modeling is to extract and learn information about the opponent's preferences from his bidding history. From the multi-agent perspective, this amounts to the agent updating his beliefs about the opponent's behavior. To learn such information, we have implemented two different opponent modeling techniques, frequency modeling and Bayesian learning. The simplest of these techniques is frequency modeling. It is fast and makes few assumptions, but might miss some subtleties in the opponent's preferences. The only assumption this method makes is that the opponent bids rationally by starting with a bid that maximizes its own utility. Bayesian learning is a more complex modeling technique, and it needs more assumptions. Opponents are assumed to bid rationally in the same way as frequency modeling, but the modeler also needs to assume its opponent is using a certain concession rate in order to estimate the opponent's constraints. In both of these approaches, the resulting opponent model consists of a set of constraints forming the preferences of the opponent as described in section 5.3.1. Figure 5.6 illustrates how this will work. More detailed explanations of how the different modeling techniques work, will come in sections 5.4.2 and 5.4.3.

5.4.1 Extracting Information From Bids

Both modeling approaches can be interpreted as classifying systems, trying to map the change done in the last offer to a set of constraints that would make this change likely. Figure 5.7 shows the different categories that a change can

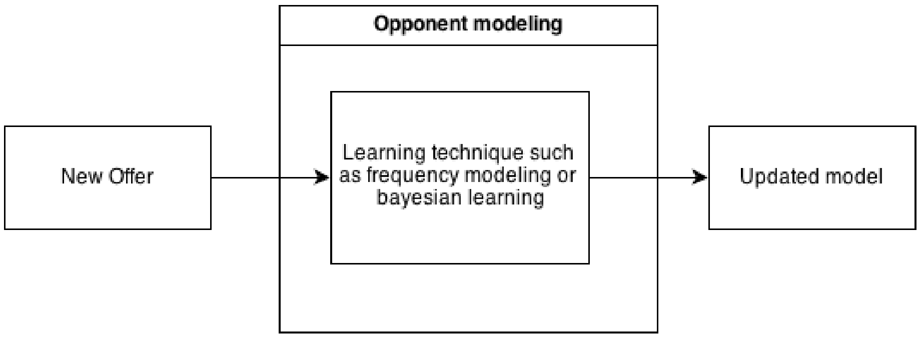

Opponent modeling
New Offer
Learning technique such as frequency modeling or bayesian learning
Updated model

Figure 5.6: Illustration of opponent modeling.

be mapped to, and figure 5.8 illustrates through an example how a change in an offer can be mapped to a constraint that can be added to the agent's opponent model.

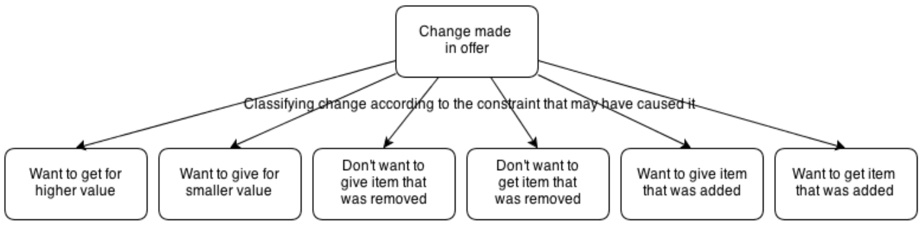

Change made in offer
Classifying change according to the constraint that may have caused it
Want to get for higher value
Want to give for smaller value
Don't want to give item that was removed
Don't want to get item that was removed
Want to give item that was added
Want to get item that was added

Figure 5.7: Illustration of the classifications that a change in an offer can be mapped to.

A learned opponent model consists of a set of constraints representing the preferences of the opponent. It is represented in the same way as the agent's own preferences as described in section 5.3.1, where each constraint has a priority, telling how much it contributes to the utility of the agent. Learning both what constraints the opponent has and what priorities these constraints have is vital for efficient negotiation. In the next two sections, we describe how the knowledge will be extracted from the negotiation interactions using two different classification approaches; Frequency modeling and Bayesian modeling.

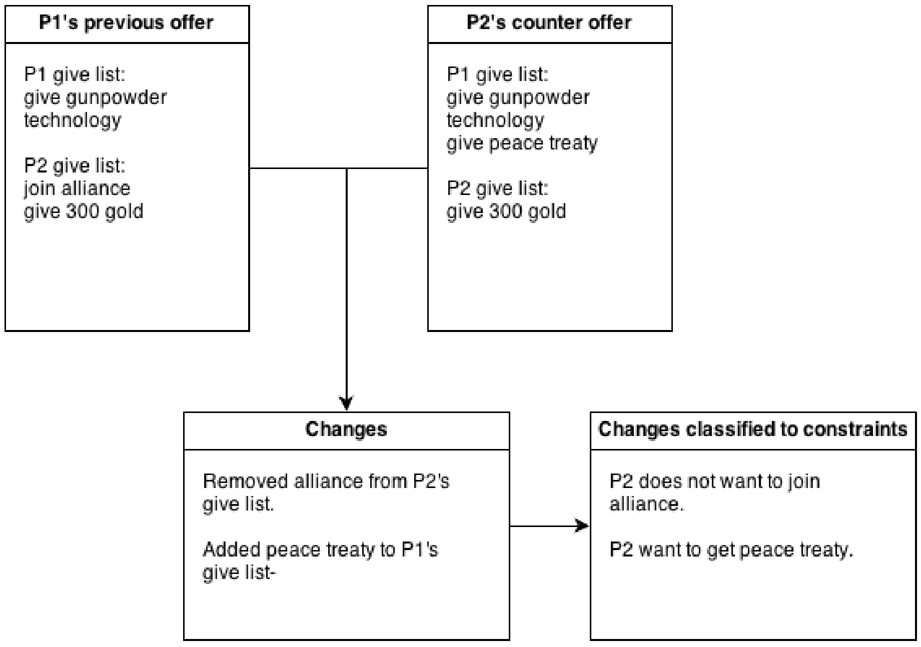

P1's previous offer
P1 give list:
give gunpowder technology
P2 give list:
join alliance
give 300 gold
P2's counter offer
P1 give list:
give gunpowder technology
give peace treaty
P2 give list:
give 300 gold
Changes
Removed alliance from P2's give list.
Added peace treaty to P1's give list-
Changes classified to constraints
P2 does not want to join alliance.
P2 want to get peace treaty.

Figure 5.8: Example of how constraints can be deduced from changes in offers.

5.4.2 Frequency Modeling

We have implemented an opponent modeling system that uses frequency modeling, similar to the system described in [28]. This is a simple approach to opponent modeling with a very low computational complexity, making it ideal for a soft real-time application such as a video game.

Our implementation of frequency modeling focuses on learning the getItemConstraints, giveItemConstraints, getValueConstraints and giveValueConstraints stating that an opponent wants to get or give an item or a certain value, ignoring constraints for not giving or not getting an item. The constraints for not getting or giving items are harder to learn and are less important when trying to improve negotiation results.Therefore, we chose to ignore them in the frequency modeling method.

Frequency modeling estimates the opponent's constraints and priorities based on the offers it has sent. Figure 5.9 illustrates how the opponent modeling works. The system starts by creating constraints for all issues that are added in the initial bid. It is assumed that the opponent is bidding rationally, and the initial bid should therefore maximize his constraints. Due to this, it is very likely that the opponent has constraints concerning these issues. This may give errors, however, as an opponent may ask for an item without having a constraint about this particular item. An example of this could be that the agent has a constraint about getting items worth at least a certain sum. In this case, he does not care about which items he gets, so long as their value is high enough. This is an example of an agent with a getValueConstraint, as described in section 5.3.1.

Issues that are repeated in the negotiations are more likely to be there because of constraints, and are more likely to have higher priorities than other issues. Our frequency modeler utilizes this by checking if items are repeated in consequent negotiation rounds. Whenever an item that was included in an offer earlier in the negotiation is still there in the latest offer, a constraint for the issue is added, or if it already exists, the priority of this issue is increased. This is similar to how the weights are increased for issues in [28]. The constraint that will be added in this case will be a getItemConstraint or giveItemConstraint as described in section 5.3.1. These constraints describe that the agent wants to get or give one specific issue such as for instance a technology or a resource. We also need to find if the opponent has constraints about the total value of the items he gets or gives away. These kind of constraints are the GET_VALUE and GIVE_VALUE constraints described in section 5.3.1. These value constraints require the sum of the value of the items received or given away to be above or below a certain value, and are not about one specific issue. Constraints on values are added or updated by checking if the total value of the items asked for is the last offer is similar to the sum from the previous offer. Whenever the sum is similar over the course of several offers, the frequency modeler will conclude that there is a constraint on this value causing the opponent to give offers with similar values. The pseudo code for this learning method is shown in algorithm 2.

The frequency modeling method has previously been compared to other learning methods including Bayesian learning methods in [27]. This article concluded that frequency modeling is more computationally efficient than the other, more complex algorithms that were compared to it while not performing any worse. Learning the constraint sets in our domain is a little harder than the domain in their system however, so this method may not be capable of learning all possible constraint configurations. We have therefore decided to compare the method with a Bayesian modeler to see if we get the same results or if Bayesian modeling is better in our domain.

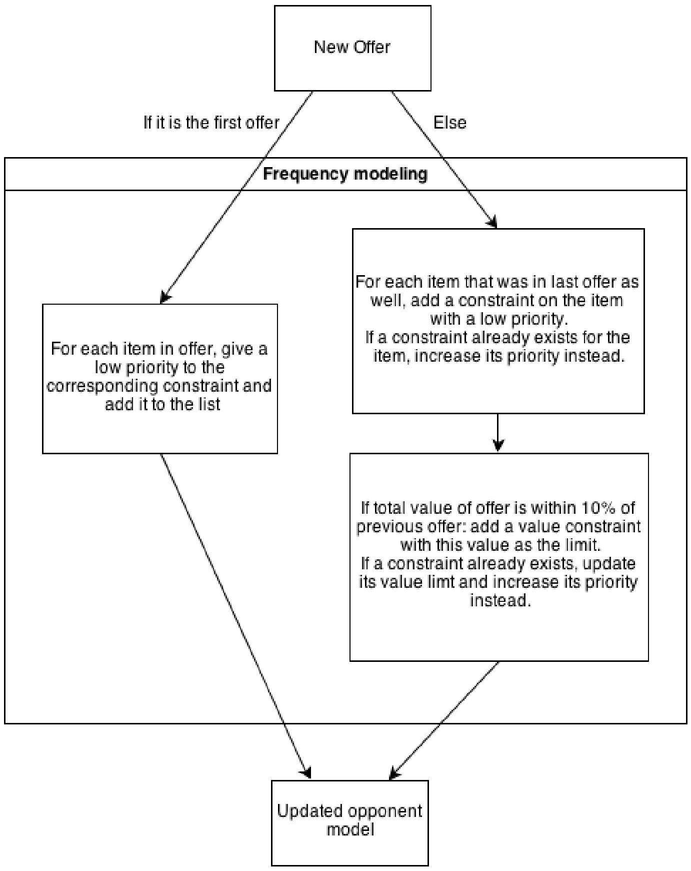

New Offer
If it is the first offer
Else
Frequency modeling
For each item in offer, give a low priority to the corresponding constraint and add it to the list
For each item that was in last offer as well, add a constraint on the item with a low priority.
If a constraint already exists for the item, increase its priority instead.
If total value of offer is within 10% of previous offer: add a value constraint with this value as the limit.
If a constraint already exists, update its value limt and increase its priority instead.
Updated opponent model

Figure 5.9: Illustration of our frequency modeling technique.

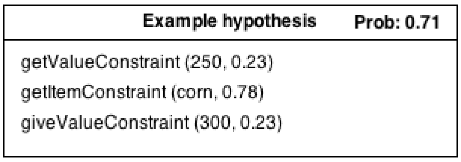
```
Data: newOpponentOffer, previousOpponentOffer, opponentModel
Result: Updated opponentModel
if newOpponentOffer is first offer then
    foreach item in newOpponentOffer do
        if constraint exists for the item in opponentModel then
            increase priority on constraint;
        else
            create constraint for item and add to opponentModel;
        end
    end
else
    foreach item in both newOpponentOffer & previousOpponentOffer do
        if constraint exists for item in opponentModel then
            increase priority on constraint;
        else
            create constraint for item and add to opponentModel;
        end
    end
    if value of newOpponentOffer within 10% of value of
    previousOpponentOffer then
        if value constraint exists in opponentModel then
            update value on constraint;
            increase priority on constraint;
        else
            create value constraint and add to opponentModel;
        end
    end
end
```

Algorithm 2: Frequency modeling pseudo code

5.4.3 Bayesian Learning

We have also implemented a version of Bayesian learning for our opponent modeling inspired by the work in [30]. The work in [30] uses Bayesian learning in a more general negotiation domain than ours, learning values and weights on issues in a multi-issue domain. We have adapted the method to learn preferences in the form of constraints to suit our system.

Using a Bayesian learner, a set of different hypotheses with corresponding probabilities are created, and the method continuously updates these probabilities to choose the most likely hypothesis as the current opponent model. A hypothesis in our case is a set of constraints representing the preferences of the opponent as described in section 5.3.1. An example hypothesis is shown in figure 5.10. The hypothesis has a list of three constraints, where each constraint has a priority and either a value or an item, depending on the type of constraint. This hypothesis can be used as the constraint set of the opponent, which represents his preferences. In order to calculate the probabilities of the hypotheses, we need to assume that the opponent is bidding rationally, that is, his first bid will maximize his utility, and he will concede gradually towards a mutually acceptable deal. We also need to assume a concession rate that the opponent is using. This assumption makes it possible to estimate the probability of a bid at a particular time, given a hypothesis ($P(bid_t|hypothesis_i)$). We will first describe a simple version of this opponent modeling to give an understanding of how Bayesian learning can be applied to video game negotiations, before we introduce our improved learning method that optimizes the learning approach to make it feasible in a large search-space.


Example hypothesis **Prob: 0.71**

getValueConstraint (250, 0.23)
getItemConstraint (corn, 0.78)
giveValueConstraint (300, 0.23)

Figure 5.10: Example of a possible hypothesis. A hypothesis is a list of constraints and captures an opponent model. Each hypothesis has a probability.

Unoptimized Bayesian Opponent Modeling

The simple version of Bayesian opponent modeling is illustrated in figure 5.11, and its pseudo-code is shown in algorithm 3. As shown in the figure, the first time an offer is received during a negotiation, all possible hypotheses are created. This means that all possible combinations of constraints are found and used as hypotheses. Each of these hypotheses will usually be created with uniform probability, but a priori knowledge could be used to set initial values to these hypotheses. Whenever an offer is received, these probabilities are updated using Bayes' rule, and when all probabilities have been updated, the most likely hypothesis is chosen as the new opponent model. This simple approach is described in more detail in the following section. It does, however, not work in a complex domain like ours, as the number of different hypotheses is exponential. Due to this, we have implemented another version of Bayesian opponent modeling that simplifies the hypothesis-space. This method is described in section 5.4.3.

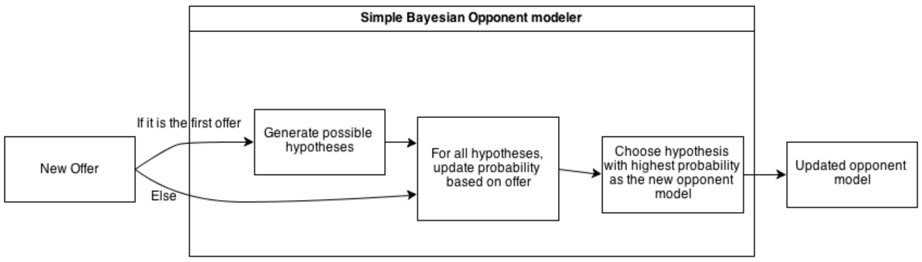

Simple Bayesian Opponent modeler
New Offer
If it is the first offer
Else
Generate possible hypotheses
For all hypotheses, update probability based on offer
Choose hypothesis with highest probability as the new opponent model
Updated opponent model

Figure 5.11: Illustration of how the unoptimized Bayesian opponent modeler works.

Detailed description

The Bayesian opponent modeler creates a set of hypotheses and calculates their probability. The hypotheses represent the preferences of our opponent. In our case, these hypotheses are sets of constraints representing a complete opponent model, while in most applications of Bayesian learning for opponent models, they are weights, values or preference functions for the negotiation issues. A large set of possible hypotheses are created by finding all possible combinations of constraints, and the probability of each of them will be calculated. In order to do this, we first need to calculate the probability of the last bid, given each of the hypotheses. That is, you assume that an hypothesis is true and calculate how likely the last bid is given this hypothesis. This probability can be calculated according to equation 5.11 where b_t is the opponent's bid at time t and h_i is hypothesis number i. This equation simply puts the utility of the bid at time t given the hypothesis into equation 5.10. The utility of the bid at time t is given by the satisfaction degree of the bid at time t with the constraints in the hypotheses used as the constraint set. We write this satisfaction degree as $Sat(h_j, b_t)$ where h_j is hypothesis number j and b_t is bid number t. This function is defined in equation 5.8 in section 5.3.1, but here we use the hypotheses h_j in place of the constraint set cs_1 and the bid b_t in place of the offer o_1. Equation 5.10 calculates the probability of the given utility level given the hypothesis. In this equation, u is the utility of the last bid, h_i is the hypothesis, $u'(t)$ is the expected utility at this time, and σ is the learning rate. The lower the value of σ is, the faster the agent learns, but this requires low error in the estimation, or it will cause us to learn false information.

In order to calculate equation 5.10, we need to estimate the expected utility $u'(t)$ given the time t . This is the utility value that the agent expects its opponent would ask for at time t, and it is dependent on the concession rate of the opponent, which is unknown. Since the agent does not know this value, and needs it in order to calculate the probabilities, the agent assumes that the opponent starts bidding rationally by asking for a bid that maximizes its own utility and then tries to concede gradually with a given concession rate. If this concession rate is known, we can calculate the $u'(t)$ value according to equation 5.9 where t is the time of the bid and *concessionRate* is the known concession rate of the opponent. The concession rate is not known, however, so the agent needs to guess what it is. Our modeler simply does this by assuming that its opponent has the same concession rate as itself, and calculates the $u'(t)$ value from this. The assumption that opponents follow a certain concession rate is used in many other opponent modeling techniques such as the ones mentioned in [30] and [26]. There have been attempts to avoid this kind of assumption, such as the system described in [32] does, but this requires learning the opponent's bidding strategy as well as his constraints, and a large set of training data is needed to learn this. What such a solution would do, is to learn the expected utility $u'(t)$ based on the training data, so that the agent does not need to make an assumption. Using an estimated value for $u'(t)$ as our agent does, is not necessarily accurate, but avoids the need for training data and strategy modeling.

$$u'(t) = 1 - (concessionRate * t) \tag{5.9}$$

$$P(u|h_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(u-u'(t))^2}{2\sigma}} \tag{5.10}$$

$$P(b_t|h_i) = P(u|h_i) \text{ where u is set to } Sat(h_i, b_t) \text{ from equation 5.8} \tag{5.11}$$

$$P(h_j|b_t) = \frac{P(h_j) * P(b_t|h_j)}{\sum_{i=1}^{m} P(h_i) * P(b_t|h_i)} \quad (5.12)$$

Once the probabilities of the latest bid, given all the different hypotheses, are calculated according to equation 5.11, this info can be used to calculate the probabilities of these hypotheses given the bid b_t. To do this we use Bayes' rule, which is shown in equation 5.12. In this equation h_j is hypothesis number j, while b_t is the bid at time t. The probabilities computed from this equation will be used to update the probability of each hypothesis, and we can then choose the most likely hypothesis as our opponent model. Algorithm 3 shows how Bayesian modeling works. We first create all possible hypotheses (all possible combinations of constraints and priorities) with a uniform probability, we then calculate the probability of the latest bid given each of the hypotheses, before we update the probability of each hypothesis. When all probabilities have been updated, we choose the most likely hypothesis as the current opponent model.

```
Data: newOffer, opponentModel
Result: Updated opponentModel
if newOffer is first offer then
    create hypothesesArray containing all possible hypotheses with a
    uniform probability distribution;
end
foreach hypothesis $h_i$ in hypothesesArray do
    foreach hypothesis $h_j$ in hypothesesArray do
        calculate P(u(newOffer) |$h_j$); (using equation 5.10)
    end
    calculate P($h_i$| newOffer) using P(u(newOffer) |$h_j$);
end
Find hypotheses h with the largest probability and choose it as the
opponent model;
```

Algorithm 3: Pseudo code for a simple version of Bayesian learning. The algorithm we have implemented is more complex, as it needs to split the hypotheses space

Optimized Bayesian Opponent Modeling

There is one big problem with the simple Bayesian learning described so far: The number of possible hypotheses is exponentially large. Each hypothesis is a combination of different constraints. As there are m different issues, and we have four different constraint types per issue (GIVE_ITEM, GET_ITEM, NOT_GIVE_ITEM and NOT_GET_ITEM), this amounts to $(4 * m)!$ different combinations of constraints, which gives an exponentially large number of hypotheses. In addition to this, there are the value constraints (GET_VALUE and GIVE_VALUE) which can have any value as their limit, and each constraint also has a, which can be any number between 0 and 1. All of this amounts to an endless number of possible constraints. Using such a large search space is unfeasible, so in order to minimize the number of possible hypotheses we have made two major changes to how hypotheses are represented in our system. These changes are described in the following section.

Simplifying the Hypothesis Space

The first technique is simply to make the priority of a constraint a fuzzy value, similar to what was done in [26]. We have three possible values for each priority, high, medium and low, and only make hypotheses with constraints that have one of these priority values. This reduces the possible amount of constraints, and hence hypotheses, from endless to three per issue and constraint type, at the expense of some accuracy in the learning of the opponent's priorities.

This first technique is not enough in itself however, as the complexity is still at least $(4 * m)!$. We have therefore employed a second technique as well which reduces the complexity all the way down to just $6 * m$ at the cost of a more complex learning algorithm. This is a drastic change in the size of the hypothesis space, and is necessary in order to make Bayesian modeling viable in our domain. This second technique achieves such a reduction in the hypothesis space by redefining the hypotheses in a similar way to what was done in [30]. Each hypothesis now consists of a single constraint instead of a combination of constraints for each issue, and we create one only a few hypotheses per issue. As we no longer need all combinations of constraints, this reduces the number of possible hypotheses greatly. The transformation from the original hypothesis-space to our minimized one is illustrated in figure 5.12. As mentioned earlier, there are four different constraint types for each issue, and the agent needs to create one hypothesis for each possible constraint type with each of the three possible priority values (high, medium or low). This would give 12 different hypotheses per issue, for a total of $12 * m$ different hypotheses. This can be further simplified however, as issues are items that are owned by one of the players, and can therefore only have either GIVE_ITEM and NOT_GIVE_ITEM types of constraints or GET_ITEM and NOT_GET_ITEM types of constraints. If the agent takes advantage of this, it only needs to create 6 different hypotheses per issue, ending up with $6 * m$ hypotheses. As a result of this technique, our learning algorithm becomes slightly more complex, as it needs to create an opponent model from a set of simpler hypotheses rather than just choosing one.

The new opponent modeling technique is illustrated in figure 5.13. It is similar to the one described previously, but creates hypotheses per issue. When all probabilities have been updated, it finds the most probable hypothesis per issue and for each of these hypotheses the constraint from the hypothesis is added to a new opponent model if its probability is above a certain threshold. The newly created opponent model consisting of the constraints from all chosen hypotheses is then used as the new opponent model, discarding the old model from the last negotiation round. Choosing only the most probable hypothesis for each issue ensures that we do not choose several hypotheses that are contradictory or superfluous, i.e. constraints to get and not get the same issue, or constraints to get the same issue, but with different priorities. Using a threshold when choosing which hypotheses to include, ensures that we do not add constraints for all issues, but only for the ones where constraints are probable.

In order to calculate the probability of each of the hypotheses, we need to change the Bayesian formula used earlier to fit our problem. We now refer to a hypothesis as $h_{k,j}$ where k is the issue number that the hypothesis belongs to, and j is the hypotheses number. There will be m different issues and n different hypotheses per issue. For each issue k and for each hypotheses j in this issue, we calculate Bayes' rule according to equation 5.16. This equation looks significantly different from the original Bayes' rule, but is essentially the same calculation. $P(h_{k,j})$ is the probability of hypotheses number j for issue k, while $P(\bar{u}_{<-k>}(b_t) + Sat(h_{k,j}, b_t)|h_{k,j})$ is the probability of the utility given by bid b_t when hypotheses $h_{k,j}$ is true. $\bar{u}_{<-k>}(b_t)$ is the expected utility of the bid for all issues except k, while $Sat(h_{k,j}, b_t)$ is the satisfaction degree given by bid b_t if hypothesis $h_{k,j}$ is true. The $Sat(h_{k,j}, b_t)$ function is given by equation 5.8 in section 5.3.1, using the hypothesis $h_{k,j}$ as the constraint set and b_t as the offer. Constraint satisfaction in our system is described in more detail in section 5.3.1. The expected utility is computed using equation 5.15, which is a special case of equation 5.14 where we skip issue number k. This function is simply the sum of equation 5.13 for all issues i. Equation 5.13 gives the expected utility $\bar{h_i}(b_t)$ of one issue for the bid b_t. It sums the probabilities of

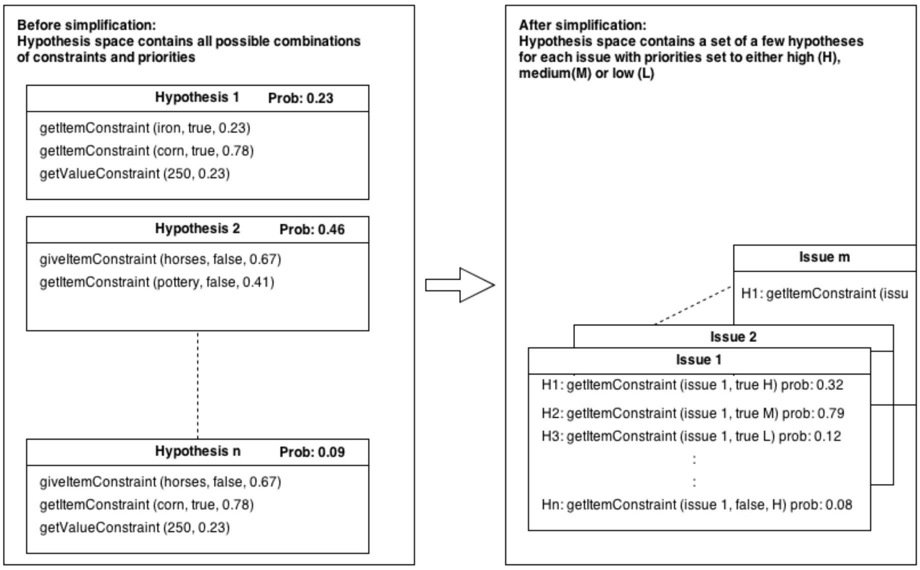

Before simplification:
Hypothesis space contains all possible combinations of constraints and priorities
Hypothesis 1 Prob: 0.23
getItemConstraint (iron, true, 0.23)
getItemConstraint (corn, true, 0.78)
getValueConstraint (250, 0.23)
Hypothesis 2 Prob: 0.46
giveItemConstraint (horses, false, 0.67)
getItemConstraint (pottery, false, 0.41)
Hypothesis n Prob: 0.09
giveItemConstraint (horses, false, 0.67)
getItemConstraint (corn, true, 0.78)
getValueConstraint (250, 0.23)
After simplification:
Hypothesis space contains a set of a few hypotheses for each issue with priorities set to either high (H), medium(M) or low (L)
Issue m
H1: getItemConstraint (issu
Issue 2
Issue 1
H1: getItemConstraint (issue 1, true H) prob: 0.32
H2: getItemConstraint (issue 1, true M) prob: 0.79
H3: getItemConstraint (issue 1, true L) prob: 0.12
Hn: getItemConstraint (issue 1, false, H) prob: 0.08

Figure 5.12: Illustration of how we simplify the hypothesis space

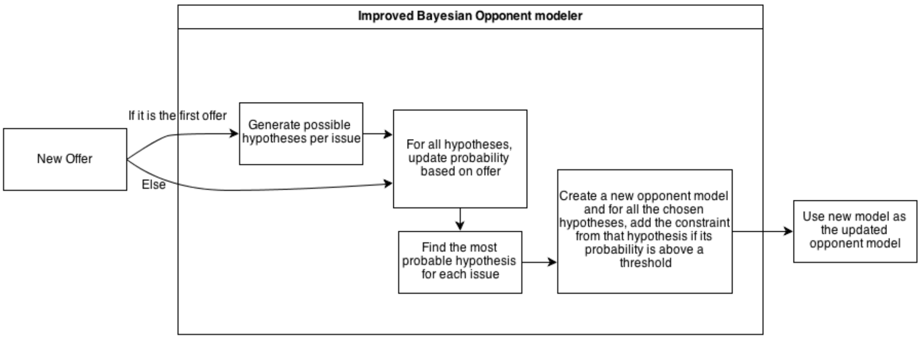

Improved Bayesian Opponent modeler
New Offer
If it is the first offer
Else
Generate possible hypotheses per issue
For all hypotheses, update probability based on offer
Find the most probable hypothesis for each issue
Create a new opponent model and for all the chosen hypotheses, add the constraint from that hypothesis if its probability is above a threshold
Use new model as the updated opponent model

Figure 5.13: Illustration of how the optimized Bayesian opponent modeling works. The hypotheses are split per issue and when all probabilities have been updated, the most likely hypotheses are combined into a new opponent model.

each hypothesis multiplied by the satisfaction degree given by the bid if that hypothesis is true. As equation 5.14 sums this function for all issues, it gives the expected utility for the bid for all issues combined. In equation 5.16, we exclude issue k, which is the one we are examining, and instead just add the satisfaction of the bid given this issue, ignoring this issue's probability. This means that we are assuming that the hypothesis we are examining is true. The utility given by $P(\bar{u}_{<-k>}(b_t) + Sat(h_{k,j}, b_t)|h_{k,j})$ in equation 5.16 is then used as the u value in equation 5.10.

$$\bar{u_i}(b_t) = \sum_{j=1}^{n} P(\bar{h_{ij}}) * Sat(\bar{h_{ij}}, b_t) \tag{5.13}$$

$$\bar{u}(b_t) = \sum_{i=1}^{m} \bar{u_i}(b_t) \tag{5.14}$$

$$\bar{u}_{<-k>}(b_t) = \sum_{i=1,2..,k-1,k+1,..m}^{m} \bar{u_i}(b_t) \tag{5.15}$$

$$P(h_{k,j}|b_t) = \frac{P(h_{k,j}) * P(\bar{u}_{<-k>}(b_t) + Sat(h_{k,j}, b_t)|h_{k,j})}{\sum_{i=1}^{n} P(h_{k,i}) * P(\bar{u}_{<-k>}(b_t) + Sat(h_{k,i}, b_t)|h_{k,i})} \tag{5.16}$$

By separating the constraints per issue, and by making priorities a fuzzy value, we have minimized the searching space significantly, making Bayesian learning a feasible method of opponent modeling for our negotiation system. The final algorithm is shown in algorithm 4.

```
Data: newOpponentOffer, opponentModel
Result: Updated opponentModel
if newOpponentOffer is first offer then
    create array with all possible hypotheses per issue with uniform
    probability;
end
foreach issue k do
    foreach hypothesis $h_{k,i}$ for this issue do
        foreach hypothesis $h_{k,j}$ for this issue do
            calculate P(u(newOpponentOffer) |$h_{k,j}$);
        end
        calculate P($h_{k,i}$| newOpponentOffer) using
        P(u(newOpponentOffer) |$h_{k,j}$);
    end
end
foreach issue k do
    Find most likely hypothesis: if hypothesis probability is above
    threshold value then
        add hypothesis to opponentModel;
    end
end
```

Algorithm 4: Pseudo code for our Bayesian learning method

5.4.4 Pareto Optimality

Pareto optimality is an important measure of how good a negotiation outcome is. One of our research questions (RQ2) is whether we can find a negotiation approach that guarantees Pareto optimality. A Pareto optimal offer is defined as an offer that can not be changed to give one agent a higher utility without the other agent losing utility. In our system, the utility is defined as the constraint satisfaction a certain offer provides. To make the reasoning simpler, we assume full knowledge of the preferences of the opponent. We also assume exhaustive searching of all possible offers, which is necessary to guarantee that the best possible deal has been found. If these assumptions are true, our algorithm will be able to find the offer that satisfies equation 5.17, where $ownSat(o_i)$ is the agent's constraint satisfaction of the offer o_i given by equation 5.1. Similarly,

$opponentSat(o_i)$ is the opponent's constraint satisfaction given by the same equation, and $opponentSat(o_{i-1})$ is the opponent's satisfaction degree from the previous offer o_{i-1}.

$$\arg\max_{o_i}(ownSat(o) \text{ where } opponentSat(o_i) > previousOpponentSat(o_{i-1})) \tag{5.17}$$

$$\arg\max_{o_i}(opponentSat(o_i)) \tag{5.18}$$

If equation 5.17 outputs more than one offer satisfying this equation, we choose the one that gives our opponent the best utility, that is we satisfy equation 5.18 with the resulting offers from equation 5.17 as the offers. Using these equations, the agent always concedes minimally, and if a better offer for the opponent exists, it will either be worse for us or it will be chosen by the equations. This means that we always suggest offers that could not be better for one player without being worse for the other. Due to this, we should be operating on the Pareto optimal border, and the suggested solution will be Pareto optimal. However, this scenario is not realistic as we do not know for sure what the opponent's constraints are, and our search can not be exhaustive as this would be NP-complete. This means that we are never sure whether the offer returned by the search actually is the best possible offer, and thereby we do not know whether we are on the Pareto optimal border. The opponent modeling attempts to counteract this problem to some degree, but as it is just a model and does not necessarily generate the constraints accurately, it does not guarantee to provide Pareto optimal results. A close to correct model should however lead to results that are more often Pareto optimal, or close to Pareto optimal, than not, even if we can not guarantee optimality.

5.4.5 Social Welfare

Another important measure for negotiation outcomes is social welfare. Research question RQ2 states that we want to find out whether our system can give results that maximize social welfare. Social welfare was defined in section 3.3.5 as the equation:

$$sw(\omega) = \sum_{i \in Ag} u_i(\omega)$$

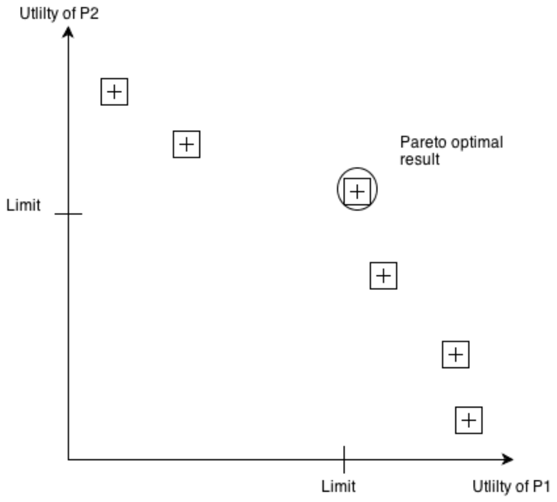


Figure 5.14: Illustration of how a negotiation can give Pareto optimal results. The Players find offers on the Pareto optimal border, and suggest these as counter offers until someone accepts an offer, which will then be Pareto optimal. This will only happen if the agents know their opponent's preferences and use exhaustive searching. The limit in the figure is the agents' lower satisfaction limit for acceptable deals as defined in section 5.3.1.

The deal that maximizes social welfare, is the deal that satisfies the equation:

$$deal = \arg\max_{alldeals}(sw(\omega)) = \arg\max_{alldeals}(\sum_{i \in Ag} u_i(\omega))$$

We have found that our system will not generally do this, and here we discuss why it does not. In our solution, the agents will accept offers that are higher than their current concession limit. This limit is set by the utility given by their own last offer minus a concession rate c. This means that whenever an offer that is above this limit is made, our system will accept immediately without considering whether there exists a better solution. As long as the concession rate is not infinitesimally small, it is possible that there exists a possible offer that gives a higher social welfare that could be found later in the negotiation if we did not accept this offer (see figure 5.15 for an example). The accepted offer would then not maximize social welfare, and ergo, our solution does not guarantee a maximization of social welfare. In order to remedy this, we would have to only accept offers if they are as good as, or better than, the last offer we made ourselves. If we had full information and exhaustive searching, this should result in maximization of social welfare, but we do not have this, and such a solution would make negotiations lengthier, which in our domain could have negative consequences. We have therefore decided not to use such a criterion for accepting offers.

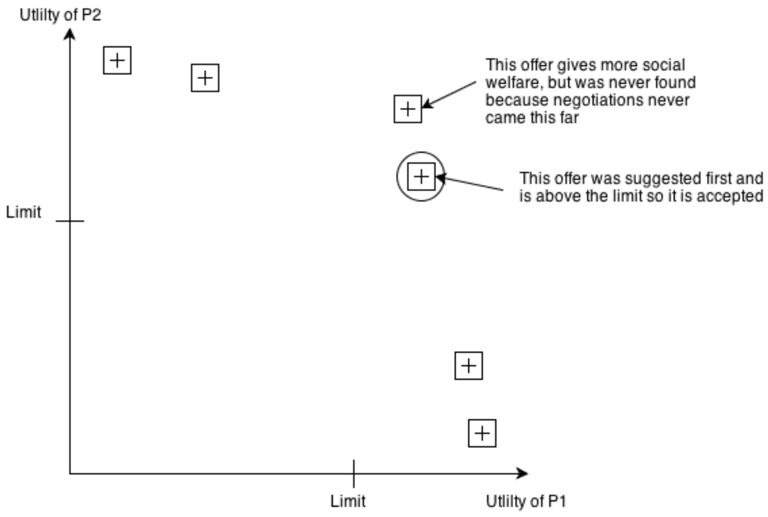

Utility of P2
This offer gives more social welfare, but was never found because negotiations never came this far
This offer was suggested first and is above the limit so it is accepted
Limit
Limit
Utility of P1

Figure 5.15: Illustration of how a negotiation might miss a result that maximizes social welfare. An offer that is good enough to accept may arrive before another solution that maximizes social welfare.

5.4.6 Negotiation Scenarios

We have chosen to outline a set of negotiation scenarios our approach needs to handle. These examples give a simple overview of how our negotiation system reasons, what information it uses to decide on offers, what kind of constraints we use to specify our preferences, and generally how the negotiation should work.

Negotiation scenario 1

This scenario has three players, P1, P2 and P3. P1 is in a war with P2, and has decided to negotiate for resources with P3 to increase its chances of winning this war. P1 knows that P3 has the technology gunpowder, as information about what items a player can trade is public knowledge in the game. This technology could significantly improve P1's chances of winning so it decides to get this item. The player creates a set of constraints c_1. for what it wants to achieve from the negotiation, prioritizing the acquisition of the gunpowder technology. It also decides on a satisfaction degree $\tau_1 = 0.8$, which is the lower limit for accepting a deal. P1 will not accept a deal that gives a satisfaction degree below this limit. The limit is set statically in our system, but could potentially be set dynamically based on the agent's situation in any further work.

$$c_1 = \begin{cases} \text{Get gunpowder tech} & \rho = 1 \\ \text{Give no unit} & \rho = 0.9 \\ \text{Give for a value less than X} & \rho = 0.8 \\ \text{Give no city} & \rho = 1 \\ \text{Other player shall not join war} & \rho = 0.6 \end{cases}$$

P1 then creates an offer o_1 that satisfies these constraints completely, and sends it to P3.

o_1 = (P1 gives music tech & 300 gold , P3 gives gunpowder tech)

P3 receives this offer and checks what constraints it has for making deals. P3 will make constraints to ensure that negotiating will not obstruct its own plans. P3 is currently at peace with both P1 and P2 and does not want a war at the time. Its constraint generator therefore creates the constraints c_2 and the minimum satisfaction degree $\tau_2 = 0.7$.

$$c_2 = \begin{cases} \text{Give no unit} & \rho = 1 \\ \text{Give no city} & \rho = 1 \\ \text{Join no war} & \rho = 0.7 \\ \text{Join no alliance} & \rho = 0.6 \\ \text{Get new technology} & \rho = 0.3 \\ \text{Value of received items} \geq \text{Value of given items} & \rho = 1 \end{cases}$$

Based on these constraints, P3 finds that o_1 breaks the last constraint because P3 values gunpowder higher than the sum of the music technology and 300 gold. The satisfaction degree is therefore too low and it tries to make a counter offer

o_2 instead. This counter offer is based on changing the original offer until it satisfies the constraints. P3 also models the constraints of P1 based on the fact that it knows P1 made an offer that completely satisfies its own constraints, it makes the model mc_1 because it sees that P1 wants gunpowder. The priority of this constraint is still unknown.

$$mc_1 = \left\{ \text{Get gunpowder tech} \qquad \rho = ? \right.$$

o_2 = (P1 gives music tech & 20 gold per turn & 300 gold , P3 gives gunpowder tech)

P1 receives this offer and calculates that the value of the items it is giving is higher than its limit X, given by its third constraint. The deal is therefore unacceptable and it makes a new counter offer that is less costly to itself. In order to make an efficient counter offer it models P3's constraints in the model mc_2. P1 now thinks that P2 wants a deal that gives it more value than o_1, so it creates a constraint on the value of the trade (a GET_VALUE constraint from section 5.3.1), where Y is the limit value of the constraint. It also has a constraint on gold per turn, as this point was added to the deal, and should therefore be assumed to be in P3's constraints. The variable Z in this constraint is an unknown limit. It could be assumed to be around 20 as P3 suggested 20 gold per turn. Based on these constraints it tries to make an offer that satisfies both constraint sets.

$$mc_2 = \begin{cases} \text{Value of the trade larger than Y} & \rho = ? \\ \text{Get gold per turn} \geq Z \quad \rho = ? \end{cases}$$

o_3 = (P1 gives banking tech & 20 gold per turn , P3 gives gunpowder tech)

P3 receives this and finds that it satisfies its constraints with a high enough satisfaction degree, so it accepts the deal and P1's plan is completed.

Negotiation scenario 2

This scenario has three players P1, P2 and P3. P1 has found that if it gets P2 to join in a war against P3, they would be able to win the war and P1 would end up as the strongest player afterwards so long as the payment for the deal is not too high. It creates a set of constraints c_1 for a deal that includes P2 joining a war against P3, as well as a minimum satisfaction degree $\tau_1 = 0.9$. P1 is not willing to join an alliance to win this war, as it does not plan to continue being friendly with P2.

$$c_1 = \begin{cases} \text{Other player join war on P3} & \rho = 1 \\ \text{Give no city} & \rho = 0.9 \\ \text{Join no alliance} & \rho = 1 \\ \text{Give for a value less than X} & \rho = 0.8 \end{cases}$$

P1 creates an offer o_1 satisfying the constraints and sends it to P2.

o_1 = (P1 gives 1 rifleman & 200 gold , P2 joins war on P3)

When P2 receives this offer, it makes a set of constraints. This player wants to form an alliance with P1 to secure its position when the war is over. It makes the constraint set c_2 and the minimum satisfaction degree $\tau_2 = 0.9$.

$$c_2 = \begin{cases} \text{Join alliance with P1} & \rho = 1 \\ \text{Value of received items} \geq \text{Value of given items} & \rho = 0.7 \\ \text{Give no technology} & \rho = 1 \\ \text{Give no city} & \rho = 1 \end{cases}$$

It finds that the second constraint is not satisfied in o_1 because the value of joining a war is very high. It therefore tries to make a counter offer o_2 based on its constraints. To do this, it first makes a model of the opponent's constraints mc_1 based on its first offer. It uses this model to try and make a counter offer that fits both its own constraints and P1's constraints. The offer contains an alliance with P1, which has a very high value for P2.

$$mc_1 = \{ \text{ P2 join war on P3} \qquad \rho = ?$$

o_2 = (P1 gives 1 rifleman & joins alliance with P2 , P2 joins war on P3)

P1 will not accept this offer as it has a constraint to make a deal with no alliance. It therefore tries to make a new deal o_3 that pays P2 more without joining the alliance. To do this it models P2's constraints. The second constraint means that P2 is probably looking for a trade worth more than X, where X is the total value of what P1 is giving in the first offer, as this was not enough to make a deal. Based on these constraints it sees that P2 is unlikely to accept a deal without an alliance, but it does not know the priority of the constraint so it will try to make a deal that pays a lot to make up for it.

$$mc_2 = \begin{cases} \text{Join alliance with P1} & \rho = ? \\ \text{Value of trade larger than X} & \rho = ? \end{cases}$$

o_3 = (P1 gives 1 rifleman & Banking tech & 500 gold , P2 joins war on P3)

P2 cannot accept a deal without an alliance so it makes a new counter offer

o_4. To do this, it updates its modeled constraints m_1 based on the new offer. Because P1 removed the alliance from the deal it adds no alliance to the constraints with an unknown priority. Because it needs an alliance to satisfy its own deal, it tries to make the deal more acceptable by adding payment for it.

$$mc_1 = \begin{cases} \text{P2 join war on P3} & \rho = ? \\ \text{Join no alliance} & \rho = ? \end{cases}$$

o_4 = (Joins alliance with P2 , P2 joins war on P3 & 300 gold)

P1, not being able to accept a deal with an alliance, updates its modeled constraints for P2, increasing the priority of the join alliance constraint and tries to make a new counter offer that is better for the other player than o_3 while not containing an alliance. It fails to find such an offer that also satisfies its own constraints to a degree higher than τ_1, and decides to decline the offer, ending the negotiation unsuccessfully.

Negotiation scenario 3

In this scenario we have two players P1 and P2. P1 wishes to improve its relations with P2 and it has decided to convince P2 to join the same religion as P1 to achieve this goal. It creates the constraints c_1 and the minimum satisfaction degree $\tau_1 = 0.7$. It does not have a lot of constraints as it is very willing to give a lot to P2 in order to increase their relation.

$$c_1 = \begin{cases} \text{Other player join our religion} & \rho = 1 \\ \text{Give no city} & \rho = 0.9 \\ \text{Give for a value less than X} & \rho = 0.8 \end{cases}$$

P1 creates an offer o_1 satisfying these constraints that it sends to P2.

o_1 = (P1 gives 5 gold per turn & horse resource , P2 joins P1's religion)

P2 receives this offer and finds a set of constraints that fit its own plans. P2 creates the constraints c_2 and the minimum satisfaction degree $\tau_2 = 0.8$. P2 is currently short of money, and makes constraints for improving its economy.

$$c_2 = \begin{cases} \text{Value of received items} \geq \text{Value of given items} \quad \rho = 1 \\ \text{Get technology Banking} \quad \rho = 0.4 \\ \text{Get gold per turn} \geq 15 \quad \rho = 0.3 \\ \text{Get gold} \geq 400 \quad \rho = 0.1 \end{cases}$$

Using the constraints, it evaluates the offer from P1 and finds that the satisfaction degree is too low as none of the last three constraints are satisfied. All of these constraints have quite low priority, so not all of them need to be satisfied in the deal, but when all of them are missing, the satisfaction degree is too low. P2 then makes a model mc$_1$ of P1's constraints and creates a counter offer o$_2$ that changes the deal to better fit its requirements while satisfying these constraints.

$mc_1 = \{$ P2 join P1's religion $\quad \rho = ?$

o$_2$ = (P1 gives 15 gold per turn & Banking technology , P2 joins P1's religion)

P1 gets this offer, but finds that the value it has to give is too high, and therefore makes a counter offer o$_3$ that is closer to this deal but satisfies its constraints. To create this deal it makes a model mc$_2$ of P2's constraints, assuming that P2 wants the banking tech as well as a deal worth more than X, where X is the value of what P1 pays in o$_1$.

$$mc_2 = \begin{cases} \text{Value of received items} \geq \text{X} & \rho = ? \\ \text{Get technology Banking} & \rho = ? \end{cases}$$

o$_3$ = (P1 gives 8 gold per turn & Banking technology , P2 joins P1's religion)

This offer is cheap enough to fit the third constraint in c$_1$, while also satisfying the model of P2's constraints. P2 receives this offer and finds that its satisfaction degree is high enough and accepts it. The result of the two players having the same religion is that they now have improved relations to each other, thereby satisfying P1's plan.

5.5 Example Run of our System

We have also decided to provide the output from an example run of our system to show how it works in practice. The output has been edited slightly to improve readability.

```
#### PLAYER 5 (FREQUENCY) VS PLAYER 1 (FREQUENCY)

### NEGOTIATION BETWEEN PLAYER 5 AND PLAYER 1
```

```
## PLAYER 5 OFFER

# WANT TO GIVE
- NOTHING
- LIST VALUE: 0

# WANT TO GET
- TRADE_WAR AGAINST P4
- TRADE_WAR AGAINST P6
- LIST VALUE: 46250

# SENDING OFFER

### NEGOTIATION BETWEEN PLAYER 1 AND PLAYER 5

## RECEIVED OFFER FROM 5

# SATISFACTION OF RECEIVED OFFER: 0.000000
# CONCESSION LIMIT: 0.900000

## PLAYER 1 COUNTER OFFER

# PLAYER 1 CONSTRAINTS
- GET_VALUE MINIMUM 10000 WITH PRIORITY 0.8
- NOT GIVE_ITEM TRADE_WAR AGAINST P6 WITH PRIORITY 0.4

# PLAYER 1 OPPONENT MODEL OF PLAYER 5
- GET_ITEM TRADE_WAR AGAINST P4 WITH PRIORITY 0.1
- GET_ITEM TRADE_WAR AGAINST P6 WITH PRIORITY 0.1

# WANT TO GIVE
- NOTHING
- LIST VALUE: 0

# WANT TO GET
- TRADE_TECHNOLOGIES #86
- TRADE_TECHNOLOGIES #52
- LIST VALUE: 15750
```

```
# SATISFACTION OF NEW OFFER: 1.000000
# SENDING NEW OFFER

### NEGOTIATION BETWEEN PLAYER 5 AND PLAYER 1

## RECEIVED OFFER FROM 1

# SATISFACTION OF RECEIVED OFFER: 0.000000
# CONCESSION LIMIT: 0.900000

## PLAYER 5 COUNTER OFFER

# PLAYER 5 CONSTRAINTS
- GET_ITEM TRADE_WAR AGAINST P4 WITH PRIORITY 0.5
- GET_ITEM TRADE_WAR AGAINST P6 WITH PRIORITY 0.2

# PLAYER 5 OPPONENT MODEL OF PLAYER 1
- GET_ITEM TRADE_TECHNOLOGIES #86 WITH PRIORITY 0.1
- GET_ITEM TRADE_TECHNOLOGIES #52 WITH PRIORITY 0.1

# WANT TO GIVE
- TRADE_TECHNOLOGIES #86
- LIST VALUE: 9430

# WANT TO GET
- TRADE_WAR AGAINST P4
- TRADE_WAR AGAINST P6
- LIST VALUE: 46250

# SATISFACTION OF NEW OFFER: 1.000000
# SENDING NEW OFFER

### NEGOTIATION BETWEEN PLAYER 1 AND PLAYER 5

## RECEIVED OFFER FROM 5

# SATISFACTION OF RECEIVED OFFER: 0.000000
# CONCESSION LIMIT: 0.900000
```

```
## PLAYER 1 COUNTER OFFER

# PLAYER 1 CONSTRAINTS
- GET_VALUE MINIMUM 10000 WITH PRIORITY 0.8
- NOT GIVE_ITEM TRADE_WAR AGAINST P6 WITH PRIORITY 0.4

# PLAYER 1 OPPONENT MODEL OF PLAYER 5
- GET_ITEM TRADE_WAR AGAINST P4 WITH PRIORITY 0.3
- GET_ITEM TRADE_WAR AGAINST P6 WITH PRIORITY 0.3
- GET_VALUE MINIMUM 46250 WITH PRIORITY 0.1

# WANT TO GIVE
- TRADE_WAR AGAINST P4
- LIST VALUE: 24440

# WANT TO GET
- TRADE_TECHNOLOGIES #86
- TRADE_TECHNOLOGIES #52
- LIST VALUE: 15750

# SATISFACTION OF NEW OFFER: 1.000000

# SENDING NEW OFFER

### NEGOTIATION BETWEEN PLAYER 5 AND PLAYER 1

## RECEIVED OFFER FROM 1

# SATISFACTION OF RECEIVED OFFER: 0.714286
# CONCESSION LIMIT: 0.900000

## PLAYER 5 COUNTER OFFER

# PLAYER 5 CONSTRAINTS
- GET_ITEM TRADE_WAR AGAINST P4 WITH PRIORITY 0.5
- GET_ITEM TRADE_WAR AGAINST P6 WITH PRIORITY 0.2

# PLAYER 5 OPPONENT MODEL OF PLAYER 1
```

```
- GET_ITEM TRADE_TECHNOLOGIES #86 WITH PRIORITY 0.3
- GET_ITEM TRADE_TECHNOLOGIES #52 WITH PRIORITY 0.3
- GET_VALUE MINIMUM 15750 WITH PRIORITY 0.1

# WANT TO GIVE
- TRADE_TECHNOLOGIES #86
- TRADE_TECHNOLOGIES #52
- LIST VALUE: 15750

# WANT TO GET
- TRADE_WAR AGAINST P4
- TRADE_WAR AGAINST P6
- LIST VALUE: 46250

# SATISFACTION OF NEW OFFER: 1.000000

# SENDING NEW OFFER

### NEGOTIATION BETWEEN PLAYER 1 AND PLAYER 5

## RECEIVED OFFER FROM 5

# SATISFACTION OF RECEIVED OFFER: 0.666667
# CONCESSION LIMIT: 0.900000

## PLAYER 1 COUNTER OFFER

# PLAYER 1 CONSTRAINTS
- GET_VALUE MINIMUM 10000 WITH PRIORITY 0.8
- NOT GIVE_ITEM TRADE_WAR AGAINST P6 WITH PRIORITY 0.4

# PLAYER 1 OPPONENT MODEL OF PLAYER 5
- GET_ITEM TRADE_WAR AGAINST P4 WITH PRIORITY 0.7
- GET_ITEM TRADE_WAR AGAINST P6 WITH PRIORITY 0.7
- GET_VALUE MINIMUM 46250 WITH PRIORITY 0.1
- GIVE_ITEM TRADE_TECHNOLOGIES #86 WITH PRIORITY 0.1
- GIVE_ITEM TRADE_TECHNOLOGIES #52 WITH PRIORITY 0.1

# WANT TO GIVE
```

```
- TRADE_WAR AGAINST P4
- TRADE_GOLD
- TRADE_GOLD_PER_TURN
- TRADE_RESOURCES #4
- TRADE_RESOURCES #15
- TRADE_RESOURCES #16
- TRADE_EMBARGO AGAINST P4
- TRADE_MAPS
- TRADE_WAR AGAINST P6
- LIST VALUE: 48260

# WANT TO GET
- TRADE_TECHNOLOGIES #86
- TRADE_TECHNOLOGIES #52
- LIST VALUE: 15750

# SATISFACTION OF NEW OFFER: 0.666667

# SENDING NEW OFFER

### NEGOTIATION BETWEEN PLAYER 5 AND PLAYER 1

## RECEIVED OFFER FROM 1

# SATISFACTION OF RECEIVED OFFER: 1.000000
# CONCESSION LIMIT: 0.900000

## ACCEPTING DEAL

### ENDING NEGOTIATION
```

Chapter 6

Experiments

This section describes the experiments we performed in order to evaluate our research questions and test the performance of our system. It presents our methods of evaluation of the test results, as well as the negotiation scenarios and how they were performed. The data sets for testing will not be presented in this report as they are too large.

6.1 Evaluation Method

When it comes to evaluating the performance of a negotiation system, there are many possible ways of doing this. There exists frameworks, such as the GENIUS environment [40], in which testing general negotiation systems in a variety of different and difficult problems is made simpler, giving a unified scale on which to compare these systems by. In some domains such as ours, however, the negotiation is too specialized to apply to such general problems. Another approach is to check for common solution properties such as Pareto optimality or maximization of social welfare. For simple types of negotiations, it is possible to make automatic checks for this to quantitatively check whether the results have these properties. In other domains it may be possible to prove whether the results will have these properties. We have given such proofs in sections 5.4.4 and 5.4.5. In our domain, automatically checking for Pareto optimality or maximization of social welfare is too computationally complex, so we focus on comparing the performance of different modeling approaches. After each negotiation round, the negotiating agents calculate their utility for the outcome. We will use this utility to compare how well our different opponent modeling techniques fare against each other in a set of different negotiation scenarios. As we can not compare our system to other general negotiation systems, we have decided to manually compute what negotiation outputs would maximize social welfare, and compare our results to these optimal results. The social welfare maximizing results are calculated by manually examining each data set and finding the offer that maximizes equation 6.1, where $sw(\omega)$ is the social welfare of an offer ω and is calculated by equation 6.2. This should give a reasonable grounds for assessing the quality of our results. We also record the number of offers it takes to reach a deal, the average success-rate of the negotiations, the average time it takes to reach a deal and the correctness of the opponent models.

$$optimal = max(sw(\omega)) = max(\sum_{i \in Ag} u_i(\omega)) \tag{6.1}$$

where Ag denotes all agents.

$$sw(\omega) = \sum_{i \in Ag} u_i(\omega) \tag{6.2}$$

where $u_i(\omega)$ is the satisfaction of the offer ω (see section 3.3.5).

6.1.1 Calculating Results

We have a number of different results we want to record when performing experiments, most of which require some calculation to find. This section therefore presents the formulas and algorithms to calculate these results for the sake of easier references later on. In the experiments presented in section 6.3, these algorithms will be used to calculate the results.

As shown in algorithms 5 and 6, our experiments will run all different combinations of opponent modeling types against each other. We have implemented two different opponent modeling types, frequency modeling and Bayesian modeling. In the experiments, we also include agents using no modeling in order to show whether opponent modeling actually makes an improvement to our system. We also included agents with full knowledge about their opponent's constraints in order to see how close the performance of our agents gets to agents' with perfect knowledge. These four agents types will be run against each other in our experiments and their performance will be evaluated using the measures presented in equations 6.3–6.8, where n is the number of negotiations and m is the number of offers made in a negotiation. A successful negotiation is defined as a negotiation where both agents meet an agreement.

$$SuccessRate = \frac{\text{number of successful negotiations}}{\text{number of negotiations}} \tag{6.3}$$

$$AverageSatisfaction = \frac{\sum_{i=1}^{n} \text{End satisfaction in negotiation i}}{\text{number of negotiations}} \tag{6.4}$$

$$AverageNumOfRounds = \frac{\sum_{i=1}^{n} \text{Number of offers made in negotiation i}}{\text{number of negotiations}} \tag{6.5}$$

$$AverageTimeToOffer = \frac{\sum_{i=1}^{n} \frac{\sum_{j=1}^{m} \text{time to make offer j}}{\text{number of offers in negotiation i}}}{\text{number of negotiations}} \tag{6.6}$$

$$p_{correct} = \frac{\text{number of correct constraints in model}}{\text{number of constraints}} \tag{6.7}$$

$$p_{wrong} = \frac{\text{number of wrong constraints in model}}{\text{number of constraints in model}} \tag{6.8}$$

6.2 Negotiation Scenarios

This section shows the negotiation scenarios we use for our experiments. Each scenario consists of two players each of which has a set of items and a set of constraints describing their preferences. In these scenarios, player 1 in the list is the one starting the negotiation, while player 2 is the opponent. We also show

```
Data: Opponent modeling types, scenarios
Result: result stats for all agents
if newOpponentOffer is first offer then
    create array with all possible hypotheses per issue with uniform probability;
end
foreach opponent modeling type $om_i$ do
    foreach opponent modeling type $om_j$ do
        foreach scenario $S_k$ do
            run negotiation scenario; if(negotiation comes to a deal)
            increase $successfulCount$; increase $numOfRoundsCount$ by
            number of rounds in negotiation; increase $satisfactionSum$ by
            end satisfaction; foreach Offer made in negotiation do
                increase $timeToOfferSum$ by time it took to make the offer;
            end
            increase $OfferTimePerScenarioSum$ by
            $timeToOfferSum/$number of offers made ;
        end
        $SuccessRate = successfulCount/$number of scenarios;
        $AverageNumOfRounds = numOfRoundsCount/$number of
        scenarios; $AverageSatisfaction = satisfactionSum/$number of
        scenarios; $AverageTimeToOffer =$
        $OfferTimePerScenarioSum/$number of scenarios
    end
end
```

Algorithm 5: Pseudo code for calculating results of our negotiation. Calculates success-rate, average satisfaction, average time used to make an offer and average number of offers to get a deal for all combinations of opponent modeling agents. The different opponent modeling types are our *frequency modeling* and *Bayesian modeling* agents as well as *agents using no opponent model* and *agents using the actual constraints of the opponent* (perfect knowledge).

```
Data: Opponent modeling types, scenarios
Result: result stats for all agents
if newOpponentOffer is first offer then
    create array with all possible hypotheses per issue with uniform
    probability;
end
foreach opponent modeling type $om_i$ do
    foreach opponent modeling type $om_j$ do
        foreach scenario $S_k$ do
            run negotiation scenario; foreach constraint in opponent
            model do
                if constraint found in correct constraints then
                    increase $CorrectConstraintsCount$;
                else
                    increase $wrongConstraintsCount$
                end
            end
            $P_{correct} = CorrectConstraintsCount/NumOfConstraints$;
            $P_{wrong} =$
            $wrongConstraintsCount/NumOfConstraintsInOpponentModel$
        end
    end
end
```

Algorithm 6: Pseudo code for calculating the model correctness. When calculating this, we only use our two opponent modeling types, *frequency modeling* and *Bayesian modeling.* We calculate the percentage of correct constraints found, as well as the percentage of constraints found that are wrong

the social welfare maximizing deal for each scenario, and the utility this deal would give each player.

Scenario 1 is a relatively simple scenario, where we would expect the modeling agents to be able to find the constraints of the opponents, especially P1's constraints are easy to find, while only two of P2's three constraints are easy to find, as constraints about not giving items can be difficult. We do however expect our Bayesian modeler to be able to find these constraints as well. The scenario requires a trade-off to reach a deal, and it it easier for P2 to concede than it is for P1.

Scenario 1

players: P1 and P2 year 2000

P1 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- defensive pact val:60
- item 81 val:7020
- resource 7 val:310

P2 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- defensive pact val:60
- item 47 val:7280
- item 48 val:6820

- item 51 val:8780
- item 88 val:2090
- war with P5 val:11420

P1 constraints:

- get maps pri:0.6
- get item 47 pri:0.7 (technology)
- get item 48 pri: 0.1 (technology)

P2 constraints:

- get defensive pact pri:0.8
- give defensive pact pri:0.8
- not give item 47 pri:0.1

Social welfare maximizing deal (Pareto optimal):
P1 gives:

- defensive pact

P2 gives:

- defensive pact
- maps
- item 47
- item 48

Utility for P1: 1.0
Utility for P2: 0.941

Scenario 2 is more difficult to model. P1 has few constraints, and one of them is a value constraint, which is difficult to find. The frequency modeler may be able to find this during the negotiation, but with some uncertainty. P2 has more constraints, half of which are easy to find, the other half is difficult, but the Bayesian modeler should be able to find them. This scenario also requires a trade-off to make a deal, as P1 has a constraint on not giving for a value above 1000, and P2 wants items for a much higher value. It is easiest for P2 to concede as its constraint for getting item 81 has a low priority.

Scenario 2

players: P1 and P2 year 2000

P1 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- defensive pact val:60
- item 81 val:7020
- resource 7 val:310

P2 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- defensive pact val:60
- item 47 val:7280
- item 48 val:6820
- item 51 val:8780

- item 88 val:2090
- war with P5 val:11420

P1 constraints:

- get war with P5 pri:0.9
- give for a value less than 1000 pri:0.8

P2 constraints:

- get item 81 pri:0.1
- get resource 7 pri:0.8
- not get defensive pact pri:0.5
- not give defensive pact pri:0.5

Social welfare maximizing deal (Pareto optimal):
P1 gives:

- resource 7

P2 gives:

- war with P5

Utility for P1: 1.0
Utility for P2: 0.947

Scenario 3 is a difficult one to model correctly, as P1 has only value constraints, which are difficult to find. We would expect our frequency modeling agents to perform better than the Bayesian modeling agents on finding these constraints. P2's constraints are easier for the Bayesian modeling agents. This scenario is one of the two that require no trade-off to make a deal, meaning we would expect a good average satisfaction from it.

Scenario 3

players: P1 and P2 year 2000

P1 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- defensive pact val:60
- item 81 val:7020
- resource 7 val:310

P2 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- defensive pact val:60
- item 47 val:7280
- item 48 val:6820
- item 51 val:8780
- item 88 val:2090
- war with P5 val:11420

P1 constraints:

- get for a value above 7500 pri:0.9
- give for a value less than 7500 pri:0.9

P2 constraints:

- not give item 47 pri:0.9
- not give war with P5 pri:0.8
- get maps pri:0.4
- get item 81 pri:0.5

Social welfare maximizing deal (Pareto optimal):
P1 gives:

- maps
- item 81

P2 gives:

- item 51

Utility for P1: 1.0
Utility for P2: 1.0

Scenario 4 is similar to the previous scenario in that P1's constraints are difficult, but easier for the frequency modeling agents than for the Bayesian modeling agents, while P2's constraints are easier for the Bayesian modeling agents. This scenario is simpler than the last one however, as both players have some simple constraints in addition to the difficult ones, and we would therefore expect better model correctness in this scenario compared to scenario 3. The scenario also requires a trade-off in order to make a deal, and it is easiest for P2 to concede, the concession necessary is small however.

Scenario 4

players: P1 and P2 year 1902

P1 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- item 20 val:400
- item 75 val:3040
- war with p4 val:8930
- embargo against p4 val:730

P2 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0 val:
- item 14 val:1650
- item 19 val:800
- item 74 val:2740

P1 constraints:

- get item 14 pri:0.8
- get item 74 pri:0.6
- give for value less than 1000 pri:0.8

P2 constraints:

- get embargo against p4 pri:0.8
- get item 20 pri:0.1
- not get war with P4 pri:0.4

Social welfare maximizing deal (Pareto optimal):
P1 gives:

- embargo against P4

P2 gives:

- item 14
- item 74

Utility for P1: 1.0
Utility for P2: 0.923

Scenario 5 has many constraints, where most of them are simple, meaning we expect a good result for the modeling on this scenario. There are some more difficult constraints, but the overall correctness should be high. The scenario does require a small trade-off to make a deal, as P2 wants to get maps from its opponent, while P1 does not want to give this item. It is easiest for P1 to concede in this scenario, but the necessary concession is small for either of the players, so we could expect any of them to concede.

Scenario 5

players: P1 and P2 year 2049

P1 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- defensive pact val:54
- item 46 val:4960
- item 52 val:6320

- item 86 val:9430
- resource 22 val:300
- war with P4 val:14010

P2 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- resource 4 val:160
- resource 15 val:300
- resource 16 val:300
- war with P4 val:24440
- war with P6 val:21810

P1 constraints:

- get war with P4 pri:0.6
- get war with P6 pri:0.6
- give resource 22 pri:0.2
- not give maps pri:0.1

P2 constraints:

- get war with P4 pri:0.4
- get maps pri:0.7
- give maps pri:0.2
- get item 86 pri:0.1

- get resource 22 pri 0.3
- get for value above 10000 pri:0.9

Social welfare maximizing deal (Pareto optimal):
P1 gives:

- war with P4
- resource 22
- maps
- item 86

P2 gives:

- war with P4
- war with P6
- maps

Utility for P1: 0.933
Utility for P2: 1.0

Scenario 6 has one player with very simple constraints, and one with very difficult constraints. We would expect all modeling approaches to model P1's constraints correctly, but P2's constraints are much harder. Its first constraint should be easy for the Bayesian modeler, but difficult for the frequency modeler, while its second constraint can not be found by the Bayesian modeler, but may be found by the frequency modeler. This means that we would expect quite low model correctness for both modeling types in this scenario, and this might also make it difficult to succeed in the negotiation, making a lower success rate more likely. In addition to this, the scenario requires a large trade-off in order to make a deal, where it is easiest for P1 to concede.

Scenario 6

players: P1 and P2 year 2049

P1 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- defensive pact val:54
- item 46 val:4960
- item 52 val:6320
- item 86 val:9430
- resource 22 val:300
- war with P4 val:14010

P2 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- resource 4 val:160
- resource 15 val:300
- resource 16 val:300
- war with P4 val:24440
- war with P6 val:21810

P1 constraints:

- get war with P4 pri:0.5
- get war with P6 pri:0.2

P2 constraints:

- not give war with P6 pri:0.4
- get for a value above 10000 pri:0.8

Social welfare maximizing deal (Pareto optimal):
P1 gives:

- item 46
- item 52

P2 gives:

- war with P4

Utility for P1: 0.71
Utility for P2: 1.0

Scenario 7 is similar to scenario 6 in that it has very few constraints, and many of them are difficult to model, especially for the Bayesian modeler. This scenario is actually even harder to model, as it mostly consists of value constraints. We would expect agents using frequency modeling to do better than Bayesian modeling agents in the scenario, but both modeling types will have a hard time finding a good model, and this may lower the success rate of this scenario.

Scenario 7

players: P1 and P2 year 2049

P1 tradeable items:

- gold val:0

- gold per turn val:0
- maps val:0
- defensive pact val:54
- item 46 val:4960
- item 52 val:6320
- item 86 val:9430
- resource 22 val:300
- war with P4 val:14010

P2 tradeable items:

- gold val:0
- gold per turn val:0
- maps val:0
- resource 4 val:160
- resource 15 val:300
- resource 16 val:300
- war with P4 val:24440
- war with P6 val:21810

P1 constraints:

- get for a value above 20000 pri:0.7
- not give war with P4 pri:0.4

P2 constraints:

- get for a value above 15000 pri:1.0

Social welfare maximizing deal (Pareto optimal):
P1 gives:

- item 52
- item 86

P2 gives:

- war with P4

Utility for P1: 1.0
Utility for P2: 1.0

6.3 Negotiation Experiments

The negotiation experiments are designed to test our research questions about the negotiation system and how well it performs when negotiating with other agents.

In the experiments, we set up seven different scenarios in the Civilization IV game where each player has a set of predefined items to negotiate about and a set of predefined preferences in the form of constraints. These will be our test data and are described in section 6.2. The agents will then negotiate, trying to achieve as high a utility for themselves as possible. Each of these scenarios will be run for all possible combinations of agent types (see table 7.1), so that all the techniques will be compared to all other techniques. The agent types will be agents using our frequency modeling or our Bayesian modeling techniques described in section 5.4, as well as agents with no information and no modeling, and agents with full information and no modeling. After the negotiations, the outcome will be recorded, including the satisfaction level of each player, the number of offers, and the average computation time it took to reach the offers. We will then evaluate our research questions, comparing the results from the different opponent modeling types and the no information and full information solutions, finding whether our opponent modeling techniques make a valid contribution to the negotiation system or not (RQ3), whether the results can be achieved fast enough to make it viable in a soft real-time system (RQ1), and how good results we are able to achieve (RQ2).

The experiments will be run on a laptop computer with the following specifications:

CPU Intel® Core^{TM}2 Duo P9500 @ 2.53GHz

RAM 3GB

GPU NVIDIA® GeForce® GTX 260M

Operating System Microsoft® Windows® XP Professional SP3

6.3.1 Experiment N1: Model correctness in *agent vs agent* negotiations

In this experiment will test the correctness of our opponent modeling techniques when negotiating with other agents. Before each negotiation, both agents will write out their preferences in the form of constraint sets, and during the negotiations, they will write out their opponent models for comparison against the correct constraints.

In order to find the correctness of the model, we count the number of constraints in the model that are in the actual constraints as well, calculating the percentage of correct constraints in the model $P_{correct}$. When checking value constraints, we check how close they are to the actual constraint, increasing the correctness the closer to the actual constraint value it is. We also need to count how many constraints the model contains that are not in the actual constraints of the opponent, finding the percentage of wrong constraints in our model P_{wrong}. A higher value for $P_{correct}$ is better, while a lower value for P_{wrong} is better. Equations 6.7 and 6.8 show this calculation. Algorithm 6 illustrates how we compute these values. We check the correctness of the models both after the first offer and after the last offer. This allows us to evaluate whether the opponent models improve during a negotiation, and how good they are in the beginning of the negotiations.

6.3.2 Experiment N2: Testing negotiation performance in ***agent vs agent*** negotiations

This experiment is for evaluating how well our negotiation system performs. Specifically, we will compare our modeling approaches with negotiation performance without opponent modeling to see how much of an improvement our modeling gives, as this is one of our major research questions (RQ3). The results will also be compared with the performance of negotiating agents with perfect knowledge of the opponent's preferences. This will be used to evaluate how close our system gets to optimal results. All negotiation outcomes will be evaluated based on the utility, $AverageSatisfaction$, achieved at the end of the negotiations, the number of offers it takes to come to a deal $AverageNumOfRounds$, the time it takes to make an offer $AverageTimeToOffer$ and the success rate of the negotiations $SuccessRate$. These values will be calculated according to equations 6.3–6.6 using algorithm 5.

Chapter 7

Results and Discussion

In this chapter we will present the results from our experiments and discuss what we found and why we got the results that we got. Below is an overview of the different agent types that we have tested for easier lookup in the tables.

1	No modeling
2	Frequency modeling
3	Bayesian modeling
4	Full information

Table 7.1: Agent types

7.1 Experiment N1: Model correctness in *agent vs agent* negotiations

This experiment intends to verify how well the opponent modeling actually performs. We have tested both our modeling types against all the different agents, including agents not using modeling and agents with full information. For each scenario, we have calculated the average correctness for each modeling type both after the first offer and after the last one. The correctness values we compute are the $P_{correct}$ and P_{wrong} values explained in equations 6.7 and 6.8, showing the percentage of the opponent's constraints we find and the percentage of wrong constraints in our model. Using these two values, we can evaluate how successful our modeling approaches are.

7.1.1 Results

This section presents the results from our experiment, showing the $P_{correct}$ and P_{wrong} values after the first and after the last offer made in the negotiations for each modeling type. The experiment has been run in negotiations against each of the four agent types, and the the $P_{correct}$ and P_{wrong} values in the tables are averages over all negotiations for each scenario. The average values are shown for both modeling types and for each of our seven scenarios. A description of the scenarios can be found in section 6.2.

Data set	Frequency modeling first offer	Frequency modeling last offer	Bayesian modeling first offer	Bayesian modeling last offer
1	0.834	0.834	1.000	1.000
2	0.500	0.500	0.625	0.750
3	0.250	0.427	0.250	0.000
4	0.667	0.667	0.750	0.750
5	0.792	0.820	0.958	0.958
6	0.500	0.659	0.875	0.875
7	0.000	0.525	0.375	0.375

Table 7.2: Opponent modeling correctness in AI versus AI negotiation. This table shows the $p_{correct}$ values

Data set	Frequency modeling first offer	Frequency modeling last offer	Bayesian modeling first offer	Bayesian modeling last offer
1	0.000	0.469	0.800	0.800
2	0.000	0.428	0.883	0.833
3	0.500	0.500	0.933	1.000
4	0.000	0.417	0.826	0.826
5	0.084	0.334	0.749	0.749
6	0.500	0.678	0.897	0.897
7	1.000	0.673	0.956	0.956

Table 7.3: Opponent modeling correctness in AI versus AI negotiation. This table shows the p_{wrong} values

7.1.2 Discussion

The percentage of correct constraints found by our modeling approaches for the different scenarios and the different modeling approaches is shown in table 7.2. Table 7.3 shows the percentage of the modeled constraints that are wrong, that is, how many of the constraints that have been found by the modeler are not correct.

Frequency modeling

Examining the results for the frequency modeling agents, it is noticeable that the percentage of correct constraints $p_{correct}$ is generally lower than that of the Bayesian modeler, but that the percentage of wrong constraints p_{wrong} is also much lower. This can be seen in the tables, where table 7.2, showing the $p_{correct}$ values, has lower values for frequency modeling in almost all the scenarios. Table 7.3, showing the p_{wrong} values, also has lower values in almost all the scenarios. The lower percentage of correct constraints can probably be attributed to the difficulty of finding constraints of the type NOT_GIVE_ITEM or NOT_GET_ITEM. These are not visible purely from the offers suggested by the opponent, and are therefore very difficult to find. The Bayesian modeler finds these, but while doing this, it also finds a large set of false constraints. Due to this it has a much higher p_{wrong} value than the frequency modeler. None of the models find all the constraints very often, as the frequency modeler is cannot find constraints such as NOT_GIVE_ITEM or NOT_GET_ITEM, while the Bayesian modeler is will not find value constraints. In order to make a good opponent modeler in this domain, one would have to find a modeling approach that is able to overcome both of these difficulties, maybe by combining several different approaches.

A very good feature of the frequency modeler is that it gives very low p_{wrong} values, especially at the start of the negotiation, meaning that it finds few false constraints. This is opposed to the Bayesian modeler which finds a large amount of false constraints. An example of this is scenario 1, where the p_{wrong} value is 0.0 after the first offer for the frequency modeler, while the value is 0.8 for the Bayesian modeler, meaning it finds a lot more false constraints.

Another strong point of the frequency modeler is that the $p_{correct}$ value often improves during the negotiation, as the modeler finds more constraints, especially value constraints that are not revealed by the initial offer. An example of this is scenario 3, where it starts off poorly, with a $p_{correct}$ value of 0.25 after the first offer, but this is improved to 0.427 during the negotiation. This makes it easier for the frequency modeler to find good offers later in the negotiation. It does however, have a problem, its p_{wrong} value also increases quite often during the negotiation, meaning it finds more false constraints during the negotiation. This problem can be seen in scenario 1, where the p_{wrong} value increases from 0.0 to 0.469 during the negotiation. The process of finding more constraints during the negotiation can apparently be both positive and negative to the end result. The method should therefore be tweaked more in any future work to minimize the number of false constraints being found.

Bayesian Modeling

The Bayesian modeler seems to give overall better values for $p_{correct}$ than the frequency modeler, sometimes even finding all of the constraints, such as in scenario 1. However, the values are generally either very high or very low, with scenario 3 even giving it a $p_{correct}$ value of 0.0, meaning it either finds most of the constraints or very few of them. This modeling approach generally finds a lot of constraints that frequency modeling does not find, such as constraints for not giving items, explaining to some degree why this modeling approach often gives higher values for $p_{correct}$ than the frequency modeler. It does not currently find value constraints however, which causes it to perform very poorly on some scenarios such as scenario 3 and scenario 7, where there are mostly value constraints. This should explain the large variation between the results on different scenarios.

The Bayesian modeler often gets a high value for p_{wrong}, meaning it finds a lot of false constraints. This can be seen very clearly in scenario 3, where it actually gets a p_{wrong} value of 1.0 after the last offer. This problem is made smaller by the often high number of correct constraints, but can still be problematic. The many false constraints can sometimes even be positive, as making a lot of constraints about the opponent wanting items can make up for the problem of not finding the value constraints of the opponents. Due to this, we have seen in the results from experiment N2 that the Bayesian opponent modeler generally performs similarly to the frequency modeler when it comes to average satisfaction or success rate. Both of the methods often struggle making offers however, as they may give up if they have no more constraints to satisfy without ruining their own satisfaction. Looking at the models made by the Bayesian modeler, it is easy to see that this may be a problem, as it often makes many constraints about not giving and not getting items, meaning that when it runs out of sensible concessions it may start to remove items from its offers, either making them unattractive to itself or its opponents. As an example of this we show an opponent model made by the Bayesian modeler:

- GIVE_ITEM TRADE_GOLD ShouldGive: FALSE
- GIVE_ITEM TRADE_GOLD_PER_TURN ShouldGive: FALSE
- GIVE_ITEM TRADE_MAPS ShouldGive: FALSE
- GIVE_ITEM TRADE_DEFENSIVE_PACT ShouldGive: FALSE
- GIVE_ITEM TRADE_TECHNOLOGIES ShouldGive: FALSE
- GIVE_ITEM TRADE_TECHNOLOGIES ShouldGive: FALSE
- GIVE_ITEM TRADE_TECHNOLOGIES ShouldGive: FALSE
- GIVE_ITEM TRADE_RESOURCES ShouldGive: TRUE
- GIVE_ITEM TRADE_WAR ShouldGive: FALSE
- GET_ITEM TRADE_GOLD ShouldGet: FALSE
- GET_ITEM TRADE_GOLD_PER_TURN ShouldGet: FALSE
- GET_ITEM TRADE_MAPS ShouldGet: FALSE
- GET_ITEM TRADE_RESOURCES ShouldGet: FALSE
- GET_ITEM TRADE_RESOURCES ShouldGet: FALSE
- GET_ITEM TRADE_RESOURCES ShouldGet: FALSE
- GET_ITEM TRADE_WAR ShouldGet: TRUE
- GET_ITEM TRADE_WAR ShouldGet: TRUE

In this case, the true constraints are:

- GIVE_ITEM TRADE_MAPS ShouldGive: FALSE

- GIVE_ITEM TRADE_RESOURCES ShouldGive: TRUE
- GET_ITEM TRADE_WAR ShouldGet: TRUE
- GET_ITEM TRADE_WAR ShouldGet: TRUE

It is clear from the example that all the constraints were found, but most of the models consist of false constraints. The problem here is that the agent may think it is improving the deal for its opponent when it removes an item like TRADE_GOLD from its offer, while this actually does not make any difference for the opponent. This could make the negotiation slower, as concession will be slow, and increases the risk of the opponent giving up. A more serious issue may occur when the model only has constraints that contradict with the agent's own constraints left to be satisfied. In this case, it will start conceding on issues based on false information, which will not make the opponent any more satisfied, and usually ends with the agent giving up. Normally a deal can be found before this happens, but if the modeler has not found enough correct constraints to make a satisfactory deal for the opponent, this may happen, and will usually cause the negotiation to fail. As mentioned in the discussion on experiment N2, the problem of agents giving up like this could be fixed by making them try giving more when they do not find any more good concessions to make, but the problem could also possibly be minimized by reducing the number of false offers in the opponent modeling.

The reason for the large amount of false constraints in the model is that it is possible to make many constraints that are satisfied by an offer, even though they are actually irrelevant to this offer, and these constraints will have the exact same probability as the correct constraints. For instance, the modeler would find that an offer suggesting to give the opponent the resource stone makes a constraint about not wanting to give gold just as probable as a constraint about wanting to give stone. In order to fix this problem, the modeling technique would need to be tweaked so that constraints that are unintentionally satisfied become less likely. The simplest approach to fixing this would be to have priori knowledge about what kind of constraints are the most probable, but this cannot be guaranteed in any real situations, and is therefore an undesirable approach. Another solution could be to tweak the learning algorithm itself so that the probabilities are not only affected by the satisfaction level of the constraint given the offer, but also how closely the constraint is related to the offer. A system that does this may find less false constraints, but will probably also miss a lot of the more difficult constraints that our system finds.

We also notice that the Bayesian modeler does not generally improve during the negotiation. It finds a large set of constraints based on the first offer, and these are not changed throughout the negotiation. It does reasonably well on the first offer however, so refining the model gradually may not be necessary.

Detailed discussion of each scenario

Looking at each scenario in detail, we see that in the first scenario both modeling techniques got a high $P_{correct}$ value, but the Bayesian modeler was clearly better with a value of 1.0 versus a value of 0.834. This was expected in our discussion of each scenario in section 6.2 as scenario 1 is simple and should be relatively easy to model, but it does contain constraints about not giving items, and this can be difficult for the frequency modeler to find, leading to the frequency modeler to get a lower $P_{correct}$ compared to the Bayesian modeler.

Scenario 2 was expected to show similar results, but with a more pronounced difference as it is a more difficult scenario to model. There are more constraints about not giving items in this scenario, causing the frequency modeler to perform poorly with only a $P_{correct}$ value of 0.5 compared to the Bayesian modeler's score of 0.625. In this scenario we expected to see the frequency modeler improve its correctness during the negotiation as it tries to find the value constraint of its opponent, but this did not happen. From row 2 in table 7.3, we see however, that it did find a lot of false constraints, causing its p_{wrong} value to increase from 0.0 to 0.428.

Scenario 3 was expected to be much more of a problem for our modeling approaches, and to be harder for the Bayesian modeler than for the frequency modeler as the frequency modeler may find the value constraints eventually during the negotiations. This prediction seems to be true, as the Bayesian modeler performs very poorly, with $P_{correct}$ values of 0.25 and 0.0, while the frequency modeler started with the same value of 0.25, but increased its correctness to 0.427 during the negotiation.

For scenario 4, we predicted that the results would be similar to scenario 3, but with higher scores, as this scenario has more easy constraints in addition to the hard constraints. The actual results were a bit different however, as our Bayesian modeler did quite well with a $P_{correct}$ value of 0.75, and once again, the frequency modeler failed to learn the value constraint, thereby being unable to improve its score of 0.667 during the negotiation. Similarly to scenario 2, the frequency modeler found more false constraints instead, ending with a P_{wrong} value of 0.417.

The next scenario, however, was expected to be much easier, as it contains a large amount of easy constraints and a few harder ones. This is evident in the results , where both modeling techniques did quite well. The Bayesian modeler is clearly the best technique in this scenario with a $P_{correct}$ value of 0.958, but the frequency modeler also does well, and even improves its $P_{correct}$ value during the negotiation, ending with a value of 0.82.

Scenario 6 was predicted to be much more difficult compared to the previous scenarios, mostly due to its small number of constraints, most of which are difficult to find. In the actual results, both modeling techniques did worse than in the last scenario, but compared to some of the other difficult scenarios such as scenario 3, the results are surprisingly good with the Bayesian modeler gaining a $P_{correct}$ value of 0.875 and the frequency modeler improving its score from 0.5 to 0.678 during the negotiations. It seems then that we have overestimated the difficulty of this scenario.

The last scenario is largely similar to scenario 6, but was predicted to be even harder, especially for the Bayesian modeler which will struggle with the many value constraints in the scenario. The frequency modeler was expected to be able to find some of these constraints during the negotiation, and thereby doing a bit better than the Bayesian modeler. This is evident in the results, where the Bayesian gets a low $P_{correct}$ score of 0.375, while the frequency modeler is able to improve its model from the dismal start of 0.0 to a value of 0.525.

7.2 Experiment N2: Testing outcomes in *agent vs agent* negotiations

This experiment tests our system's performance on a varied set of scenarios containing the trade-able objects and the constraints of both players. Two AI controlled players are facing off against each other, trying to maximize the satisfaction of their own constraints. The experiment records the average success rate over each of the seven scenarios described in section 6.2 for each combination of agent type, as well as the average satisfaction degree each player achieved, the average number of rounds and the average time per negotiation. The different agent types can be seen in table 7.1. In order to know how good these results actually are, we have also decided to manually calculate the optimal results for these values so that we can compare our results to this. These values are presented below. We then present the results of the experiments before we discuss what we have noticed in these results. The end of this chapter gives an overall discussion of the results from all the experiments.

7.2.1 Optimal results

We have calculated what the optimal results would be for all the scenarios by examining each scenario and manually calculating what deal would maximize social welfare. All of the deals that maximize social welfare are also Pareto optimal, but they are not the only Pareto optimal deals.

Social welfare is defined as the function:

$$sw(\omega) = \sum_{i \in Ag} u_i(\omega)$$

where Ag denotes all agents. The deal that maximizes social welfare, is the deal that satisfies the equation:

$$optimal = \arg\max_{alldeals}(sw(\omega)) = \arg\max_{alldeals}(\sum_{i \in Ag} u_i(\omega))$$

The social welfare maximizing deals are included in the scenario descriptions in section 6.2. For each of them, we have calculated the satisfaction degree for each player. We found that it should be possible to make a satisfactory deal on all the negotiation scenarios, meaning that our system should be able to achieve a success-rate of 100%. We see in the discussion of these scenarios in section 6.2 that it is often easier for player 2 to concede than for player 1, where player 1 is the agent starting the negotiation, and comes first in the scenario description, while player 2 is its opponent. This is due to conflicting constraints in our scenarios, where player 2 has a lower priority on the constraints, making it easier for this player to concede. This means that in the social welfare maximizing deals, the player making the first offer is generally better off than its opponent. There are exceptions to this however, and especially scenario 6 gives a very low satisfaction to player 1 in the social welfare maximizing deal as shown in the description of this scenario in section 6.2. Due to this particular scenario, player 2 gets a higher average satisfaction than player 1 does in all scenarios. The average satisfaction of the social welfare maximizing deal for player 1 throughout the scenarios is 0.949, while the average satisfaction for player 2 is 0.973. This particular issue is an artifact of our scenarios, and could be avoided by making a more balanced set of scenarios, but it is not a problem for the testing so we have decided not to change it. The imbalance in the scenarios should also be evident in our results if our system gets close to producing social welfare maximizing deals, as player 2 should be getting a higher average satisfaction than player 1. The satisfaction values of the optimal deals given in section 6.2 can be compared to the average satisfaction in our results to evaluate how well our system does.

7.2.2 Actual Results

Here are the actual results from our test run. In order to make the tables smaller, we represent the different opponent modeling agents as a number. These agent types and their number representation can be seen in table 7.1. In the result tables 7.4, 7.5 and 7.7 we show the results from two agents facing off against each other, where the identifiers in columns and rows show what agents are negotiating with each other. Each negotiation type faces off against each negotiation type once for each scenario, including facing off against an agent with the same negotiation type as itself. The results in the tables are averages over all the seven scenarios. Table 7.6 is a little different from the other tables. It shows the time used for making offers for each agent against each other agent. In this table, the agent in the column is making the offer against the agent in the row.

		Player 1			
		1	2	3	4
Player 2	1	42.9%			
	2	57.1%	71.4%		
	3	71.4%	71.4%	71.4%	
	4	85.7%	71.4%	57.1%	85.7%

Table 7.4: Success-rate in AI versus AI negotiations

		Player 1			
		1	2	3	4
Player 2	1	11.6			
	2	10.2	4.57		
	3	8.14	4.57	4.71	
	4	6.14	4.57	4.71	4.29

Table 7.5: Average number of offers in AI versus AI negotiations

		Player making offer			
		1	2	3	4
Opponent	1	0.0280	0.0355	0.0177	0.0260
	2	0.0933	0.0843	0.0614	0.0651
	3	0.1101	0.0838	0.0569	0.0330
	4	0.1106	0.0887	0.0805	0.0716

Table 7.6: Average time to make an offer in AI versus AI negotiations. All times are shown in seconds. The player making the offer is in the column.

		Player 1			
		1	2	3	4
Player 2	1	0.429/0.420			
	2	0.705/0.658	0.705/0.651		
	3	0.705/0.651	0.705/0.651	0.666/0.647	
	4	0.848/0.783	0.705/0.651	0.562/0.515	0.848/0.800

Table 7.7: Average satisfaction in AI versus AI negotiations

7.2.3 Discussion

Success Rate

Table 7.4 shows the success rate of the negotiations in the experiment. We have calculated the average success rate over all scenarios for each combination of agent types. The success rate is calculated according to equation 6.3 in section 6.1.1.

The first thing we noticed when examining the results, is that the agents that don't use modeling (agent type 1 in the table) get a much lower success rate. In particular, the experiment where agent type 1 against agent type 1 gives the lowest success rate in the experiment with 42.9 per cent. This was expected, and gives a positive answer to our research question RQ3, about whether opponent modeling improves the results of the negotiations. The results show a clear improvement when using opponent modeling over not using it. Compared to the agents that have full information about their opponents (agent type 4), the opponent modeling agents (agent types 2 and 3) generally achieve a lower success rate. The results are, however, quite close as agent types 2 and 3 get a success rate of 71.4 per cent in most of the cases, while agent type 4 has its best success rates at 85.7 per cent and its worst success rate at 57.1 per cent. It therefore seems that our opponent modeling approach is working quite well, but not quite as good as having full information.

We also notice that Bayesian modeling and frequency modeling seem to be performing equally good in most cases. They pretty much get the same result of 71.4 per cent in most match-ups. There are exceptions however, and especially one curious case of Bayesian vs full information (agent types 3 versus 4) performs very poorly with a score of only 57.1 per cent. An important factor for the similar results in the table is the low number of scenarios in our experiments. We only have seven scenarios. This means that the difference between the best results of the full information agent and the slightly worse results of the opponent modeling agents is only the difference of five successful negotiation out of seven scenarios versus in six out of seven. This also means that the lower success rate for agent type 2 versus 4 can be attributed to randomness. The opponent modeling agents seem to generally succeed on five out of our seven scenarios (71.4 per cent), meaning that succeeding on four out of seven negotiations is not a surprise. In order to combat this problem, a larger set of scenarios should be used for testing, but this has not been done in this project, as automatic testing is not possible in the Civilization IV domain, and experiments are therefore very time consuming.

Not even agents using full information (agent type 4) perform perfectly, with the best success rate being six out of seven scenarios (85.7 per cent). This may be caused by our searching algorithm being a local search, which is not exhaustive and may not find the best offer given the current knowledge. In addition to this, the negotiation may fail because the opponent gave up, which gives both agents a lower success rate. This seems to be a problem in both of our opponent modeling types. A reason for this may be that the opponent modeler makes a small number of constraints for the opponent, and tries to satisfy these when conceding as described in section 5.3.3. If it cannot satisfy these constraints while simultaneously satisfying its own constraints, it will give up. This means that in many of our scenarios, especially the ones with fewer constraints to satisfy, the opponent modeling agents are more prone to giving up. This shows as the full information agents do much worse against the opponent modelers than against agents of its own type or against agents that do not use models. Notice especially that the full information agent does well against agents with no modeling (agent type 1 versus 4 in table 7.4). These agents do not give up very easily, as they will keep trying to give more as long as there is more to give, and may therefore be a little more likely to find a possible deal, giving a higher success rate. In order to fix this problem, we would need to change our algorithm for creating offers into an algorithm that is less dependent on the opponent model. An agent that tries giving things that are not in its opponent model instead of giving up would probably end up with a much higher success rate in this experiment. This would also increase the success rate of the full information agent, as it only ever fails when its opponent gives up before it has produced a good enough deal.

Number of Offers

In order to evaluate how fast our agents negotiate, we have also calculated the average number of offers made over all scenarios for each combination of agent types. These results are shown in table 7.5. Agents using modeling (agent type 2 or 3) or agents having full information (agent type 4) generally need much fewer offers before reaching a deal than agents that don't use any information about their opponents. This can be seen from table 7.5, where agent 1, which does not use modeling, gets the worst result against agents of the same type with 11.6 offers in average before finding a deal. When negotiating against other agent types, the results improve, and the best results come when agents with full information (agent type 4) negotiate with each other, needing only 4.29 offers in average. This is as expected, as agents using no modeling (agent type 1) will just try to add any items to make its offer more attractive to its opponent, not knowing what the opponent wants. In cases where the agents have value constraints, this may not be a big problem, but when negotiating with agents that want specific items, it will generally take many offers before a good deal can be found. We also see in these results that our opponent modeling agents need almost as few offers as the full information agents, needing only 4.57 or 4.71 offers on average in most of the match-ups, meaning that the modeling definitely helps the agents finding an acceptable offer faster.

Time use

The time used to make an offer is shown in table 7.6. The time is shown for each agent type, with separate results against each agent type. From this table, we see that the time spent making offers is generally very low with the highest value being 0.1106 seconds, which is important for our system, because low response times are important, especially when the AI's are negotiating with each other while the players are waiting. Notice that the complex opponent modeling such as Bayesian learning is not slower than the simpler frequency modeling or even the no modeling agent. The highest time use for the Bayesian modeler (agent type 3 in the table) is 0.0805, while the highest value for the frequency modeler (agent type 2) is 0.0887, but the lowest time use is 0.0177 for the Bayesian modeler and 0.0355 for the frequency modeler, meaning that the Bayesian modeler can be faster. The slowest agent is surprisingly the agent using no modeling (agent type 1) with the highest time used being 0.1106 and the lowest being 0.0280. This is surprising because the Bayesian modeler needs to do a large amount of extra calculations compared to the other agents. The fastest agent is clearly the full information agent, which is to be expected, as it already knows what the opponent wants and only needs to make an offer that satisfies this. The difference between this and the opponent modeling agents is small however, as its highest time use is 0.0716, only 0.0171 seconds faster than the frequency modelers fastest time. This means that the modeling imposes a very small overhead. A possible explanation for the slow offer making by the agent not using information could be that this agent needs to search through all possible items it can add to- or remove from the deal, while the modeling agents and the full information agents only need to choose the best constraint of the opponent and try to satisfy it, which is much faster as long as it is not a value constraint. As most modeled constraints are item constraints and not value constraints, the average performance of these agents is much lower as long as the modeling itself is fast.

Satisfaction

Table 7.7 shows the average satisfaction achieved over all scenarios for each combination of agents. The number to the left is the satisfaction achieved by the agent shown in the row, while the number to the right is the satisfaction achieved by the agent shown in the column. The results here are closely correlated to the results in the success rate, as every unsuccessful negotiation gives a satisfaction of zero to both players. The curious case of Bayesian versus full information (agent type 3 versus 4 in the table) giving poor results is therefore as evident here as it was in the success rate results, with player 1 getting a satisfaction of 0.562 and player 2 getting a satisfaction of 0.515. We see that the agents not using modeling get a much lower satisfaction when negotiating with each other than when negotiating against agents with modeling or full information, ending with an average satisfaction of 0.429 for player 1 and 0.42 for player 2. An important point to note here is that when negotiating against against the modeling agents (agents 2 and 3) and with the full information agent (agent 4), the agent not using modeling (agent 1) does not get poor results, but instead gets higher satisfaction levels than its opponents. This can be seen in the player 1 column in table 7.7, where player 1 gets the values 0.429, 0.705, 0.705 and 0.848, while its opponent gets the values 0.420, 0.658, 0.651 and 0.783. This result will be discussed further later in this section.

It is noticeable that player 1 generally does better than player 2 in our scenarios, where player 1 is the agent starting the negotiations, and player 2 is its opponent. This seems not to align with the optimal results discussed in section 7.2.1, where player 2's average satisfaction is higher than the average satisfaction of player 1. By closer examination of the scenarios however, we notice that in most of the scenarios, player 1 does better than player 2 in the optimal results. There is actually only two scenarios where player 1 is worse off than player 2. One of these scenarios, scenario 6, was shown to be very difficult for player 1 in the scenario discussion in section 6.2. This scenario makes such a big difference that it is the reason for player 1 ending up below player 2 in the optimal results. In our actual results however, this does not happen, possibly due to a poor opponent model. In the results player 1 actually performs consistently better than player 2 in this scenario (scores for each scenario are omitted in this report for readability, and only average values are shown in the tables). As the other scenarios we have made mostly favor player 1, this explains why the average values are higher for player 1 than for player 2 across all combinations of agents,even though it is different from the optimal result we have calculated. On closer examination of scenario 6 from section 6.2, we find that player 1's constraints are very easy to find, and all our modeling approaches will find them on the first attempt, while player 2's constraints are much harder and will often not be found correctly. This makes it easier for player 2 to concede to something player 1 wants, which ends with player 1 getting a higher satisfaction than player 2. In addition to this, there are only two issues to give that will satisfy these constraints, war with player 4 and war with player 6. One of these is conflicting with player 2's constraints, but not important enough to stop the agent from giving it. This means that in most cases in this scenario, player 2 will concede by offering war with P6, which is against its own constraints. This deal will usually be accepted, ending the negotiation with player 2 losing out on satisfaction compared to its opponent, which has not yet formed a good opponent model.

This result where the first agent that finds a good model concedes first and thereby gets a lower score, makes it seem like finding a good opponent model fast is bad for the agent. The idea is strengthened more by the fact that even when player 1 is an agent with no modeling, it does better than its opponent in our scenarios. This means that even though modeling significantly increases the success rate, it does not give an advantage over the opponent in terms of satisfaction. These results are slightly biased by the fact that in our tests, the agent without modeling (agent type 1) has consistently been tested as player 1, and as mentioned above, player 1 is in a better position than player 2 in most of our scenarios regardless of what types of agents are negotiating. We hypothesized that the agents with full information or opponent modeling would be able to outperform agent type 1, even when not being in the best position, but this does not seem to be the case. It therefore seems that agents do not utilize their information well enough to make it an advantage over less informed agents. This is a problem with our negotiation strategy, and not our opponent modeling techniques, and shows that in any further work, more emphasis should be put on the strategy.

In order to combat the problem, we should try to make the agents less willing to concede, more willing to explore the negotiation space rather than trying to appease its opponent, and possibly better at abusing the information it finds. Such improvements would be important steps in any future work in this domain. Using a more advanced negotiation strategy such as the well established Zeuthen strategy [5, pp.43] could significantly improve these results.

7.3 Overall Discussion

Overall, our experiments have shown that while there is still a lot of room for improvement, our system is capable of efficient multi issue negotiation in the Civilization IV domain without impairing the user experience, and both of our opponent modeling techniques show significantly improved results compared to negotiating without information. The perhaps biggest issue we have found with our approach, is that the agents are not capable of using their opponent models to gain an advantage over their opponents. The models make it much easier to make a deal. However, as we have shown in the discussion sections above, the agent with the best model often makes the most concessions, and therefore often loses out on utility in the end. If further work is to be done in this domain, making a smarter negotiating algorithm capable of utilizing the modeled information better would be key to achieving better results. Another big issue is the problem of agents sometimes giving up. This could be remedied by making better opponent models, but also by having a smarter algorithm for making offers that recognizes the fact that it might still be worth trying to make an offer when the opponent model is not good enough.

When it comes to the speed of the negotiation, which is of major importance in a video game, our algorithms have shown to be more than fast enough for the extra computing time to be unnoticeable. In fact, it seems that the opponent modeling agents often compute offers faster than agents not using opponent modeling, as having a model significantly speeds up the process of making an offer. In addition to this, negotiations require fewer rounds to find a suitable deal when using opponent modeling than without it.

Comparing our two opponent modeling techniques, we find that both methods have their strengths and weaknesses. The frequency modeling finds few false constraints, but struggles finding some specific constraint types, while the Bayesian is better at finding the different constraint types but also finds many false constraints as well as struggling to find value constraints. The higher number of false constraints does however, not seem to be a major disadvantage as these are generally irrelevant constraints that do not impact what will be offered. Due to the advantages and disadvantages of both modeling types, a possibility for future work in this area could be to combine these two techniques to overcome their weaknesses. Time usage seems to be an insignificant factor in this domain as both methods are more than fast enough. This means that going for a simpler technique such as frequency modeling to save processing time is not necessary. The time complexity may however, make more of a difference in future work if more complex computations are to be used for other parts of the negotiations or if the system is to be used in a more demanding real time system where response times need to be shorter.

Chapter 8

Conclusion

We have made a negotiation system specially adapted to the domain of Civilization IV that expands and improves the capabilities of the built-in AI.

The primary goal of the project was to explore the use of multi-issue negotiation systems with limited information in a complex, soft real-time system such as the video game Civilization IV. As stated in RQ1 our research aims to find whether such a system is feasible. The experiments show that the negotiation system performs adequately given the difficult domain, and runs fast enough to be feasible for use in real-time systems. The algorithms used computed fast enough so that the extra computing time did not become noticeable while playing.

An important feature of any negotiation system, and the focus of our second research question, RQ2, is whether it can guarantee Pareto optimality or maximization of social welfare. We have found that, in theory, using our methods with full information and exhaustive searching would guarantee Pareto optimality. However, due to the nature of the domain, we have limited information and even when we are using opponent models, we cannot guarantee their accuracy, thus we cannot guarantee Pareto optimality.

When it comes to maximizing social welfare, we have found that our system is not able to do this, even with full information. In our solution the agents always have a concession limit for the satisfaction level that the acceptable offers should have, and whenever an offer is above this limit, the AI accepts without considering whether there exists a better offer or not. This means that it might accept an offer that is not necessarily the best possible offer and the social welfare is therefore not maximized.

Our research has focused on the use of opponent modeling to improve the negotiation results, and thus, in our last research question, RQ3, we ask whether these techniques actually improve the results. Two opponent modeling methods have been evaluated, Bayesian modeling and frequency modeling. The experiments showed that both methods can improve the negotiation results significantly, but that they were far from flawless. As mentioned before, both algorithms are more than fast enough to be used in a real-time domain. In addition to this, negotiations using opponent modeling required fewer rounds before finding a suitable deal in our experiments compared to negotiations without opponent modeling. Our system does not, however, make use of the opponent model as much as it should. Having a model of the opponent should give the agent an advantage, but this is not the case, meaning our system does not utilize the information well enough.

Overall our project has shown that using these multi agent techniques for negotiation is feasible in Civilization IV, and the use of opponent modeling can significantly improve the negotiation results in this domain.

Chapter 9

Future Work

There are many possibilities for improving our current solution. For instance, the opponent modeling should be improved to find all types of constraint. In our current solution, neither the frequency modeling or the Bayesian modeling are able to find all the different types of constraints. The opponent modeling could also be further improved to find less false-positive constraints. Since both opponent modeling methods have their own strengths and weaknesses, a possibility here would be to combine the two to overcome their weaknesses. This would increase the complexity, which may result in higher computing times. Due to this, such a solution may not be suitable for real-time systems anymore.

The decision making mechanism in negotiations could also be improved by making it utilize the opponent model in a more efficient way. In our solution, having a very accurate opponent model actually makes the AI worse off because it ends up conceding more effectively than its opponent. This information should be used to give the AI an advantage. The decision making should also be improved so that it does not give up as easily as it does now. In order to improve the solution, a well known negotiation strategy such as the Zeuthen strategy could be used.

The decision making on when to negotiate and what to negotiate about is very simple in our solution. It currently utilizes the original AI's decision making, which does not think forward or reason about what it actually needs. For further improvements of the system, its decision making could be expanded to do such reasoning i.e. by using a planning system.

The representation of constraints can also be made more general and fuzzy, giving room for negotiations to find more different and unique offers and solutions.

Bibliography

[1] Michael Wooldridge. *An Introduction to MultiAgent Systems.* Wiley Publishing, 2nd edition, 2009. ISBN 0470519460, 9780470519462.

[2] Nicholas R. Jennings and Michael Wooldridge. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[3] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944. ISBN 0691119937.

[4] Martin J. Osborne. *An Introduction to Game Theory.* Oxford Univ. Press, 2004. ISBN 978-0-19-512895-6.

[5] Jeffrey S Rosenschein and Gilad Zlotkin. *Rules of encounter: designing conventions for automated negotiation among computers.* the MIT Press, 1994.

[6] S.J. Russell, P. Norvig, E. Davis, S.J. Russell, and S.J. Russell. *Artificial intelligence: a modern approach.* Prentice hall Upper Saddle River, NJ, 2010.

[7] Thomas M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[8] R. Walpole, R. Myers, S. Myers, and K. Ye. *Probability & Statistics For Engineers & Scientists, Eight Edition.* Pearson Education International, 2007. ISBN 0132047675.

[9] Reid Garfield Smith. *A framework for problem-solving in a distributed processing environment.* PhD thesis, Stanford, CA, USA, 1979. AAI7912412.

[10] Thomas W Malone, Richard E Fikes, and Michael T Howard. Enterprise: A market-like task scheduler for distributed computing environments. 1983.

[11] M. Beer, M. D'inverno, M. Luck, N. Jennings, C. Preist, and M. Schroeder. Negotiation in multi-agent systems. *The Knowledge Engineering Review*, 14(03):285–289, 1999.

[12] S. Kraus and D. Lehmann. Designing and building a negotiating automated agent. *Computational Intelligence*, 11(1):132–171, 2007.

[13] X. Luo, N.R. Jennings, N. Shadbolt, H. Leung, and J.H. Lee. A fuzzy constraint based model for bilateral, multi-issue negotiations in semi-competitive environments. *Artificial Intelligence*, 148(1):53–102, 2003.

[14] V. Tamma, S. Phelps, I. Dickinson, and M. Wooldridge. Ontologies for supporting negotiation in e-commerce. *Engineering applications of artificial intelligence*, 18(2):223–236, 2005.

[15] J. Shaheed. Creating a diplomat. *Master's Thesis, Imperial College, London, UK*, 2004.

[16] S. Saha. Improving agreements in multi-issue negotiation. *Journal of Electronic Commerce Research.(to appear)*, 2006.

[17] Shaheen Fatima, Michael Wooldridge, and Nicholas R. Jennings. 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. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[18] S.S. Fatima, M. Wooldridge, and N.R. Jennings. Approximate and online multi-issue negotiation. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 156. ACM, 2007.

[19] H. Raiffa. *The art and science of negotiation*. Belknap Press, 1982.

[20] D.G. Pruitt. *Negotiation behavior*, volume 47. Academic Press New York, 1981.

[21] Peyman Faratin, Carles Sierra, and Nick R. Jennings. Negotiation decision functions for autonomous agents. *INTERNATIONAL JOURNAL OF ROBOTICS AND AUTONOMOUS SYSTEMS*, 24:3–4, 1998.

[22] Jan Richter, Matthias Klusch, and Ryszard Kowalczyk. On monotonic mixed tactics and strategies for multi-issue negotiation. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1609–1610. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[23] K. Hindriks, C. Jonker, and D. Tykhonov. Analysis of negotiation dynamics. *Cooperative Information Agents XI*, pages 27–35, 2007.

[24] Michal Chalamish and Sarit Kraus. Automed: an automated mediator for multi-issue bilateral negotiations. *Autonomous Agents and Multi-Agent Systems*, 24(3):536–564, 2012.

[25] C.M. Jonker, V. Robu, and J. Treur. An agent architecture for multi-attribute negotiation using incomplete preference information. *Autonomous Agents and Multi-Agent Systems*, 15(2):221–252, 2007.

[26] Hamid Jazayeriy, Masrah Azmi-Murad, Nasir Sulaiman, and Nur Izura Udizir. The learning of an opponent's approximate preferences in bilateral automated negotiation. *Journal of theoretical and applied electronic commerce research*, 6(3):65–84, 2011.

[27] Tim Baarslag, Mark Hendrikx, Koen Hindriks, and Catholijn Jonker. Measuring the performance of online opponent models in automated bilateral negotiation. In *Proceedings of the 25th Australasian joint conference on Advances in Artificial Intelligence*, AI'12, pages 1–14, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-35100-6.

[28] Thijs van Krimpen, Daphne Looije, and Siamak Hajizadeh. Hardheaded. In *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pages 223–227. Springer, 2013.

[29] S Fatima, Michael Wooldridge, and Nicholas Jennings. Optimal negotiation strategies for agents with incomplete information. *Intelligent Agents VIII*, pages 377–392, 2002.

[30] K. Hindriks and D. Tykhonov. 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, pages 331–338, 2008.

[31] Dajun Zeng and Katia Sycara. Bayesian learning in negotiation. *International Journal of Human-Computer Studies*, 48(1):125–141, 1998.

[32] Robert M Coehoorn and Nicholas R Jennings. Learning on opponent's preferences to make effective multi-issue negotiation trade-offs. In *Proceedings of the 6th international conference on Electronic commerce*, pages 59–68. ACM, 2004.

[33] Y. Oshrat, R. Lin, and S. Kraus. Facing the challenge of human-agent negotiations via effective general opponent modeling. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 377–384. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[34] Mark Klein, Peyman Faratin, Hiroki Sayama, and Yaneer Bar-Yam. Negotiating complex contracts. *Group Decision and Negotiation*, 12(2):111–125, 2003.

[35] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *Systems, Man and Cybernetics, IEEE Transactions on*, 21 (3):660–674, 1991.

[36] Finn V Jensen. *An introduction to Bayesian networks*, volume 74. UCL press London, 1996.

[37] Sahibsingh A Dudani. The distance-weighted k-nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, (4):325–327, 1976.

[38] David B Leake. *Case-based reasoning.* John Wiley and Sons Ltd., 2003.

[39] G. Lai, C. Li, K. Sycara, and J. Giampapa. Literature review on multi-attribute negotiations. *Robotics Inst., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-RI-TR-04-66*, 2004.

[40] Raz Lin, Sarit Kraus, Tim Baarslag, Dmytro Tykhonov, Koen Hindriks, and Catholijn M Jonker. Genius: An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence*, 2012.