

# Introduction to Automated Negotiation

Dave de Jonge

IIIA-CSIC, Barcelona, Spain

July 1, 2025  
v0.2

© 2025 Dave de Jonge.

This work is licensed under CC BY-NC-ND 4.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/4.0/>

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Characteristics of Negotiation . . . . .	7
1.2	History of Automated Negotiation . . . . .	9
<b>2</b>	<b>Basic Negotiations</b>	<b>13</b>
2.1	Informal Description . . . . .	13
2.2	Formal Model . . . . .	15
2.2.1	The Offer Space . . . . .	15
2.2.2	The Alternating Offers Protocol . . . . .	17
2.2.3	Utility Functions . . . . .	21
2.2.4	Reservation Values . . . . .	26
2.2.5	Discount Factors . . . . .	27
2.2.6	Knowledge . . . . .	28
2.2.7	Negotiation Domains . . . . .	29
2.3	Pareto Optimality and Individual Rationality . . . . .	32
2.4	Competitiveness . . . . .	35
2.5	Simulation Framework . . . . .	38
<b>3</b>	<b>Negotiation Strategies</b>	<b>41</b>
3.1	The BOA Model . . . . .	42
3.2	Bidding Strategies . . . . .	44
3.2.1	Time-Based Strategies . . . . .	45
3.2.2	Adaptive Strategies . . . . .	53
3.2.3	Imitative Strategies . . . . .	55
3.3	Acceptance Strategies . . . . .	66
3.4	Reproposing . . . . .	72
<b>4</b>	<b>Opponent Modeling</b>	<b>77</b>
4.1	Learning the Opponent's Utility Function . . . . .	77

4.1.1	Bayesian Learning . . . . .	78
4.1.2	Scalable Bayesian Learning . . . . .	87
4.1.3	Frequency Analysis . . . . .	91
4.2	Learning the Opponent's Strategy . . . . .	92
4.2.1	Gaussian Processes . . . . .	93
4.2.2	Choosing the Optimal Target Value for an Adaptive Negotiation Strategy . . . . .	96
4.3	Learning the Opponent's Strategy from Previous Negotiation Sessions . . . . .	98
<b>5</b>	<b>Game Theory</b>	<b>99</b>
5.1	Normal-Form Games . . . . .	99
5.2	Extensive-Form Games . . . . .	99
5.3	Automated Negotiation as a Game . . . . .	99
5.4	Bargaining Solutions . . . . .	99
<b>6</b>	<b>Evaluation of Negotiation Algorithms</b>	<b>101</b>
<b>7</b>	<b>Advanced Negotiations</b>	<b>103</b>
7.1	Multilateral Negotiation . . . . .	103
7.2	Negotiation and Search . . . . .	103
7.3	Non-linear and Computationally Complex Utility Functions .	103

# Preface

This book is targeted towards computer science students who are completely new to the topic of automated negotiation. It does not require any prerequisite knowledge, except for elementary mathematics and basic programming skills. I have made this book available for free, so feel free to share it with anyone you like.

Please note that this book is meant as an organic document that keeps expanding over time. Therefore, I recommend to regularly check the website of this book to see if there is any updated version available.

This book comes with an simple toy-world negotiation framework implemented in Python that can be used by the readers to implement their own negotiation algorithms and perform experiments with them. This framework is small and simple enough that any reader who does not like to work in Python should be able to re-implement it very quickly in any other programming language of their choice. It can be downloaded from the website of this book:

[https://www.iiia.csic.es/~davedejonge/intro\\_to\\_nego](https://www.iiia.csic.es/~davedejonge/intro_to_nego)

If you have any questions or comments on this book, please send me an e-mail: [davedejonge@iiia.csic.es](mailto:davedejonge@iiia.csic.es). I am more than happy to hear your suggestions so that I can improve this work. Especially, if you feel that something is not clearly explained, or that something important is missing, please let me know!



# Chapter 1

## Introduction

### 1.1 Characteristics of Negotiation

Whenever we talk about ‘negotiation’ we are referring to any form of communication between multiple ‘agents’ (which can be either humans or software) with the goal of coordinating their actions, so that they can achieve a better outcome for themselves than what they could possibly achieve without such coordination.

A simple example is the scenario of a group of friends that want to go to the cinema together. In order to achieve that goal, they have to make a number of decisions together: which cinema to go to, which movie to watch, and at what time. If they do not manage to come to an agreement on all these decisions, then they will not be able to go to the cinema together. Clearly, coordination is essential to achieve the desired outcome.

In particular, we say that agents are negotiating whenever the following conditions are satisfied:

1. There is more than one agent.
2. These agents are able to communicate with each other.
3. The agents need to make one or more choices out of a number of options.
4. Each agent has its own individual preferences over the options.
5. Each agent is autonomous.

The need for the first three of these conditions should be obvious. The fourth assumption is essential, because if an agent does not have its own preferences, then it would not have any reason to participate in the negotiations. It could simply let all the other agents make the decision. Note

however, that this does not mean their preferences need to be *different*. For example, suppose two friends called Alice and Bob want to choose a movie to watch together. Even if they each want to see the same movie, they may still need to communicate this preference to one another in order to ensure that they are each *aware* of this fact. For example, Alice could propose to Bob to see *The Godfather*, and then Bob could accept that proposal. In other words, they still need a short negotiation, to establish their decision. The key point here, is that the two agents a priori do not know that their preferences are the same.

Nevertheless, in the rest of this book we will almost always assume that there is some amount of conflict among the agents. After all, a scenario in which all agents exactly agree on their preferences is not a very interesting test case for scientific research. A commonly used example of a scenario in which two negotiators have conflicting interests, is the case of a buyer and a seller that are negotiating the price of a car. In this case the agents' preferences are diametrically opposed: the seller wants to sell the car for the highest possible price, while the buyer wants to buy it for the lowest possible price. Despite their conflicting interests, the two agents still aim to find a compromise that is acceptable to each of them individually and that they each prefer over the situation that the car is not sold at all.

The fifth assumption means that each agent has at least some partial freedom to do whatever it wants. If one of the agents does not have any such freedom at all, then it would mean that that agent would essentially be a slave to the others and it would not have any negotiation power. For example, a car seller cannot force the buyer to buy the car. The buyer has the autonomy to refuse any offer he or she doesn't like. Similarly, the buyer cannot force the seller to sell the car either. The seller too has the autonomy to reject any offer from the buyer.

As a counter example, we can imagine a swarm of robots that are searching through the ruins of a collapsed building in order to find survivors. If these robots are fully controlled by a central computer, then there is no need for negotiation. The central computer simply dictates what all the robots should do.

It should be noted that there are many situations in daily life in which the above conditions hold, and therefore can be seen as a type of negotiation, even though we normally wouldn't think of them as a negotiation. In fact, any time two or more people make a joint decision, it is essentially a negotiation. So, whenever you ask someone a question like "shall we eat at 19:00?" or "Do you want to go to the cinema?" you are essentially starting a negotiation.



Another nice example of a negotiation scenario that we typically do not think of as a negotiation, is when you do your groceries at the supermarket. In this scenario there are indeed multiple agents, namely the customer and the supermarket. These two agents jointly aim to come to an agreement about which products the supermarket will sell to the user. Each of these agents has a certain amount of autonomy: the supermarket can choose which products it offers and for what price. The customer, on the other hand, can choose which of those products he or she will buy. Furthermore, each agent has their own preferences: the supermarket aims to make the highest possible financial profit, while the customer has preferences over which products he or she wants to buy, and prefers to buy them for the lowest possible price. The least obvious requirement, is perhaps the requirement of communication, as it might not be obvious at first sight that the two agents are indeed communicating. However, the supermarket is communicating to the customer by means of labels and price tags on their products. Every time the customer sees a label saying something like “1 kg of beef, \$6” this can be seen as a *proposal* made by the supermarket to the customer. The customer can then either *accept* that proposal by taking the product from the shelf and adding it to their shopping cart, or *reject* it by walking along without taking the product. This is, essentially, a form of negotiation. Of course, it is a somewhat limited form of negotiation since the supermarket is the only agent here that can make proposals, while the customer can only accept or reject those proposals, but cannot make any counter-proposals to the supermarket.

In the literature one sometimes distinguishes between *negotiation* and *bargaining*. The exact definitions differ per author, where ‘bargaining’ is often used exclusively to refer to the exchange of proposals that can be accepted or rejected, while ‘negotiation’ often refers to a more general process in which the agents may use a broader form of communication that allows them to express their respective interests, or allows them to convince the other agents to change their points of view. In the rest of this book, however, we will not distinguish between the two concepts and simply always use the term ‘negotiation’ even where some authors might argue that ‘bargaining’ would be the more appropriate term.

## 1.2 History of Automated Negotiation

Of course, in this book we are not just interested in negotiation, but rather in *automated* negotiation. That is, the study of how to develop computer

programs that can perform negotiations autonomously, either with other computer programs or with humans (although in this book we will focus mainly on negotiations between computers only).

The topic of automated negotiation dates back to the 1950's, starting with the work of John Nash [35]. Back in those days, however, automated negotiation was mainly studied from a purely theoretical point of view, rather than from an algorithmic point of view. The typical approach followed by Nash and other researchers of his time, would be to argue that the outcome of a certain negotiation scenario should satisfy a certain set of mathematical axioms. They would then formally prove that there exists a unique outcome satisfying those axioms. Several different solution concepts were proposed in this way, based on different sets of axioms [28, 25, 13].

This changed in 1998 with the seminal paper by Faratin et al. [22]. Rather than trying to find theoretically optimal outcomes, they took a more practical approach and proposed a number of possible negotiation strategies, which we will discuss in Chapter 3. This was a great step forwards towards realistic applications of automated negotiation, because it takes into account that real agents would typically would not have complete domain knowledge and would not be willing to share strategic information with each other.

Another pivotal event in the history of automated negotiation was the inception of the Automated Negotiating Agents Competition (ANAC) in 2010 [9] and the development of the Genius framework [30] on which ANAC was run. Since then, ANAC has been held almost every year at major A.I. conferences such as IJCAI and AAMAS and has greatly boosted the number of papers published on the topic of automated negotiation. Furthermore, ANAC has led to the development of hundreds of negotiating agents and a plethora of different opponent modeling techniques, which are still used by many researchers as a baseline against which they can test new negotiation algorithms.

Initially, most research on automated negotiation focused on the most basic type of negotiations with two agents negotiating over a small set of possible agreements with linear utility functions [9]. However, over the years, more and more researchers have started investigating more complex negotiation scenarios. For example, several researchers have studied negotiation domains with non-linear utility functions and with an extremely large number of possible agreements [27, 31]. This was even taken a step further by considering domains in which the calculation of the utility of just a single proposal is already computationally complex problem [19, 20, 18].

Other researchers have focused on multi-lateral negotiations (negotiations between 3 or more agents) [36, 21, 19, 4], or the use of machine learn-

ing algorithms such as deep learning and reinforcement learning to train negotiation algorithms [40, 10].

Most of these developments have also been closely mirrored by the various editions of ANAC. For example, ANAC 2014 involved negotiations with non-linear utility functions and extremely large search spaces [24], while from 2015 to 2018 ANAC focused on multi-lateral negotiations[23]. Then, in 2019 and 2020 the focus shifted back to small, bilateral negotiations, but in which each agent only had *partial* knowledge about its own utility function [3]. After that, several editions focused on the use of machine learning to allow the agents to learn the characteristics of their opponents, from earlier negotiations [37]. Furthermore, from 2017 onward the ANAC competition was divided into a ‘main league’ and one or more sub-leagues focusing on more specialized negotiation problems, such as high computational complexity in the game of Diplomacy [16], multi-lateral negotiations in a supply-chain environment [33] negotiations between computers and humans [32], and negotiations in the game of Werewolves [3].

For a long time, the Genius framework, which was written in Java, was the main platform that researchers used for their experiments in the field of automated negotiation. It was especially useful because it included a large set of hand-crafted test-domains that were used in the ANAC competitions and a large set of agents that participated in those competitions. This immediately gave researchers access to a vast library of benchmark test cases and baseline algorithms for their experiments.

However, it has recently been shown, both experimentally [14] and theoretically [15], that a very simple negotiation strategy called MiCRO is able to achieve near-optimal results on the Genius test domains even without using any form of machine learning or opponent modeling. It was therefore argued that those hand-crafted test cases should no longer be used.

The Genius framework is no longer maintained, and has now been superseded by the NegMas framework [34] as the main platform for research on automated negotiation. It is written in Python, but it still includes the possibility to run the Java agents from the Genius framework. Furthermore, it allows generating random test domains which are harder to tackle than the hand-crafted ones from Genius. Another framework, called GeniusWeb, was also developed by the makers of Genius, but this framework never gained much traction.



## Chapter 2

# Basic Negotiations

In this chapter we discuss the basic ideas of automated negotiation. For now we will focus mainly on **bilateral** negotiation. That is, negotiations between exactly two agents, as opposed to **multilateral** negotiation, which takes place between more than two agents. The only exception is that some of the mathematical definitions below will be given for arbitrary numbers of agents, because it would not simplify anything if we presented them for only two agents.

We here focus on bilateral negotiation because they are the simplest to explain, because they have been studied much more extensively in the literature and because they are sufficient to explain the most basic aspects of automated negotiation. We will discuss multilateral negotiations later on in Chapter 7.1.

### 2.1 Informal Description

Imagine there are two agents, which we will call the ‘buyer’ and the ‘seller’ respectively, that are negotiating the price of a second-hand car. The negotiations start with one agent proposing an offer to the other agent. For example, the seller might start by proposing a price of \$10,000. Next, the buyer can do two things: to accept the proposal, or to reject it. If the buyer accepts the proposal, then then it becomes a formally binding agreement and the negotiations are over. Otherwise, if she rejects the proposal, then she can make a counter-proposal. For example, she might propose a price of \$5,000. Next, it is again the seller’s turn. The seller now also has the choice between accepting the last proposal, or rejecting it and making a new proposal. For example, she could then propose a price of \$9,500. This will

continue until they come to an agreement, or one of the agents decides to withdraw from the negotiations, or a given deadline has passed, or when a fixed maximum number of proposals have been made.

In this example we assumed the agents negotiated according to the so-called **alternating offers protocol** (AOP) [38], meaning that the agents take turns making proposals. Specifically, it means that an agent is not allowed to make two proposals in a row. After making a proposal the agent first needs to wait for the other agent to respond and make a counter-proposal before she can make a new proposal herself. While this is certainly not the only protocol for automated negotiation, it does seem to be the one that is most commonly used in the literature.

In the field of automated negotiation we typically assume there is a fixed set of possible offers that the agents can propose to one another. This set is called the **offer space** (or sometimes **agreement space**). In the example of the car sale, the offer space consisted of every possible price that the seller could possibly ask, or that the buyer could possibly offer. So, this could be the set of all integers. One important thing to notice about this example, is that the agents were negotiating over just one issue: the price of the car. This is what we call a **single-issue** negotiation. In many cases in the literature, however, one studies **multi-issue negotiations**. That is, negotiations in which each proposal may involve multiple different components. For example, suppose there are two friends, Alice and Bob, that want to go to the cinema together. They need to agree on three different issues:

- Which movie they will see.
- Where they will see this movie (in which cinema).
- When they will see this movie (which day of the week and at which time).

One way to conduct such multi-issue negotiations would be to negotiate each issue separately, one by one. However, a more common approach in the literature is to just negotiate all issues at the same time. This means that each proposal indicates a value for all three issues at the same time. For example, Alice might start by proposing to see *The Godfather* in cinema *Rialto* on Friday at 20:00. Bob might then reject this proposal, and instead propose to see *Casablanca*, in cinema *Paradiso*, on Saturday at 18:00, etcetera.

We should remark that in this book we will use the term **offer** to refer to a potential outcome of a negotiation. That is, something that can be proposed or accepted or rejected. So, in the scenario of the car sale, the price of \$10,000 would be an example of an offer, while in the scenario of

the two friends who are going to the cinema, the tuple (*The Godfather*, Rialto, Fri 20:00) would be an example of an offer. Furthermore we will use the term **proposal** to refer to the *action* of proposing an offer. Finally, we use the term **agreement** to refer to an offer that has been accepted as the final outcome of the negotiation between the two agents. We should note however, that the literature is not very consistent on this matter. Other authors may use these terms in different ways, or they may use alternative terms such as **deal**, **contract**, or **bid** with their meanings being different for each author.

## 2.2 Formal Model

In order to be able to implement an agent that can negotiate, we first need to have a formalization of what ‘negotiation’ means exactly. We will here discuss this formal model. We assume there are exactly two **agents**, which we denote by  $a_1$  and  $a_2$  respectively.

### 2.2.1 The Offer Space

In order to implement a negotiating agent, the first thing we need to know is which offers the agents can possibly propose. This is known as the **offer space** or **agreement space** and is usually denoted by  $\Omega$ . In the example of a single-issue car sale, the set of possible offers was the set of all positive integers  $\mathbb{N}$ , where each number  $k \in \mathbb{N}$  represents a proposal to trade the car for a price of  $k$  dollars. A single offer from the offer space is usually denoted by  $\omega$ .

In the case of a multi-issue negotiation, the offer space can be written as the cartesian product of smaller sets that we call **issues**:

$$\Omega = I_1 \times I_2 \times \cdots \times I_m$$

so each offer  $\omega$  is a tuple:

$$\omega = (x_1, x_2, \dots, x_m)$$

where each  $x_j \in I_j$ . For each issue, we will refer to its elements as its **options**.

For example, the scenario in which two friends are planning to watch a movie together, can be modeled as a negotiation over the following three

issues, representing the movie, the cinema, and the time slot, respectively:

$$\begin{aligned} I_1 &= \{The\ Godfather, Casablanca, The\ Big\ Lebowski\} \\ I_2 &= \{Rialto, Paradiso\} \\ I_3 &= \{Fri\ 18:00, Fri\ 20:00, Fri\ 22:00, Sat\ 18:00, Sat\ 20:00, Sat\ 22:00\} \end{aligned}$$

We see that the issue ‘movie’ has 3 options, the issue ‘cinema’ has 2 options, and the issue ‘time slot’ has 6 options. So, the offer space contains  $3 \times 2 \times 6 = 36$  possible offers.

Note that issues may or may not have a natural ordering. For example, the issue  $I_3$  above, representing the time slot, is naturally ordered from early to late. On the other hand, the other two issues  $I_1$  and  $I_2$  do not have any ordering (of course, we could put them in any order we like, such as an alphabetical order, but that is not very meaningful for the negotiations).

Furthermore, note that the division of an offer space into separate issues can sometimes be a bit arbitrary. For example, rather than having one issue representing the time slot, we could have instead defined two separate issues: one issue for the day of the week, and one issue for the time. So, we could have defined the offer space as a product of the following 4 issues:

$$\begin{aligned} I_1 &= \{The\ Godfather, Casablanca, The\ Big\ Lebowski\} \\ I_2 &= \{Rialto, Paradiso\} \\ I_3 &= \{Fri, Sat\} \\ I_4 &= \{18:00, 20:00, 22:00\} \end{aligned}$$

This would not have made any difference. This also works in the other direction: if we wanted, we could have just ignored the separate issues altogether and model the entire domain as one single issue containing 36 different options, without any structure. However, as we will see in Section 2.2.3.3, decomposing the offer space into separate issues has the advantage that it allows us to define simple utility functions that are linear combinations of smaller functions that are each defined over a single issue.

Also note that in a real-world scenario there may exist constraints among the issues. For example, Rialto might only screen *The Godfather* on Saturdays, and Paradiso might not screen any movie at all on Friday at 18:00. So, in that case not *every* combination of options would be possible, and the offer space  $\Omega$  would only be a *subset* of the Cartesian product  $I_1 \times I_2 \times \dots \times I_m$ . However, in most of the literature such constraints are not taken into account and one therefore assumes that all possible combinations of options are allowed.



### 2.2.2 The Alternating Offers Protocol

The next thing we need to specify is the **negotiation protocol**. That is, the rules that determine when which agent is allowed to propose or accept which offer, and when a proposal will be considered a formally binding agreement.

The most commonly used protocol for *bilateral* negotiations, is the alternating offers protocol (AOP) which we have already seen above. In this protocol the agents take turns, so the protocol needs to specify which of the two agents will make the first proposal. In this section we will, without loss of generality, assume that this is always agent  $a_1$ .

At the start of the negotiations, agent  $a_1$  can choose any  $\omega \in \Omega$  from the offer space and propose it to  $a_2$ . Next it is agent  $a_2$ 's turn. Agent  $a_2$  can now either accept the previous proposal from  $a_1$ , or propose an alternative offer  $\omega' \in \Omega$ . If  $a_2$  accepts the previous offer  $\omega$  then the negotiations are over and  $\omega$  will be considered a formally binding agreement. Otherwise, if  $a_2$  does not accept  $\omega$  and instead makes a new proposal, then we say that  $a_2$  **rejects** the offer  $\omega$ . Next, it is again  $a_1$ 's turn. This time,  $a_1$  can choose between accepting the previously received proposal  $\omega'$ , or rejecting it and proposing a new offer  $\omega''$  from the offer space  $\Omega$ .

This continues until one of the following stopping criteria is satisfied:

1. A proposal is accepted.
2. A given temporal deadline  $T$  has passed.
3. A maximum number of rounds  $N$  have passed.

In the first case we say the negotiations have **succeeded**, while in the other two cases we say the negotiations have **failed**, meaning that the agents did not manage to come to any agreement. When we say that a ‘*round*’ has passed, we mean that an agent has proposed or accepted an offer. So, if  $N = 10$  it means that each agent can make at most 5 proposals (or 4 proposals and an acceptance).

We should remark here, that many authors assume there is only a temporal deadline, but no maximum number of rounds, or vice versa. However, if there is no temporal deadline then we can equivalently just say that  $T = \infty$ . Similarly, if there is no maximum number of rounds, then this is equivalent to saying that  $N = \infty$ . So, we can always say, without loss of generality, that there is a temporal deadline as well as a maximum number of rounds, as long as we allow these values to be infinite.

In the rest of this book we will use the notation  $(i, \mathbf{p}, \omega, t)$  to indicate that agent  $a_i$  proposes offer  $\omega$  at time  $t$ , and we will use the notation  $(i, \mathbf{a}, \omega, t)$  to indicate that agent  $a_i$  accepts offer  $\omega$  at time  $t$ . We follow the convention that  $t = 0$  represents the time at which the negotiations start.

**Definition 1.** We define a **negotiation action** to be a tuple

$$(i, \eta, \omega, t) \in \{1, 2\} \times \{\mathbf{p}, \mathbf{a}\} \times \Omega \times \mathbb{R}$$

where  $i$  represents the index of the agent performing the action, and  $\eta$  represents the **type** of the action, which can be either the symbol  $\mathbf{p}$  ('propose'), or the symbol  $\mathbf{a}$  ('accept'). Furthermore,  $\omega$  is the offer that is being proposed or accepted, and  $t$  is the time at which the agent proposes or accepts the offer. We define a **proposal** to be negotiation action for which  $\eta = \mathbf{p}$  and we define an **acceptance** as a negotiation action for which  $\eta = \mathbf{a}$ .

Some authors also include a third type of action, besides 'propose' and 'accept', which is called 'withdraw'. If an agent withdraws, it means that the agent chooses to end the negotiations immediately, without agreement. So, this also adds a fourth stopping criterion to the three that we mentioned above. However, since this type action does not play an important role in the rest of this book, we prefer not to include it here, so as to keep the formalization simple.

**Definition 2.** A **history**  $h$  is a list of negotiation actions, sorted in chronological order (i.e. in order of increasing values of  $t$ ).

A history is defined as a list of negotiation actions, which themselves are defined as 4-tuples, so a history is a list of 4-tuples. Furthermore, in the case of multi-issue negotiations, the offers  $\omega$  inside those tuples are themselves tuples as well. For example, a history with 10 negotiation actions could look as follows:

$$\begin{aligned} h &= \left( ac_1, ac_2, \dots, ac_9, ac_{10} \right) \\ &= \left( (1, \mathbf{p}, \omega_1, t_1), (2, \mathbf{p}, \omega_2, t_2), \dots, (1, \mathbf{p}, \omega_9, t_9), (2, \mathbf{a}, \omega_9, t_{10}) \right) \\ &= \left( (1, \mathbf{p}, (x_1^1, x_1^2, x_1^3), t_1), (2, \mathbf{p}, (x_2^1, x_2^2, x_2^3), t_2), \right. \\ &\quad \left. \dots, (1, \mathbf{p}, (x_9^1, x_9^2, x_9^3), t_9), (2, \mathbf{a}, (x_9^1, x_9^2, x_9^3), t_{10}) \right) \end{aligned}$$

where each  $ac_k$  is a negotiation action, and each  $x_k^j \in I_j$  is an option from the  $j$ -th issue in the  $k$ -th proposal. In this example we assumed that the domain has three issues. Note that in the 10-th action agent  $a_2$  accepts the offer  $\omega_9$  that was proposed by  $a_1$  directly before that.

In the rest of this book we may sometimes abuse notation and terminology and consider  $h$  to be a *set* of negotiation actions rather than a list, so we may write, for example,  $ac_k \in h$ .

We can now formally define the AOP as follows.

**Definition 3.** *We say a history  $h$  satisfies the AOP (with deadline  $T$  and maximum number of rounds  $N$ ) if and only if the following conditions all hold:*

1. *For any two different negotiation actions  $ac = (i, \eta, \omega, t)$  and  $ac' = (j, \eta', \omega', t')$  in  $h$  (so  $ac \neq ac'$ ), we have that  $t \neq t'$ .*
2. *For every pair of consecutive negotiation actions in  $h$ , denoted  $(i, \eta, \omega, t)$  and  $(j, \eta', \omega', t')$ , we have that  $i \neq j$ .*
3. *A negotiation action with  $\eta = \mathbf{a}$  can only appear as the last element of the history.*
4. *If  $(i, \eta, \omega, t)$  and  $(j, \eta', \omega', t')$  are the second-last and last elements of the history respectively and  $\eta' = \mathbf{a}$ , then we must have  $\omega = \omega'$ .*
5. *For all negotiation actions  $(i, \eta, \omega, t)$  in  $h$  we have  $t \leq T$ .*
6. *The length of  $h$  can be at most  $N$ .*

The first rule says that only one proposal or acceptance can be made at a time. The second rule says that the agents take alternating turns. The third rule says that the negotiations are over as soon as one agent accepts an offer. The fourth rule says that an agent can only accept the offer from the last proposal and not from any earlier proposals. The fifth rule says that the negotiations are over when the deadline  $T$  has passed, and the last rule says that the negotiations are over as soon as  $N$  negotiation actions have been made.

The alternating offers protocol is also displayed as a finite-state machine in Figure 2.1.

**Definition 4.** *Let  $h$  be a history that obeys the AOP and let  $(i_k, \eta_k, \omega_k, t_k)$  be its last element. Then we say that  $h$  ended with **agreement** if and only if  $\eta_k = \mathbf{a}$ . In that case we say that  $\omega_k$  is the **accepted offer**.*

### 2.2.2.1 Some Remarks

Some authors model the action of rejecting a proposal and the action of making a counter-proposal as two separate actions. However, since in the AOP a counter-proposal is always preceded by a rejection this is not really necessary. So, we will here follow the convention that the act of rejecting the previous proposal and the act of making a new proposal are modeled as one single action.

At first sight, it may seem a bit unrealistic to assume that there is a single deadline  $T$  which is imposed upon the two agents. After all, in a

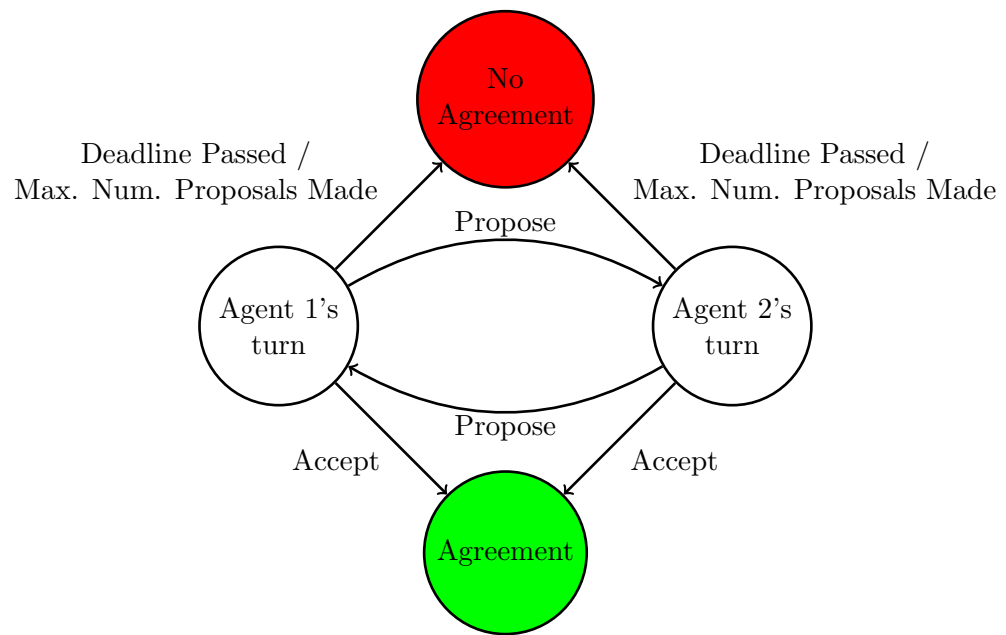


Figure 2.1: The alternating offers protocol as a finite-state machine.

real-world negotiation, who would impose a deadline onto the two agents? However, we can imagine that in a real scenario each agent  $a_i$  *itself* has its own individual deadline  $T_i$ , which is may be determined by various external factors. In that case, we can simply define the global deadline  $T$  as the individual deadline that comes the earliest. That is:  $T := \min\{T_1, T_2\}$ .

Arguably less realistic, is the assumption that the agents have a maximum number of proposals  $N$  that they can make. The main advantage of this assumption is that it makes it easier to analyze the negotiations using mathematical or game-theoretical techniques that require a fixed and commonly-known number of rounds. However, I personally can't really imagine any real-world situation in which two negotiators would only be allowed to make a certain number of proposals which is known to both negotiators in advance. The only similar scenario I can imagine, is that one of the agents is human and therefore would get tired after rejecting a certain number of proposals and give up. However, even in that case I don't think there would be a clearly fixed number  $N$  that is known by both agents in advance. Instead, I think it would be more realistic to model this with a random variable that assigns a probability  $P(N)$  to every possible value of  $N$ , to model the probability that the human would be too tired to continue after  $N$  rounds. Furthermore, one major disadvantage of including a maximum number of proposals, is that it implies an asymmetry between the two negotiators, since the agent that has the last turn will not be able to make any new proposals, and thus will be forced to either accept the last proposal or to end the negotiations without agreement.

Finally, we should remark that according to the definition of the AOP that we used here, an agent is only allowed to accept the *last* proposal it received from its opponent. That is, an agent is not allowed to accept any proposals that it received from the opponent in any of the *earlier* rounds. So, if an agent does not immediately accept a certain offer  $\omega$  proposed by the opponent, then the possibility of accepting that offer may be lost forever. While this may seem overly strict, in practice this rule is not much of a restriction because if the agent later changes its mind and does want to accept  $\omega$ , then he can simply propose that offer again itself. Since the opponent already proposed it earlier, there are good reasons to believe that the opponent will be willing to accept it.

### 2.2.3 Utility Functions

The negotiation protocol defines what the agents are *allowed* to do, but does not specify anything about how an agent would choose between its various

legal actions. That is, it does not specify the agents' *preferences*. Such preferences are typically modeled by means of *utility functions*. If we see negotiations as a game, and we see the negotiation protocol as the rules of the game, then the utility functions specify, for each agent, its *goal* in the game.

Clearly, each agent has its own preferences over the set of possible agreements. For example, in the case of a negotiation between a buyer and seller over the price of a car, the seller prefers to sell the car for the highest possible price, while the buyer prefers to sell the car for the lowest possible price. To model these preferences we assume that each agent has its own personal **utility function**  $u_i$ , which is a map from the set of offers to the set of real numbers:

$$u_i : \Omega \rightarrow \mathbb{R}$$

A higher utility function represents a more desired outcome. So, each agent aims to make an agreement for which his utility value is as high as possible. In the example of the car sale, the seller would have a utility function that strictly increases as a function of the price, while the buyer has a utility function that strictly decreases as a function of the price.

In the rest of this paper it will turn out useful to use the notation  $\omega_i^{max}$  for the offer most preferred by agent  $a_i$ , and the notation  $\omega_i^{min}$  for the offer least preferred by agent  $a_i$ :

$$\omega_i^{max} := \arg \max_{\omega \in \Omega} \{u_i(\omega)\} \quad (2.1)$$

$$\omega_i^{min} := \arg \min_{\omega \in \Omega} \{u_i(\omega)\} \quad (2.2)$$

### 2.2.3.1 Von Neumann-Morgenstern Utilities

When we only look at a single negotiation, the interpretation of the utility functions is clear: they represent the agents' respective preferences over the possible outcomes of that negotiation. However, you typically do not implement a negotiation algorithm to use it only one time and then throw it away. Ideally, it should be possible to use the same negotiation algorithm more than once. But then, how do we interpret the utility functions? After all, if we use the algorithm, say, five times, then it may make five different agreements. But how do we determine which combination of five agreements is the best?

While there are many possibilities, the most obvious and most commonly used interpretation is that the agent would prefer those outcomes that maximize the *sum* of their utility values (or equivalently: the *average*). That

is, if the algorithm is used  $n$  times, then the agent  $a_i$  aims to maximize  $\sum_{k=1}^n u_i(\omega_k)$ , where  $u_i$  is the utility function of the agent and  $\omega_k$  the agreement reached in the  $k^{th}$  negotiation. Utility functions that are interpreted in this way are called **von Neumann - Morgenstern utilities**. In the rest of this book we will always assume that utility functions are such von Neumann-Morgenstern utilities, unless specified otherwise.

One important aspect of von Neumann-Morgenstern utilities is that we can add any arbitrary constant to them or multiply them with any arbitrary positive constant, without changing the actual preferences. In other words, if  $a$  and  $b$  are two arbitrary real numbers (but with  $a > 0$ ) and  $u_i$  is the utility function of our agent, then it should not make any difference if we used the utility function  $a \cdot u_i + b$  instead of  $u_i$ . This, in turn, means that if the offer space  $\Omega$  is finite, then we can always normalize the utility function such that the offer with highest utility has utility value  $u_i(\omega_i^{max}) = 1$  and the offer with lowest utility has utility value  $u_i(\omega_i^{min}) = 0$ . We will call this a **normalized utility function**.

Note that if  $u_i$  is some arbitrary utility function, then it is easy to check that the utility function  $u'_i$  defined as follows is a *normalized* utility function.

$$u'_i := \frac{u_i - u_i(\omega_i^{min})}{u_i(\omega_i^{max}) - u_i(\omega_i^{min})}$$

Since any von Neumann-Morgenstern utility function over a finite offer space can be normalized, it is often assumed in the literature that the agents' utility functions are indeed normalized.

### 2.2.3.2 Self-interested Agents

In the rest of this book, we will assume that agents are always *purely self-interested* with respect to their utility functions. This means that each agent only aims to maximize its own utility function, and does not care at all if its opponents also receive high utility values.

Of course, the point of automated negotiation is that agents need to compromise. An agent that only makes proposals that yield high utility for itself and low utility for its opponent will never be able to come to an agreement and therefore only end up with low utility. So, in negotiations, even a purely-self interested agent still needs to take the other agents' preferences into account as well. However, the point is that when an agent makes a concession to its opponent, it does that not because it *wants* the opponent to receive more utility, but rather only because it *needs* to concede, to secure high utility for itself.

Now, this may *sound* like we are only trying to model very selfish and anti-social agents that do not care about each others' welfare. However, it is extremely important to understand that this is not the case. That is, '*self-interested*' does not mean the same as '*selfish*'.

For example, suppose that we have two agents  $a_1$  and  $a_2$  with respective utility functions  $u_1$  and  $u_2$ . Furthermore, suppose that agent  $a_1$  is a social agent that cares just as much about the opponent's utility as it cares about its own. So, it aims to maximize the sum  $u_1 + u_2$  of the two utility functions (this is also known as the *social welfare*). Now, note that we can simply define a new utility function  $u'_1$  for agent  $a_1$  as follows:

$$u'_1 := u_1 + u_2$$

We now see that, even though  $a_1$  is a very social agent, we can at the same time say that, *with respect to utility function  $u'_1$* , it is purely self-interested. In other words, the question whether or not an agent is self-interested depends entirely on how we define its utility function and has nothing to do with the question whether or not it is *selfish*.

### 2.2.3.3 Linear Utility Functions

In the case of multi-issue negotiations, one often assumes **linear utility functions**. We say a utility function is linear, if it is composed as a linear combination of several smaller functions, each one defined over one of the issues of the domain. That is:

$$u_i(\omega) = \sum_{j=1}^m v_i^j(x_j)$$

where:

$$\omega = (x_1, x_2, \dots, x_m) \in I_1 \times I_2 \times \dots \times I_m$$

and each  $v_i^j$  is a function that maps issue  $I_j$  to the real numbers:  $v_i^j : I_j \rightarrow \mathbb{R}$ . We will call these functions  $v_i^j$  the **evaluation functions**. The superscript  $j$  refers to the issue  $I_j$  for which it is defined, while the subscript  $i$  refers to the agent  $a_i$  to which it belongs.

Alternatively, linear utility functions are often written as:

$$u_i(\omega) = \sum_{j=1}^m w_i^j \cdot v_i^j(x_j) \tag{2.3}$$



where the  $w_i^j$  are the so-called **weights**, which typically sum to one:  $\sum_{j=1}^m w_i^j = 1$ . However, this expression is not fundamentally different from the expression without weights, as the weights can simply be ‘absorbed’ inside the evaluation functions  $v_i^j$ . That is, to re-write the second expression into the form of the first expression, we simply define  $v_i^{j'} := w_i^j \cdot v_i^j$ .

Nevertheless, the second expression is often preferred, because it allows to emphasize that an agent might consider some issues more important than other issues, by giving them a higher weight. Furthermore, in this form it is easier to define utility functions that are normalized, because all you need to do is choose the weights and evaluation functions such that the following conditions are met:

- All evaluation functions  $v_i^j$  are mapped into the interval  $[0, 1]$ .
- Each issue  $I_j$  has at least one option  $x_j \in I_j$  for which  $v_i^j(x_j) = 0$ .
- Each issue  $I_j$  has at least one option  $x_j \in I_j$  for which  $v_i^j(x_j) = 1$ .
- The weights sum to one:  $\sum_{j=1}^m w_i^j = 1$

Just be careful not to confuse the notation  $w$  for weights, with the notation  $\omega$  for offers.

One should realize, that when we say a utility function is linear, it only refers to the fact that it is a linear combination of evaluation functions  $v_i^j$ , while those evaluation functions themselves may still be non-linear. In fact, it often does not even make sense to ask if a certain evaluation function is linear or not, unless its options are numerical. For example, say that Alice’s preferences over which movie to watch are given as follows:

$$\begin{aligned} v_{Alice}^1(The\ Godfather) &= 0 \\ v_{Alice}^1(Casablanca) &= 1 \\ v_{Alice}^1(The\ Big\ Lebowski) &= 0.7 \end{aligned}$$

There is no way to tell if this function is linear or not. This is because the options of this issue (*The Godfather*, *Casablanca* and *The Big Lebowski*) are non-numerical. For the same reason it normally does not make sense to ask if a utility function is linear if that function is defined over an offer space that only consists of a single issue.

In the rest of this book, we will sometimes abuse notation and write  $v_i^j(\omega)$  when we actually mean  $v_i^j(x_j)$ , where  $x_j$  is the  $j$ -th component of  $\omega$ . That is:

$$v_i^j(x_1, x_2, \dots, x_j, \dots, x_m) \quad := \quad v_i^j(x_j)$$

### 2.2.4 Reservation Values

In many real negotiation scenario it may happen that some proposals are so bad that you would rather not to make any agreement at all than to accept any of them.

For example, in the example of a car sale, if the seller asks a ridiculously high price, then the buyer would prefer not to buy the car at all than to pay that price. This can be either because the buyer knows she can get a better deal elsewhere, or because she simply doesn't have that amount of money, or because she would prefer not to own a car at all, rather than to pay that much.

This means that a negotiating agent should not only be able to compare the various possible offers with each other, but should also be able to compare them with the situation that the negotiations end without agreement. For this, we define the *reservation value*.

**Definition 5.** *An agent's **reservation value** is the amount of utility it receives when the negotiations end without agreement.*

This definition implies that a rational agent would never accept any proposal that yields a utility value smaller than that agent's reservation value. After all, the agent *by definition* prefers to not make any agreement at all than to accept that proposal. Another way to look at it, is to say that the reservation value  $rv_i$  is the minimum amount of utility that the agent  $a_i$  is guaranteed to get. After all,  $a_i$  can always choose to withdraw from the negotiations, or to reject any proposals it receives. Therefore, a rational agent would only propose or accept any offer that offer yields more utility than that its reservation value.

Here is another example. Suppose two friends, Alice and Bob, want to go out for dinner together and they are discussing where to go. They have three options: a Chinese restaurant, an Italian restaurant, or a Mexican restaurant. Let us denote this as follows:

$$\Omega = \{CHI, ITA, MEX\}.$$

Unfortunately, they have different preferences, so they will have to find a compromise. If they can't agree about where they will eat, then they will each just have to stay home and eat alone. Let's suppose that Alice assigns the following utility values to the options:

$$u_{Alice}(CHI) = 1, \quad u_{Alice}(ITA) = 4, \quad u_{Alice}(MEX) = 5$$

and that her reservation value is 3, which we denote as:

$$rv_{Alice} = 3$$

The fact that she assigns the lowest utility to Chinese food means that this is her least preferred option. In fact, the utility she assigns to Chinese food is even lower than her reservation value. This means that she dislikes Chinese food so much, that she would prefer to just eat alone at home than to eat Chinese food with Bob. Furthermore, we see that she prefers Mexican food over Italian food. However, the utility she assigns to Italian food is still higher than her reservation value, which means that she still prefers to eat Italian food with Bob, than to stay at home.

The situation that the negotiations end without agreement is often called the **conflict outcome**, or **disagreement**.

### 2.2.5 Discount Factors

In the literature, many authors have studied models of negotiation in which the utility obtained by the agents does not only depend on the agreement they make, but also on the time at which they make that agreement. That is, the faster they make the agreement, the higher their respective utilities. This is typically modeled by introducing so-called **discount factors**. In a negotiation with discount factors, when the agents come to an agreement  $\omega$  each agent receives a **discounted utility**  $u_i(\omega, t)$  defined as:

$$u_i(\omega, t) := u_i(\omega) \cdot \delta^t$$

where  $\delta \in (0, 1]$  is called the discount factor,  $t$  is the time at which the agents come to an agreement and the function  $u_i$  on the right-hand side is the ordinary utility function as defined previously, which in this context is also referred to as the **undiscounted utility**. Note that since  $\delta$  is between 0 and 1, the discounted utility decreases over time. Furthermore, note that if  $\delta = 1$  then the discounted utility is just the same as the undiscounted utility, so this is equivalent to saying that there is no discount factor at all.

Furthermore, when studying negotiations with discount factors, it is sometimes also assumed that the reservation values are discounted as well. This means that if one of the two agents decides to withdraw from the negotiations at time  $t$ , then each agent  $a_i$  receives its respective **discounted reservation value**  $rv_i \cdot \delta_i^t$ . In that case it may indeed be beneficial for an agent to withdraw from the negotiations early, if it seems unlikely that they will come to a good deal. This is why some authors include a ‘withdraw’ action in the AOP, as we briefly discussed in Section 2.2.2.

I personally feel that the presence of discount factors is a somewhat unrealistic assumption. It seems to me that most researchers only make this assumption in order to obtain more interesting results, rather than because it yields a realistic model of negotiation.

For example, Rubinstein [39] used discount factors because it enabled him to find a mathematically optimal solution for certain negotiation scenarios. More generally, the advantage of discount factors is that they force the agents to concede quicker. After all, without discount factors an agent could simply refuse to make any concessions until very close to the deadline.

Some people might argue that discount factors could be used to model a human's impatience. However, that argument of course only holds in the case that you are modeling negotiations with humans. Furthermore, I don't think it is very obvious that a human's impatience is indeed accurately modeled by an exponentially decreasing discount factor.

Another argument that some people might use in favor of discount factors, is that they can model the fact that certain goods such as fish or flowers are perishable, so their value quickly decreases over time. However, I don't think that that is a strong argument, since the typical time scale for the decay of such products is several days, which is much longer than the time span of a typical negotiation involving such products, which might take place in a matter of seconds, or at most minutes.

### 2.2.6 Knowledge

The final ingredient that is still missing before we can fully specify a negotiation scenario, is the question how much knowledge each agent has about the other agents' utility functions, reservation values and discount factors (if present).

Authors that mainly focus on the theoretical aspects of negotiation, often assume full knowledge about the utility functions and reservation values because it is typically much harder to derive formal mathematical results under partial knowledge.

On the other hand, authors that focus more on algorithms and experiments often assume that each agent only knows its own utility function and reservation value, while it does not know anything about its opponent's utility function or reservation value, except maybe that the opponent's utility function is linear. Furthermore, they may sometimes assume that some of the issues are ordered, and that each agent knows, for each such issue, whether the opponent's preference over the options of that issue are increasing or decreasing w.r.t the ordering (e.g. Alice knows that Bob prefers to

go to the cinema as late as possible).

Of course, for many commercial applications it would be unrealistic to assume the agents know each other's utility functions. After all, each agent would aim to exploit the other one as much as possible and would therefore try to hide its utility function. Nevertheless, theoretical research that does assume full knowledge is still very valuable, since it allows us to determine a theoretical 'upper bound' to what an agent could hypothetically achieve in the ideal case of full knowledge (for example, *the Nash bargaining solution* [35] which we will discuss later on in this book). This, in turn, allows us to quantify how well practical algorithms are able to approach that upper bound [15].

Furthermore, one can argue that the assumption of having no knowledge about the opponent's utility at all, is also unrealistic. For example, a car dealer knows that some cars are more valuable than other cars and understands that the customer's preference is largely determined by his budget. I would therefore argue that in many negotiation scenarios the most realistic model lies somewhere in between. A real negotiator would not know the *exact* utility function of its opponent, but would have at least some background knowledge about the negotiation domain, from which it could make some basic assumption about the opponent's preferences. Another good example of this, is given in [17] and [18] in which two logistics companies negotiate the exchange of truck loads. Their utility functions depend on expenses like fuel price and truck driver salaries. While neither company knows exactly how much the other company pays for fuel and salaries, they do know that these prices cannot be radically different between the two companies. So, they can each make an educated guess about the opponent's utility function.

### 2.2.7 Negotiation Domains

**Definition 6.** A *negotiation domain*  $\mathcal{D}$  for  $n$  agents consists of the following components:

- An offer space  $\Omega$ .
- For each agent  $a_i \in \{a_1, a_2, \dots, a_n\}$ :
  - a utility function  $u_i : \Omega \rightarrow \mathbb{R}$
  - a reservation value  $rv_i \in \mathbb{R}$
  - a discount factor  $\delta_i \in (0, 1]$

A negotiation domain with two agents (i.e.  $n = 2$ ) is called a **bilateral negotiation domain** and a negotiation domain with more than two agents

(i.e.  $n > 2$ ) is called a **multilateral negotiation domain**.

**Definition 7.** *In a negotiation domain for  $n$  agents, each offer  $\omega$  corresponds to an  $n$ -tuple which we call the **utility vector** and which consists of the utility values of all agents:*

$$(u_1(\omega), u_2(\omega), \dots, u_n(\omega))$$

It is often instructive (in the case of bilateral negotiations) to plot the utility vectors of a given negotiation domain in a diagram such as in Figure 2.2. We will call this a **utility space diagram** or simply a **utility diagram**. In such diagrams, each black dot represents one offer. For example, if an offer  $\omega$  yields utility values  $u_1(\omega) = 0.3$  and  $u_2(\omega) = 0.6$  for the two agents respectively, then that offer is represented by a black dot with coordinates  $(0.3, 0.6)$ . Furthermore, in such diagrams we may draw the reservation values of the agents with a horizontal line and a vertical line respectively. For example, if agent  $a_1$  has a reservation value of  $rv_1 = 0.1$ , then we draw a vertical line at  $x = 0.1$  and if agent  $a_2$  has a reservation value of  $rv_2 = 0.2$ , then we draw a horizontal line at  $y = 0.2$ .

Whenever we refer to such diagrams we may use somewhat sloppy language and use the term ‘offer’ or the symbol  $\omega$  when we technically mean the *utility vector* of that offer.

Of course, it is important to remember that we often assume that neither of the two agents knows the utility function of the other and therefore neither of the two agents would be able to draw such a diagram. In other words, such diagrams are typically only meaningful to you, as the researcher, but not to the agents themselves.

A bilateral negotiation domain is called a **split-the-pie** domain if it satisfies  $\forall \omega \in \Omega : u_1(\omega) + u_2(\omega) = 1$ . It is called this way, because it is as if the two agents are negotiating about how to divide a pie among them. The size of the pie is 1, and each agent’s utility is proportional to the size of the pie she gets. So, if  $a_1$  gets, say, 40% of the pie then her utility is 0.4 and therefore  $a_2$  gets 60% of the pie, corresponding to a utility of 0.6. Another example of split-the-pie domain is the scenario of the seller and the buyer that are negotiating the price of a car. A utility diagram of a split-the-pie domain is displayed in Figure 2.3.

### 2.2.7.1 Single-Issue Domains vs. Multi-Issue Domains

It is sometimes argued that multi-issue negotiations are more complex than single-issue negotiations, because they involve making trade-offs between the various different issues. However, this is somewhat misleading.

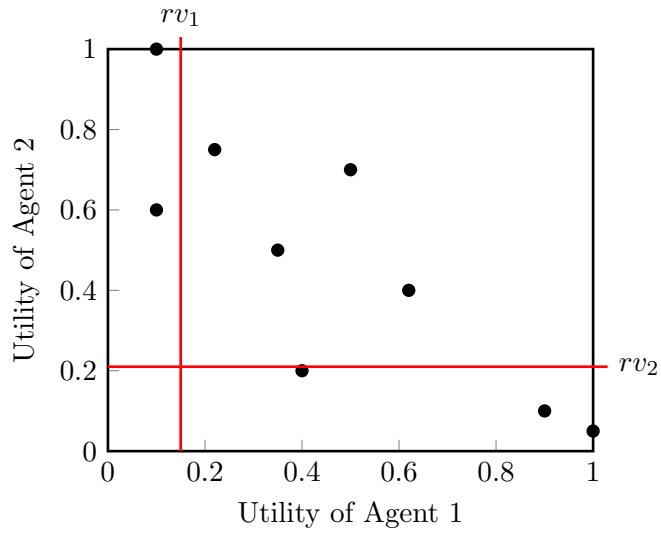


Figure 2.2: Utility space diagram. Every dot is the utility vector of one offer  $\omega$  in the offer space  $\Omega$ . The red lines represent the reservation values of the two respective agents.

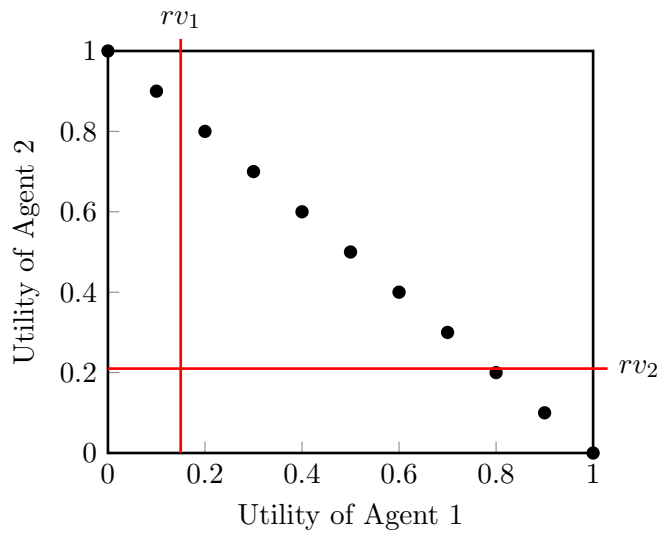


Figure 2.3: Utility space diagram of a split-the-pie domain. Note that all utility vectors lie on the line  $y = 1 - x$ .

Of course, if you compare a single-issue domain  $\mathcal{D}_1$  that contains 10 different offers, with a multi-issue domains  $\mathcal{D}_2$  that contains 3 issues with 10 options each, then indeed a negotiation over the multi-issue domain will be more complex because it involves  $10^3 = 1,000$  offers in total. However, this is not because there are multiple issues, but rather because the domain simply contains more offers.

In fact, if we compare domain  $\mathcal{D}_2$  with a single-issue domain  $\mathcal{D}_3$  of the same size (i.e. with 1,000 offers), then I would even say that the single-issue domain  $\mathcal{D}_3$  is more complex, especially if the utility functions of  $\mathcal{D}_2$  are linear. After all, in that case, to describe the utility functions of  $\mathcal{D}_2$  we only need 33 parameters (the three weights, plus 10 numbers for each issue  $I_j$  to represent the values  $v_i^j(x_j)$ ). On the other hand, to describe the utility functions in the single-issue domain  $\mathcal{D}_3$  we need 1,000 parameters: one for each offer. As we will see later on in Chapter 4, this means that for many opponent modeling algorithms it is much easier to learn the opponent's utility function in the multi-issue domain. In fact, many existing opponent modeling algorithms would not even work on single-issue domains.

I therefore argue that *if a single-issue domain and a multi-issue domain each have the same size, then, in general, the single-issue domain would typically be more complex than the multi-issue domain.*

One exception to this rule, however, would be if we assume that all issues are ordered and that we know, for each issue, the opponent's preference ordering over that issue. In that case a single-issue domain would be easier to handle.

## 2.3 Pareto Optimality and Individual Rationality

In this section we discuss two important properties that any agreement between two agents should ideally satisfy: *individual rationality*, and *Pareto optimality*.

As mentioned before, a rational agent would never accept an offer that yields a utility value lower than its reservation value. This motivates the definition of individual rationality.

**Definition 8.** *In any negotiation domain an offer  $\omega$  is said to be **rational for agent**  $a_i$  if that agent's utility for that offer is strictly greater than that agent's reservation value:*

$$u_i(\omega) > rv_i$$

*Furthermore, we say an offer  $\omega$  is **individually rational** if it is rational*



for all agents:

$$\forall i \in \{1, 2, \dots, n\} : u_i(\omega) > rv_i$$

You may find this terminology a bit confusing, since *individual* rationality actually refers to *all* agents, but this is an established term in the literature.

The importance of individual rationality, is that in a bilateral negotiation only the individually rational offers could ever become an agreement. After all, if an offer is not individually rational, then at least one of the two agents would never accept or propose it (unless, of course, the agent is very badly programmed).

In a multilateral negotiation, on the other hand, this depends on the details of the protocol. If the protocol prescribes that *all* agents need to agree with an offer for it to become an agreement, then again we have that only individually rational offers can become agreements. However, there are scenarios and protocols in which it is possible for *subsets* of agents to make agreements. In such cases, of course, an agreement only needs to be rational for that subset of agents.

The set of individually rational offers can be visualized easily in a utility diagram, since it is the set of all offers that lie above the horizontal line representing  $rv_2$ , as well as to the right of the vertical line representing  $rv_1$ . See Figure 2.4.

Before we can define the concept of Pareto optimality, we first have to define the concept of *domination*. Suppose that we have two offers,  $\omega$  and  $\omega'$ , such that each agent prefers  $\omega$  over  $\omega'$ . We then say that  $\omega$  *dominates*  $\omega'$ , or that  $\omega'$  is *dominated* by  $\omega$ . We can give a precise definition as follows.

**Definition 9.** We say that an offer  $\omega$  **dominates** another offer  $\omega'$  if:

$$\forall i \in \{1, 2, \dots, n\} : u_i(\omega) \geq u_i(\omega')$$

and there is at least one agent for which this inequality is strict:

$$\exists i \in \{1, 2, \dots, n\} : u_i(\omega) > u_i(\omega')$$

We say an offer  $\omega'$  **is dominated** by  $\omega$ , if  $\omega$  dominates  $\omega'$ .

In a utility diagram, this can be visualized as follows: first, draw a vertical line through the point representing  $\omega'$ , next, draw a horizontal line through  $\omega'$ . Now, if  $\omega$  lies on or above the horizontal line, and also lies on or to the right of the vertical line, then  $\omega$  dominates  $\omega'$ . See Figure 2.5.

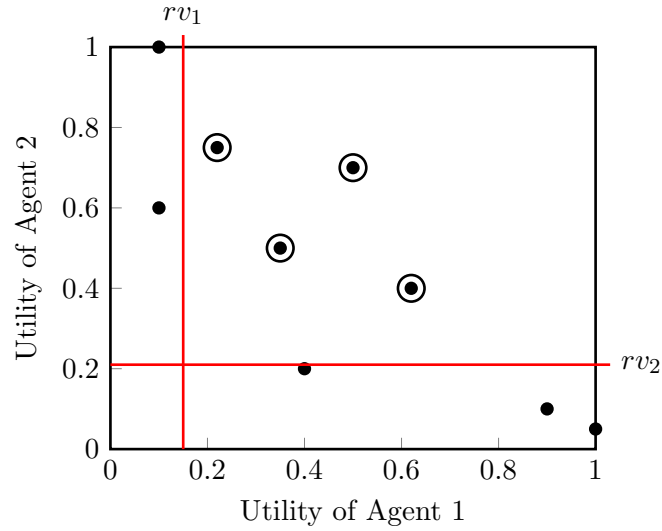


Figure 2.4: The individually rational offers are those for which their utility vector lies above the horizontal line representing  $rv_2$  and to the right of the vertical line representing  $rv_1$ . Here these utility vectors are all drawn with a circle around them.

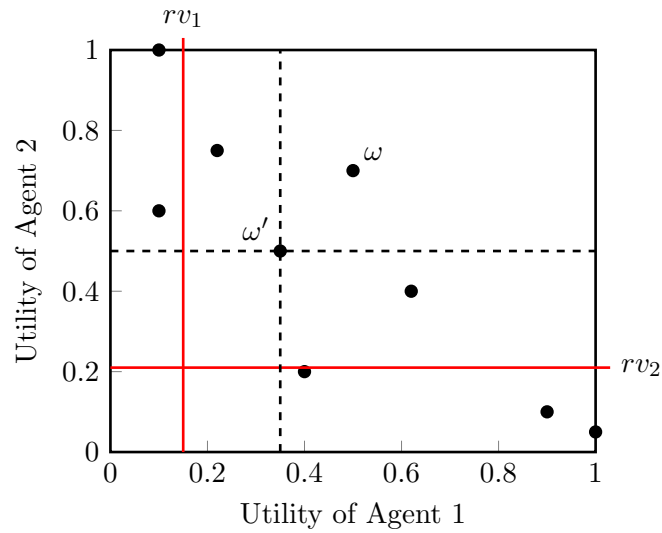


Figure 2.5: Example of domination. The offer  $\omega$  lies to the top-right of  $\omega'$  and we therefore say that  $\omega$  dominates  $\omega'$ .

Clearly, if the agents agree upon an offer  $\omega'$  that is dominated by some other offer  $\omega$ , then this outcome would not be optimal, since at least one agent would actually prefer  $\omega$  as the final agreement and none of the other agents would have any objection against  $\omega$  instead of  $\omega'$ . So, ideally, agents would only agree upon offers that are not dominated by any other offer. Such offers are called *Pareto-optimal*.

**Definition 10.** *An offer  $\omega$  is **Pareto optimal** if it is not dominated by any other offer.*

However, unlike individual rationality, Pareto optimality is hard to guarantee in practice, if the agents don't know each other's utility functions. So, many negotiation algorithms still often make deals that are not Pareto optimal.

To visualize Pareto optimality, again draw a horizontal line and a vertical line through a given offer  $\omega$ . The lines divide the space into four quarters. If the top-right quarter (including the lines themselves) is empty, then  $\omega$  is Pareto optimal. See Figure 2.6.

**Definition 11.** *For any negotiation domain  $\mathcal{D}$ , its **Pareto set**  $\Omega^p$  is the set of all Pareto-optimal offers. The **Pareto frontier** is the set of all utility vectors of the Pareto-optimal offers.*

Note that the Pareto set is a subset of  $\Omega$ , while the Pareto frontier is a subset of  $\mathbb{R}^n$ . See Figure 2.7 for the visualization of a Pareto frontier.

## 2.4 Competitiveness

In some negotiation domains it is easier to find good offers that are acceptable to all agents than in other domains. For example, if the domain contains a single offer  $\omega^*$  that yields the maximum utility to all agents (i.e.  $\omega^* = \omega_1^{max} = \omega_2^{max}$ ), then it is obvious that that specific offer should be the one that the agents agree upon. After all, no agent would benefit from switching to any other agreement. The interests of all agents are aligned and therefore we say the domain has zero *competitiveness* or *opposition* (we will use these two terms interchangeably).

On the other hand, in a split-the-pie domain there is high opposition, because the interests of the two agents are diametrically opposed. The better an offer is for one agent, the worse it is for the other. In fact, we can construct even more competitive domains where there is no good intermediate solution and every offer is really bad for at least one agent of the agents. See Figure 2.8.

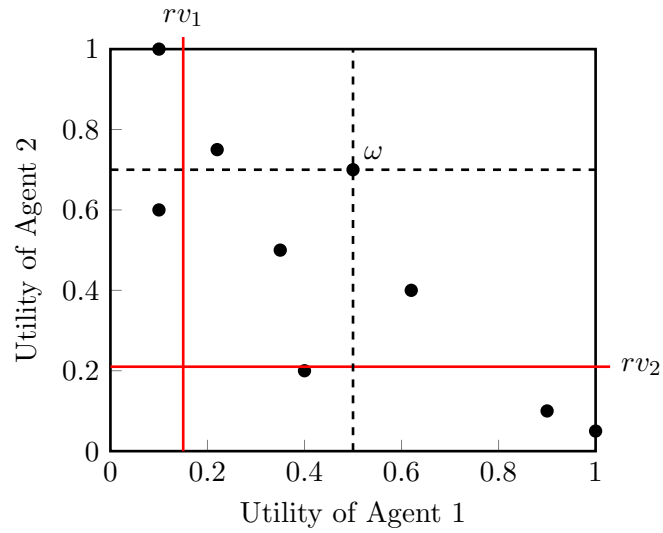


Figure 2.6: The offer  $\omega$  is Pareto optimal because it is not dominated by any other offer. We can see this because the area that lies above the horizontal dashed line and to the right of the vertical dashed line is empty.

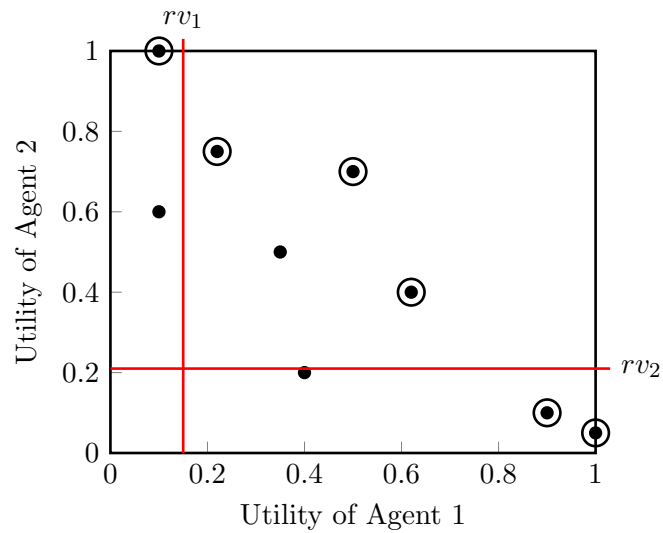


Figure 2.7: Pareto-frontier. All offers that are Pareto-optimal have been drawn here with a circle around them.

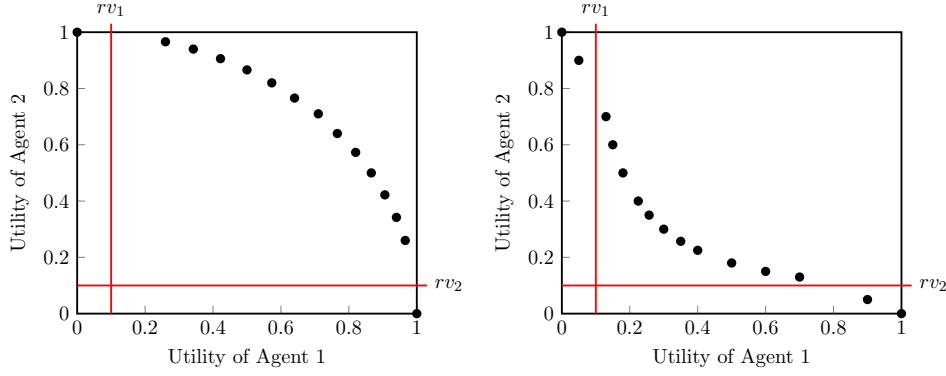


Figure 2.8: Left: a domain with low opposition. Right: a domain with high opposition.

In other words, the ‘competitiveness’ or ‘opposition’ of a domain measures how easy it is for all agents to receive high utility. Now, it would be nice to have a formula that allows us to quantify, for any given domain its competitiveness. It turns out, however, that many different such formulas have been proposed in the literature, so we will discuss a couple of them. For simplicity, we will assume the utility functions are normalized. Each of the expressions we discuss here is based on the idea that we first pick some ‘ideal’ offer, and then measure the difference between the utility vector of that ideal offer and the ‘utopian’ utility vector  $(1, 1, \dots, 1)$  that assigns the maximum utility to each agent. The higher this value, the higher the opposition of the domain.

Perhaps the most commonly used definition of opposition is one based on the Euclidean distance [46]. That is:

$$opp(\mathcal{D}) \quad := \quad \min_{\omega \in \Omega} \sqrt{\sum_{i=1}^n (1 - u_i(\omega))^2} \quad (2.4)$$

While this definition may initially seem intuitive, one could argue that it is not entirely satisfactory. For example, suppose that the minimum Euclidean distance is attained for some offer with utility vector  $(0.6, 0.6)$ . Now, it is easy to see that if we change the domain a bit, by replacing this offer with a new offer with utility vector  $(0.6, 0.65)$ , then according to this Euclidean measure, the domain would become less competitive, even though we have only increased the utility of *one* of the two agents. Moreover, we can even slightly decrease the utility of the other agent, to get  $(0.59, 0.65)$

and the Euclidean opposition measure would still indicate that this domain is less competitive than the original one. One could argue that this result is somewhat contrary to what you might expect from an accurate measure of opposition.

One alternative definition is the following [1]:

$$\text{opp}(\mathcal{D}) := \min_{\omega \in \Omega} 1 - \min_{i \in \{1, 2, \dots, n\}} u_i(\omega) \quad (2.5)$$

Here, the distance to the ‘utopian’ outcome is defined as the difference between 1 and the utility obtained by the agent that receives lowest utility. The advantage of this measure is that to decrease the competitiveness of a domain, we need to increase the utility of *all* agents.

Yet another definition [37] also uses the Euclidean distance, but defines the ‘ideal offer’ as the one that minimizes  $|u_1(\omega) - u_2(\omega)|$  among all Pareto optimal offers. That is:

$$\text{opp}(\mathcal{D}) := \sqrt{\sum_{i=1}^n (1 - u_i(\omega^*))^2} \quad (2.6)$$

where:

$$\omega^* := \min_{\omega \in \Omega^p} |u_1(\omega) - u_2(\omega)| \quad (2.7)$$

In the end, there is no obvious way to determine which of these measures is the ‘best’. I would say that this question mainly depends on the purpose that you have in mind for which you want to measure opposition.

## 2.5 Simulation Framework

In order to implement negotiation algorithms and perform experiments on them, we need a framework that allows us to run a simulation of a negotiation between agents. A commonly used framework for this is the NegMas platform [34].

However, for this book we have implemented a very simple, toy-world negotiation simulator in Python. It can be downloaded from the web page of this book:

[https://www.iiia.csic.es/~davedejonge/intro\\_to\\_nego](https://www.iiia.csic.es/~davedejonge/intro_to_nego)

It does not rely on any libraries so you don’t need to install anything, except of course Python itself, and any development environment that is suitable for Python. We will use this simulator for various exercises throughout this book.

**Exercise 1.** Download the python code of the NegoSimulator and run the file negoSimulator.py. This will run a simulation of a negotiation between two agents that just make random proposals.





## Chapter 3

# Negotiation Strategies

We are now finally ready to discuss how we can actually implement a negotiation algorithm. This is probably the most important chapter of this book. We will describe several possible strategies and we will see that each of them has its own advantages and disadvantages.

The goal of this chapter is to discuss how we can develop our own agent, that will be able to negotiate with arbitrary unknown opponents. We will here always follow the convention that our agent is denoted as  $a_1$ , while its opponent is denoted as  $a_2$ .

It is important to understand that the only goal of our agent is always to maximize its own utility, so it does not care about other concepts such as fairness or social welfare, as explained in Section 2.2.3.2, and we assume the same for the opponent.

There are many kinds of negotiation scenarios that we could consider, but in this chapter we will always make the following assumptions:

- Negotiations are bilateral (so our agent is negotiating with only one opponent).
- Negotiations take place according to the alternating offers protocol (See Section 2.2.2).
- Each of the two agents involved in the negotiation knows its *own* utility function and its own reservation value, but neither of them knows the utility function or reservation value of the other.
- The offer space  $\Omega$  is finite.
- The agents have a finite deadline  $T$  for the negotiations.
- There is no maximum number of negotiation rounds (or equivalently,  $N = \infty$ ).
- There are no discount factors (or equivalently, the discount factors are equal to 1).

On the other hand, we will not make any assumptions about whether the negotiation domain is a single-issue or multi-issue domain, nor about the type of utility functions the agents have (linear or non-linear).

We make these assumptions because they yield the simplest types of negotiation scenarios that are still interesting enough to allow us to discuss the most commonly used negotiation strategies. More advanced negotiation scenarios will be discussed later on in this book.

### 3.1 The BOA Model

When implementing a negotiation algorithm, it is often useful to think of it as consisting of three separate components:

- A **Bidding strategy**: a strategy to determine when our agent will propose which offer to the opponent.
- An **Opponent modeling algorithm**: an algorithm that allows our agent to approximately learn the opponent's utility function and/or its bidding strategy.
- An **Acceptance strategy**: A strategy to determine which proposals received from the opponent should be accepted by our agent and which ones should be rejected.

This model is known as the BOA model [6]. A typical BOA agent would be implemented as follows:

1. Receive an offer  $\omega_{rec}$  proposed by the opponent.
2. Use the opponent modeling algorithm to update a model of the opponent's strategy and utility function, based on the received proposal.
3. Use the bidding strategy, in combination with the model of the opponent, to determine which counter offer  $\omega_{next}$  to propose next.
4. Use the acceptance strategy to determine whether or not to accept the received offer  $\omega_{rec}$ . If yes, then accept  $\omega_{rec}$ , if not, then propose  $\omega_{next}$ .

An implementation in pseudo-code is displayed in Algorithm 1. In the following sections we will present more specific strategies, but they all follow the same structure. One thing that may seem counter-intuitive, is that this algorithm first decides which offer to propose next, before it decides whether or not to accept the received offer. This is, because the decision whether or not to accept the received proposal often depends on which proposal you are going to make next.

In the following section we will discuss various bidding strategies and present some example implementations in pseudo-code. These examples will

---

**Algorithm 1** BOA Agent for the Alternating Offers protocol. Generic implementation of a method that is called every turn and determines whether the agent should accept the last proposal received from the opponent or reject it and, in case of rejection, which counter-offer to propose next.

---

**Input:**

$\Omega$	▷ The offer space.
$u_1$	▷ The agent's own utility function.
$rv_1$	▷ The agent's own reservation value.
$T$	▷ The deadline.
$\mathcal{M}$	▷ A model of the opponent.
$t$	▷ The current time.
$h$	▷ The history: a list containing all proposals that have so far been proposed by both agents, sorted in chronological order.
$\omega_{rec}$	▷ The offer last proposed by the opponent (if any). Note that it is also contained in the history $h$ , but for clarity we also display it here separately.

// OPPONENT MODELING

// First, update the opponent model

1:  $\mathcal{M} \leftarrow \text{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$ 

// BIDDING STRATEGY

// Next, apply a bidding strategy to select the next offer to propose.

2:  $\omega_{next} \leftarrow \text{biddingStrategy}(\Omega, u_1, rv_1, T, \mathcal{M}, t, h)$ 

// ACCEPTANCE STRATEGY

// Then, determine whether or not to accept the opponent's last

// proposal. We store this decision in a boolean variable *acceptOffer*.3:  $\text{acceptOffer} \leftarrow \text{acceptanceStrategy}(\Omega, u_1, T, \mathcal{M}, t, \omega_{rec}, \omega_{next})$ 

// RETURN SELECTED ACTION

// Finally, return the selected action (accept or propose).

4: **if** *acceptOffer* **then**5:     RETURN (**a**,  $\omega_{rec}$ )6: **else**7:     RETURN (**p**,  $\omega_{next}$ )8: **end if**


---

also include various acceptance strategies, but we will not discuss them yet because we defer that discussion until Section 3.3. Furthermore, opponent modeling algorithms will be discussed in Chapter 4.

## 3.2 Bidding Strategies

In this section we will discuss the various negotiation strategies that have been studied in the literature. These strategies can be classified into the following three categories:

1. Time-based strategies.
2. Adaptive strategies.
3. Imitative strategies.

We certainly do not claim that these are the only possible strategies, but they are the most commonly studied ones. In fact, in their seminal paper [22] Faratin et al. also proposed a fourth type of strategy, known as a *resource-based* strategy, but these seem to have been given considerably less attention in the literature, so we will not discuss them in this book.

The basic idea behind all three types of strategy above is the same: our agent starts by proposing an offer that gives the highest possible utility to itself but, as time passes, our agent will propose offers that yield less and less utility to itself, which will hopefully make it more likely for the opponent to accept one of those offers. Every time an agent makes a new proposal that yields less utility to itself than any of its previous proposals, we say the agent is making a **concession**, or that the agent is **conceding**.

The big question is how to determine *how much* to concede in every turn. On the one hand, our agent obviously should not concede too much, because its aim is to make a deal that gives itself the highest possible utility. An agent that concedes too much will only make deals that yield very little utility. But on the other hand, if our agent doesn't concede enough, there is the risk that it may not come to any agreement at all, which would often result in even less utility. Therefore, the key to a strong negotiation strategy is to make exactly the right trade-off between conceding enough, and not conceding too much. In the rest of this book we will refer to a strategy that concedes very little as a **hardheaded strategy**, while we will refer to a strategy that concedes very much as a **conceding strategy**.

### 3.2.1 Time-Based Strategies

Time-based strategies are the simplest kind of negotiation strategy. A time-based strategy makes use of a function  $\lambda : \mathbb{R} \rightarrow \mathbb{R}$ , known as the **aspiration function**, which would typically be strictly decreasing. This aspiration function controls the amount of concession the agent makes as a function of time. Specifically, the idea is that at any given time  $t$  our agent  $a_1$  will propose an offer  $\omega$  that concedes as much as possible, under the constraint that his utility value  $u_1(\omega)$  must remain greater than, or equal to  $\lambda(t)$ .

Time-based agents can be either hardheaded or conceding, depending on the shape of the aspiration function. The faster  $\lambda$  decreases, the more conceding the agent will be. We will discuss this in more detail below.

#### 3.2.1.1 Choosing the Next Offer to Propose

Given an aspiration function  $\lambda$ , we need to implement a precise rule how to choose the next offer to propose  $\omega_{next}$  based on this function. One example would be to do it according to the following expression:

$$\omega_{next} = \arg \max_{\omega \in \Omega} \{ \hat{u}_2(\omega) \mid u_1(\omega) \geq \lambda(t) \wedge \omega \notin \Omega_t^{prop} \} \quad (3.1)$$

where  $\hat{u}_2$  is an estimation that our agent  $a_1$  makes of the opponent's utility function  $u_2$ , by means of its opponent modeling algorithm. The details about how such opponent modeling techniques work will be discussed in Chapter 4. For now, we will just see it as a 'black box' that magically gives us an approximation of the opponent's utility function. Furthermore,  $\Omega_t^{prop}$  is the set of all offers that have already been proposed by  $a_1$  before time  $t$ .

$$\Omega_t^{prop} := \{ \omega \in \Omega \mid \exists t' \in [0, t] : (1, \mathbf{p}, \omega, t') \in h \} \quad (3.2)$$

In Equation (3.1) we can clearly see how  $\lambda(t)$  controls the trade-off between demanding a high utility for yourself and conceding more utility to the opponent. On the one hand our agent is maximizing the opponent's estimated utility  $\hat{u}_2$ , but on the other hand this is restricted by the constraint that our agent should not propose any offer that yield less utility than  $\lambda(t)$ .

The constraint  $\omega \notin \Omega_t^{prop}$  ensures that, if the best candidate has already been proposed, then instead of repeating that proposal, our agent will propose the second best candidate. After all, the opponent modeling algorithm may not be accurate, so even if  $\hat{u}_2(\omega)$  is greater than  $\hat{u}_2(\omega')$  it may happen that the opponent actually prefers  $\omega'$ , so, if it has the chance, our agent should also try to propose  $\omega'$ .

Of course, it may happen that there is no offer at all that satisfies the criteria, because all offers for which  $u_1(\omega) \geq \lambda(t)$  holds have already been proposed. In that case our agent can simply repeat the same proposal as in the last turn, or propose an arbitrary one that it has already proposed before.

The main disadvantage of Eq. (3.1), however, is that it depends on having an accurate opponent modeling algorithm. Therefore, alternatively, one can instead use the following expression.

$$\omega_{next} = \arg \min_{\omega \in \Omega} \{ u_1(\omega) \mid u_1(\omega) \geq \lambda(t) \wedge \omega \notin \Omega_t^{prop} \} \quad (3.3)$$

That is, it picks the offer with the *lowest* utility value that is still greater than or equal to  $\lambda(t)$ . There are two scenarios in which this alternative approach would make sense:

1. In domains where the utility functions of the two agents are strongly negatively correlated (that is, domains in which any offer that yields high utility to our agent, yields low utility to the opponent, and vice versa).
2. In domains with a very small offer space.

An example of the first scenario is the case where a buyer and a seller negotiate the price of a car, or any other split-the-pie domain. In such cases, finding the offer that yields the highest utility to the opponent is (approximately) equivalent to finding the offer that yields the lowest utility to our agent. So, Eq. (3.3) would yield approximately the same proposals as Eq. (3.1), but without using any opponent modeling algorithm. Of course, the problem is that we have to *know* that the utility functions are strongly correlated, so we need to have at least some prior knowledge about the opponent's utility function.

In the second scenario Eq. (3.3) may work, because there is enough time for our agent to propose *all* the offers, one by one. For example, if it takes about 100 milliseconds for an agent to make a proposal, and the deadline is set to 1 minute, then there is time to propose 6,000 different offers. So, if the offer space contains less than 6,000 different offers, then there is enough time for the two agents to propose all offers. In that case this approach may work even when there is no strong correlation between the utility functions, because it simply doesn't matter if our agent sometimes proposes offers that are bad for the opponent. If there is a better offer available, then our agent will simply propose that offer in any of the following turns. On the other hand, if the domain is too large (or the deadline too short), then this

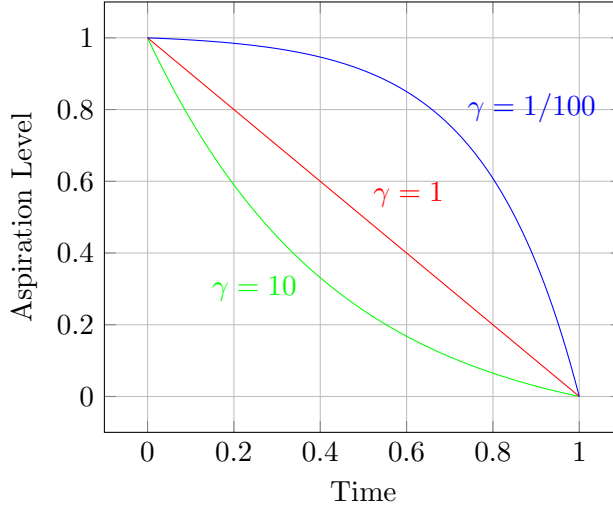


Figure 3.1: Aspiration functions with  $\alpha = 1$ ,  $\beta = 0$ ,  $T = 1$ , and several different values for  $\gamma$ .

approach may fail because our agent cannot propose all offers, and therefore risks failing to propose those offers that are acceptable to the opponent.

### 3.2.1.2 Choosing the Aspiration Function

The aspiration function can be any monotonically decreasing function, but a good example would be the following:

$$\lambda(t) = (\alpha - \beta) \cdot \frac{1 - \gamma^{1 - \frac{t}{T}}}{1 - \gamma} + \beta \quad (3.4)$$

where  $T$  is the deadline of the negotiations, and  $\alpha$ ,  $\beta$  and  $\gamma$  are three parameters that can be freely chosen, but with  $\alpha > \beta$  and  $\gamma > 0$ . We have plotted this expression in Figure 3.1 for various different values of  $\gamma$ . An example implementation of a time-based agent is displayed in Algorithm 2.

Let us now discuss how to interpret the parameters  $\alpha$ ,  $\beta$ , and  $\gamma$ , and how to choose their values. For this, first note that if  $t = 0$  then we have  $\lambda(0) = \alpha$ . Therefore,  $\alpha$  represents the minimum utility our agent will demand for itself at the start of the negotiations. Similarly, if  $t = T$  then we have  $\lambda(t) = \beta$ . This means that  $\beta$  represents the utility our agent will demand for itself at the end of the negotiations, when the deadline is near. We will call this the **target value**. A high target value represents a hardheaded strategy, while

---

**Algorithm 2** Time-based bidding Strategy.

---

**Parameters:**  $\alpha, \beta, \gamma$

**Input:**

$\Omega$	▷ The offer space.
$u_1$	▷ The agent's own utility function.
$T$	▷ The deadline.
$t$	▷ The current time.
$h$	▷ The history of all proposals made so far.
$\omega_{rec}$	▷ The offer last proposed by the opponent (if any).

// OPPONENT MODELING

1:  $\mathcal{M} \leftarrow \text{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$   
 2:  $\hat{u}_2 \leftarrow \text{getEstimatedOpponentUtility}(\mathcal{M})$

// BIDDING STRATEGY

// Calculate the aspiration level.

3:  $\lambda \leftarrow (\alpha - \beta) \cdot \frac{1 - \gamma^{1-t/T}}{1 - \gamma} + \beta$   
 // Obtain the set of offers we have already proposed.  
 4:  $\Omega^{prop} \leftarrow \text{getOffersProposedByUs}(h)$   
 // Find the next offer to propose.  
 5:  $\omega_{next} \leftarrow \arg \max_{\omega \in \Omega} \{\hat{u}_2(\omega) \mid u_1(\omega) \geq \lambda \wedge \omega \notin \Omega^{prop}\}$

// ACCEPTANCE STRATEGY

// Get the last proposal received from the opponent, and accept it if  
 // it yields more utility to us than our aspiration level.

6:  $\text{acceptOffer} \leftarrow u(\omega_{rec}) \geq \lambda$

// RETURN SELECTED ACTION

7: **if**  $\text{acceptOffer}$  **then**  
 8:     RETURN ( $\mathbf{a}, \omega_{rec}$ )   //accept the received offer  
 9: **else**  
 10:     RETURN ( $\mathbf{p}, \omega_{next}$ )   //propose a new offer  
 11: **end if**

---



a lower target value represents a conceding strategy. Finally, the parameter  $\gamma$  determines how quickly the agent concedes from  $\alpha$  to  $\beta$ .

Typically, the value chosen for  $\alpha$  is exactly the utility of the offer that the agent prefers most:  $\alpha = u_1(\omega_1^{max})$ . After all, a typical negotiator would start with the proposal that yields the highest utility for itself. While it is certainly possible to start with a lower offer, there does not seem to be much reason to do so. So, the other two parameters are more important.

Regarding the value for  $\beta$ , it should be obvious that it should always be greater than the agent's reservation value, because our agent should never propose any offer that yields less utility than that. One common choice is to set  $\beta$  *exactly* equal to the reservation value. The reasoning behind this is that making a deal that is just slightly above the reservation value is always better than making no deal at all, and thus one should be willing to concede all the way to the reservation value as the deadline gets close. While this reasoning may make sense if we focus only on one single negotiation in isolation, this choice is actually not optimal at all if we consider that our agent may be involved in many different negotiations and that our opponents may remember our agent's behavior from previous encounters and may be learning how to negotiate optimally against our agent.

The problem is this: if our agent always chooses  $\beta = rv_1$ , then its opponents may anticipate this. That is, the opponents know that our agent will be conceding all the way to its reservation value and therefore they can exploit it by simply not conceding at all, or very little, and waiting until the very last moment before accepting any of our agent's proposals.

For example, consider a split-the-pie domain where the maximum utility is 1, and our reservation value is 0. If our agent plays a strategy with  $\beta = 0$  and the opponent chooses a strategy with  $\beta = 0.99$  then all negotiations would end with an agreement that gives our agent a utility of 0.01 and the opponent 0.99 (assuming such an offer exists).

It is therefore often wiser to choose a higher target value (i.e. choose a more hardheaded strategy). This may sometimes cause the negotiations to fail, but in the long run that may actually be a good thing, because it sends a signal to our opponents that they will need to make concessions if they want to make an agreement with our agent.

On the other hand, choosing the target value too high will not work well either. It could work against a very conceding time-based agent (i.e. one with a low target value), but it will fail to come to an agreement if the opponent also chooses a high target value. For example, if both agents choose a target value of 0.99 (when the maximum utility is 1), then they can only come to an agreement if there exists an offer that yields a utility

of 0.99 to both agents. It is rare to encounter a negotiation domain where this is the case.

Figure 3.2 visualizes the evolution of the aspiration levels of two time-based agents during a negotiation. The aspiration level of  $a_1$  is indicated with a vertical blue line that over time moves from the right to the left, while the aspiration level of  $a_2$  is indicated with a horizontal blue line that over time moves from the top to the bottom. Note that in this example,  $a_1$  follows a conceding strategy, while  $a_2$  follows a hardheaded strategy. We see that they end up with an agreement that yields more utility to the hardheaded agent than to the conceding agent.

The parameter  $\gamma$  is the **concession parameter**. It determines how fast our agent will concede towards its target value. If  $\gamma$  is very small (e.g. 0.01) our agent will initially concede very slowly, as we can see in Figure 3.1, and only start making large concessions towards the end of the negotiations. On the other hand, if  $\gamma$  is very large, our agent will immediately start making large concessions. Finally, a value of  $\gamma = 1$  represents an agent that concedes linearly.<sup>1</sup>

In order to exploit the opponent as much as possible, our agent should make sure it concedes slower than the opponent. This suggests that we would always want a low value of  $\gamma$ . However, if we choose  $\gamma$  too low, then our agent may start conceding so late, that by the time it finally makes a substantial concession there is no more time for the opponent to accept it.

For example, suppose we choose an intermediate target value of  $\beta = 0.5$ , but our concession parameter is so low, that at 10 milliseconds before the deadline the aspiration value is still at  $\lambda(t) = 0.90$ . While in theory the aspiration level will continue to decrease to 0.5 in the last 10 milliseconds, this time might not be enough for our agent to actually exchange more proposals and come to an agreement. After all, every time our agent makes a proposal, it will take a small amount of time for that message to arrive at the opponent and then the opponent will still need some time to process it, and to send an ‘accept’ message back. This means that the optimal value of  $\gamma$  largely depends on the speed at which the agents can send messages and at which they are able to process them. In other words, it largely depends on practical considerations related to the infrastructure on which the agents are implemented.

Furthermore, if we choose  $\gamma$  very low, then our agent’s aspiration level will remain very high for a long time, which means that for a long time there

---

<sup>1</sup>Technically, the expression in Eq. 3.4 is not defined for  $\gamma = 1$ , but it can be shown that  $\lim_{\gamma \rightarrow 1} f(t) = (\alpha - \beta) \cdot (1 - t/T) + \beta$ , which is a linear function of  $t$ .

might not be any agreement possible. Then, when the deadline gets near, our agent will suddenly concede very fast towards its target value, meaning that the only possible agreement would be one close to the target value. But in that way we might miss out on any opportunities to obtain a better deal. Our agent would only be able to make a deal near its target value, or no deal at all. By choosing a somewhat higher value of  $\gamma$  our agent has the time to propose several intermediate offers that yield utilities of, say, 0.8, 0.7 and 0.6, which could be accepted by the opponent before our agent reaches its target level.

Another reason why a low value of  $\gamma$  might not be optimal is when there are discount factors (see Section 2.2.5), because in that case we would prefer our agent to come to an agreement as quickly as possible. Yet another example could be in the case that the opponent is participating in multiple negotiations in parallel. For example, when a seller has one item to sell, and is negotiating with multiple potential buyers at the same time. In that case our agent, as a buyer, would also want to come to an agreement as soon as possible, before the seller sells the item to one of the other buyers.

Time-based strategies with a low value of  $\gamma$ , but with  $\beta = rv_1$  are also known as **Boulware strategies**.

Finally, it should be noted that Eq. (3.4) is sometimes adapted so that the agent reaches its target level already a bit *before* the deadline, at a time  $T'$  slightly less than  $T$ , which we will call the **target time**. After the target time, the aspiration level will just remain constant:

$$\lambda(t) = \begin{cases} (\alpha - \beta) \cdot \frac{1 - \gamma^{1-t/T'}}{1 - \gamma} + \beta & \text{if } t \in [0, T'] \\ \beta & \text{if } t \in [T', T] \end{cases} \quad (3.5)$$

This is to ensure that our agent will indeed concede all the way to its target level, but not any further. Furthermore, it ensures that after  $a_1$  proposes its ultimate offer (with utility equal to or very close to  $\beta$ ) at time  $T'$  so that there is enough time left for the opponent to accept that offer.

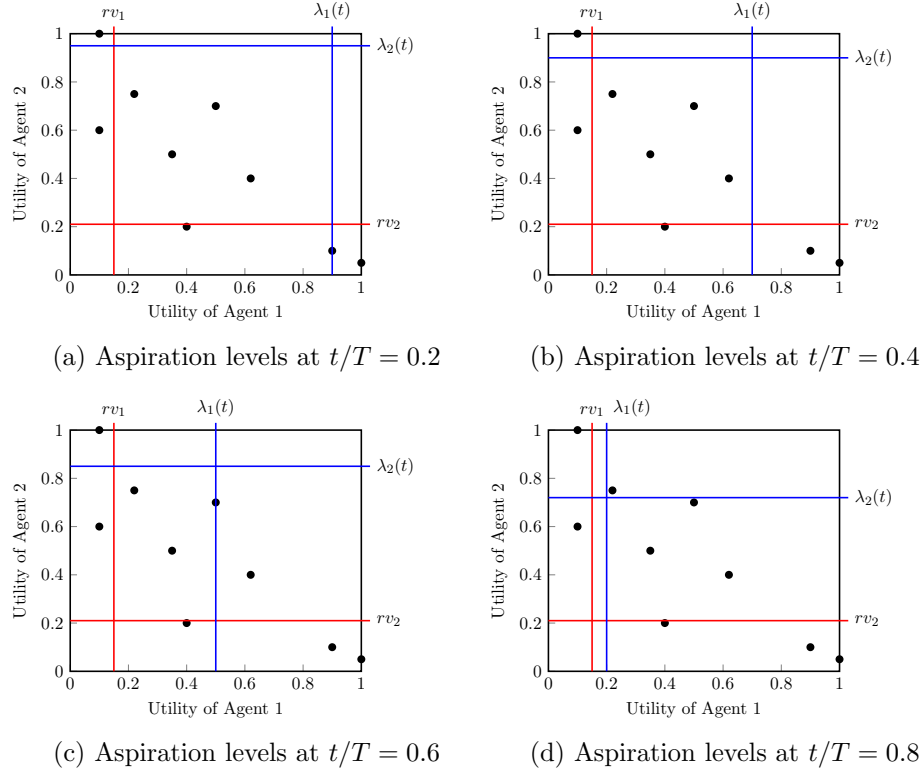


Figure 3.2: Negotiation between a conceding agent ( $a_1$ ) and a hardheaded agent ( $a_2$ ). Their aspiration levels are indicated with a vertical blue line and a horizontal blue line respectively. We see that the aspiration level of the conceding agent drops much further than the aspiration level of the hardheaded agent. The negotiations continue until they reach a point at which there is an offer that is acceptable to both agents. That is, when there is an offer for which its utility vector lies above the horizontal blue line, as well as to the right of the vertical blue line. In this example that happens at  $t/T = 0.8$ . Note that the agreement yields more utility to the hardheaded agent than to the conceding agent.

**Exercise 2. Time-based Agent.** Use the NegoSimulator framework (Section 2.5) to implement an agent that applies a time-based negotiation strategy. Note that the framework already comes with the source code of a RandomAgent, so you can just copy its code and adapt it according to Algorithm 2.

Since we haven't discussed opponent modeling algorithms yet, you can use Equation (3.3) to determine the next offer, which doesn't require opponent modeling.

Alternatively, you can use the DummyOpponentUtilityModel that comes with the framework. This is a fake opponent model that takes the opponent's real utility function as its input and returns a random approximation of that function.

Run several negotiations between time-based agents and experiment with different parameter settings. Which values for the parameters  $\alpha$ ,  $\beta$  and  $\gamma$  give the best results?

### 3.2.2 Adaptive Strategies

We will now describe another type of strategy, known as an **adaptive strategy**. Adaptive strategies have probably received the most attention in the literature, and most agents that were successful in the various ANAC competition have been of this type.

In order to explain this type of strategy, let us first suppose that our opponent  $a_2$  plays a time-based strategy with target value  $\beta_2$ . This means that, if we wait long enough, the opponent will be willing to propose or accept any offer  $\omega$  for which  $u_2(\omega) \geq \beta_2$ . Now, let  $\omega^*$  denote the offer that maximizes  $a_1$ 's own utility  $u_1$  among those offers. That is:

$$\omega^* := \arg \max_{\omega \in \Omega} \{u_1(\omega) \mid u_2(\omega) \geq \beta_2\} \quad (3.6)$$

This means that  $a_1$  cannot possibly receive any utility higher than  $u_1(\omega^*)$ . After all, by Eq. (3.6) we know that for any offer  $\omega$  that yields a higher utility to  $a_1$ , we would have  $u_2(\omega) < \beta_2$ , and agent  $a_2$  would never propose or accept any such offer, by definition of  $\beta_2$ . On the other hand, however, it also means that if we get close enough to the deadline, then  $a_2$  will be willing to accept the offer  $\omega^*$  and therefore, *ideally*,  $a_1$  should not propose or accept any offers that yield less utility than  $u_1(\omega^*)$ . So, against this opponent, a theoretically optimal strategy for  $a_1$  would be one that concedes no further than  $u_1(\omega^*)$ . For example, a time-based strategy with target value  $\beta_1 = u_1(\omega^*)$ .

Unfortunately, however, there are two problems with this idea. Firstly,  $a_1$  typically does not know the target value  $\beta_2$  of its opponent, and secondly  $a_1$  typically also does not know the utility function  $u_2$  of its opponent. Therefore,  $a_1$  cannot directly determine the ideal offer  $\omega^*$ .

Instead, however,  $a_1$  can try to infer it, using opponent modeling algorithms (which we will discuss in Chapter 4). The idea is then simple: every time our agent receives a proposal from the opponent, our agent uses it to update the opponent model to obtain a more accurate approximation of  $u_2$  and  $\beta_2$ , which it can then use to obtain a better prediction of  $\omega^*$ . Then, our agent sets its target value equal to  $u_1(\omega^*)$  (unless it is below our agent's reservation value, of course), and finally it uses this to determine our aspiration level at that moment, according to some formula such as Eq. (3.4).

This approach is called an *adaptive strategy*, because it tries to adapt to its opponent. Just like a time-based strategy it applies an aspiration level that decreases over time, but the difference is that the target value is not constant. Instead, it is updated every time we gain more information about the opponent's strategy and utility function.

In theory, if we are 100% sure that our opponent is using a time-based strategy, and we have an opponent modeling algorithm that can predict  $\omega^*$  with 100% accuracy, then an adaptive strategy is the theoretically optimal strategy against that opponent (in game theory terminology: it is a *best response*, see Chapter 5). After all, it concedes exactly enough to ensure a deal, but no further than that, so it always achieves the maximum amount of utility that can possibly be achieved against that opponent.

Of course, in practice we don't really have a 100% accurate opponent modeling algorithm. But besides that, another problem with the reasoning above is that it assumes the opponent does not know anything about our agent. The problem, is that if the opponent can somehow anticipate that we are using a purely adaptive strategy, then he may be able to exploit this knowledge by choosing a very hardheaded strategy. For example, in a split-the-pie domain where both agents have a reservation value of 0, he could choose a target value of  $\beta = 0.99$ . If we then apply a purely adaptive strategy, then our agent would always come to an agreement for which it gets no more than 0.01 utility.

Therefore, in practice, many adaptive strategies have a 'minimum target'  $\beta^{min}$  and they make sure that their target  $\beta$  is never lower than that. That is:

$$\beta = \max\{ u_1(\omega^*) , \beta^{min} \}$$

This means that such strategies are more of a hybrid between a time-based

strategy and a *purely* adaptive strategy.

Furthermore, since our opponent modeling will probably not be 100% accurate, we may need to add another term  $\epsilon$  to our target utility  $u_1(\omega^*)$ , where  $\epsilon > 0$  and where  $\epsilon$  decreases as we gain more and more knowledge about the opponent from the offers it proposes to us. So we would get:

$$\beta = \max\{ u_1(\omega^*) + \epsilon, \beta^{\min} \}$$

This is to prevent that an inaccurate estimation at the beginning of the negotiations causes our agent to concede too much.

Yet another problem with adaptive strategies, is that they kind of assume the opponent is following a purely time-based strategy, which allows the adaptive strategy to predict the optimal target value. This, however, gets much more complicated if the opponent is also playing an adaptive strategy. In that case we have two agents that are each trying to adapt to the other.

A basic implementation of an adaptive strategy is displayed in Algorithm 3.

**Exercise 3. Adaptive Agent.** Use the NegoSimulator framework to implement an agent that applies an adaptive negotiation strategy. Note that the framework already comes with the source code of a RandomAgent, so you can just copy its code and adapt it according to Algorithm 3.

Since we haven't discussed opponent modeling algorithms yet, you can again use the DummyOpponentUtilityModel that comes with the framework (See Exercise 2) to estimate the opponent's utility function.

Furthermore, to estimate the optimal target value  $\beta^*$  you can use the SimpleOpponentStrategyModel that also comes with the NegoSimulator framework. This class implements a very naive linear extrapolation algorithm to predict how far the opponent will concede.

Experiment with several parameter settings and run a number of negotiations between your adaptive agent and your time-based agent(s) from Exercise 2.

### 3.2.3 Imitative Strategies

Above, we have seen that if we know the opponent plays a time-based strategy, then the best response for our agent would be to play an adaptive strategy. On the other hand, if the opponent is playing an adaptive strategy,

---

**Algorithm 3** Adaptive Strategy.

---

**Parameters:**  $\alpha, \beta^{min}, \gamma$ **Input:**

$\Omega$  ▷ The offer space.  
 $u_1$  ▷ The agent's own utility function.  
 $rv_1$  ▷ The agent's own reservation value.  
 $T$  ▷ The deadline.  
 $\mathcal{M}$  ▷ A model of the opponent.  
 $t$  ▷ The current time.  
 $h$  ▷ The history.  
 $\omega_{rec}$  ▷ The offer last proposed by the opponent (if any).

//OPPONENT MODELING

//Update the opponent model.

1:  $\mathcal{M} \leftarrow \text{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$ 2:  $\hat{u}_2 \leftarrow \text{getEstimatedOpponentUtility}(\mathcal{M})$ 

//Use the opponent model to estimate the optimal target value.

 $\hat{\beta}^* \leftarrow \text{estimateOptimalTarget}(\mathcal{M})$ 

//BIDDING STRATEGY

//Calculate the aspiration value

3:  $\beta \leftarrow \max\{\hat{\beta}^*, \beta^{min}\}$ 4:  $\lambda \leftarrow (\alpha - \beta) \cdot \frac{1 - \gamma^{1-t/T}}{1 - \gamma} + \beta$ 

// Obtain the set of offers we have already proposed.

5:  $\Omega^{prop} \leftarrow \text{getOffersProposedByUs}(h)$ 

// Find the next offer to propose.

6:  $\omega_{next} \leftarrow \arg \max_{\omega \in \Omega} \{\hat{u}_2(\omega) \mid u_1(\omega) \geq \lambda \wedge \omega \notin \Omega^{prop}\}$ 

// ACCEPTANCE STRATEGY

// Get the last proposal received from the opponent, and accept it if

// it yields more utility to us than our aspiration level.

7:  $\text{acceptOffer} \leftarrow u(\omega_{rec}) \geq \lambda$ 

// RETURN SELECTED ACTION

8: **if**  $\text{acceptOffer}$  **then**9:     **RETURN**  $(\mathbf{a}, \omega_{rec})$ 10: **else**11:     **RETURN**  $(\mathbf{p}, \omega_{next})$ 12: **end if**

---



then the best choice for our agent would be to play a hardheaded time-based strategy which can exploit the opponent's adaptiveness. Now, the question is how to choose between these two strategies when we don't know what strategy our opponent will choose.

If one agent plays a hardheaded time-based strategy and the other plays an adaptive strategy, then the time-based agent would typically receive a higher utility than the adaptive agent. Therefore, one might be inclined to argue that choosing a hardheaded time-based strategy is better. But the problem is that the opponent could follow exactly the same reasoning, and therefore choose a hardheaded strategy as well. But then we end up with two agents each playing a hardheaded strategy, and in that case it is unlikely that the two agents will come to an agreement, since neither of the two would be willing to make any considerable concessions.

For this reason, some might reason that it is better to play an adaptive strategy. But then again, the opponent might reason in the same way and also choose an adaptive strategy. In that case we would miss out on the opportunity of exploiting him. Furthermore, if we always choose an adaptive strategy, then that could be exploited by the opponent by always choosing a hardheaded strategy. In other words, choosing between a hardheaded strategy and an adaptive strategy is a bit of a chicken-and-egg problem. The problem is that each of these strategies work well against the other, but neither of them is optimal when the opponent picks the same strategy.

We have seen that one way out would be to choose a hybrid approach that applies an adaptive strategy with a minimum target  $\beta^{min}$ , but then we still need to answer the question how to choose the optimal value for  $\beta^{min}$ . Another approach would be to flip a coin and decide between the two strategies randomly. We will discuss this option in more depth in Chapter 5.

In this section, however, we will discuss an entirely different type of strategy that is designed specifically to play well against itself. Such strategies are known as **imitative strategies** [22]. Rather than trying to *adapt* to the opponent (play hardheaded when the opponent plays conceding and vice versa), imitative agents instead try to *imitate* the opponent. That is, when the opponent is hardheaded then play hardheaded as well, and when the opponent is conceding, play conceding as well. The rationale behind this, is that if the opponent plays too hardheaded, then our agent can 'punish' it by also playing hardheaded, and when the opponent plays conceding, then our agent rewards the opponent by playing conceding as well.

Of course, this is all based on the assumption that the opponent does not play a rigid time-based strategy, but rather observes our agent's actions and is able to adapt itself to our agent's strategy.

We will discuss two kinds of imitative strategies, namely the *Classic Tit-for-Tat* strategy and the *MiCRO* strategy.

### 3.2.3.1 Classic Tit-for-Tat

In game theory, Tit-for-That (TFT) strategies are strategies in which a player imitates the moves of the other player. This strategy has been proven especially useful in the iterated prisoner's dilemma [2].

In the context of negotiation, this would mean that whenever our opponent makes a large concession, our agent replies to this by also making a large concession, and whenever our opponent makes a small concession (or no concession at all), then our agent replies with a small concession as well (or no concession at all).

Before we continue, recall that  $\Omega_t^{prop}$  denotes the set of offers that have been proposed by our agent  $a_1$  up until time  $t$  (see Eq.(3.2)). Similarly, we define  $\Omega_t^{rec}$  to be the set of offers that have been *received* by our agent  $a_1$  up until time  $t$ . In other words, it is set of offers that have been proposed by the *opponent*  $a_2$  up until time  $t$ . Formally:

$$\Omega_t^{rec} := \{\omega \in \Omega \mid \exists t' \in [0, t] : (2, \mathbf{p}, \omega, t') \in h\} \quad (3.7)$$

Now, in order to give a concrete implementation of a classic tit-for-tat negotiation strategy, we need a function  $\sigma_1$  that, given  $\Omega_t^{prop}$  returns a value  $\sigma_1(\Omega_t^{prop}) \in \mathbb{R}$  that measures how much agent  $a_1$  has so far conceded. Furthermore, we need a function  $\sigma_2$  that, given  $\Omega_t^{rec}$  returns a value  $\sigma_2(\Omega_t^{rec}) \in \mathbb{R}$  that measures the amount of concession made by  $a_2$ .

$$\sigma_1, \sigma_2 : 2^\Omega \rightarrow \mathbb{R}$$

In general, for any agent, when we say it makes a large ‘concession’, this can be interpreted in two ways: it can mean that it makes a proposal with high utility for the opponent, or it can mean that it makes a proposal with low utility for itself. In a single-issue negotiation where the agents’ interests are strictly opposing, such as the bargaining over the price of a second-hand car, we don’t have to worry about this distinction, because any concession of the first type is automatically also one of the second type and vice versa.

However, in more complex negotiation scenarios, where not every offer is Pareto-optimal, and where the agents do not know each others’ utility function, these two concepts are different.

This means that for  $\sigma_1$  there are two obvious choices. Namely, we could define it in terms of our agent’s own utility, or in terms of our opponent’s

(estimated) utility:

$$\sigma_1(\Omega_t^{prop}) = \max \{u_1(\omega_1^{max}) - u_1(\omega) \mid \omega \in \Omega_t^{prop}\} \quad (3.8)$$

or:

$$\sigma_1(\Omega_t^{prop}) = \max \{\hat{u}_2(\omega) - \hat{u}_2(\omega_2^{min}) \mid \omega \in \Omega_t^{prop}\} \quad (3.9)$$

where  $\hat{u}_2$  is an estimation of the opponent's utility function  $u_2$ , made by an opponent modeling algorithm and where  $\omega_1^{max}$  and  $\omega_2^{min}$  are defined by Equations (2.1) and (2.2).

In the first case, our 'concession' corresponds to the lowest amount of utility our agent has so far asked for itself, while in the second case it corresponds to the highest amount of utility it has so far offered to the opponent.

Similarly, we can measure the opponent's concession using either our agent's own utility function, or the opponent's estimated utility function:

$$\sigma_2(\Omega_t^{rec}) = \max \{u_1(\omega) - u_1(\omega_1^{min}) \mid \omega \in \Omega_t^{rec}\} \quad (3.10)$$

or:

$$\sigma_2(\Omega_t^{rec}) = \max \{\hat{u}_2(\omega_2^{max}) - \hat{u}_2(\omega) \mid \omega \in \Omega_t^{rec}\} \quad (3.11)$$

Here, in the first case, the opponent's 'concession' corresponds to the highest amount of utility the opponent has so far offered to our agent, while in the second case it corresponds to the lowest amount of utility the opponent has so far asked for itself.

Note that here,  $\sigma_2$  is a function used by *our* agent  $a_1$  to measure the opponent's concession. In other words, it exists in the 'mind' of our agent  $a_1$  and the opponent itself may actually use an entirely different function to measure its own concession (if it even uses a Tit-for-Tat strategy at all).

Whenever it is our agent's turn, its goal is to propose an offer  $\omega_{next}$  such that the total amount of concession that our agent has made so far remains slightly higher than our opponent's. We therefore define, for any offer  $\omega \in \Omega$ , its *concession gain*:

$$\Delta\sigma_t(\omega) := \sigma_1(\Omega_t^{prop} \cup \{\omega\}) - \sigma_2(\Omega_t^{rec})$$

which allows us to quantify, for any offer  $\omega$ , the difference between our agent's concession after proposing  $\omega$  and the concession made by the opponent.

Finally, the Tit-for-Tat strategy chooses our agent's next offer to propose  $\omega_{next}$  by selecting it from a set of possible offers that satisfy some criterion

regarding to the concession gain. Again, there is no unique way to do this, so we provide two examples:

$$\omega_{next} = \arg \max_{\omega} \{ u_1(\omega) \mid \Delta\sigma_t(\omega) > \epsilon_{min} \wedge u_1(\omega) > rv_1 \} \quad (3.12)$$

or:

$$\omega_{next} = \arg \max_{\omega} \{ \hat{u}_2(\omega) \mid \Delta\sigma_t(\omega) \in (\epsilon_{min}, \epsilon_{max}) \wedge u_1(\omega) > rv_1 \} \quad (3.13)$$

where  $\epsilon_{min}$  and  $\epsilon_{max}$  are a minimum and a maximum required concession gain, respectively. In the first case our agent would select the offer that maximizes its own utility, under the constraint that it should also concede enough to the opponent. In the second case, our agent would select an offer that maximizes the *opponent's* estimated utility, but that requires we also limit ourselves to a maximum concession gain, to prevent our agent from conceding too much.

In each of these expressions,  $\epsilon_{min}$  can be equal to 0, but  $\Delta\sigma_t(\omega)$  must remain strictly greater than 0. This is, because otherwise if it happens that both agents have made exactly the same amount of concession, then neither of them will be willing to concede more, and they get stuck in a deadlock (if they both use the same strategy). Therefore, each of the two agents should always strive to concede slightly more than the other.

We have now seen that for a concrete implementation of Tit-for-Tat we need to make 3 choices: an expression for  $\sigma_1$ , an expression for  $\sigma_2$ , and a method to choose  $\omega_{next}$  (e.g. Eq. (3.12) or Eq. (3.13)).

At first sight, we might be tempted to choose the expressions that only depend on our agent's own utility function (i.e. Eqs. (3.8), (3.10) and (3.12)), so that we don't have to rely on any opponent modeling algorithms. However, it turns out that this doesn't work very well. The problem is that in that case, if both agents make sufficiently small concessions in each turn, then the final outcome would always be an offer that satisfies  $u_1 \approx \frac{1}{2}u_1(\omega_1^{max}) + \frac{1}{2}u_1(\omega_1^{min})$ . This can be seen easily as follows. Suppose for simplicity that we have a normalized utility function (i.e.  $u_1(\omega_1^{min}) = 0$  and  $u_1(\omega_1^{max}) = 1$ ). Now, if the opponent  $a_2$  makes an offer that yields a utility of 0.1 to our agent, then our agent  $a_1$  would reply with an offer that yields a utility of  $1-0.1=0.9$  to itself. Next, if  $a_2$  makes a proposal with utility of, say, 0.3 for  $a_1$ , then  $a_1$  replies with an offer with utility  $1-0.3=0.7$ . Then, if  $a_2$  proposes an offer with utility 0.35, our agent  $a_1$  will reply with an offer that yields  $1-0.35=0.65$ , next, if  $a_2$  proposes an offer with utility 0.55 then  $a_2$  replies with an offer that yields  $1-0.55 = 0.45$ . It is easy to see that, no matter which offers the opponent proposes, this always either converges

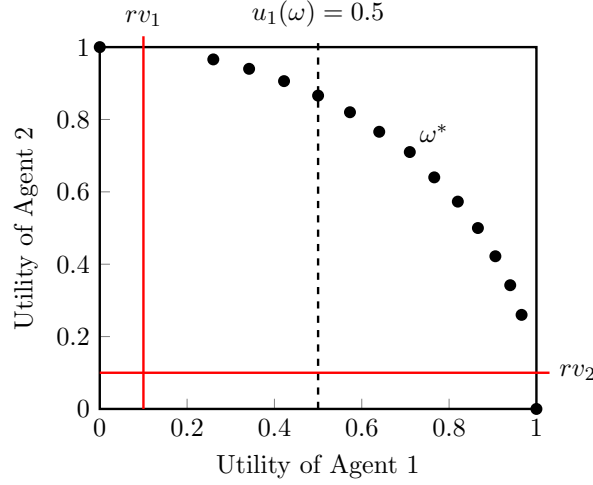


Figure 3.3: An example of a domain with low opposition. Here, the outcome with  $u_1(\omega) = 0.5$  is highly unfair for  $a_1$ , since the opponent would receive  $u_2(\omega) = 0.87$  for that same offer. Especially, since there exists a much fairer offer, here indicated as  $\omega^*$ , for which both agents would receive 0.7.

to an agreement with utility 0.5 for  $a_1$ , or the two agents' proposals don't converge at all, which means there will be no agreement.

Now, it happens that in many negotiation domains, if an offer yields 0.5 to one agent, then it yields much more utility to the other agent. This happens specifically in domains with low opposition, where there exist offers for which both agents receive a normalized utility greater than 0.5. This is illustrated in Figure 3.3. In other words, our agent would receive much less utility than what it could potentially achieve with a better algorithm.

Furthermore, if we already know that that this algorithm can only make agreements with a utility value of 0.5 for our agent, then we could just as well play a time-based strategy with target value of  $\beta = 0.5$ . This would at least give our agent the possibility of reaching agreements with higher utility.

So, what if we choose one of the other options? Well, if we choose the opponent's estimated utility  $\hat{u}_2$  to calculate our own concession  $\sigma_1$  as well as our opponent's concession  $\sigma_2$ , then we end up with essentially the same problem. In that case (assuming we have accurate opponent modeling algorithms), the only possible agreement the agents could make, would be one with  $u_2(\omega) \approx 0.5$ . While this may seem good, because such a solution

would typically yield high utility for our own agent, the problem is that it would therefore be also less likely that the opponent would be willing to accept such a deal.

A better idea seems to be to use our own utility to measure our own concession and the opponent's utility to measure the opponent's concession, or vice versa. In either of these two cases the proposals would converge to some deal  $\omega$  with  $u_1(\omega) \approx u_2(\omega)$ , which would typically be better.

The problem with that, however, is that its success depends on the accuracy of our opponent modeling algorithms. If we cannot estimate  $u_2$  accurately, then our agent could be making concessions that are too large, yielding suboptimal agreements, or it could be making concessions that are too small, preventing the agents from coming to an agreement at all.

An alternative approach to reach good outcomes using TFT, is to use *relative* concessions, instead of absolute ones [8]. By this we mean that we first pick some ideal outcome  $\omega^*$ , such as the maximum social welfare solution, of the Nash bargaining solution (see Section 5.4) and then we measure concession relative to that ideal outcome:

$$\sigma_1(\Omega_t^{prop}) = \max \left\{ \frac{u_1(\omega_1^{max}) - u_1(\omega)}{u_1(\omega_1^{max}) - u_1(\omega^*)} \mid \omega \in \Omega_t^{prop} \right\} \quad (3.14)$$

$$\sigma_2(\Omega_t^{rec}) = \max \left\{ \frac{u_1(\omega) - u_1(\omega_1^{min})}{u_1(\omega^*) - u_1(\omega_1^{min})} \mid \omega \in \Omega_t^{rec} \right\} \quad (3.15)$$

Note that this does require you to know which outcome  $\omega^*$  would be ideal, which would still depend on the opponent's utility function. However, it requires much less knowledge about  $u_2$  than if we used Eqs. (3.9) and (3.11).

It may also be worth mentioning that in the paper that originally proposed the TFT negotiation strategy [22], the authors proposed a variant in which the agents' concessions were calculated only in terms of the *last few* proposals by each agent, rather than *all* their proposals up to time  $t$ .

As explained before, the main idea of Tit-for-Tat is that it works well against itself. However, if the opponent uses a hardheaded time-based strategy, then Tit-for-Tat is likely to fail, because neither of the two agents will be making big concessions. If the opponent applies an adaptive strategy, or a conceding time-based strategy, Tit-for-Tat will likely come to an agreement, but it will not be able to exploit the opponent as much as a hardheaded strategy could have done.

Furthermore, even if we have a good opponent strategy, and the opponent is indeed using TFT as well, then the success of our agent also heavily relies on the accuracy of the *opponent's* opponent modeling algorithms (i.e.

the algorithm used by our opponent to estimate our utility function). After all, the opponent might *intend* to make an offer that yields a lot of utility to our agent, but due to an inaccurate opponent model he might end up proposing one that actually yields very low utility to our agent, which would then respond with a counter-proposal that yields very low utility to the opponent. This would prevent them to reach an agreement, even though both agents have the intention to make large concessions.

**Exercise 4. Tit-for-Tat Agent.** Implement an agent that applies one of the various Tit-for-Tat strategies explained in this section. Since we haven't discussed opponent modeling algorithms yet, you can use the `DummyOpponentUtilityModel` that comes with the `NegoSimulator` framework (See Exercise 2). Let your agent negotiate against the `RandomAgent` or against one of your agents from Exercises 2 and 3, or against a copy of itself.

### 3.2.3.2 The MiCRO Strategy

We have seen above, that classic TFT strategies depend heavily on the quality of the opponent modeling algorithms of both agents. However, recently a new kind of TFT strategy has been proposed based on the idea that our agent does not know anything about the opponent's utility function at all and moreover, that the opponent also does not know anything about *our* agent's utility function [14]. This strategy was called MiCRO, which stands for *Minimal Concession in Reply to new Offers*. Despite its simplicity and the fact that it does not require any opponent modeling at all, it has shown some remarkably good results.

MiCRO works as follows. Before the negotiations begin, our agent  $a_1$  creates a list  $(\omega_1, \omega_2, \dots, \omega_K)$  containing all offers in the domain, sorted in order of decreasing utility for itself. That is,  $u_1(\omega_1) \geq u_1(\omega_2) \geq \dots \geq u_1(\omega_K)$ . Then, when the negotiations start, our agent will first propose the offer with highest utility for itself. That is,  $\omega_1$ , which is the first offer on its list. Then, in the following rounds, every time the opponent makes a new proposal, our agent will respond by proposing the next offer on its list. So, it will first propose  $\omega_2$ , then  $\omega_3$ , then  $\omega_4$ , etcetera. However, whenever the opponent  $a_2$  proposes an offer that  $a_2$  has already proposed before,  $a_1$  will reply by also repeating an earlier proposal.

More precisely, whenever it is  $a_1$ 's turn to make a proposal, it counts how many *different* offers it has so far received from the opponent (we denote

---

**Algorithm 4** A Classic Tit-for-Tat strategy.

---

**Parameters:**  $\epsilon_{min}$

**Input:**

$\Omega$	▷ The offer space.
$u_1$	▷ The agent's own utility function.
$rv_1$	▷ The agent's own reservation value.
$T$	▷ The deadline.
$\mathcal{M}$	▷ A model of the opponent.
$t$	▷ The current time.
$h$	▷ The history.
$\omega_{rec}$	▷ The offer last proposed by the opponent (if any).

```

//OPPONENT MODELING
1:  $\mathcal{M} \leftarrow \text{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$ 
2:  $\hat{u}_2 \leftarrow \text{getEstimatedOpponentUtility}(\mathcal{M})$ 

//BIDDING STRATEGY
// Get the next offer to propose according to Equation (3.12)
// We split this calculation into two parts:
// 1) Get a set of candidate offers  $C$ .
// 2) Find the offer that maximizes our utility.
// Note that the calculation of  $\Delta\sigma_t(\omega)$  depends on the chosen
// expressions for  $\sigma_1$  and  $\sigma_2$ .
3:  $C \leftarrow \{ \omega \in \Omega \mid \Delta\sigma_t(\omega) > \epsilon_{min} \wedge u_1(\omega) > rv_1 \}$ 
4: if  $C = \emptyset$  then
5:    $\omega_{next} \leftarrow \dots$  // Use any alternative method to pick an offer here.
6: else
7:    $\omega_{next} \leftarrow \arg \max_{\omega} \{ u_1(\omega) \mid \omega \in C \}$ 
8: end if

//ACCEPTANCE STRATEGY
//Get the last proposal received from the opponent, and accept it if and
//only if it is at least as good as the offer the agent is about to propose.
9:  $\text{acceptOffer} \leftarrow u(\omega_{rec}) \geq u(\omega_{next})$ 

// RETURN SELECTED ACTION
10: if  $\text{acceptOffer}$  then
11:   RETURN ( $\mathbf{a}$ ,  $\omega_{rec}$ )
12: else
13:   RETURN ( $\mathbf{p}$ ,  $\omega_{next}$ )
14: end if

```

---



this number by  $n$ ), and how many *different* offers it has so far proposed to the opponent (we denote this number by  $m$ ). That is,  $n := |\Omega_t^{rec}|$  and  $m := |\Omega_t^{prop}|$ . Then, if  $m \leq n$ , our agent will propose  $\omega_{m+1}$ . On the other hand, if  $m > n$  then it will pick a random integer  $r$  such that  $1 \leq r \leq m$  and propose  $\omega_r$ .

An implementation of the MiCRO strategy is given in Algorithm 5.

The intuition behind MiCRO is that, like any other TFT algorithm, it tries to make a concession whenever the opponent makes a concession. However, since it assumes neither of the two agents know anything about the other agent's utility function, MiCRO does not care *how large* the opponent's concessions are. After all, the size of the opponent's concession as perceived by our agent says nothing about the size of the concession the opponent *intended* to make. The opponent might make a large concession in terms of its own utility  $u_2$ , but this may result in a very small concession measured in our agent's own utility  $u_1$ . For the same reason MiCRO never makes large concessions to its opponent. In fact, it always makes exactly the smallest possible concession: it just proposes the next offer on its list. Another difference between MiCRO and classic TFT is that MiCRO uses a different definition of 'concession'. That is, even if the opponent's new proposal offers less utility to  $a_1$  than the opponent's previous proposal, MiCRO still considers this a concession, as long as it is different from any of the opponent's previous offers. After all, if the opponent makes offers in order of decreasing utility for itself, then every new proposal is indeed a concession from his point of view.

Note that MiCRO can indeed be seen as a TFT algorithm, with the following concession measures:

$$\begin{aligned}\sigma_1(\Omega_t^{prop}) &= |\Omega_t^{prop}| \\ \sigma_2(\Omega_t^{rec}) &= |\Omega_t^{rec}|\end{aligned}$$

and that uses Eq. (3.12) to select the next offer to propose, with  $\epsilon_{min} = 0$ .

At first sight, it may seem that MiCRO must be very slow in large negotiation domains, since it makes only minimal concessions. If a domain contains tens of thousands of offers, then you may therefore expect it to take a long time before MiCRO has conceded enough for the opponent to be willing to accept any of MiCRO's proposals. However, in practice it turns out to be rather the opposite. When two MiCRO agents negotiate against each other they typically come to an agreement much faster than most other negotiation strategies. The reason for this, is that MiCRO does not spend any time updating any opponent modeling algorithms. In each turn it just

performs a few very simple calculations and then proposes the next offer on its list, which makes it very fast.

Another main advantage of MiCRO is that it is very simple to implement, since it does not require any complicated machine learning algorithms for opponent modeling and it also does not require any parameters to be fine-tuned.

However, the biggest advantage of MiCRO, is that it makes a nearly optimal trade-off. On the one hand it is very hardheaded because it only makes minimal concessions and only keeps conceding as long as the opponent also keeps conceding. Yet, unlike hardheaded time-based agents, which often fail to come to an agreement against other hardheaded agents, MiCRO almost always comes to an agreement when negotiating against itself. This is because in that case both agents would always keep making concessions until sooner or later they reach an agreement.

There are just two possible scenarios where a negotiation between two agents that both apply the MiCRO strategy would fail. The first scenario is when one of the two agents has a very high reservation value so at some point it can't continue conceding because it has already reached its reservation value before the agents could reach an agreement. The second scenario is when the deadline is too short compared to the size of the domain, so there is no time to concede far enough to reach an agreement. However, as explained above, MiCRO is typically much faster than other strategies, so in this scenario many other strategies might also suffer to concede fast enough.

Apart from these two possible scenarios, the main disadvantage of MiCRO is that it will still fail to make an agreement against a hardheaded time-based agent that at some point refuses to concede any further before they reach an agreement.

**Exercise 5. MiCRO.** Implement an agent based on the MiCRO strategy in the NegoSimulator framework and let it negotiate against the RandomAgent, or against any of the agents from the previous exercises, or against a copy of itself.

### 3.3 Acceptance Strategies

In the previous sections we have discussed a number of bidding strategies. In doing so, we also showed a number of different *acceptance* strategies in

---

**Algorithm 5** The MiCRO strategy. Note that  $offers[m]$  here corresponds to  $\omega_{m+1}$  in the text.

---

**Input:**

$offers$	▷ A list containing all possible offers, sorted in order of decreasing utility.
$u_1$	▷ The agent's own utility function.
$rv_1$	▷ The agent's own reservation value.
$h$	▷ The history.
$\omega_{rec}$	▷ The offer last proposed by the opponent (if any).

```

1:  $m \leftarrow countUniqueOffersProposedByMe(h)$ 
2:  $n \leftarrow countUniqueOffersProposedByOpponent(h)$ 

  // If we have not proposed more unique offers than
  // the opponent and the next offer on our list is better than  $rv_1$ ,
  // then we will propose a new offer.
  // We store this decision in a boolean variable readyToConcede.
3:  $readyToConcede \leftarrow m \leq n$  and  $u_1(offers[m]) > rv_1$ 

  //BIDDING STRATEGY
  // If we are ready to concede then propose the next offer on the list.
  // Otherwise, pick a random offer that we have already proposed before.
4: if  $readyToConcede$  then
5:    $\omega_{next} \leftarrow offers[m]$ 
6: else
7:    $r \leftarrow getRandomInteger(0, m)$       ▷ Pick random integer  $r$  with  $0 \leq r < m$ .
8:    $\omega_{next} \leftarrow offers[r]$ 
9: end if

  //ACCEPTANCE STRATEGY
  // Determine the lowest utility we are willing to accept.
10: if  $readyToConcede$  then
11:    $\lambda \leftarrow u_1(offers[m])$       ▷ The utility of the offer we are about to propose next.
12: else
13:    $\lambda \leftarrow u_1(offers[m-1])$       ▷ The lowest utility among all offers we have already proposed.
14: end if
15:  $acceptOffer \leftarrow u(\omega_{rec}) \geq \lambda$ 

  // RETURN SELECTED ACTION
16: if  $acceptOffer$  then
17:   RETURN ( $\mathbf{a}$ ,  $\omega_{rec}$ )
18: else
19:   RETURN ( $\mathbf{p}$ ,  $\omega_{next}$ )
20: end if

```

---

the various examples (Algorithms 2–5). In this section we will discuss these acceptance strategies in a bit more detail.

In the following, let  $\omega_{next}$  denote the next offer to make, as decided by the bidding strategy, and let  $\omega_{rec}$  denote the last received offer.

Perhaps the most commonly used acceptance strategy in the literature is the  $AC_{next}$  strategy that simply accepts  $\omega_{rec}$  if and only if it is better than, or equal to  $\omega_{next}$ :

**Definition 12.** *The  $AC_{next}$  acceptance strategy accepts if and only if:*

$$u_1(\omega_{rec}) \geq u_1(\omega_{next}) \quad (3.16)$$

At first sight, this makes perfect sense, because it simply let the bidding strategy do all the work to decide which offers our agent will consider acceptable. However, the problem with this strategy, is that it can lead to somewhat illogical decisions when the strategy is not purely monotonic. By ‘monotonic’ we mean that the offers proposed by the agent keep always keep decreasing in terms of the utility for that agent. More precisely:

**Definition 13.** *A bidding strategy for agent  $a_i$  is **monotonic**, if for any history  $h$ , and any two proposals  $(i, \mathbf{p}, \omega, t) \in h$ ,  $(i, \mathbf{p}, \omega', t') \in h$  generated by that strategy, for which  $t < t'$  we have  $u_i(\omega) > u_i(\omega')$*

While each of the bidding strategies we discussed above *in general* proposes offers in order of decreasing utility, it is certainly not the case that *every* proposal is always followed by a proposal with lower utility. Therefore, none of these strategies are monotonic.

The problem with  $AC_{next}$  and non-monotonic bidding strategies is illustrated in Figure 3.4. Before we explain the problem, we should first highlight a few important details about this figure. Firstly, note that the vertical axis does not represent  $a_2$ ’s *true* utility  $u_2$ , but rather its *estimated* utility  $\hat{u}_2$ , as estimated by agent 1’s opponent modeling algorithm. Secondly, note that we have zoomed in a bit so that the horizontal axis shows only values between 0.65 and 0.77. Finally, note that we have drawn the aspiration levels of agent 1 in the diagram at three different times:  $t_1$ ,  $t_2$ , and  $t_3$ , where  $t_1 < t_2 < t_3$ .

Now, let us suppose that our agent  $a_1$  uses a time-based strategy, based on Equation (3.1). Furthermore, suppose that at some time  $t_1$  the aspiration level  $\lambda_1(t_1)$  of our agent is 0.74 and our agent proposes the offer  $\omega_1$  with utility  $u_1(\omega_1) = 0.79$ . Next, suppose that agent  $a_2$  rejects this proposal, so after a small amount of time our agent gets to propose a new offer in the next

turn, at time  $t_2$ . Meanwhile, our agent's aspiration level has dropped to, say,  $\lambda_1(t_2) = 0.69$ . We see in the diagram that there are several offers with utility between 0.69 and 0.74 that can now be proposed but, according to Eq. (3.1), our agent will propose the one with highest estimated opponent utility  $\hat{u}_2$ . This offer is denoted by  $\omega_2$  and we see that  $u_1(\omega_2) = 0.7$ . Again, suppose this offer is rejected and instead  $a_2$  makes a counter-proposal, which is denoted  $\omega_{rec}$  in the diagram, with utility  $u_1(\omega_{rec}) = 0.71$ . Then, in the next turn, at time  $t_3$ , suppose the aspiration level has dropped to 0.67. Among all offers with  $u_1(\omega) > 0.67$  that we have not proposed yet, the one with highest estimated opponent utility  $\hat{u}_2$  is now  $\omega_3$ , with utility  $u_1(\omega_3) = 0.7$ . So, the bidding strategy will select  $\omega_3$  to propose next.

Now, if our agent uses  $AC_{next}$ , it will compare  $\omega_{rec}$  with  $\omega_3$ . This means our agent will *reject*  $\omega_{rec}$ , because  $\omega_3$  yields more utility. But this clearly does not make sense, because our agent has already proposed  $\omega_2$  which yielded less utility than  $\omega_{rec}$ . So, if our agent was willing to propose  $\omega_2$  with utility 0.7, then it should certainly be willing to accept  $\omega_{rec}$  with utility 0.71. In fact, according to its aspiration level it should be willing to propose or accept any offer with utility higher than 0.67.

Rejecting offer  $\omega_{rec}$  only makes sense if our agent thinks it could obtain a better deal in the future, but if that's the case then our agent should have never proposed  $\omega_2$ , and its aspiration level should not have dropped to 0.67.

The problem illustrated above can be resolved easily by using the aspiration level *itself* to make the acceptance decision, rather than using the offer  $\omega_{next}$  that was chosen based on the aspiration level. Indeed, we used this acceptance strategy in Algorithms 2 and 3. We will denote this strategy by  $AC_{asp}$ .

**Definition 14.** *The  $AC_{asp}$  acceptance strategy accepts if and only if:*

$$u_1(\omega_{rec}) \geq \lambda(t) \quad (3.17)$$

where  $\lambda$  is the aspiration function and  $t$  is the time at which the decision is made.

Of course, the problem with  $AC_{asp}$  is that it only works if your bidding strategy indeed uses an aspiration function. For other bidding strategies, such as Tit-for-Tat or MiCRO, that do not make use of aspiration functions, there is another straightforward solution. Namely, to accept any offer that is better than the offer you are going to propose next, *or* better than any of the offers you have already proposed before. We will denote this strategy by  $AC_{low}$ .

**Definition 15.** *The  $AC_{low}$  acceptance strategy accepts if and only if:*

$$u_1(\omega_{rec}) > \min\{u_1(\omega) \mid \omega \in \Omega_t^{prop} \cup \{\omega_{next}\}\} \quad (3.18)$$

where  $t$  is the time at which the decision is made and  $\Omega_t^{prop}$  denotes the set of offers so far proposed by our agent (as defined by Eq. (3.2)).

Note that we used this acceptance strategy in our implementation of MiCRO in Algorithm 5 (although this may not be immediately obvious from the notation).

The strategies  $AC_{next}$ ,  $AC_{asp}$  and  $AC_{low}$  are all based on the same principle: only accept an offer if you would also be willing to *propose* that same offer yourself. While this principle makes sense, it may be somewhat too strict when the negotiations are close to the deadline. In that case it can be beneficial to even accept offers that are actually somewhat less valuable than those offers that you are willing to propose.

The idea is that near the deadline, proposing an offer is more risky than accepting an offer, because an acceptance yields a guaranteed amount of utility, while a proposal could be rejected by the opponent, so it brings along the risk that the negotiations may fail. The closer we get to the deadline, the more important this risk becomes.

Therefore, one could argue that when you decide to make a proposal, you should ask for a bit more utility than what you would be willing to accept, in order to offset the increased risk. This can be modeled by a parametrized version of  $AC_{next}$  [7], which has two parameters  $\alpha$  and  $\beta$  and which is denoted by  $AC_{next}(\alpha, \beta)$ .

**Definition 16.** *Let  $\alpha, \beta \in \mathbb{R}$  be two real numbers. Then the  $AC_{next}(\alpha, \beta)$  acceptance strategy accepts if and only if:*

$$\alpha \cdot u_1(\omega_{rec}) + \beta \geq u_1(\omega_{next}) \quad (3.19)$$

Note that if  $\alpha = 1$  and  $\beta = 0$ , then  $AC_{next}(\alpha, \beta)$  is just identical to  $AC_{next}$ . Typically, the values of  $\alpha$  and  $\beta$  would both be non-negative. While there is no mathematical reason why they could not be negative, there does not seem to be any obvious reason to ever consider such values. After all, it does not make a lot of sense to propose an offer with a utility of, say,  $u_1(\omega) = 0.6$  if you are not willing to accept an offer with that same amount of utility, or better. The same generalization can also be applied to  $AC_{asp}$  or  $AC_{low}$ . That is, we could define  $AC_{asp}(\alpha, \beta)$  or  $AC_{low}(\alpha, \beta)$  in an analogous manner. Of course, an obvious disadvantage of such parametrized strategies, is that it requires choosing the right values of  $\alpha$  and  $\beta$ , which may be difficult.

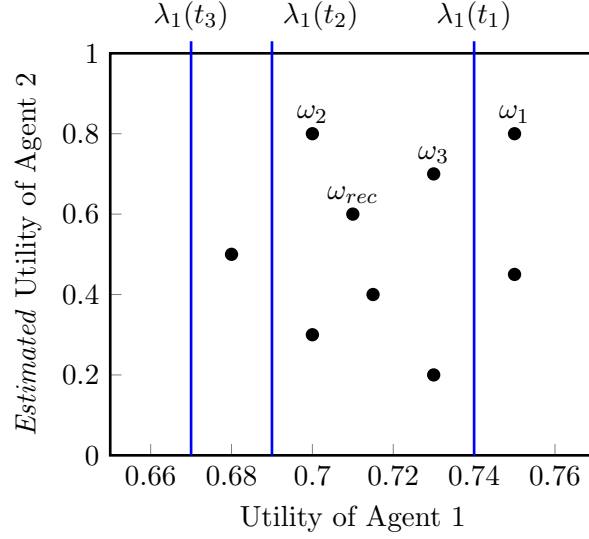


Figure 3.4: The problem with  $AC_{next}$ . At  $t_1$  agent 1 proposes  $\omega_1$ , at  $t_2$  agent 1 proposes  $\omega_2$ , and at  $t_3$  agent 1 has the choice between proposing  $\omega_3$  or accepting  $\omega_{rec}$ . According to  $AC_{next}$ , the agent should reject. However, this does not make sense, since he has already proposed  $\omega_2$  which is actually worse than  $\omega_{rec}$ .

Another reason why it could be advantageous for our agent to accept offers that yield less utility than the offers it is willing to propose, is that this would allow our agent to apply a very hardheaded bidding strategy, in order to entice the opponent to make large concessions, while at the same time it still allows our agent to come to an agreement in case the opponent is not willing to make such concessions. In other words, it allows our agent to pretend to be more hardheaded than what he really is.

**Exercise 6.  $AC_{low}$ .** Adapt the implementation of your Tit-for-Tat agent from Exercise 4 to apply the  $AC_{low}$  acceptance strategy instead of  $AC_{next}$ .

### 3.4 Reproposing

We will now discuss a simple technique that can be added on top of any of the previously described negotiation strategies, that can make them somewhat better. This approach was described, for example, in [43] and in [15].

Let us explain it with an example. Suppose that we have a negotiation domain with 10 possible offers:  $\Omega = \{\omega_1, \omega_2, \dots, \omega_{10}\}$  and suppose that our agent's utility function is given by  $u_1(\omega_j) = 0.1j$ . That is,  $u_1(\omega_1) = 0.1$ ,  $u_1(\omega_2) = 0.2$ , etcetera, so our agent's most preferred offer is  $\omega_{10}$ . Furthermore, suppose that our agent  $a_1$  follows a time-based strategy with a linear aspiration function ( $\gamma = 1$ ) and without opponent modeling, as given by Eq. (3.3).

Now, suppose that, from the point of view of  $a_1$ , the negotiations proceed as follows (see also Figure 3.5):

1. At  $t = 0.0$ :  $\lambda_1(t) = 1.0$   $a_1$  proposes  $\omega_{10}$
2. At  $t = 0.05$ :  $a_2$  proposes  $\omega_4$
3. At  $t = 0.10$ :  $\lambda_1(t) = 0.9$   $a_1$  proposes  $\omega_9$
4. At  $t = 0.15$ :  $a_2$  proposes  $\omega_6$
5. At  $t = 0.20$ :  $\lambda_1(t) = 0.8$   $a_1$  proposes  $\omega_8$
6. At  $t = 0.30$ :  $a_2$  proposes  $\omega_2$
7. At  $t = 0.50$ :  $\lambda_1(t) = 0.5$   $a_1$  proposes ...

At time  $t = 0.50$ , our agent's strategy prescribes that it should propose  $\omega_5$ . Ideally, however,  $a_1$  would like to accept  $\omega_6$  instead, because that would yield more utility. The problem is that the AOP does not allow that, because it only allows accepting the *last* received offer, which is  $\omega_2$ . Note that earlier our agent did not accept  $\omega_6$ , because at the moment he received that offer, his aspiration level was still at  $\lambda_1(t) = 0.8$  which was greater than  $u_1(\omega_6) = 0.6$ .

The solution, is to override the bidding strategy and propose  $\omega_6$  instead of  $\omega_5$ . Since  $\omega_6$  was already proposed before by  $a_2$ , it is very likely that  $a_2$  will now accept it, and therefore it should indeed be better for  $a_1$  to propose  $\omega_6$ , than to propose  $\omega_5$ . We call this *reproposing* because the agent is proposing an offer that was already proposed earlier by the opponent. Algorithm 6 shows how this technique can be implemented on top of any generic agent.

**Definition 17.** We say an agent  $a_i$  **reproposes** an offer  $\omega$  if  $a_i$  proposes it, while it was earlier already proposed by the other agent  $a_j$  and  $a_i$  itself has not yet proposed it since then.



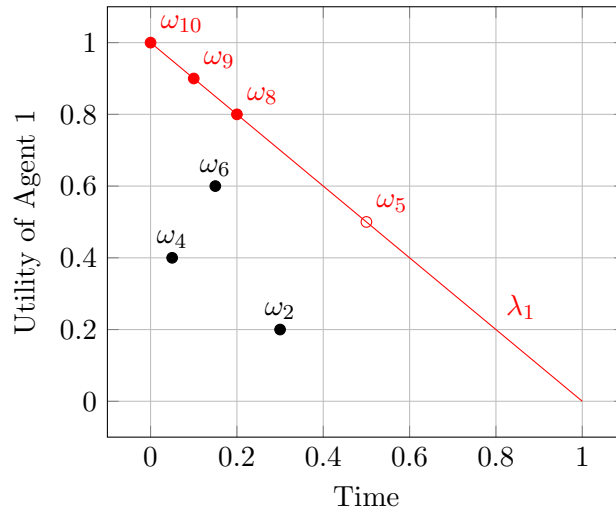


Figure 3.5: The benefit of reproposing. The red dots represent proposals made by  $a_1$ , the red line represents  $a_1$ 's aspiration level  $\lambda_1$  as a function of time, and the black dots represent proposals made by  $a_2$ . At time  $t = 0.5$ , the bidding strategy of  $a_1$  suggests to propose  $\omega_5$ . However, it makes more sense for  $a_1$  to propose  $\omega_6$ , which earlier was already proposed by  $a_2$ . Note that at that time  $a_1$  cannot *accept*  $\omega_6$ , because the AOP only allows accepting the last received proposal, which was  $\omega_2$ .

**Exercise 7. Reproposing** Adapt the agents that you have implemented in the previous exercises to make them apply the reproposing technique, as described in Algorithm 6.

---

**Algorithm 6** Generic BOA Agent for the alternating offers protocol that applies reproposing.

---

**Input:**

$\Omega$	▷ The offer space.
$u_1$	▷ The agent's own utility function.
$rv_1$	▷ The agent's own reservation value.
$T$	▷ The deadline.
$\mathcal{M}$	▷ A model of the opponent.
$t$	▷ The current time.
$h$	▷ The history.
$\omega_{rec}$	▷ The offer last proposed by the opponent (if any).

//OPPONENT MODELING

1:  $\mathcal{M} \leftarrow \text{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$

//BIDDING STRATEGY

2:  $\omega_{next} \leftarrow \text{biddingStrategy}(\Omega, u_1, rv_1, T, \mathcal{M}, t, h)$

//CHECK IF WE CAN FIND A BETTER OFFER TO REPROPOSE

//From the history, extract the set of all offers that have so far been  
 //proposed by this agent.

3:  $\Omega^{prop} \leftarrow \text{getProposedOffers}(h)$   
 //From the history, extract the set of all offers that have so far been  
 //proposed by the opponent.

4:  $\Omega^{rec} \leftarrow \text{getReceivedOffers}(h)$

5: **if**  $\Omega^{rec} \setminus \Omega^{prop} \neq \emptyset$  **then**

6:      $\omega_{rep} \leftarrow \arg \max \{u_1(\omega) \mid \omega \in \Omega^{rec} \setminus \Omega^{prop}\}$

7:     **if**  $u_1(\omega_{rep}) \geq u_1(\omega_{next})$  **then**

8:          $\omega_{next} \leftarrow \omega_{rep}$

9:     **end if**

10: **end if**

//ACCEPTANCE STRATEGY

11:  $\text{acceptOffer} \leftarrow \text{acceptanceStrategy}(\Omega, u_1, T, \mathcal{M}, t, h, \omega_{rec}, \omega_{next})$

// RETURN SELECTED ACTION

//Finally, return the selected action (accept or propose).

12: **if**  $\text{acceptOffer}$  **then**

13:     RETURN (**a**,  $\omega_{rec}$ )

14: **else**

15:     RETURN (**p**,  $\omega_{next}$ )

16: **end if**

---



## Chapter 4

# Opponent Modeling

In this chapter we will discuss various techniques that have been proposed in the literature to model the opponent.

We can distinguish between three types of opponent modeling:

1. Learning the opponent's utility function, during the negotiation.
2. Learning the opponent's strategy, during the negotiation.
3. Learning the opponent's strategy from earlier negotiations.

We will discuss each of these types respectively in the following three sections.

Note that we do not discuss learning the opponent's utility function from earlier negotiations, because in most scenarios studied in the literature the utility function would change with every new negotiation, so this wouldn't make sense.

### 4.1 Learning the Opponent's Utility Function

In this section we will discuss several techniques that can be used by our agent to learn the opponent's utility function, based on the proposals that it receives from its opponent.

Specifically, we will discuss the following techniques:

1. Bayesian learning.
2. Scalable Bayesian learning.
3. Frequency Analysis.

We should note that all these techniques assume that the negotiations take place over a multi-issue domain and that the opponent's utility function

$u_2$  is linear, so it is of the form of Eq. (2.3). Therefore, these techniques are not applicable to other types of negotiation domains.

#### 4.1.1 Bayesian Learning

Bayesian learning [26] is one of the earliest and still most commonly used techniques in automated negotiation to learn the opponent's utility function.

The idea is as follows. Suppose that we have some given set of possible utility functions  $U$  and, based on the proposals  $\pi_1, \pi_2, \dots, \pi_k$  that our agent has so far received from its opponent, we want to calculate the probability, for each function  $u \in U$ , that that function  $u$  is the actual utility function  $u_2$  of the opponent. That is, for each  $u \in U$  we want to calculate a probability  $P(u|\pi_1, \pi_2, \dots, \pi_k)$ .

##### 4.1.1.1 Bayesian Learning in General

Bayesian learning is a technique that is much older than automated negotiation and that has been used in many other applications. So, before we explain how it can be applied to automated negotiation, we will here first explain how it works in general.

The goal of Bayesian learning is, given a set of hypotheses  $Y$ , a sequence of observations  $\vec{o} = (o_1, o_2, \dots, o_k)$ , and a *prior probability*  $P(y)$  for each hypothesis  $y \in Y$ , to calculate the *posterior probability*  $P(y|\vec{o})$  that the hypothesis  $y$  is true. Here,  $P(y)$  denotes the probability that we assign to hypothesis  $y$  *before* making any observations, while  $P(y|\vec{o})$  represents the probability we assign to  $y$  *after* making the observations  $o_1, o_2, \dots, o_k$ .

For example, suppose that somebody draws a card from a standard deck of 52 playing cards, without showing it to us. Then, for us, the prior probability that this card is the ace of spades would be  $P(A\spadesuit) = \frac{1}{52}$ . Next, suppose that this person tells us that the card is indeed a *spades* card. Now, with this new information, the probability for us that it is the *ace* of spades is suddenly four times higher:  $P(A\spadesuit | \spadesuit) = \frac{1}{13}$ .

In this example it was straightforward to calculate  $P(y|o)$  directly. However, in practice, it often happens that it is much easier to calculate  $P(o|y)$  instead. In such cases we can use a theorem known as *Bayes' rule* to express  $P(y|o)$  in terms of  $P(o|y)$  and  $P(y)$ .

To derive Bayes' rule, we start from the following identities, which are well-known from basic probability theory, and which hold for any arbitrary 'events'  $y$  and  $o$ :

$$P(y, o) = P(y | o) \cdot P(o) = P(o | y) \cdot P(y) \quad (4.1)$$

$$P(o) = \sum_{y' \in Y} P(o | y') \cdot P(y') \quad (4.2)$$

From Equation (4.1) we can then directly derive:

$$P(y | o) = \frac{P(o | y) \cdot P(y)}{P(o)}$$

and then using Equation (4.2) we obtain Bayes' rule:

$$P(y | o) = \frac{P(o | y) \cdot P(y)}{\sum_{y' \in Y} P(o | y') \cdot P(y')}$$

Note that indeed, this rule allows us to express  $P(y|o)$  on the left-hand side in terms of  $P(o|y)$  and  $P(y)$  on the right-hand side.

If there are multiple observations  $o_1, o_2, \dots, o_k$ , then this becomes:

$$P(y | o_1, o_2, \dots, o_k) = \frac{P(o_1, o_2, \dots, o_k | y) \cdot P(y)}{\sum_{y' \in Y} P(o_1, o_2, \dots, o_k | y') \cdot P(y')} \quad (4.3)$$

and if it holds that for any given hypothesis  $y$ , the probabilities of observations  $o_1, o_2, \dots, o_k$ , are all independent, then we can write this as:

$$P(y | o_1, o_2, \dots, o_k) = \frac{P(o_1|y) \cdot P(o_2|y) \cdot \dots \cdot P(o_k|y) \cdot P(y)}{\sum_{y' \in Y} P(o_1|y') \cdot P(o_2|y') \cdot \dots \cdot P(o_k|y') \cdot P(y')} \quad (4.4)$$

Now, suppose that we have already calculated, for each hypothesis  $y \in Y$ , the probability  $P(y|o_1, o_2, \dots, o_k)$ , which we will denote as  $P(y|\vec{o})$ . Next, suppose we make a new observation  $o_{k+1}$ . We now want to update the probability of each hypothesis, taking into account this new observation. That is, for all  $y \in Y$  we now want to calculate  $P(y|\vec{o}, o_{k+1})$ , given  $P(y|\vec{o})$ .

To do this, first note that the denominator of Eq. (4.4) is just a normalization constant that ensures that the sum of all probabilities equals 1, which is the same for every hypothesis  $y \in Y$ . Ignoring this constant for a moment, we can define the *unnormalized* probability  $\tilde{P}(y|\vec{o})$  as:

$$\tilde{P}(y|\vec{o}) := P(y) \cdot P(o_1|y) \cdot P(o_2|y) \cdot \dots \cdot P(o_k|y) \quad (4.5)$$

which is just the numerator of the right-hand side of Eq. (4.4).

We now see that to update this unnormalized probability after a new observation  $o_{k+1}$  we just need to multiply it with  $P(o_{k+1}|y)$ . That is:

$$\tilde{P}(y|\vec{o}, o_{k+1}) = \tilde{P}(y|\vec{o}) \cdot P(o_{k+1}|y) \quad (4.6)$$

Then, after we have done this for every possible hypothesis  $y \in Y$  we can calculate the true probabilities  $P(y|\vec{o}, o_{k+1})$  by normalizing:

$$P(y|\vec{o}, o_{k+1}) = \frac{\tilde{P}(y|\vec{o}, o_{k+1})}{\sum_{y' \in Y'} \tilde{P}(y'|\vec{o}, o_{k+1})} \quad (4.7)$$

#### 4.1.1.2 Implementation

We will here discuss how Bayesian learning can be implemented.

First determine, for every  $y \in Y$ , the prior probability  $P(y)$ . Since initially we haven't made any observations yet,  $\vec{o}$  will be empty and thus by Eq. (4.5) we have  $\tilde{P}(y|\vec{o}) = P(y)$ , for all  $y \in Y$ .

Then, every time we make a new observation  $o_{k+1}$ , we take the following steps:

1. For each  $y \in Y$ , calculate:

$$\tilde{P}(y|\vec{o}, o_{k+1}) = \tilde{P}(y|\vec{o}) \cdot P(o_{k+1}|y)$$

2. Calculate the sum:

$$S = \sum_{y \in Y} \tilde{P}(y|\vec{o}, o_{k+1})$$

3. For each  $y \in Y$ , calculate:

$$P(y|\vec{o}, o_{k+1}) = \frac{1}{S} \cdot \tilde{P}(y|\vec{o}, o_{k+1})$$

Note that this requires two lists of size  $|Y|$  each: one list to store all the values of  $\tilde{P}(y|\vec{o})$  and one to store the values of  $P(y|\vec{o})$ .

However, this can be done a bit more efficiently. To see how, first note that we can modify the implementation as follows.

Every time we make a new observation  $o_{k+1}$ , we take the following steps:

1. Pick an arbitrary number  $C_{k+1}$ .
2. For each  $y \in Y$ , calculate:

$$\tilde{P}(y|\vec{o}, o_{k+1}) = \tilde{P}(y|\vec{o}) \cdot P(o_{k+1}|y) \cdot C_{k+1}$$

3. Calculate the sum:

$$S = \sum_{y \in Y} \tilde{P}(y|\vec{o}, o_{k+1})$$



4. For each  $y \in Y$ , calculate:

$$P(y|\vec{o}, o_{k+1}) = \frac{1}{S} \cdot \tilde{P}(y|\vec{o}, o_{k+1})$$

Note that the fact that in Step 2 each  $\tilde{P}(y|\vec{o}, o_{k+1})$  is multiplied by a constant  $C_{k+1}$  does not affect the correctness of the calculations, because it means the sum  $S$  in Step 3 will also be multiplied by the same constant, which means that in step 4 this constant will cancel out against itself.

Furthermore, note that every time we make a new observation we can choose a different value for this constant, and that instead of Eq. (4.5), we are now calculating the unnormalized probability  $\tilde{P}(y|\vec{o})$  as:

$$\tilde{P}(y|\vec{o}) = P(y) \cdot C_1 \cdot P(o_1|y) \cdot C_2 \cdot P(o_2|y) \cdot \dots \cdot C_k \cdot P(o_k|y) \quad (4.8)$$

This means that if we choose each  $C_{k+1}$  as follows:

$$C_{k+1} = \frac{1}{\prod_{i=1}^k C_i} \cdot \frac{1}{\sum_{y' \in Y} P(y'|\vec{o})} \quad (4.9)$$

then, by combining Eq. (4.8) and Eq. (4.9) with Eq. (4.3), we see that for every  $y \in Y$  we now have:

$$C_{k+1} \cdot \tilde{P}(y|\vec{o}) = P(y|\vec{o})$$

Knowing this, we can simplify our implementation, since it is now equivalent to the following:

1. For each  $y \in Y$ , calculate:

$$\tilde{P}(y|\vec{o}, o_{k+1}) = P(y|\vec{o}) \cdot P(o_{k+1}|y) \quad (4.10)$$

2. Calculate the sum:

$$S = \sum_{y \in Y} \tilde{P}(y|\vec{o}, o_{k+1})$$

3. For each  $y \in Y$ , calculate:

$$P(y|\vec{o}, o_{k+1}) = \frac{1}{S} \cdot \tilde{P}(y|\vec{o}, o_{k+1})$$

While this looks very similar to our original implementation, the difference is that step 1 now involves  $P(y|\vec{o})$ , rather than  $\tilde{P}(y|\vec{o})$ . The great advantage of this, is that we now only need one list of size  $|Y|$ . In Step 1 we can use

this list to store the values of  $\tilde{P}(y|\vec{o}, o_{k+1})$  and then in Step 3 we can simply overwrite it to store the values of  $P(y|\vec{o}, o_{k+1})$ . In our initial implementation this was not possible, because we needed to keep the values of  $\tilde{P}(y|\vec{o}, o_{k+1})$  for the next iteration. Also note that we do not actually need to calculate the constants  $C_{k+1}$ , since this last implementation does not use them. We only mentioned these constants and Eq. (4.9) to show the correctness of the last implementation.

#### 4.1.1.3 Bayesian Learning for Automated Negotiation

We will now explain how Bayesian Learning can be applied in automated negotiation to learn the utility function of the opponent.

In general, to apply Bayesian learning, we need the following ingredients:

- A set of possible observations  $O$ .
- A set of hypotheses  $Y$ .
- For any hypothesis  $y \in Y$ , a prior probability  $P(y)$ .
- A formula that allows us to calculate, for any hypothesis  $y \in Y$ , and any observation  $o \in O$ , the probability  $P(o | y)$ .

In the context of automated negotiation, the observations that our agent makes are the proposals that it receives from the opponent. Recall that such a proposal  $\pi$  is defined as a tuple of the form  $(2, \mathbf{p}, \omega, t)$  for some offer  $\omega$  and some time  $t$ . So we have:

$$O = \{(2, \mathbf{p}, \omega, t) \mid \omega \in \Omega, t \in [0, T]\}$$

The set of hypotheses would be some set of possible utility functions  $U$  for the opponent. To stress that each hypothesis is now a utility function, we will from now on use the symbol  $U$  to denote the set of hypotheses instead of  $Y$ . We will discuss how to choose these utility functions below in Section 4.1.1.4.

For the prior probabilities, the simplest approach is to assign them all an equal probability. That is:  $P(u) = \frac{1}{|U|}$ . However, depending on the domain of application, you could also choose different prior probabilities that take into account some background knowledge you may have about that specific application.

Finally, we need to determine how to calculate  $P(\pi|u)$  for any arbitrary proposal  $\pi \in O$  and utility function  $u \in U$ . That is, we have to make an assumption about which proposals the opponent would make, if he had utility function  $u$ . In other words, we have to make some assumptions about

his strategy. In order to do this, the authors of [26] modeled the opponent's strategy as a linear time-based strategy. So, at any time  $t$  they *expect* the opponent to propose an offer  $\omega$  with normalized utility  $u_2(\omega) = 1 - c \cdot \frac{t}{T}$ , where  $c$  is some constant between 0 and 1. However, since this is of course not guaranteed to be exactly true, they assumed the opponent's *actual* proposal at any time  $t$  was drawn from the following probability distribution function:

$$P((2, \mathbf{p}, \omega, t) \mid u) = \mathcal{N}(u(\omega) \mid 1 - c \cdot \frac{t}{T}, \sigma) \quad (4.11)$$

where the notation  $\mathcal{N}(r \mid \mu, \sigma)$  represents the probability of drawing the number  $r$  from a Gaussian probability distribution with mean  $\mu$  and standard deviation  $\sigma$ .

With this equation the Bayesian opponent model can be implemented straightforwardly using Equations (4.10) and (4.7). An example implementation is given in Algorithm 7.

Finally, whenever our agent needs to have an estimation  $\hat{u}_2(\omega)$  of the opponent's utility for some offer  $\omega$ , it can be calculated by taking the expectation value over all hypothetical utility functions  $u \in U$ :

$$\hat{u}_2(\omega) = \sum_{u \in U} P(u \mid \vec{\pi}) \cdot u(\omega) \quad (4.12)$$

where  $\vec{\pi}$  is the list of all proposals our agent has so far received from the opponent.

#### 4.1.1.4 Choosing the Utility Hypotheses

We now know how to apply Bayesian learning for some given set of hypothetical utility functions  $U$ . However, we still need to discuss how to choose this set.

To do this, let us first assume that the negotiation domain is a multi-issue domain with  $m$  issues and that we know that the opponent's utility function  $u_2$  is linear, so it can be expressed in the form of Eq. (2.3). Therefore, it can be described in terms of its weights  $w_2^1, w_2^2, \dots, w_2^m$  and its evaluation functions  $v_2^1, v_2^2, \dots, v_2^m$ .

To simplify the notation a bit, in the rest of this section we will suppress the subscript 2 and just write  $w^j$  instead of  $w_2^j$  and  $v^j$  instead of  $v_2^j$ , since we are exclusively talking about the *opponent's* utility anyway.

Furthermore, we will use the notation  $x_{j,l}$  to denote the  $l$ -th option for issue  $I_j$ . For example, if  $I_1$  represents a movie to choose:

$$I_1 = \{The\ Godfather, Casablanca, The\ Big\ Lebowski\}$$

---

**Algorithm 7** Opponent modeling algorithm based on Bayesian learning

---

**Parameters:**

- $\sigma$  ▷ Standard deviation of the Gaussian distribution.
- $c$  ▷ Concession speed of hypothesized opponent strategy.
- $U$  ▷ A set of hypothetical utility functions for the opponent.

**Input:**

- $T$  ▷ The deadline.
- $t$  ▷ The current time.
- $\omega_{rec}$  ▷ The last received offer.
- $probs$  ▷ A map that maps each  $u \in U$  to the probability value  $P(u \mid \pi_1, \pi_2, \dots, \pi_k)$  as calculated in the previous iteration.

// Ensure that we initially assign the same probability to each  
// hypothesis.

```

1: if this is our first turn then
2:   for  $u \in U$  do
3:      $probs[u] \leftarrow \frac{1}{|U|}$ 
4:   end for
5: end if

```

// Update all the values in  $probs$ , given the newly received offer  $\omega_{rec}$   
// and simultaneously calculate the sum of all these values.

```

6:  $sum \leftarrow 0$ 
7: for  $u \in U$  do
8:    $probs[u] \leftarrow probs[u] \cdot \mathcal{N}(u(\omega_{rec}) \mid 1 - c \cdot \frac{t}{T}, \sigma)$ 
9:    $sum \leftarrow sum + probs[u]$ 
10: end for

```

// Ensure that all probabilities are normalized.

```

11: for  $u \in U$  do
12:    $probs[u] \leftarrow probs[u] / sum$ 
13: end for

```

```

14: return  $probs$ 

```

---

Then we have:

$$x_{1,1} = \textit{The Godfather} \quad x_{1,2} = \textit{Casablanca} \quad x_{1,3} = \textit{The Big Lebowski}$$

In addition, if  $v^j$  is the evaluation function of agent  $a_2$  for issue  $I_j$  then we use the notation  $v^{j,l}$  as a shorthand for the value it assigns to option  $x_{j,l}$ . That is:

$$v^{j,l} := v_2^j(x_{j,l})$$

So, to fully specify a linear utility function, we need to specify the value of each weight  $w^j$  and each  $v^{j,l}$ . This means that if the domain has  $m$  issues and each issue has  $s$  options, then we need to specify  $m + m \cdot s$  parameters. For example, if  $m = 4$  and  $s = 3$ :

$$\begin{aligned} w^1 &= 0.3, & w^2 &= 0.5, & w^3 &= 0.1, & w^4 &= 0.1 \\ v^{1,1} &= 0.0, & v^{2,1} &= 0.3, & v^{3,1} &= 0.3, & v^{4,1} &= 1.0 \\ v^{1,2} &= 0.4, & v^{2,2} &= 0.7, & v^{3,2} &= 0.0, & v^{4,2} &= 1.0 \\ v^{1,3} &= 1.0, & v^{2,3} &= 0.9, & v^{3,3} &= 0.0, & v^{4,3} &= 0.2 \end{aligned}$$

Now, one way to select a finite set of hypothetical utility functions, is to restrict each of these parameters to only have values in some finite domain, such as the set  $\{0, 0.1, 0.2, \dots, 0.9, 1.0\}$ . Since this set has 11 possible values, this gives us a total of  $11^{m+m \cdot s}$  possible utility functions. Unfortunately, however, this is an astronomically large number, even for small domains with only  $m = 3$  and  $s = 4$ . This is a problem because, as can be seen in Algorithm 7, we need to loop over all elements of  $U$ , which is clearly unfeasible for such a large set.

The authors of [26] therefore made some simplifying assumption to decrease this number. For example, they assumed that all issues are *ordered* sets, and that the evaluation functions are *triangular*. That is, if  $x_{j,n}$  denotes  $a_2$ 's most preferred option of issue  $I_j$ , then they assume the evaluation function  $v^j$  first increases linearly from 0 to 1 until the option  $x_{j,n}$  is reached, after which it decreases linearly from 1 to 0. Figure 4.1 displays a few examples of such functions. Formally, for any issue  $I_j$  with size  $s_j := |I_j|$  and any integer  $n$  with  $1 \leq n \leq s_j$ , the triangular function  $\Lambda_j^n$  is defined as:

$$\Lambda_j^n(x_{j,l}) = \begin{cases} \frac{l-1}{n-1} & \text{if } l < n \\ 1 & \text{if } l = n \\ \frac{s_j-(l-1)}{s_j-(n-1)} & \text{if } l > n \end{cases} \quad (4.13)$$

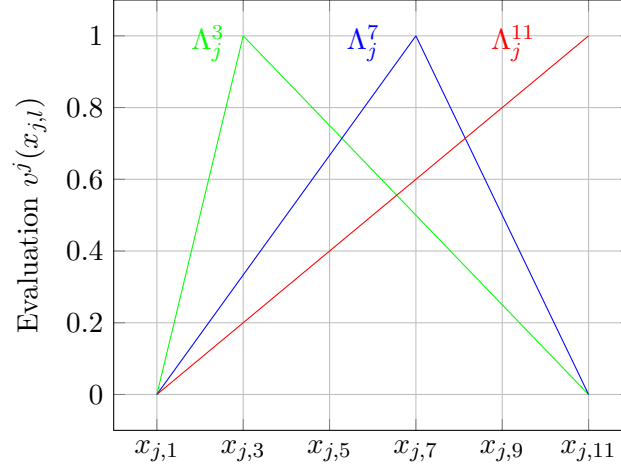


Figure 4.1: Some examples of triangular evaluation functions for an issue  $I_j$  with 11 options.

This assumption of triangular evaluation functions greatly reduces the size of the set  $U$ , because now to specify a single evaluation function  $v^j$ , we only need to specify the most preferred option  $x_{j,n} \in I_j$ , rather than specifying a number  $v^{j,l}$  for every single option  $x_{j,l} \in I_j$ . This reduces the number of possible evaluation functions for  $I_j$  from  $11^{s_j}$  to just  $s_j$ . And therefore it reduces the total number of utility functions to  $11^m \cdot s^m$  (if all issues have the same size  $s$ ).

With these reductions the set  $U$  becomes small enough to apply Bayesian learning in practice to small domains with just a few issues. However, since the set  $U$  still grows exponentially with the number of issues, this approach is still not feasible scenarios with many issues. Luckily however, the authors of [26] also proposed a more scalable version of Bayesian opponent modeling, which we will discuss next.

**Exercise 8. Bayesian Learning.** Implement the Bayesian learning algorithm discussed above. Next, run some negotiations with your time-based agent and/or Tit-for-Tat agent from Exercises 2 and 4, but using this new opponent modeling algorithm, instead of the DummyOpponentUtilityModel.

### 4.1.2 Scalable Bayesian Learning

Before we explain the scalable version of Bayesian learning *for automated negotiation*, let us first take a step back and focus again on the general case.

Let us assume we have some set of hypotheses  $Y$  and that each hypothesis  $y \in Y$  can be decomposed into a number of sub-hypotheses:  $y = (y_1, y_2, \dots, y_m)$ , so the hypothesis space can be decomposed as the Cartesian product of a number of sub-hypothesis spaces:  $Y = Y^1 \times Y^2 \times \dots \times Y^m$ .

For example, the hypothesis that a given playing card is the ace of spaces can be written as  $y = (A, \spadesuit)$ .

Now, the probability  $P(y \mid \vec{o})$  can be written as:

$$P(y \mid \vec{o}) = \prod_{j=1}^m P(y_j \mid \vec{o})$$

and the Bayesian update rule (4.10) can be applied to each sub-hypothesis separately:

$$\tilde{P}(y_j \mid \vec{o}, o_{k+1}) = P(y_j \mid \vec{o}) \cdot P(o_{k+1} \mid y_j) \quad (4.14)$$

The question, now, is how to calculate  $P(o_{k+1} \mid y_j)$ . After all, we typically need the full hypothesis  $y$  to be able to calculate the probability of some observation.

Before answering that question, let us first return to the topic of automated negotiation. In the previous section we have seen that each hypothesis  $y$  corresponds to a utility function  $u$ , which is defined by a number of parameters: for each issue  $I_j$  a weight  $w^j$  and an evaluation function  $v^j$ .

This means that the hypothesis space can be written as:

$$Y = Y_w^1 \times Y_w^2 \times \dots \times Y_w^m \times Y_v^1 \times Y_v^2 \times \dots \times Y_v^m$$

where each  $Y_w^j$  is a set of possible values for weight  $w^j$ , and each  $Y_v^j$  is a set of possible evaluation functions defined over issue  $I_j$ .

For example, if we assume that each weight must be an integer multiple of 0.1 and must be between 0 and 1, then we have:

$$Y_w^1 = Y_w^2 = \dots = Y_w^m = \{0, 0.1, 0.2, \dots, 0.9, 1.0\}$$

Similarly, if we assume that each evaluation function must be a triangular function (See Eq. (4.13)), then for each  $Y_v^j$  we have:

$$Y_v^j = \{\Lambda_j^1, \Lambda_j^2, \dots, \Lambda_j^{s_j}\}$$

where  $s_j$  is the size of issue  $I_j$ .

So a hypothesis  $y$  is now a tuple  $(w^1, w^2, \dots, w^m, v^1, v^2, \dots, v^m)$ , where each  $w^j$  is a value from the set of weight hypotheses  $Y_w^j$  and each  $v^j$  is an evaluation function from the set of evaluation hypotheses  $Y_v^j$ . Furthermore, each such hypothesis  $y$  corresponds to a utility function  $u_y$ :

$$u_y(\omega) := \sum_{j=1}^m w^j \cdot v^j(\omega)$$

Recall from Sec. 2.2.3.3 that we may abuse notation by writing  $v^j(\omega)$  when we actually mean  $v^j(x_j)$ , where  $x_j$  is the  $j$ -th component of  $\omega$ .

For a given hypothesis  $y$  and a given sequence of received proposals  $\vec{\pi}$  we can now express the posterior probability as:

$$P(y|\vec{\pi}) = \prod_{j=1}^m P(w^j|\vec{\pi}) \cdot \prod_{j=1}^m P(v^j|\vec{\pi})$$

and each probability  $P(w^j|\vec{\pi})$  and  $P(v^j|\vec{\pi})$  can be updated separately. For example, for each weight  $w^j$  the update rule (4.10) now becomes:

$$\tilde{P}(w^j|\vec{\pi}, \pi_{k+1}) = P(w^j|\vec{\pi}) \cdot P(\pi_{k+1}|w^j) \quad (4.15)$$

and similarly, for the evaluation functions  $v^j$ :

$$\tilde{P}(v^j|\vec{\pi}, \pi_{k+1}) = P(v^j|\vec{\pi}) \cdot P(\pi_{k+1}|v^j) \quad (4.16)$$

Note that these two equations are just special cases of Eq. (4.14), specific to automated negotiation. So, our original question how to calculate  $P(o_{k+1} | y_j)$  can now be reformulated as the question how to calculate  $P(\pi_{k+1}|w^j)$  and  $P(\pi_{k+1}|v^j)$ .

To answer this, we first define for each issue  $I_j$  its *expected* weight  $\bar{w}^j$  and its *expected* evaluation function  $\bar{v}^j$  as follows:

$$\bar{w}^j := \sum_{w^j \in Y_w^j} w^j \cdot P(w^j | \vec{\pi}) \quad (4.17)$$

$$\bar{v}^j(\omega) := \sum_{v^j \in Y_v^j} v^j(\omega) \cdot P(v^j | \vec{\pi}) \quad (4.18)$$

which in turn can be used to define the expected utility function  $\bar{u}$ :

$$\bar{u}(\omega) := \sum_{j=1}^m \bar{w}^j \cdot \bar{v}^j(\omega) \quad (4.19)$$



Next, this allows us to define, for any issue  $I_l$  and weight-hypothesis  $w^l \in Y_w^l$  a function  $\bar{u}_{[w^l]}$  as follows:

$$\bar{u}_{[w^l]}(\omega) := \sum_{\substack{j=1 \\ j \neq l}}^m \bar{w}^j \cdot \bar{v}^j(\omega) + w^l \cdot \bar{v}^l(\omega)$$

That is,  $\bar{u}_{[w^l]}(\omega)$  is the utility value calculated by taking, for each issue  $I_j$ , the *expectation* value of the weight  $w^j$ , and the expectation value of  $v^j(\omega)$ , except for issue  $I_l$ , for which we use the hypothesized weight  $w^l$ .

Similarly, we can define:

$$\bar{u}_{[v^l]}(\omega) := \sum_{\substack{j=1 \\ j \neq l}}^m \bar{w}^j \cdot \bar{v}^j(\omega) + \bar{w}^l \cdot v^l(\omega)$$

Then, for any  $w^j \in Y_w^j$  we can calculate  $P(\pi_{k+1}|w^j)$  as in Eq. (4.11). but with the variable  $u$  replaced by  $\bar{u}_{[w^j]}$ . That is:

$$P((2, \mathbf{p}, \omega, t) | w^j) := \mathcal{N}(\bar{u}_{[w^j]}(\omega) | 1 - c \cdot \frac{t}{T}, \sigma) \quad (4.20)$$

Similarly,  $P(\pi_{k+1}|v^j)$  can be calculated as:

$$P((2, \mathbf{p}, \omega, t) | v^j) := \mathcal{N}(\bar{u}_{[v^j]}(\omega) | 1 - c \cdot \frac{t}{T}, \sigma) \quad (4.21)$$

See Algorithm 8 for an implementation.

It should be noted, however, that these equations are just approximations. They are based on the assumption that the current expected utility function  $\bar{u}$  is already a good approximation to the opponent's true utility function  $u_2$ .

While scalable Bayesian learning largely solves the problem of scalability, the main disadvantage is that we need to make a lot of assumptions. For example, we need to assume that the opponent's utility function is linear, that the issues are ordered and that the opponent has triangular evaluation functions. Furthermore, it depends on the chosen model of the opponent's bidding strategy and on the chosen standard deviation  $\sigma$  for the Gaussian distribution.

---

**Algorithm 8** Opponent modeling algorithm based on Scalable Bayesian learning. This function is called every time a new proposal is received, in order to update our agent's model of the opponent's utility function.

---

**Parameters:**

- $\sigma$   $\triangleright$  Standard deviation of the Gaussian distribution.
- $c$   $\triangleright$  Concession speed of hypothesized opponent strategy.

**Input:**

- $T$   $\triangleright$  The deadline.
- $t$   $\triangleright$  The current time.
- $\omega_{rec}$   $\triangleright$  The last received offer.
- $weight\_hyps$   $\triangleright$  A double array that contains for each issue  $I_j$  a list of possible weights. So,  $weight\_hyps[j]$  is a single array that represents  $Y_w^j$ .
- $weight\_probs$   $\triangleright$  A double array that contains for each issue  $I_j$  and each possible weight  $w^j \in Y_w^j$  a probability value  $P(w^j|\vec{\pi})$ .
- $eval\_hyps$   $\triangleright$  A double array that contains for each issue  $I_j$  a list of possible evaluation functions. So,  $eval\_hyps[j]$  is a single array that represents  $Y_v^j$ .
- $eval\_probs$   $\triangleright$  A double array that contains for each issue  $I_j$  and each possible evaluation function  $v^j \in Y_v^j$  a probability value  $P(v^j|\vec{\pi})$ .

```

// Calculate the values of  $\bar{w}^j$  and  $\bar{v}^j(\omega_{rec})$  according to Eqs. (4.17) and (4.18)
1: for each issue  $I_j$  of the domain do
2:    $\bar{w}^j \leftarrow \sum_{l=1}^{|Y_w^j|} weight\_hyps[j][l] \cdot weight\_probs[j][l]$ 
3:    $\bar{v}^j \leftarrow \sum_{l=1}^{|Y_v^j|} eval\_hyps[j][l] \cdot eval\_probs[j][l](\omega_{rec})$ 
4: end for

5: for each issue  $I_j$  of the domain do

6:   for  $l \in \{0, 1, \dots, |Y_w^j| - 1\}$  do
7:      $\bar{u}_{[w^j]} \leftarrow \sum_{i=1, i \neq j}^m \bar{w}^i \cdot \bar{v}^i + weight\_hyps[j][l] \cdot \bar{v}^j$ 
8:      $weight\_probs[j][l] \leftarrow$ 
        $weight\_probs[j][l] \cdot \mathcal{N}(\bar{u}_{[w^j]} \mid 1 - c \cdot \frac{t}{T}, \sigma)$  // Eq. (4.15)
9:   end for
10:   $normalize(weight\_probs[j])$ 

11:  for  $l \in \{0, 1, \dots, |Y_v^j| - 1\}$  do
12:     $\bar{u}_{[v^j]} \leftarrow \sum_{i=1, i \neq j}^m \bar{w}^i \cdot \bar{v}^i + \bar{w}^j \cdot eval\_hyps[j][l](\omega_{rec})$ 
13:     $eval\_probs[j][l] \leftarrow$ 
       $eval\_probs[j][l] \cdot \mathcal{N}(\bar{u}_{[v^j]} \mid 1 - c \cdot \frac{t}{T}, \sigma)$  // Eq. (4.16)
14:  end for
15:   $normalize(eval\_probs[j])$ 

16: end for
17: return ( $weight\_probs, eval\_probs$ )

```

---

**Exercise 9. Scalable Bayesian Learning.** Implement the scalable Bayesian learning algorithm discussed in this section. Next, run some negotiations with your time-based agent and/or Tit-for-Tat agent from Exercises 2 and 4, but using this new opponent modeling algorithm, instead of the dummy opponent model or the regular Bayesian learning algorithm from Exercise 8.

### 4.1.3 Frequency Analysis

In this section we will discuss a simpler alternative to Bayesian learning, called *frequency analysis*, which is based on the idea that the opponent's evaluation functions and weights can be estimated from the frequency with which the opponent proposes the respective options for each issue. While this method is perhaps not as elegant or sophisticated as Bayesian learning, it turns out that in practice it often performs equally well, or even better [5].

The basic idea of frequency analysis is that for any issue  $I_j$  and any option  $x_{j,l} \in I_j$  of that issue, the value  $v_2^j(x_{j,l})$  that the opponent assigns to it can be estimated from the number of times that the opponent makes proposals containing that option.

For example, in the scenario that Alice and Bob are negotiating about a visit to the cinema, if Alice keeps making proposals that include the movie *The Godfather*, then that is a clear indication that Alice probably likes that movie very much.

Furthermore, to estimate the opponent's weights  $w_2^j$ , the idea is that if the opponent proposes many different options for the same issue  $I_j$ , then this is an indication that that issue is probably not very important to the opponent, so the weight  $w_2^j$  should have a low value.

For example, if Alice first proposes to see the movie at 18:00, but then proposes to see it at 20:00, and then proposes to see it at 22:00, then apparently she does not really care much about the time at which the movie starts.

As usual, there are many ways how these ideas can be implemented. As an example, we here present the implementation by van Galen Last [41].<sup>1</sup>

Let  $k$  denote the total number of proposals made by the opponent:

$$k := |\{(i, \eta, \omega, t) \in h \mid i = 2 \wedge \eta = \mathbf{p}\}|$$

<sup>1</sup>The cited paper itself actually does not explain this opponent modeling algorithm, but it can be found in the source code of their agent, which can be found at <https://tracinsy.ewi.tudelft.nl/pubtrac/Genius/browser/src/main/java/agents/anac/y2010/AgentSmith>

and let  $x_{j,l}$  denote the  $l$ -th option for issue  $I_j$ . Furthermore, let  $f_h(x_{j,l})$  denote the number of times that the opponent has proposed an offer that contained  $x_{j,l}$ :

$$f_h(x_{j,l}) := |\{(i, \eta, \omega, t) \in h \mid i = 2 \wedge \eta = \mathbf{p} \wedge x_{j,l} \in \omega\}|$$

Then, each value  $v_2^j(x_{j,l})$  can be estimated as the number of times the option  $x_{j,l}$  has been proposed by the opponent, divided by the total number of proposals made by the opponent:

$$\hat{v}_2^j(x_{j,l}) = \frac{f_h(x_{j,l})}{k}$$

and each weight  $w_2^j$  can be estimated as:

$$\hat{w}_2^j = \frac{\max \{f_h(x_{j,l}) \mid x_{j,l} \in I_j\}}{k}$$

Note that this approach in general will not yield a normalized utility function, so you may optionally still want to apply some normalization to these weights and evaluation functions.

**Exercise 10. Frequency Analysis.** Implement the frequency analysis algorithm discussed in this section. Next, run some negotiations with your time-based agent and/or Tit-for-Tat agent from Exercises 2 and 4, but using this new opponent modeling algorithm.

## 4.2 Learning the Opponent's Strategy

In this section we will discuss how to model the opponent's bidding strategy, based on the proposals he makes during the negotiations. More precisely, given the set of proposals that our agent received from the opponent until time some time  $t$ , we aim to predict which offers the opponent will propose later on, between time  $t$  and the deadline.

The ability to make such predictions is essential for the implementation of an adaptive negotiation strategy, as explained in Section 3.2.2.

To formalize this, let

$$\pi_1 = (2, \mathbf{p}, \omega_1, t_1), \quad \pi_2 = (2, \mathbf{p}, \omega_2, t_2), \quad \dots, \quad \pi_k = (2, \mathbf{p}, \omega_k, t_k)$$

denote the sequence of proposals that our agent has received from its opponent and let  $z_1, z_2, \dots, z_k$  denote their corresponding utility values, for *our* agent. That is:

$$z_i := u_1(\omega_i)$$

Then our goal is to implement an algorithm that can take as its input the sequence

$$(z_1, t_1), (z_2, t_2), \dots, (z_k, t_k)$$

plus some arbitrary time  $t_{k+1}$  in the future, and that outputs a prediction for the corresponding utility value  $z_{k+1}$ .

However, in general it is unlikely that we can make such a prediction perfectly, so rather than outputting the actual value  $z_{k+1}$ , a typical opponent modeling algorithm would instead output a probability distribution  $P(z_{k+1})$  over all the possible values of  $z_{k+1}$ .

Many different techniques to do this have been proposed in the literature. For example, Agent K [29], the winner of ANAC 2010, used an extrapolation algorithm based on the average and standard deviation of the values of  $z_i$ . Other agents used non-linear regression (IAMhaggler [45]), or wavelet decomposition and cubic smoothing splines (OMAC [12]). Here, however, we will only focus on the technique of Gaussian Processes (IAMHaggler2011 [44]).

### 4.2.1 Gaussian Processes

Due to the technical nature of this topic we cannot discuss Gaussian processes in detail, so will only give a global idea of how this technique works. For a more detailed discussion we refer to [42] or [11].

The idea behind Gaussian processes is that we assume that at any given time the probability that the opponent will propose an offer  $\omega$  with utility  $u_1(\omega) = z$  is given by a Gaussian distribution:

$$P(z) = \mathcal{N}(z \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

Now, in order to be able to use this for our purposes, we first need to determine an expression for the probability that the opponent proposes a certain *sequence* of offers with utility values  $z_1, z_2, \dots, z_k$  respectively.

If we could assume that each offer is drawn *independently* from the same normal distribution, then this would be easy, as we could simply multiply the probabilities. This would yield the following expression:

$$P(z_1, z_2, \dots, z_k) = \frac{1}{(2\pi)^{k/2}} \cdot \frac{1}{\sigma^k} \cdot e^{-\frac{(z_1-\mu)^2 + (z_2-\mu)^2 + \dots + (z_k-\mu)^2}{2\sigma^2}}$$

which can be rewritten using vector-notation:

$$P(\vec{z}) = \frac{1}{(2\pi)^{k/2}} \cdot \frac{1}{\sigma^k} \cdot e^{-\frac{1}{2\sigma^2}(\vec{z}-\vec{\mu})^T \mathbf{I}(\vec{z}-\vec{\mu})} \quad (4.22)$$

where  $\mathbf{I}$  is the  $k \times k$  identity matrix and  $\vec{\mu} = (\mu, \mu, \dots, \mu)^T$  is the  $k$ -dimensional column vector containing just  $k$  copies of the number  $\mu$ .

However, the offers proposed by the opponent are typically not independent. After all, it is fair to assume that the opponent is following some negotiation strategy that concedes over time with respect to his utility  $u_2$  and that this utility function is at least to some extent correlated with our own utility  $u_1$ .

For example, in the extreme case that the opponent follows a strictly monotonic bidding strategy and that the negotiation domain is a split-the-pie domain, then our agent would perceive the offers it receives from the opponent as strictly increasing over time, i.e.  $z_1 \leq z_2 \leq \dots \leq z_k$ . So, their values are clearly not independent.

Of course, in practice many negotiation scenarios will not be split-the-pie domains in which the utility functions are that strongly correlated. Nevertheless, it is still reasonable to assume that there will at least be some correlation. In fact, we have to make this assumption, because if there is no correlation between the two utility functions at all, then there would be no way for our agent to make any predictions based on the received proposals. After all, the utility values of the received proposals would just appear as a completely random sequence with no pattern whatsoever.

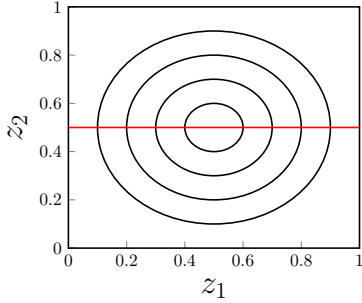
We will therefore assume that, *in general*, two consecutive proposals  $\pi_i$  and  $\pi_{i+1}$  will often have similar values:  $z_i \approx z_{i+1}$ . To state this more formally, we will assume that the closer two proposals  $\pi_i$  and  $\pi_j$  are to each other in time, the stronger the correlation between the corresponding random variables  $z_i$  and  $z_j$ .

Whenever a sequence of Gaussian random variables is not independent, we can model their joint distribution by replacing the identity matrix in Eq. (4.22) with some other matrix  $\mathbf{K}$  (which has to be symmetric and positive semi-definite) so that the expression for the joint probability becomes:

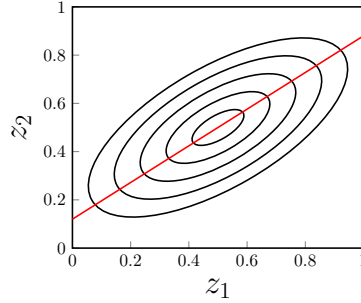
$$P(\vec{z}) = \frac{1}{(2\pi)^{k/2}} \cdot \frac{1}{|\mathbf{K}|^{1/2}} \cdot e^{-\frac{1}{2}(\vec{z}-\vec{\mu})^T \mathbf{K}^{-1}(\vec{z}-\vec{\mu})} \quad (4.23)$$

where  $|\mathbf{K}|$  is the determinant of  $\mathbf{K}$ .

The fact that this matrix indeed introduces a dependency between each pair of variables  $z_i$  and  $z_j$  can be seen clearly from Figure 4.2. In this



(a) Countour plot of a multi-variate Gaussian distribution with identity matrix.



(b) Countour plot of multi-variate Gaussian distribution with alternative covariance matrix.

Figure 4.2: Multi-variate Gaussian distributions.

figure we have drawn two contour plots for a Gaussian distribution over just two variables  $z_1$  and  $z_2$ . Figure 4.2a shows the case where  $\mathbf{K}$  is just the identity matrix, so this corresponds to Eq. (4.22). We see that for any arbitrary value of  $z_1$ , the probability distribution for  $z_2$  is maximized at the same value  $z_2 = 0.5$  (indicated with a red line). Similarly, for any value of  $z_2$  the probability distribution for  $z_1$  is maximized at the same value  $z_1 = 0.5$ . In other words, the probability distribution for  $z_2$  does not depend on  $z_1$  and vice versa.

On the other hand, in Figure 4.2b, where we have drawn the contour plot of a Gaussian distribution with an alternative matrix  $\mathbf{K}$  we see that as  $z_1$  increases, the value of  $z_2$  with maximum probability also increases (again, indicated with a red line). That is, the larger the value of  $z_1$ , the greater the expectation value of  $z_2$ .

Furthermore, note that if we use Eq. (4.23) to calculate the covariance  $\mathbb{E}\left((z_i - \mu) \cdot (z_j - \mu)\right)$  between any two variables  $z_i$  and  $z_j$  then the result will be exactly the element  $K_{i,j}$  of the matrix  $\mathbf{K}$ . For this reason,  $\mathbf{K}$  is called the *covariance* matrix. From this it follows immediately that if  $\mathbf{K}$  is the identity matrix, then there is no covariance among any two different variables  $z_i$  and  $z_j$ , which means that they are indeed independent.

The question now, is how to choose the correct matrix  $\mathbf{K}$ . For this, we use a so-called *kernel* function. A kernel function is a function  $\kappa : \mathbb{R}^2 \rightarrow \mathbb{R}$  that represents how the correlation between any two variables  $z_i$  and  $z_j$  depends on the times  $t_i$  and  $t_j$ . That is, we set:

$$K_{i,j} := \kappa(t_i, t_j) \quad (4.24)$$

where  $K_{i,j}$  is an entry of the matrix  $\mathbf{K}$ , representing the covariance between variables  $z_i$  and  $z_j$ , and  $t_i$  and  $t_j$  are the times of the proposals  $\pi_i$  and  $\pi_j$ .

Of course, we have now only replaced our original question “*How do we select the correct covariance matrix?*” by a new question: “*How do we select the correct kernel function?*”.

We will not go into the details of how to select the best such kernel function. We will just mention that it should be consistent with our requirement that the smaller the difference between  $t_i$  and  $t_j$ , the more the two variables  $z_i$  and  $z_j$  should be correlated. So, this should be reflected in the kernel function: the smaller  $|t_i - t_j|$ , the greater  $\kappa(t_i, t_j)$ . Furthermore, let us mention that Williams et al. [42] used a so-called *Matérn* kernel.

Once we have determined the covariance matrix, we know the expression for  $P(\vec{z})$ . The next step, is to use this to calculate an expression for  $P(z_{k+1} \mid z_1, z_2, \dots, z_k)$ . This is indeed the expression that we are looking for, because it calculates the probability of some future value  $z_{k+1}$ , given the observed sequence  $z_1, z_2, \dots, z_k$ .

The expression for  $P(z_{k+1} \mid z_1, z_2, \dots, z_k)$  can be obtained directly from the expression for  $P(\vec{z})$  using straightforward, but somewhat tedious, algebra. We will not go into the details of this calculation here, but the key point is that  $P(z_{k+1} \mid z_1, z_2, \dots, z_k)$  will again be a Gaussian distribution. Therefore, this distribution is determined by just two parameters  $\mu$  and  $\sigma$ , representing the mean and standard deviation.

Note that, technically, the probability  $P(z_{k+1} \mid z_1, z_2, \dots, z_k)$  also depends on the times  $t_1, t_2, \dots, t_k$ , of the received proposals, as well as on the chosen future time  $t_{k+1}$ , because they determine the covariance matrix  $\mathbf{K}$ , through the kernel function  $\kappa$ , as in Eq. (4.24). We may therefore write this probability more correctly as  $P(z_{k+1} \mid \pi_1, \pi_2, \dots, \pi_k, t_{k+1})$ .

Finally, let us mention that instead of using *all* received proposals from the opponent as their input, Williams et al. [42] divided time into a number of time-windows and only used the proposal with highest utility from each time window. This has the advantage that it reduces noise in the data, and it also reduces the size of the input data, which in turn reduces the required computation time.

#### 4.2.2 Choosing the Optimal Target Value for an Adaptive Negotiation Strategy

The typical use case for Gaussian processes, is to determine an optimal target value  $\beta^*$  for an adaptive negotiation strategy. Let us here explain in more detail how that can be done.



In order to do this, we first have to select a time point  $t_{k+1}$  which is close to the deadline  $T$ . This will allow us to predict the utility value of the last offer that the opponent will propose to us. The output of our Gaussian process algorithm will then consist of the two parameters  $\mu$  and  $\sigma$ , which are the mean and the standard deviation of the Gaussian probability distribution that represents the probability that the opponent will propose an offer  $\omega_{k+1}$  at time  $t_{k+1}$  with utility  $z_{k+1}$ :

$$P(z_{k+1} \mid \pi_1, \pi_2, \dots, \pi_k, t_{k+1}) = \mathcal{N}(z_{k+1} \mid \mu, \sigma)$$

Now, let us suppose for a moment that we know the exact value  $z_{k+1}$  of the offer  $\omega_{k+1}$  that the opponent will propose at time  $t_{k+1}$ , and furthermore that we have a good approximation  $\hat{u}_2$  of the opponent's utility function, so we can ensure that our own proposals are Pareto-optimal. In that case we can assume that the opponent will accept any Pareto-optimal offer  $\omega$  for which  $u_1(\omega) < z_{k+1}$ . After all, if, *for our agent*, the offer  $\omega$  is worse than the offer  $\omega_{k+1}$  that the opponent would propose, then by Pareto-optimality, *for the opponent*, the offer  $\omega$  would be *better* than the offer  $\omega_{k+1}$  that he would propose. So, it is fair to assume that the opponent would be willing to accept  $\omega$ .

Of course, in reality we only have a *probability distribution* for  $z_{k+1}$ , so we can calculate, for any offer  $\omega$  with utility  $u_1(\omega) = z$  the *probability* that the opponent will accept it, by integrating over all values of  $z_{k+1}$  that are greater than  $z$ . That is:

$$P_a(z) = \int_z^\infty P(z_{k+1} \mid \pi_1, \pi_2, \dots, \pi_k) dz_{k+1}$$

where  $P_a(z)$  denotes the probability that  $a_2$  would accept an offer  $\omega$  with utility  $u_1(\omega) = z$ .

Let us now make the pessimistic assumption that if our target value is  $\beta$ , then we will indeed need to concede all the way to that value and we will not be able to get any agreement with higher utility than that. Therefore, our expected utility would be given by  $\beta \cdot P_a(\beta)$ . That is, the utility  $\beta$  in case of agreement, multiplied by the probability that the opponent will indeed accept such an agreement. We can now determine our optimal target value  $\beta^*$  as follows:

$$\beta^* = \arg \max_{\beta} \beta \cdot P_a(\beta)$$

### **4.3 Learning the Opponent's Strategy from Previous Negotiation Sessions**

COMING SOON!

## Chapter 5

# Game Theory

### 5.1 Normal-Form Games

COMING SOON!

### 5.2 Extensive-Form Games

COMING SOON!

### 5.3 Automated Negotiation as a Game

COMING SOON!

### 5.4 Bargaining Solutions

COMING SOON!



## Chapter 6

# Evaluation of Negotiation Algorithms

COMING SOON!



## Chapter 7

# Advanced Negotiations

### 7.1 Multilateral Negotiation

COMING SOON!

### 7.2 Negotiation and Search

COMING SOON!

### 7.3 Non-linear and Computationally Complex Utility Functions

COMING SOON!





# Bibliography

- [1] Bo An and Victor R. Lesser. Yushu: A heuristic-based agent for automated negotiating competition. In Takayuki Ito, Minjie Zhang, Valentin Robu, Shaheen Fatima, and Tokuro Matsuo, editors, *New Trends in Agent-Based Complex Automated Negotiations*, volume 383 of *Studies in Computational Intelligence*, pages 145–149. Springer, 2012.
- [2] R Axelrod and WD Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.
- [3] Reyhan Aydoğan, Tim Baarslag, Katsuhide Fujita, Johnathan Mell, Jonathan Gratch, Dave de Jonge, Yasser Mohammad, Shinji Nakadai, Satoshi Morinaga, Hirotaka Osawa, Claus Aranha, and Catholijn Jonker. Challenges and main results of the automated negotiating agents competition (anac) 2019. In *Multi-Agent Systems and Agreement Technologies. 17th International Conference EUMAS 2020 and 7th International Conference AT 2020. Thessaloniki, Greece September 14-15, 2020. Revised Selected Papers*, Cham, 2020. Springer International Publishing.
- [4] Reyhan Aydoğan, David Festen, Koen V Hindriks, and Catholijn M Jonker. Alternating offers protocols for multilateral negotiation. *Modern approaches to agent-based complex automated negotiation*, pages 153–167, 2017.
- [5] Tim Baarslag, Mark Hendrikx, Koen V. Hindriks, and Catholijn M. Jonker. Predicting the performance of opponent models in automated negotiation. In *2013 IEEE/WIC/ACM International Conferences on Intelligent Agent Technology, IAT 2013, 17-20 November 2013, Atlanta, Georgia, USA*, pages 59–66. IEEE Computer Society, 2013.
- [6] Tim Baarslag, Koen Hindriks, Mark Hendrikx, Alexander Dirkzwager, and Catholijn Jonker. Decoupling negotiating agents to explore

- the space of negotiation strategies. In Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Takayuki Ito, Minjie Zhang, Quan Bai, and Katsuhide Fujita, editors, *Novel Insights in Agent-based Complex Automated Negotiation*, pages 61–83. Springer Japan, Tokyo, 2014.
- [7] Tim Baarslag, Koen V. Hindriks, and Catholijn M. Jonker. Acceptance conditions in automated negotiation. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 95–111. Springer, 2013.
- [8] Tim Baarslag, Koen V. Hindriks, and Catholijn M. Jonker. A tit for tat negotiation strategy for real-time bilateral negotiations. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 229–233. Springer, 2013.
- [9] Tim Baarslag, Koen V. Hindriks, Catholijn M. Jonker, Sarit Kraus, and Raz Lin. The first automated negotiating agents competition (ANAC 2010). In *New Trends in Agent-Based Complex Automated Negotiations*, volume 383 of *Studies in Computational Intelligence*, pages 113–135. Springer, Berlin, Heidelberg, 2012.
- [10] Jasper Bakker, Aron Hammond, Daan Bloembergen, and Tim Baarslag. RLBOA: A modular reinforcement learning framework for autonomous negotiating agents. In Edith Elkind, Manuela Veloso, Noa Agmon, and Matthew E. Taylor, editors, *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, pages 260–268. International Foundation for Autonomous Agents and Multiagent Systems, 2019.
- [11] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.
- [12] Siqi Chen and Gerhard Weiss. An efficient and adaptive approach to negotiation in complex environments. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31, 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 228–233, Amsterdam, The Netherlands, 2012. IOS Press.

- [13] John P Conley and Simon Wilkie. An extension of the nash bargaining solution to nonconvex problems. *Games and Economic behavior*, 13(1):26–38, 1996.
- [14] Dave de Jonge. An analysis of the linear bilateral ANAC domains using the MiCRO benchmark strategy. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 223–229. ijcai.org, 2022.
- [15] Dave de Jonge. Theoretical properties of the MiCRO negotiation strategy. *Autonomous Agents and Multi-Agent Systems*, 38(46), 2024.
- [16] Dave de Jonge, Tim Baarslag, Reyhan Aydoğan, Catholijn Jonker, Katsuhide Fujita, and Takayuki Ito. The challenge of negotiation in the game of diplomacy. In Marin Lujak, editor, *Agreement Technologies, 6th International Conference, AT 2018, Bergen, Norway, December 6-7, 2018, Revised Selected Papers*, volume 11327 of *Lecture Notes in Computer Science*, pages 100–114, Cham, 2019. Springer International Publishing.
- [17] Dave de Jonge, Filippo Bistaffa, and Jordi Levy. A heuristic algorithm for multi-agent vehicle routing with automated negotiation. In Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé, editors, *AA-MAS ’21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 404–412. ACM, 2021.
- [18] Dave de Jonge, Filippo Bistaffa, and Jordi Levy. Multi-objective vehicle routing with automated negotiation. *Applied Intelligence*, 52(14):16916–16939, Nov 2022.
- [19] Dave de Jonge and Carles Sierra. NB3: a multilateral negotiation algorithm for large, non-linear agreement spaces with limited time. *Autonomous Agents and Multi-Agent Systems*, 29(5):896–942, 2015.
- [20] Dave de Jonge and Carles Sierra. D-Brane: a diplomacy playing agent for automated negotiations research. *Applied Intelligence*, 47(1):158–177, 2017.
- [21] Ulle Endriss. Monotonic concession protocols for multilateral negotiation. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss,

- and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, Hakodate, Japan, May 8-12, 2006, pages 392–399. ACM, 2006.
- [22] Peyman Faratin, Carles Sierra, and Nicholas R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159 – 182, 1998. Multi-Agent Rationality.
  - [23] Katsuhide Fujita, Reyhan Aydoğan, Tim Baarslag, Koen Hindriks, Takayuki Ito, and Catholijn Jonker. The sixth automated negotiating agents competition (anac 2015). In *Modern Approaches to Agent-based Complex Automated Negotiation*, pages 139–151. Springer International Publishing, Cham, 2017.
  - [24] Katsuhide Fujita, Reyhan Aydoğan, Tim Baarslag, Takayuki Ito, and Catholijn M. Jonker. The fifth automated negotiating agents competition (ANAC 2014). In *Recent Advances in Agent-based Complex Automated Negotiation [revised and extended papers from the 7th International Workshop on Agent-based Complex Automated Negotiation, ACAN 2014, Paris, France, May 2014]*, volume 638 of *Studies in Computational Intelligence*, pages 211–224, Cham, 2014. Springer International Publishing.
  - [25] Maria Jose Herrero. The nash program: non-convex bargaining problems. *Journal of Economic Theory*, 49(2):266–277, 1989.
  - [26] Koen V. Hindriks and Dmytro Tykhonov. Opponent modelling in automated multi-issue negotiation using bayesian learning. In Lin Padgham, David C. Parkes, Jörg P. Müller, and Simon Parsons, editors, *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal, May 12-16, 2008, Volume 1, pages 331–338. IFAAMAS, 2008.
  - [27] Takayuki Ito, Mark Klein, and Hiromitsu Hattori. A multi-issue negotiation protocol among agents with nonlinear utility functions. *Multiagent Grid Syst.*, 4:67–83, January 2008.
  - [28] Ehud Kalai and Meir Smorodinsky. Other solutions to nash’s bargaining problem. *”Econometrica”*, ”43”(3):513–518, 1975.
  - [29] Shogo Kawaguchi, Katsuhide Fujita, and Takayuki Ito. Compromising strategy based on estimated maximum utility for automated negotiation agents competition (ANAC-10). In *Modern Approaches in Applied*

- Intelligence - 24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2011, Syracuse, NY, USA, June 28 - July 1, 2011, Proceedings, Part II*, volume 6704 of *Lecture Notes in Computer Science*, pages 501–510, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [30] 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*, 30(1):48–70, 2014.
- [31] Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Juan R. Velasco, and Enrique de la Hoz. Effective bidding and deal identification for negotiations in highly nonlinear scenarios. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1057–1064, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.
- [32] Johnathan Mell, Jonathan Gratch, Tim Baarslag, Reyhan Aydogan, and Catholijn M. Jonker. Results of the first annual human-agent league of the automated negotiating agents competition. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents, IVA 2018, Sydney, NSW, Australia, November 05-08, 2018*, pages 23–28, New York, NY, USA, 2018. Association for Computing Machinery.
- [33] Yasser Mohammad, Enrique Areyan Viqueira, Nahum Alvarez Ayerza, Amy Greenwald, Shinji Nakadai, and Satoshi Morinaga. Supply chain management world - A benchmark environment for situated negotiations. In Matteo Baldoni, Mehdi Dastani, Beishui Liao, Yuko Sakurai, and Rym Zalila-Wenkstern, editors, *PRIMA 2019: Principles and Practice of Multi-Agent Systems - 22nd International Conference, Turin, Italy, October 28-31, 2019, Proceedings*, volume 11873 of *Lecture Notes in Computer Science*, pages 153–169. Springer, 2019.
- [34] Yasser Mohammad, Shinji Nakadai, and Amy Greenwald. Negmas: A platform for automated negotiations. In Takahiro Uchiya, Quan Bai, and Ivan Marsá-Maestre, editors, *PRIMA 2020: Principles and Practice of Multi-Agent Systems - 23rd International Conference, Nagoya, Japan, November 18-20, 2020, Proceedings*, volume 12568 of *Lecture Notes in Computer Science*, pages 343–351. Springer, 2020.

- [35] J.F. Nash. The bargaining problem. *"Econometrica"*, "18":155–162, 1950.
- [36] Thuc Duong Nguyen and Nicholas R. Jennings. Coordinating multiple concurrent negotiations. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, pages 1064–1071. IEEE Computer Society, 2004.
- [37] Bram Renting, Dave de Jonge, Holger Hoos, and Catholijn Jonker. Analysis of learning agents in automated negotiation. Under review.
- [38] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. The MIT Press, Cambridge, USA, 1994.
- [39] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica: Journal of the Econometric Society*, pages 97–109, 1982.
- [40] Ayan Sengupta, Yasser Mohammad, and Shinji Nakadai. An autonomous negotiating agent framework with reinforcement learning based strategies and adaptive strategy switching mechanism. In Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé, editors, *AA-MAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 1163–1172. ACM, 2021.
- [41] Niels van Galen Last. Agent smith: Opponent model estimation in bilateral multi-issue negotiation. In Takayuki Ito, Minjie Zhang, Valentin Robu, Shaheen Fatima, and Tokuro Matsuo, editors, *New Trends in Agent-Based Complex Automated Negotiations*, volume 383 of *Studies in Computational Intelligence*, pages 167–174. Springer, 2012.
- [42] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.
- [43] Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. Using gaussian processes to optimise concession in complex negotiations against unknown opponents. In Toby Walsh, editor, *IJ-CAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 432–438. IJCAI/AAAI, 2011.

- [44] Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. Using gaussian processes to optimise concession in complex negotiations against unknown opponents. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, July 16-22, 2011*, pages 432–438, Barcelona, Catalonia, Spain, 2011. IJCAI.
- [45] Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. Iamhaggler: A negotiation agent for complex environments. In *New Trends in Agent-Based Complex Automated Negotiations*, volume 383 of *Studies in Computational Intelligence*, pages 151–158. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [46] Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. An overview of the results and insights from the third automated negotiating agents competition (ANAC2012). In Ivan Marsá-Maestre, Miguel A. López-Carmona, Takayuki Ito, Minjie Zhang, Quan Bai, and Katsuhide Fujita, editors, *Novel Insights in Agent-based Complex Automated Negotiation*, volume 535 of *Studies in Computational Intelligence*, pages 151–162. Springer, 2014.