Introduction to Automated Negotiation

Dave de Jonge

IIIA-CSIC, Barcelona, Spain
Universitat Autònoma de Barcelona, Spain

11 April 2026
v0.5

Contents

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. Also note that since this is still only a preliminary version of the final book, some notations or definitions may change in future versions of this book, or may have changed with respect to earlier versions.

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`

If you have any questions or comments on this book, please send me an e-mail: `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!

To cite this book, please use the following BibTeX:

```
@book{deJonge2025IntroToNego,
        title = "Introduction to Automated Negotiation",
        author = "de Jonge, Dave",
        year = "2025",
        publisher = "IIIA-CSIC",
        address = "Barcelona, Spain",
        doi = "10.48550/arXiv.2511.08659",
        url = "https://www.iiia.csic.es/~davedejonge/intro_to_nego"
    }
```

Summary of Notation

Basic Negotiations

$\mathbb{R}$	The set of real numbers.
$\mathbb{R}^+$	The set of positive real numbers.
$\mathbb{N}$	The set of natural numbers (including 0).
ag_i	Agent i
Ω	The set of all offers in a given negotiation domain.
ω	Offer
I_j	Issue
s_j	The size of issue I_j, i.e. $s_j := \|I_j\|$.
x_j	Option
$x_{j,l}$	The l-th option of issue I_j.
t	Time
$(i, \mathfrak{p}, \omega, t)$	Agent ag_i proposes offer ω at time t.
$(i, \mathfrak{a}, \omega, t)$	Agent ag_i accepts offer ω at time t.
η	Action type, i.e. a variable that can either adopt the value $\mathfrak{p}$ or the value $\mathfrak{a}$.
T	Deadline
N	The maximum number of rounds in a negotiation.
ϵ	Delay, i.e. the difference between the time a proposal or acceptance was sent by one agent and the time it was received by the other agent.
h	Negotiation history or action history
h_i^o	Observed negotiation- or action- history (observed by agent ag_i)
u_i	Utility function of agent ag_i.
ω_i^{max}	The most preferred offer by agent ag_i
ω_i^{min}	The least preferred offer by agent ag_i
u_i^{max}	The utility value, for ag_i of ag_i's most preferred offer.
u_i^{min}	The utility value, for ag_i of ag_i's least preferred offer.
v_i^j	Evaluation function for agent ag_i and issue I_j
w_i^j	Weight for agent ag_i and issue I_j
rv_i	Reservation value of agent ag_i (Def. 2.2.7).
δ	Discount factor.
D	Negotiation domain (Def. 2.2.8)
$\vec{u}(\omega)$	Utility vector of offer ω.
Ω^p	Pareto set, i.e. the set of all Pareto-optimal offers (Def. 2.3.4).
$opp(D)$	The amount of 'opposition' of a domain D.

Negotiation Strategies

ω_{rec}	The last offer that our agent has received from the opponent.
ω_{next}	The offer that our agent is about to propose next.
$\mathcal{M}$	Opponent model
$\lambda(t)$	Aspiration level at time t.
$\hat{u}_2$	Estimation of ag_2's utility function, as estimated by ag_1's opponent modeling algorithm.
Ω_t^{prop}	The set off all offers that have already been proposed by ag_1 before time t.
α	Initial value of the aspiration function, i.e. $\lambda(0)$.
β	Target value, i.e. the final value of the aspiration function: $\lambda(T)$.
γ	The concession parameter.
T'	Target time.
β^*	Optimal target value for our agent, based on predictions of our opponent's future proposals.
Ω_t^{rec}	The set of offers that have been *received* by agent ag_1 up until time t.
c_i	Function that measures the amount of concession made by agent ag_i.
2^Ω	Power set of the set of offers (i.e. the set of all subsets of Ω).
Δc_t	'concession gain' of agent 1 at time t.
θ_{min}	minimum required concession gain for tit-for-tat strategy.
θ_{max}	maximum required concession gain for tit-for-tat strategy.

Opponent Modeling

$\mathcal{U}$	Some set of possible utility functions.
π_j	Proposal
$\vec{\pi}$	Sequence of proposals.
$P(u\|\pi_1, \pi_2, \dots, \pi_k)$	The probability that our opponent has utility function u, given that our agent has received proposals $\pi_1, \pi_2, \dots, \pi_k$ from that opponent.
y	Hypothesis
Y	Set of possible hypotheses.
o	Observation
O	Set of possible observations.
$\vec{o}$	Sequence of observations.
$P(y\|\vec{o})$	Probability that hypothesis y holds, given the sequence of observations $\vec{o}$.
$P(o\|y)$	Probability of making observation o when hypothesis y holds.
$\tilde{P}$	Unnormalized probability.
$\mathcal{N}(r\|\mu, \sigma)$	Probability of drawing the number r from a Gaussian probability distribution with mean μ and standard deviation σ.
Λ_j^n	Triangular function over issue I_j, with peak at option $x_{j,n}$ (see Eq. (4.13)).
$\overline{w}^j$	Expectation value for the weight that the opponent assigns to issue I_j.
$\overline{v}^j$	Expected evaluation function that the opponent applies to issue I_j.
$\overline{u}$	Expected utility function for the opponent.
$f_h(x_{j,l})$	The number of times the opponent has proposed an offer containing option $x_{j,l}$.
z_j	Shorthand for the utility offered to us by the opponent in her j-th proposal to us, i.e.: $z_j := u_1(\omega_j)$.
$\vec{z}$	Sequence of offered utilities, i.e. $\vec{z} = (z_1, z_2, \dots)$
$\mathbf{I}$	Identity matrix.
$\mathbf{K}$	Covariance matrix.
$K_{i,j}$	Element of the covariance matrix at row i and column j.
κ	Kernel function.
$P_{\mathfrak{a}}(z)$	Probability that ag_2 would accept an offer ω with utility $u_1(\omega) = z$.

Game Theory

a	Action
A_i	The set of actions available to player i.

G	Normal-form game.
$BR_j(a_i)$	The set of actions that are a best response for agent j, against some action a_i of its opponent.
m	Mixed strategy
$\mathcal{M}_i$	The set of all mixed strategies for player i
$\vec{m}$	Strategy profile (of mixed strategies).
$BR_j(m_i)$	The set of mixed strategies that are a best response for agent j, against some mixed strategy m_i of its opponent.
NE	The set of all Nash equilibria of a normal-form game.
$\mathcal{G}$	A set of 2-player normal-form games.
$\mathcal{F}_A$	A 'role frequency function', i.e. $\mathcal{F}_A(G, i)$ is a number that represents the relative frequency in which an agent is going to be (or is expected to be) playing game G in the role of player i.
$\mathcal{U}_A, \mathcal{U}_B$	Meta-utility functions of agents ag_A and ag_B.
X^*	The set of tuples over some set X.
X^n	The n-fold Cartesian product of some set X, i.e. $X^1 = X$, $X^2 = X \times X, \quad X^3 = X \times X \times X, \quad$ etc...
ε	The 'empty tuple'.
$\circ$	The concatenation operator for tuples, e.g. $(a, b) \circ (c, d, e) = (a, b, c, d, e)$.
Y^T	The set of terminal tuples among some set of tuples Y.
ν	Tree node.
d	Depth of a tree node.
Γ	Turn-taking game.
$\mathcal{H}$	The set of all legal action histories of a turn-taking game.
pl	The active player function of a turn-taking game.
$\mathcal{H}_i$	The set of all non-terminal histories after which player i is the active player.
A_h	The set of actions that the active player is allowed to choose after history h.
σ	Strategy for a turn-taking game.
$\vec{\sigma}$	Strategy profile for a turn-taking game.
$h_{\vec{\sigma}}$	The unique terminal history generated by strategy profile $\vec{\sigma}$.
$\mathcal{S}_i$	The set of all possible strategies for player i in some turn-taking game.
Γ_h	The subgame of Γ at history h.
$\mathcal{H}_h$	The set of legal action histories of the subgame Γ_h.
f_i^{obs}	The observation function of player i.

O_i	The set of all possible observed histories after which it is player i's turn.
A_i^D	The set of negotiation actions for player i in negotiation domain D.
Υ_D	The utility space of a negotiation domain D.
Υ_D^p	The Pareto frontier of a negotiation domain D.
ks	The 'Kalai-Smorodinsky function' (Eq. 5.2).
$L(D)$	The line from the point (rv_1, rv_2) to the 'utopian' point (u_1^{max}, u_2^{max}).

Evaluation of Negotiation Algorithms

Π	Negotiation protocol.
$\mathcal{D}$	Set of negotiation domains.
Ag	Set of agents.
$ag_{\underline{i}}$	The i-th agent from the set of agents Ag in a tournament.
$\mathcal{R}$	Function that maps each possible negotiation scenario to its number of repetitions in the tournament.
$R^{d,\underline{i},\underline{j}}$	Shorthand for $\mathcal{R}(D_d, ag_{\underline{i}}, ag_{\underline{j}})$.
$u_1^{d,\underline{i},\underline{j},r}$	Utility obtained by agent ag_k in the r-th negotiation between agents $ag_{\underline{i}}$ and $ag_{\underline{j}}$ over domain D_d.
$U_{\underline{i}}^{d,\underline{j}}$	Average utility obtained by agent $ag_{\underline{i}}$ against opponent $ag_{\underline{j}}$ when negotiating over domain D_d.
$U_{\underline{i}}$	Tournament score of agent $ag_{\underline{i}}$.
$\mathbb{1}_{agr(d,\underline{i},\underline{j},r)}$	Indicator function that has value 1 if the r-th repetition of scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$ ended with agreement, and 0 otherwise.
$NA^{d,\underline{i},\underline{j}}$	Number of times a negotiation in the scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$ ended with agreement.
$AR_{\underline{i}}$	Agreement rate of agent $ag_{\underline{i}}$.
$UA_{\underline{i}}$	Utility-under-agreement of agent $ag_{\underline{i}}$.
$\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$	Random variables.
$S_{\mathcal{X}}$	The set of possible values of the random variable $\mathcal{X}$.
$P_{\mathcal{X}}$	Probability distribution of $\mathcal{X}$.
$\mu_{\mathcal{X}}$	Mean of $\mathcal{X}$.
$Var_{\mathcal{X}}$	Variance of $\mathcal{X}$.
$\sigma_{\mathcal{X}}$	Standard deviation of $\mathcal{X}$.
$\hat{\mu}_{\mathcal{X}}$	Estimated mean of $\mathcal{X}$.
$\hat{Var}_{\mathcal{X}}$	Estimated variance of $\mathcal{X}$ (a.k.a. sample variance).

$\hat{\sigma}_{\mathcal{X}}$	Estimated standard deviation of $\mathcal{X}$ (a.k.a. sample standard deviation).
$se_{\mu_{\mathcal{X}}}$	Standard error on the mean of $\mathcal{X}$.
$\hat{se}_{\mu_{\mathcal{X}}}$	Estimated standard error on the mean of $\mathcal{X}$.
sc	Negotiation scenario
Sc	Set of negotiation scenarios.
$u_{\underline{i},s,r}$	The r-th observation from random variable $X_{\underline{i},s}$.
$\hat{\mu}_l^{d,\underline{i},\underline{j}}$	Average utility obtained by agent ag_l in the negotiations between $ag_{\underline{i}}$ and $ag_{\underline{j}}$ over domain D_d.
$\mathfrak{t}$	t-statistic
t_{k-1}	The t-distribution with $k-1$ degrees of freedom.

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 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 had, 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 costumer 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 were 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 [40]. 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 [32, 29, 14].

This changed in 1998 with the seminal paper by Faratin et al. [25]. Rather then 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 [35] 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 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 with non-linear utility functions and with an extremely large number of possible agreements [31, 36]. 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 [21, 22, 20].

Other researchers have focused on multi-lateral negotiations (negotiations between 3 or more agents) [41, 24, 21, 4], or the use of machine learning algorithms such as deep learning and reinforcement learning to train negotiation algorithms [47, 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 [28], while from 2015 to 2018 ANAC focused on multi-lateral negotiations[27]. 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 [43]. 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 [18], multi-lateral negotiations in a supply-chain environment [38] negotiations between computers and humans [37], 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 [15] and theoretically [17], 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 [39] 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) [44], 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:

1. Which movie they will see.
2. Where they will see this movie (in which cinema).
3. 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 ag_1 and ag_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 Ω. 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 ω.

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 ω 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 see a movie together, can be modeled as a negotiation over the following three issues, representing the movie, the cinema, and the time slot, respectively:

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

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 instead have 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:

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

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, Cinema Rialto might only screen *The Godfather* on Saturdays, and Cinema 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 Ω would only be a *subset* of the Cartesian product $I_1 \times I_2 \times \cdots \times I_m$. However, in most of the literature such constraints are not taken into account and one typically 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 ag_1.

At the start of the negotiations, agent ag_1 can choose any $\omega \in \Omega$ from the offer space and propose it to ag_2. Next it is agent ag_2's turn. Agent ag_2 can now either accept the previous proposal from ag_1, or propose an alternative offer $\omega' \in \Omega$. If ag_2 accepts the previous offer ω then the negotiations are over and ω will be considered a formally binding agreement. Otherwise, if ag_2 does not accept ω and instead makes a new proposal, then we say that ag_2 **rejects** the offer ω. Next, it is again ag_1's turn. This time, ag_1 can choose between accepting the previously received proposal ω', or rejecting it and proposing a new offer ω'' from the offer space Ω.

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 $(i, \mathfrak{p}, \omega, t)$ to indicate that agent ag_i proposes offer ω at time t, and we will use the notation $(i, \mathfrak{a}, \omega, t)$ to indicate that agent ag_i accepts offer ω at time t. We follow the convention that $t = 0$ represents the time at which the negotiations start.

Definition 2.2.1. *We define a* ***negotiation action*** *to be a tuple*

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

where i represents the index of the agent performing the action, and η represents the ***type*** *of the action, which can be either the symbol $\mathfrak{p}$ ('propose'), or the symbol $\mathfrak{a}$ ('accept'). Furthermore, ω 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 a negotiation action for which $\eta = \mathfrak{p}$ and we define an* ***acceptance*** *as a negotiation action for which $\eta = \mathfrak{a}$.*

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 of action does not play an important role in the rest of this book, we prefer not to include it here, to keep the formalization simple.

Whenever two agents are negotiating with each other, they obviously need to be connected to each other through some communication channel such as the Internet or a local network. This means that whenever one agent proposes an offer, it will take some time, due to network latency, for the other agent to receive that proposal. Since this delay is typically unpredictable, we will model it as a random variable denoted ϵ. This motivates the following definition.

Definition 2.2.2. *A* ***negotiation history*** *h is a finite list that alternates between negotiation actions a_j and positive real numbers $\epsilon_j \in \mathbb{R}^+$:*

$$h = \Big((i_1, \eta_1, \omega_1, t_1) \ , \ \epsilon_1 \ , \ (i_2, \eta_2, \omega_2, t_2) \ , \ \epsilon_2 \ , \ (i_3, \eta_3, \omega_3, t_3) \ , \ \epsilon_3 \ , \ \dots \Big)$$

such that the negotiation actions appear in chronological order (i.e. for all j we have $t_j \leq t_{j+1}$).

In this definition, each ϵ_j represents the time it takes for the action $(i_j, \eta_j, \omega_j, t_j)$ to be received by the other agent. So, a proposal made at time t_j will be received by the other agent at time $t_j + \epsilon_j$. Each ϵ_j is assumed to be drawn independently from some probability distribution.

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

$$
\begin{aligned}
h &= \Big(a_1\ ,\ \epsilon_1\ ,\ a_2\ ,\ \epsilon_2,\quad \dots\quad ,\ a_9\ ,\ \epsilon_9\ ,\ a_{10}\ ,\ \epsilon_{10}\ \Big) \\
&= \Big((1,\mathfrak{p},\omega_1,t_1),\ \epsilon_1,\ (2,\mathfrak{p},\omega_2,t_2),\ \epsilon_2,\ \dots\ ,\ (1,\mathfrak{p},\omega_9,t_9),\ \epsilon_9,\ (2,\mathfrak{a},\omega_9,t_{10}),\ \epsilon_{10}\Big) \\
&= \Big((1,\mathfrak{p},(x_1^1,x_1^2,x_1^3),t_1),\ \epsilon_1,\ \ (2,\mathfrak{p},(x_2^1,x_2^2,x_2^3),t_2),\ \epsilon_2, \\
&\qquad \dots\ ,\ \ (1,\mathfrak{p},(x_9^1,x_9^2,x_9^3),t_9),\ \epsilon_9,\ \ (2,\mathfrak{a},(x_9^1,x_9^2,x_9^3),t_{10}),\ \epsilon_{10}\Big)
\end{aligned}
$$

where each a_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 ag_2 accepts the offer ω_9 that was proposed by ag_1 directly before that.

We can now formally define the AOP as follows.

Definition 2.2.3. *We say a negotiation history h satisfies the AOP (with deadline T and maximum number of rounds N) if and only if all of the following conditions hold:*

1. *For any two consecutive negotiation actions $a_j = (i_j, \eta_j, \omega_j, t_j)$ and $a_{j+1} = (i_{j+1}, \eta_{j+1}, \omega_{j+1}, t_{j+1})$ in h, we have:*
   - *(a) $i_j \neq i_{j+1}$, and*
   - *(b) $t_j + \epsilon_j < t_{j+1}$*
2. *A negotiation action with $\eta = \mathfrak{a}$ can only appear as the last action in the negotiation history.*
3. *If (i, η, ω, t) and (i', η', ω', t') are the second-last and last actions of the negotiation history respectively and $\eta' = \mathfrak{a}$, then we must have $\omega = \omega'$.*
4. *For all negotiation actions (i, η, ω, t) in h we have $t \leq T$.*
5. *The history h can contain at most N negotiation actions.*

The first rule says that the two agents have to alternate turns and that an agent can only propose or accept an offer after it has received the previous proposal from the other agent. The second rule says that the negotiations are over as soon as one agent accepts an offer. The third rule says that an agent can only accept the offer from the *previous* proposal and not from any earlier proposals. The fourth 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.

Definition 2.2.4. *Let h be a negotiation history that satisfies the AOP and let $a_k = (i_k, \eta_k, \omega_k, t_k)$ be the last negotiation action of this history. Then, the AOP defines that the negotiation has ended in* ***agreement*** *if $\eta_k = \mathfrak{a}$ and $t_k + \epsilon_k < T$. In that case we say that ω_k is the* ***accepted offer****. Otherwise, we say the negotiations have* ***failed****.*

Note that this means that even if an agent accepts an offer before the deadline, the negotiations may still fail if the other agent does not *receive* this acceptance message before the deadline.

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

It is important to note that each individual agent cannot observe the delays. That is, if agent 1 proposes an offer, then he will only know the time t at which he proposed the offer, but he will not know the time $t + \epsilon$ at which the offer was received by agent 2. On the other hand, agent 2 will only observe the time $t + \epsilon$ at which she received that proposal, but she will not know the exact time t at which it was sent. In other words, each of the agents only has a partial view of the negotiation history, and neither of them knows the full history h. This motivates the following definition.

Definition 2.2.5. *An* ***observed negotiation history*** *is a list of negotiation actions, sorted in chronological order (i.e. in order of increasing values of t). Specifically, if h is a negotiation history:*

$$h = \Big((1, \eta_1, \omega_1, t_1) \ , \ \epsilon_1 \ , \ (2, \eta_2, \omega_2, t_2) \ , \ \epsilon_2 \ , \ (3, \eta_3, \omega_3, t_3) \ , \ \epsilon_3 \ , \ \dots \Big)$$

then the corresponding observed negotiation history h_1^o for agent 1, is:

$$h_1^o = \Big((1, \eta_1, \omega_1, t_1) \ , \ (2, \eta_2, \omega_2, t_2 + \epsilon_2) \ , \ (3, \eta_3, \omega_3, t_3) \ , \ \dots \Big)$$

while the corresponding observed negotiation history h_2^o for agent 2 is:

$$h_2^o = \Big((i_1, \eta_1, \omega_1, t_1 + \epsilon_1) \ , \ (i_2, \eta_2, \omega_2, t_2) \ , \ (i_3, \eta_3, \omega_3, t_3 + \epsilon_3) \ , \ \dots \Big)$$

So, if h is the true negotiation history, then agents 1 and 2 will only be aware of their respective *observed* histories h_1^o and h_2^o.

In the rest of this book we will often just use the term ‘history’ or ‘negotiation history’ when we actually mean an *observed* negotiation history, because it should be clear from the context what we mean.

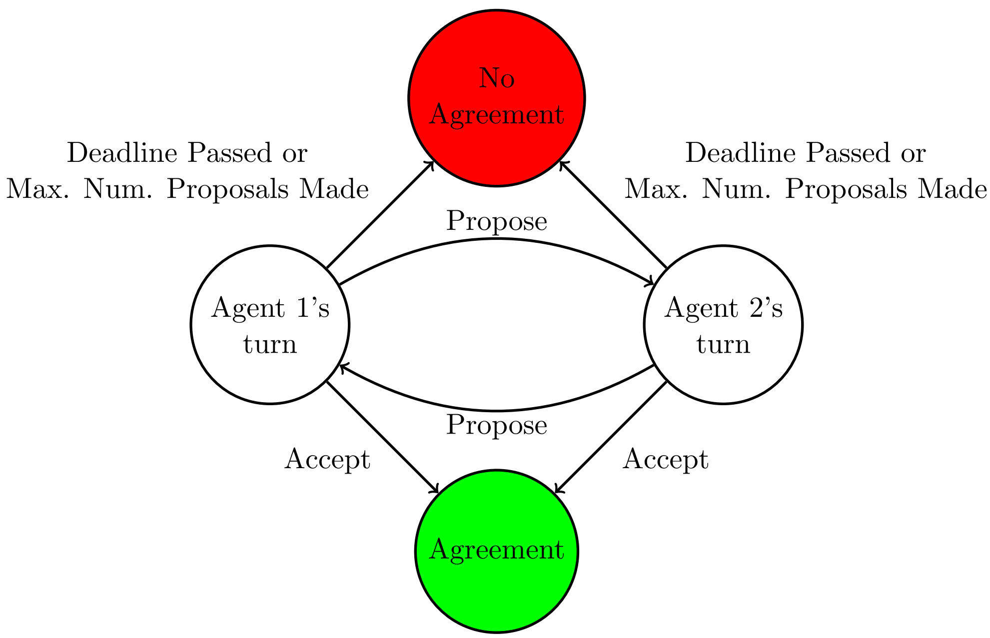


Figure 2.1: The alternating offers protocol as a finite-state machine.

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 distinction is not really necessary. So, in this book we 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 real-world negotiation, who would impose such a deadline onto the two agents? However, we can imagine that in a real-world scenario each agent ag_i *itself* has its own individual deadline T_i, which may be determined by various external factors. In that case, we can simply define the global deadline T as the individual deadline that comes earliest. That is: $T := \min\{T_1, T_2\}$. We can imagine that before the negotiations begin each agent announces his personal deadline T_i to the other agent, so that both agents will be aware of the global deadline T.

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, 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.

Opinion. I personally have never been a fan of negotiations with a fixed maximum number of proposals N. This is because I can't really imagine any real-world situation in which the two negotiators would face such a constraint and in which the number N would be known to both negotiators in advance. The only similar scenario I can imagine, is that either of the negotiators 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 negotiators 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 represent the probability that the human would be too tired to continue after N rounds.

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 ω 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 an agent ag does want to accept an offer ω that was proposed by the opponent ag' in an earlier round, then instead agent ag can simply propose that offer again itself. Since the opponent ag' already proposed it earlier, there are good reasons to believe that ag' will now be willing to accept it (more about this later in Section 3.4).

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 buy 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 value 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 book it will turn out useful to use the notation ω_i^{max} for the offer most preferred by agent ag_i, and the notation ω_i^{min} for the offer least preferred by agent ag_i:

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

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

Furthermore, we will use the notation u_i^{max} and u_i^{min} to denote the corresponding utility values of the most preferred and least preferred offers:

$$u_i^{max} := u_i(\omega_i^{max}) = \max_{\omega \in \Omega}\{u_i(\omega)\} \tag{2.3}$$

$$u_i^{min} := u_i(\omega_i^{min}) = \min_{\omega \in \Omega}\{u_i(\omega)\} \tag{2.4}$$

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 ag_i aims to maximize $\sum_{k=1}^{n} u_i(\omega_k)$, where u_i is the utility function of the agent and ω_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 does not make any difference if we use the utility function $u_i' = a \cdot u_i + b$ instead of u_i. Any set of agreements that is optimal under u_i will also be optimal under u_i'.

Definition 2.2.6. *The principle of* ***Invariance under Linear Transformations*** *says that if an agent ag_i has a von Neumann Morgenstern utility function u_i, then that agent should not behave any differently if instead it had a utility function $u_i' = au_i + b$, where $a, b \in \mathbb{R}$ can be any arbitrary real numbers, as long as $a > 0$.*

The principle of Invariance under Linear Transformation implies that if the offer space Ω is finite, then we can always normalize the utility function such that the offer with highest utility has utility value $u_i^{max} = 1$ and the offer with lowest utility has utility value $u_i^{min} = 0$. 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.

$$u_i' := \frac{u_i - u_i^{min}}{u_i^{max} - u_i^{min}}$$

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 ag_1 and ag_2 with respective utility functions u_1 and u_2. Furthermore, suppose that agent ag_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 ag_1 as follows:

$$u_1' \quad := \quad u_1 + u_2$$

We now see that, even though ag_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:

$$\omega \quad = \quad (x_1, x_2, \ldots, x_m) \quad \in \quad I_1 \times I_2 \times \cdots \times I_m$$

and each v_i^j is a function that maps issue I_j to the real numbers: $v_i^j : I_j \to \mathbb{R}$. 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 ag_i to which it belongs.

Alternatively, linear utility functions are often written as:

$$u_i(\omega) = \sum_{j=1}^{m} w_i^j \cdot v_i^j(x_j) \tag{2.5}$$

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 ω 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:

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

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 ω. That is:

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

2.2.4 Reservation Values

In many real negotiation scenarios it may happen that some proposals are so bad that you would rather not to make any agreement at all than to accept such a proposal.

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 2.2.7. *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 ag_i is guaranteed to get. After all, ag_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 yields more utility than 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**.

One thing you may be wondering now, is what an agent should do when it receives an offer ω for which the utility is exactly *equal* to the reservation value, i.e. $u_i(\omega) = rv_i$. We argue that in that case the agent should also reject the offer. After all, if he accepts the offer he will certainly receive rv_i, while if he rejects it, he is also guaranteed to obtain at least rv_i, but on top of that he also still has the possibility to get a better deal later and thus obtain more utility.

Observation. *A rational agent ag_i should never accept any offer ω for which his utility $u_i(\omega)$ is smaller than or equal to his reservation value rv_i.*

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 ω 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 δ 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 ag_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.

Opinion. 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 [45] 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 still 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* [40] 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 [17].

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 [19] and [20] 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 2.2.8. *A* ***negotiation domain*** *D for n agents consists of the following components:*

- *An offer space Ω.*
- *For each $i \in \{1, 2, \ldots, n\}$:*
  - *a utility function $u_i : \Omega \to \mathbb{R}$*
  - *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 2.2.9. *In a negotiation domain for n agents, each offer ω corresponds to an n-tuple which we call the* ***utility vector*** *and which consists of the utility values of all agents:*

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

We may also denote this vector as $\vec{u}(\omega)$.

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 ω 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 ag_1 has a reservation value of $rv_1 = 0.1$, then we draw a vertical line at $x = 0.1$ and if agent ag_2 has a reservation value of $rv_1 = 0.2$, 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 ω when we technically mean the *utility vector* of that offer.

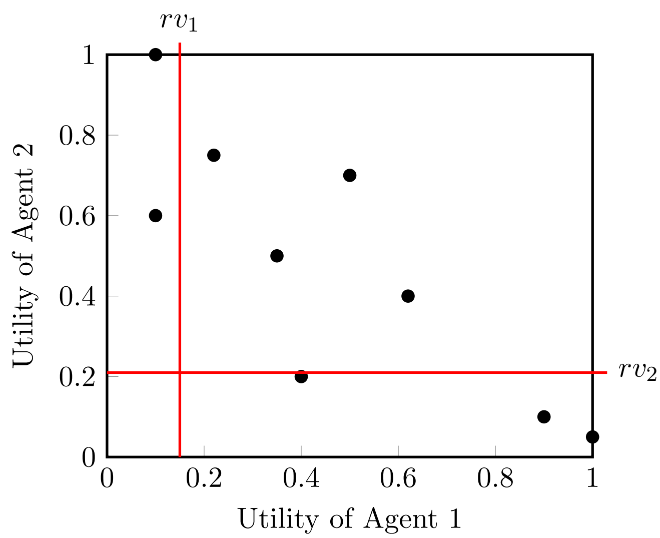


Figure 2.2: Utility space diagram. Every dot is the utility vector of one offer ω in the offer space Ω. The red lines represent the reservation values of the two respective agents.

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 ag_1 gets, say, 40% of the pie then her utility is 0.4 and therefore ag_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

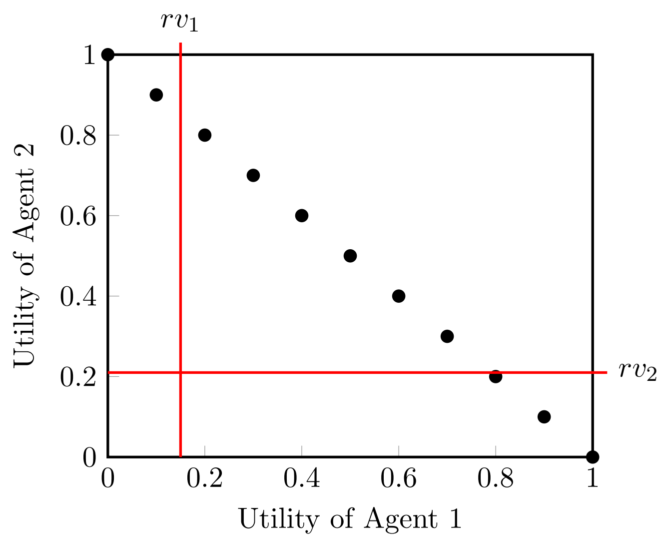


Figure 2.3: Utility space diagram of a split-the-pie domain. Note that all utility vectors lie on the line $y = 1 - x$.

various different issues. However, this is somewhat misleading.

Of course, if you compare a single-issue domain D_1 that contains 10 different offers, with a multi-issue domains 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 D_2 with a single-issue domain D_3 of the same size (i.e. with 1,000 offers), then I would even say that the single-issue domain D_3 is more complex, especially if the utility functions of D_2 are linear. After all, in that case, to describe the utility functions of 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 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.

One could 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, because we would have a full preference ordering over all offers in such a domain.

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 or equal to its reservation value. This motivates the definition of individual rationality.

Definition 2.3.1. *In any negotiation domain an offer ω is said to be* ***rational for agent*** *ag_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 ω 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

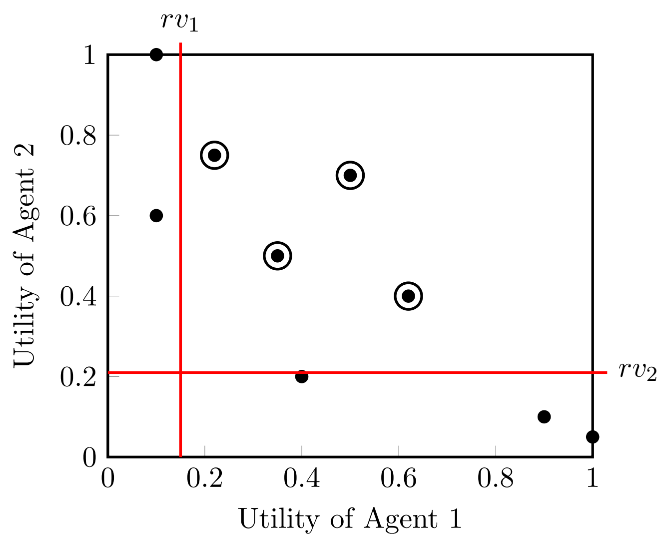


Figure 2.4: The individually rational offers are those for which their utility vector lies above the horizontal line representing rv_2 and to the right of the vertical line representing rv_1. Here these utility vectors are all drawn with a circle around them.

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, ω and ω', such that each agent prefers ω over ω'. We then say that ω *dominates* ω', or that ω' is *dominated* by ω. We can give a precise definition as follows.

Definition 2.3.2. *We say that an offer ω* ***dominates*** *another offer ω' if:*

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

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

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

We say an offer ω' ***is dominated*** *by ω, if ω dominates ω'.*

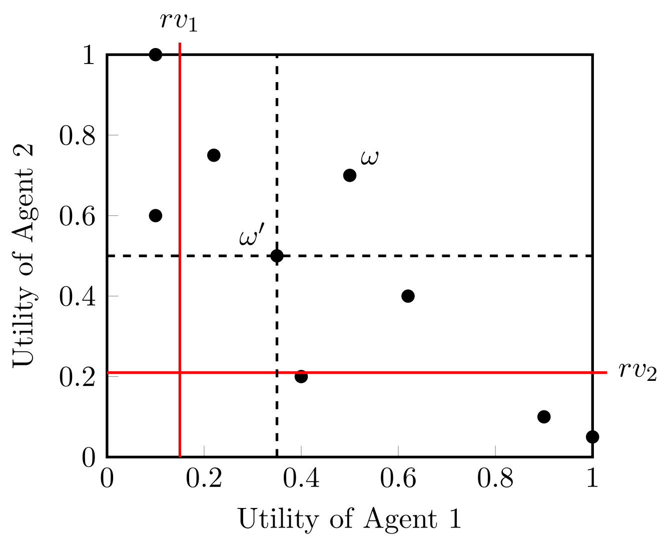


Figure 2.5: Example of domination. The offer ω lies to the top-right of ω' and we therefore say that ω dominates ω'.

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

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

Definition 2.3.3. *An offer ω 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 ω. The lines divide the space into for quarters.

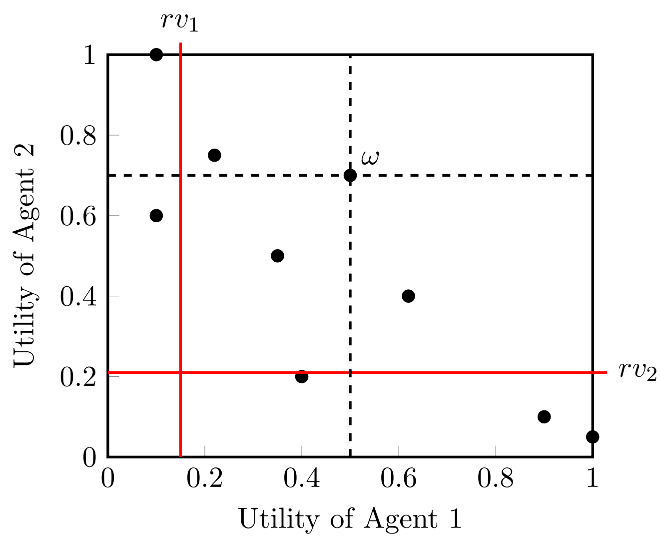


Figure 2.6: The offer ω is Pareto optimal because it is not dominated by any other offer. We can see this because the area that lies above the horizontal dashed line and to the right of the vertical dashed line is empty.

If the top-right quarter (including the lines themselves) is empty, then ω is Pareto optimal. See Figure 2.6.

Definition 2.3.4. *For any negotiation domain D, its* ***Pareto set*** Ω^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 Ω, 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 ω^* 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. In that case 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).

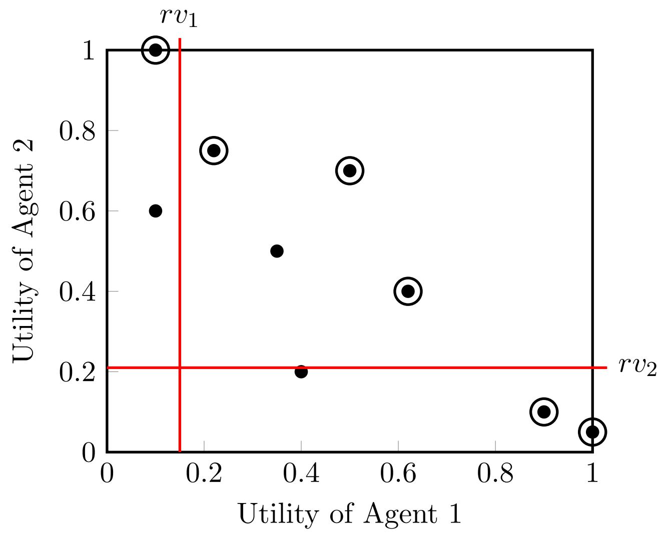


Figure 2.7: Pareto-frontier. All offers that are Pareto-optimal have been drawn here with a circle around them.

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.

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 $(1, 1, \ldots, 1)$ 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

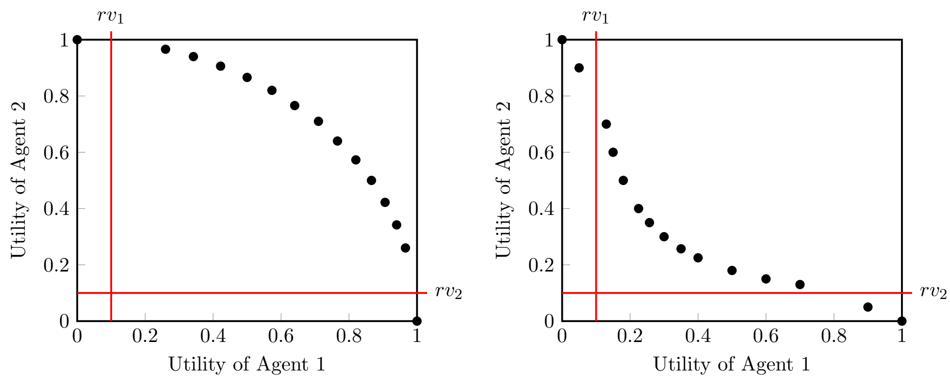


Figure 2.8: Left: a domain with low opposition. Right: a domain with high opposition.

on the Euclidean distance [54]. That is:

$$opp(D) \quad := \quad \min_{\omega \in \Omega} \sqrt{\sum_{i=1}^{n} (1 - u_i(\omega))^2} \tag{2.6}$$

While this definition may initially seem intuïtive, 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]:

$$opp(D) \quad := \quad \min_{\omega \in \Omega} \; 1 - \min_{i \in \{1,2,\ldots,n\}} u_i(\omega) \tag{2.7}$$

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 [43] 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:

$$opp(D) \quad := \quad \sqrt{\sum_{i=1}^{n}(1 - u_i(\omega^*))^2} \tag{2.8}$$

where:

$$\omega^* \quad := \quad \min_{\omega \in \Omega^p} \quad |u_1(\omega) - u_2(\omega)| \tag{2.9}$$

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 The 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 [39].

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

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. It consists of only 5 Python files:

- **nego_simulator.py** Contains the code to run a simulation of a single negotiation. When you run this code, it creates a negotiation domain and two agents which will then negotiate with each other over that domain. When this negotiation is over it prints the outcome of the negotiation (i.e. whether the agents made an agreement or not, plus the utility values obtained by the two respective agents).
- **domains.py** Contains the code that defines negotiation domains, and their components (issues, offers, utility functions and evaluation functions). Furthermore, it contains a function that constructs an example domain, and another function that can be used to load negotiation domains from json files.

- **agents.py** Contains the code of the 'Agent' class, which should be the base class of any negotiating agent that you may implement in this framework. Furthermore, it contains the implementation of an agent called 'RandomAgent'. This agent just makes random proposals and randomly accepts any offers with a probability of 1%.

- **opponent_utility_models.py** Contains the base class for any opponent modeling algorithms that aim to learn the opponent's utility function (see Section 4.1). Furthermore, it contains the implementation of a 'dummy' opponent model. This is a fake opponent modeling algorithm that you can use if you haven't implemented any real opponent modeling algorithms yet (it is 'fake' in the sense that it uses the true utility function of the opponent, while the whole point of a real opponent modeling algorithm is that you don't have access to that function).

- **opponent_strategy_models.py** Contains the base class for any opponent modeling algorithms that aim to learn the opponent's strategy (see Section 4.2), plus the implementation of one very simple such algorithm based on linear extrapolation.

Of course, in order to run experiments you will need a lot more code. For example, you will need more negotiating agents and more domains, and you will need code to run large tournaments with many agents and many domains. Furthermore, you will need code to analyze the results of your experiments.

The idea, however, is that as you continue reading this book, you will implement that code yourself, by doing the exercises. We feel that in this way you get a much better understanding of how a negotiation framework works, in a step-by-step manner, rather than if we gave you the complete framework all at once.

However, if you don't want to do those exercises, you can skip most of them and find their solutions in the folder 'Solutions to Exercises'. So, if you want to run experiments without implementing all the necessary code yourself, you can just copy-paste the required code from the solutions folder to the main folder.

> **Exercise 1. Explore the NegoSimulator framework.** Download the python code of the NegoSimulator, take a look at the five files mentioned above, and familiarize yourself with the code. In particular, make sure that you understand the source code of nego_simulator.py and agents.py.
> Next, run the file nego_simulator.py. This will run a simulation of two RandomAgents that are negotiating over the example domain.

Note that the framework allows you to define negotiation domains in two different ways:

1. Directly in the code itself.
2. In json format.

Currently, the framework comes with two example domains. One for each of these two methods.

The example of the first method is given in the function get_example_domain() in the file domains.py. This method can be useful if you want to generate negotiation domains automatically.

The second method is probably easier. For an example of this method you can open the folder called 'Domains'. The idea is that any negotiation domains that you create yourself can be stored in this folder. Each such domain should be stored in its own subfolder, which would then contain exactly one json file for each agent (so if it's a bilateral domain then it should contain exactly two json files), each defining the utility function of one of the agents. These files should all have the same structure, in the sense that they should each define the same domain name, the same number of issues, with the same issue names, and for each issue the two files should define the same options. The only things that can be different between the files are the numerical values (i.e. the issue weights, the evaluations, and the reservation values).

To obtain a NegotiationDomain object representing the domain defined in the folder named "Cinema Date", you can call:

```
my_domain = load_domain_from_folder("Domains/Cinema Date")
```

This function is defined in the file domains.py.

Exercise 2. Create your own negotiation domain. To do this, create a new subfolder inside the 'Domains' folder, with the name of your domain. Then create two json files inside this subfolder with the same structure as the json files from the Cinema Date domain.
You can give your domain any number of issues you like and you can give each issue any number of options you like. Also, you can give any names you like to the issues and their options and to the domain itself. Just make sure that the two json files are identical, except for their numerical values.
Next, adapt the file nego_simulator.py so that the two agents will negotiate over your new domain, and then run it.

Exercise 3. Create a random domain generator. Implement code that automatically generates an object of type 'NegotiationDomain' (as defined in the file domains.py), with a randomly chosen number of issues, and with a random size for each issue, and with randomly chosen utility functions and reservation values.

Next, adapt the file nego_simulator.py so that each time you run it, it will randomly generate a new negotiation domain and then the two agents will negotiate over that domain.

To make it more challenging, you can add some constraints on the competitiveness of the domain. For example, you can add the constraint that for each offer $\omega \in \Omega$ the sum of the two agents' utilities $u_1(\omega) + u_2(\omega)$ must stay below a specified value.

Exercise 4. Implement a domain visualizer. Implement a tool that, given an object of type 'NegotiationDomain' (as defined in the file domains.py), generates a corresponding utility-space diagram, like in Figure 2.2. Use this tool to visualize some of the domains generated by the tool from Exercise 3.

Hint: a useful tool for generating such diagrams in Python is pyplot, which is part of the matplotlib library.

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 ag_1, while its opponent is denoted as ag_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 Ω 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 ω_{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 ω_{next} to propose next.
4. Use the acceptance strategy to determine whether or not to accept the received offer ω_{rec}. If yes, then accept ω_{rec}, if not, then propose ω_{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 the 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:

Ω ▷ 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_1^o ▷ The observed negotiation history: a list containing all proposals that have so far been proposed by both agents, sorted in chronological order.

ω_{rec} ▷ The offer last proposed by the opponent (if any). Note that it is also contained in the history h_1^o, but for clarity we also display it here separately.

▷ OPPONENT MODELING

▷ First, update the opponent model:

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

▷ BIDDING STRATEGY

▷ Next, apply a bidding strategy to select the next offer to propose:

$\omega_{next} \leftarrow \mathit{biddingStrategy}(\Omega, u_1, rv_1, T, \mathcal{M}, t, h_1^o)$

▷ ACCEPTANCE STRATEGY

▷ Then, determine whether or not to accept the opponent's last proposal. We store this decision in a boolean variable *acceptOffer*:

$\mathit{acceptOffer} \leftarrow \mathit{acceptanceStrategy}(\Omega, u_1, T, \mathcal{M}, t, \omega_{rec}, \omega_{next})$

▷ RETURN SELECTED ACTION

▷ Finally, return the selected action (accept or propose):

if *acceptOffer* **then**

  RETURN ($\mathfrak{a}$, ω_{rec})

else

  RETURN ($\mathfrak{p}$, ω_{next})

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 [25] Faratin et al. also proposed a fourth type of strategy, known as a *resource-based* strategy, but this type seems to have been given considerably less attention in the literature, so we will not discuss it 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 ag_1 will propose an offer ω 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 λ 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 λ, we need to implement a precise rule how to choose the next offer to propose ω_{next} based on this function. One example would be to do it according to the following expression:

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

where $\hat{u}_2$ is an estimation that our agent ag_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, Ω_t^{prop} is the set off all offers that have already been proposed by ag_1 before time t.

$$\Omega_t^{prop} := \{\omega \in \Omega \mid \exists t' \in [0, t] : (1, \mathfrak{p}, \omega, t') \in h_1^o\} \tag{3.2}$$

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 ω', so, if it has the chance, our agent should also try to propose ω'.

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.

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

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

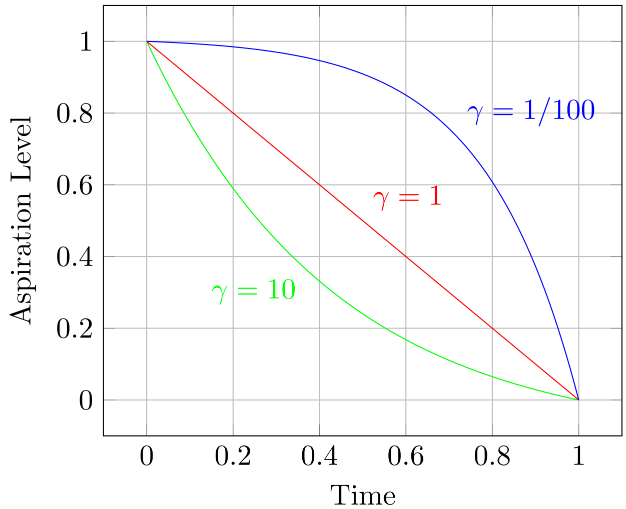


Figure 3.1: Aspiration functions with $\alpha = 1$, $\beta = 0$, $T = 1$, and several different values for γ.

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:

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

where T is the deadline of the negotiations, and α, β and γ 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 γ. An example implementation of a time-based agent is displayed in Algorithm 2.

Let us now discuss how to interpret the parameters α, β, and γ, and how to choose their values. For this, first note that if $t = 0$ then we have $\lambda(0) = \alpha$. Therefore, α 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 β 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: α, β, γ
Input:
Ω ▷ The offer space.
u_1 ▷ The agent's own utility function.
T ▷ The deadline.
t ▷ The current time.
h_1^o ▷ The observed negotiation history.
ω_{rec} ▷ The offer last proposed by the opponent (if any).

▷ OPPONENT MODELING
$\mathcal{M} \leftarrow \mathit{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$
$\hat{u}_2 \leftarrow \mathit{getEstimatedOpponentUtility}(\mathcal{M})$

▷ BIDDING STRATEGY
▷ Calculate the aspiration level:
$\lambda \leftarrow (\alpha - \beta) \cdot \frac{1-\gamma^{1-t/T}}{1-\gamma} + \beta$
▷ Obtain the set of offers we have already proposed:
$\Omega^{prop} \leftarrow \mathit{getOffersProposedByUs}(h_1^o)$
▷ Find the next offer to propose:
$\omega_{next} \leftarrow \arg\max_{\omega \in \Omega} \{\hat{u}_2(\omega) \mid u_1(\omega) \geq \lambda \ \wedge \ \omega \notin \Omega^{prop}\}$

▷ ACCEPTANCE STRATEGY
▷ Get the last proposal received from the opponent, and accept it if it yields more utility to us than our aspiration level:
$\mathit{acceptOffer} \leftarrow u(\omega_{rec}) \geq \lambda$

▷ RETURN SELECTED ACTION
if *acceptOffer* **then**
    RETURN $(\mathfrak{a}, \omega_{rec})$ ▷ accept the received offer
else
    RETURN $(\mathfrak{p}, \omega_{next})$ ▷ propose a new offer
end if

a lower target value represents a conceding strategy. Finally, the parameter γ determines how quickly the agent concedes from α to β.

Typically, the value chosen for α 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 β, 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 β *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 ag_1 is indicated with a vertical blue line that over time moves from the right to the left, while the aspiration level of ag_2 is indicated with a horizontal blue line that over time moves from the top to the bottom. Note that in this example, ag_1 follows a conceding strategy, while ag_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 γ is the **concession parameter**. It determines how fast our agent will concede towards its target value. If γ is very small (e.g. 0.01) our agent will initially concede very slowly, as we can see in Figure 3.1, and only starts making large concessions towards the end of the negotiations. On the other hand, if γ is very large, our agent will immediately start making large concessions. Finally, a value of $\gamma = 1$ represents an agent that concedes linearly.[1]

In order to exploit the opponent as much as possible, our agent should make sure it concedes slower than the opponent. This suggest that we would always want a low value of γ. However, if we choose γ 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 γ 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 γ very low, then our agent's aspiration level will remain very high for a long time, which means that for a long time there

[1] Technically, the expression in Eq. 3.4 is not defined for $\gamma = 1$, but it can be shown that $\lim_{\gamma \to 1} f(t) = (\alpha - \beta) \cdot (1 - t/T) + \beta$, 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 γ 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.

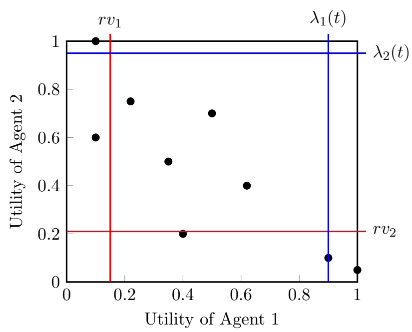
Another reason why a low value of γ might not be optimal is when there are discount factors (see Section 2.2.5), because in that case we would prefer our agent to come to an agreement as quickly as possible. Yet another example could be in the case that the opponent is participating in multiple negotiations 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 γ, but with $\beta = rv_1$ are also known as **Boulware strategies**.

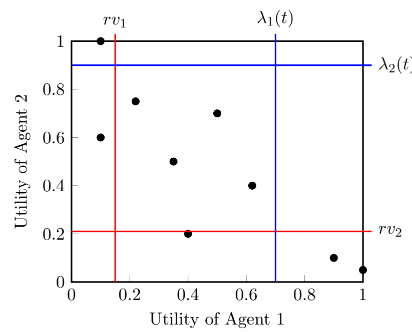
Finally, it should be noted that Eq. (3.4) is sometimes adapted so that the agent reaches its target level already a bit *before* the deadline, at a time T' slightly less than T, which we will call the **target time**. After the target time, the aspiration level will just remain constant:

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

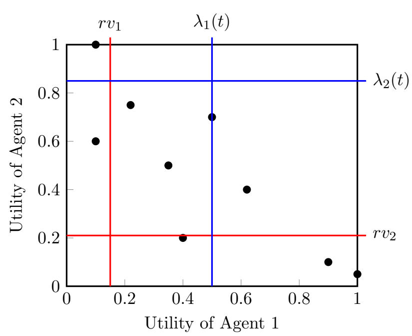
This is to ensure that our agent will indeed concede all the way to its target level, but not any further. Furthermore, it ensures that after ag_1 proposes its ultimate offer (with utility equal to or very close to β) at time T' so that there is enough time left for the opponent to accept that offer.



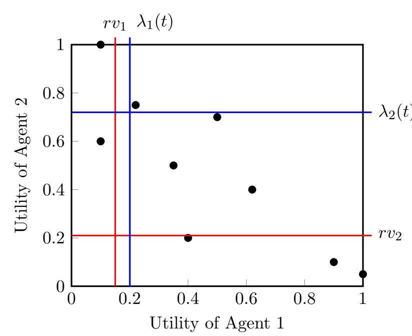
(a) Aspiration levels at $t/T = 0.2$



(b) Aspiration levels at $t/T = 0.4$



(c) Aspiration levels at $t/T = 0.6$



(d) Aspiration levels at $t/T = 0.8$

Figure 3.2: Negotiation between a conceding agent (ag_1) and a hardheaded agent (ag_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 5. 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 α, β and γ 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 ag_2 plays a time-based strategy with target value β_2. This means that, if we wait long enough, the opponent will be willing to propose or accept any offer ω for which $u_2(\omega) \geq \beta_2$. Now, let ω^* denote the offer that maximizes ag_1's own utility u_1 among those offers. That is:

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

This means that ag_1 cannot possibly receive any utility higher than $u_1(\omega^*)$. After all, by Eq. (3.6) we know that for any offer ω that yields a higher utility to ag_1, we would have $u_2(\omega) < \beta_2$, and agent ag_2 would never propose or accept any such offer, by definition of β_2. On the other hand, however, it also means that if we get close enough to the deadline, then ag_2 will be willing to accept the offer ω^* and therefore, *ideally*, ag_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 ag_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, ag_1 typically does not know the target value β_2 of its opponent, and secondly ag_1 typically also does not know the utility function u_2 of its opponent. Therefore, ag_1 cannot directly determine the ideal offer ω^*.

Instead, however, ag_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 β_2, which it can then use to obtain a better prediction of ω^*. 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 ω^* 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' β^{min} and they make sure that their target β is never lower than that. That is:

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

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 ϵ to our target utility $u_1(\omega^*)$, where $\epsilon > 0$ and where ϵ decreases as we gain more and more knowledge about the opponent from the offers it proposes to us. So we would get:

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

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 6. 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 5) to estimate the opponent's utility function.
Furthermore, to estimate the optimal target value β^* 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 5.

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: α, β^{min}, γ
Input:
Ω ▷ 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_1^o ▷ The observed negotiation history.
ω_{rec} ▷ The offer last proposed by the opponent (if any).

▷ OPPONENT MODELING
▷ Update the opponent model:
$\mathcal{M} \leftarrow \textit{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$
$\hat{u}_2 \leftarrow \textit{getEstimatedOpponentUtility}(\mathcal{M})$
▷ Use the opponent model to estimate the optimal target value:
$\hat{\beta}^* \leftarrow \textit{estimateOptimalTarget}(\mathcal{M})$

▷ BIDDING STRATEGY
▷ Calculate the aspiration value:
$\beta \leftarrow \max\{\hat{\beta}^* \ , \ \beta^{min}\}$
$\lambda \leftarrow (\alpha - \beta) \cdot \frac{1-\gamma^{1-t/T}}{1-\gamma} + \beta$
▷ Obtain the set of offers we have already proposed:
$\Omega^{prop} \leftarrow \textit{getOffersProposedByUs}(h_1^o)$
▷ Find the next offer to propose:
$\omega_{next} \leftarrow \arg\max_{\omega \in \Omega}\{\hat{u}_2(\omega) \mid u_1(\omega) \geq \lambda \ \wedge \ \omega \notin \Omega^{prop}\}$

▷ ACCEPTANCE STRATEGY
▷ Get the last proposal received from the opponent, and accept it if it yields more utility to us than our aspiration level:
$\textit{acceptOffer} \leftarrow u(\omega_{rec}) \geq \lambda$

▷ RETURN SELECTED ACTION
if *acceptOffer* **then**
  RETURN ($\mathfrak{a}$, ω_{rec})
else
  RETURN ($\mathfrak{p}$, ω_{next})
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 β^{min}, but then we still need to answer the question how to choose the optimal value for β^{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** [25]. 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 Ω_t^{prop} denotes the set of offers that have been proposed by our agent ag_1 up until time t (see Eq.(3.2)). Similarly, we define Ω_t^{rec} to be the set of offers that have been *received* by our agent ag_1 up until time t. In other words, it is set of offers that have been proposed by the *opponent* ag_2 up until time t. Formally:

$$\Omega_t^{rec} := \{\omega \in \Omega \mid \exists t' \in [0, t] : (2, \mathfrak{p}, \omega, t') \in h_1^o\} \tag{3.7}$$

Now, in order to give a concrete implementation of a classic Tit-for-Tat negotiation strategy, we need a function c_1 that, given Ω_t^{prop} returns a value $c_1(\Omega_t^{prop}) \in \mathbb{R}$ that measures how much agent ag_1 has so far conceded. Furthermore, we need a function c_2 that, given Ω_t^{rec} returns a value $c_2(\Omega_t^{rec}) \in \mathbb{R}$ that measures the amount of concession made by ag_2.

$$c_1, c_2 : 2^\Omega \to \mathbb{R}$$

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 functions, these two concepts are different.

This means that for c_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:

$$c_1(\Omega_t^{prop}) \quad := \quad \max \{u_1(\omega_1^{max}) - u_1(\omega) \mid \omega \in \Omega_t^{prop}\} \tag{3.8}$$

or:

$$c_1(\Omega_t^{prop}) \quad := \quad \max \{\hat{u}_2(\omega) - \hat{u}_2(\omega_2^{min}) \mid \omega \in \Omega_t^{prop}\} \tag{3.9}$$

where $\hat{u}_2$ is an estimation of the opponent's utility function u_2, made by an opponent modeling algorithm and where ω_1^{max} and ω_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:

$$c_2(\Omega_t^{rec}) \quad := \quad \max \{u_1(\omega) - u_1(\omega_1^{min}) \mid \omega \in \Omega_t^{rec}\} \tag{3.10}$$

or:

$$c_2(\Omega_t^{rec}) \quad := \quad \max \{\hat{u}_2(\omega_2^{max}) - \hat{u}_2(\omega) \mid \omega \in \Omega_t^{rec}\} \tag{3.11}$$

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, c_2 is a function used by *our* agent ag_1 to measure the opponent's concession. In other words, it exists in the 'mind' of our agent ag_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 ω_{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*:

$$\Delta c_t(\omega) \quad := \quad c_1(\Omega_t^{prop} \cup \{\omega\}) \quad - \quad c_2(\Omega_t^{rec})$$

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

Finally, the Tit-for-Tat strategy chooses our agent's next offer to propose ω_{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:

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

or:

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

where θ_{min} and θ_{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, θ_{min} can be equal to 0, but $\Delta c_t(\omega)$ 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 c_1, an expression for c_2, and a method to choose ω_{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 ag_2 makes an offer that yields a utility of 0.1 to our agent, then our agent ag_1 would reply with an offer that yields a utility of 1-0.1=0.9 to itself. Next, if ag_2 makes a proposal with utility of, say, 0.3 for ag_1, then ag_1 replies with an offer with utility 1-0.3=0.7. Then, if ag_2 proposes an offer with utility 0.35, our agent ag_1 will reply with an offer that yields 1-0.35=0.65, next, if ag_2 proposes an offer with utility 0.55 then ag_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

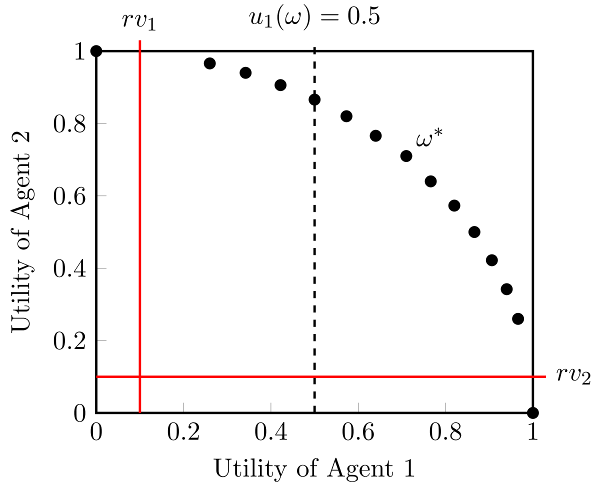


Figure 3.3: An example of a domain with low opposition. Here, the outcome with $u_1(\omega) = 0.5$ is highly unfair for ag_1, since the opponent would receive $u_2(\omega) = 0.87$ for that same offer. Especially, since there exists a much fairer offer, here indicated as ω^*, for which both agents would receive 0.7.

to an agreement with utility 0.5 for ag_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 c_1 as well as our opponent's concession c_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 ω 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 ω^*, such as the maximum social welfare solution, of the Nash bargaining solution (see Section 5.6) and then we measure concession relative to that ideal outcome:

$$c_1(\Omega_t^{prop}) = \max \{ \frac{u_1(\omega_1^{max}) - u_1(\omega)}{u_1(\omega_1^{max}) - u_1(\omega^*)} \mid \omega \in \Omega_t^{prop} \} \quad (3.14)$$

$$c_2(\Omega_t^{rec}) = \max \{ \frac{u_1(\omega) - u_1(\omega_1^{min})}{u_1(\omega^*) - u_1(\omega_1^{min})} \mid \omega \in \Omega_t^{rec} \} \quad (3.15)$$

Note that this does require you to know which outcome ω^* 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 [25], the authors proposed a variant in which the agents' concessions were calculated only in terms of the 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 7. 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 5).
Let your agent negotiate against the RandomAgent or against one of your agents from Exercises 5 and 6, 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 [15]. This strategy is called MiCRO, which stands for *Minimal Concession in Reply to new Offers.* Despite its simplicity and the fact that it does not require any form of opponent modeling, it has shown some remarkably good results.

MiCRO works as follows. Before the negotiations begin, our agent ag_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, $u_1(\omega_1) \geq u_1(\omega_2) \geq \cdots \geq u_1(\omega_K)$. Then, when the negotiations start, our agent will first propose the offer with highest utility for itself. That is, ω_1, which is the first offer on the list. Then, in the following rounds, every time the opponent proposes a *new* offer (i.e. an offer that it hasn't proposed before), our agent will respond by proposing the next offer on its list. So, it will first propose ω_2, then ω_3, then ω_4, etcetera. However, whenever the opponent ag_2 proposes an offer it has already proposed before, ag_1 will reply by also repeating an earlier proposal.

More precisely, whenever it is ag_1's turn to make a proposal, it counts

Algorithm 4 A Classic Tit-for-Tat strategy.

Parameters: θ_{min}
Input:
Ω ▷ 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_1^o ▷ The observed negotiation history.
ω_{rec} ▷ The offer last proposed by the opponent (if any).

▷ OPPONENT MODELING
$\mathcal{M} \leftarrow \mathit{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$
$\hat{u}_2 \leftarrow \mathit{getEstimatedOpponentUtility}(\mathcal{M})$

▷ 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 $\Delta c_t(\omega)$ depends on the chosen expressions for c_1 and c_2.
$C \leftarrow \{\ \omega \in \Omega \mid \Delta c_t(\omega) > \theta_{min}\ \wedge\ u_1(\omega) > rv_1\}$
if $C = \emptyset$ **then**
  $\omega_{next} \leftarrow \ldots$ ▷ Use any alternative method to pick an offer here.
else
  $\omega_{next} \leftarrow \arg\max_\omega \{\ u_1(\omega) \mid \omega \in C\}$
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 the propose:
$\mathit{acceptOffer} \leftarrow u(\omega_{rec}) \geq u(\omega_{next})$

▷ RETURN SELECTED ACTION
if *acceptOffer* **then**
  RETURN $(\mathfrak{a}, \omega_{rec})$
else
  RETURN $(\mathfrak{p}, \omega_{next})$
end if

how many *different* offers it has so far received from the opponent (we denote 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 ω_{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 ω_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 ag_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:

$$c_1(\Omega_t^{prop}) := |\Omega_t^{prop}|$$

$$c_2(\Omega_t^{rec}) := |\Omega_t^{rec}|$$

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

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 might 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.

However, perhaps the biggest advantage of MiCRO is that it is very simple to implement, since it does not require implementing any complicated opponent modeling algorithms, and that it does not require any parameters to be fine-tuned. This makes it especially ideal as a benchmark strategy for scientific experiments. After all, there is basically just one version of MiCRO, while almost any other negotiation strategy requires choosing some parameter values or choosing a particular opponent modeling algorithm. This makes it much harder to draw general conclusions about such strategies, or about agents that have been tested against such strategies.

Furthermore, what makes MiCRO particularly elegant, 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, it typically still manages to come to agreement when negotiating against itself. This is because two MiCRO agents would always keep making concessions until sooner or later they reach an agreement.

There are just two possible scenarios in which a negotiation between two MiCRO agents 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 have reached 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 make any further concessions.

> **Exercise 8. 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.

Algorithm 5 The MiCRO strategy. Note that *offers*$[m]$ here corresponds to ω_{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_1^o ▷ The observed negotiation history.
ω_{rec} ▷ The offer last proposed by the opponent (if any).

$m \leftarrow \mathit{countUniqueOffersProposedByMe}(h_1^o)$
$n \leftarrow \mathit{countUniqueOffersProposedByOpponent}(h_1^o)$

▷ 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*.
$\mathit{readyToConcede} \leftarrow m \leq n$ **and** $u_1(\mathit{offers}[m]) > rv_1$

▷ 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.
if *readyToConcede* **then**
  $\omega_{next} \leftarrow \mathit{offers}[m]$
else
  $r \leftarrow \mathit{getRandomInteger}(0, m)$ ▷ Pick a random integer r with $0 \leq r < m$.
  $\omega_{next} \leftarrow \mathit{offers}[r]$
end if

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

▷ RETURN SELECTED ACTION
if *acceptOffer* **then**
  RETURN $(\mathfrak{a}, \omega_{rec})$
else
  RETURN $(\mathfrak{p}, \omega_{next})$
end if

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

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

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

Definition 3.3.1. *The AC_{next} acceptance strategy accepts if and only if:*

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

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 3.3.2. *A bidding strategy for agent ag_i is* ***monotonic****, if for any negotiation history h, and any two proposals $(i, \mathfrak{p}, \omega, t) \in h$, $(i, \mathfrak{p}, \omega', t') \in h$ 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 ag_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 shows 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 ag_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 ω_1 with utility $u_1(\omega_1) = 0.79$. Next, suppose that agent ag_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 ω_2 and we see that $u_1(\omega_2) = 0.7$. Again, suppose this offer is rejected and instead ag_2 makes a counter-proposal, which is denoted ω_{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 ω_3, with utility $u_1(\omega_3) = 0.7$. So, the bidding strategy will select ω_3 to propose next.

Now, if our agent uses AC_{next}, it will compare ω_{rec} with ω_3. This means our agent will *reject* ω_{rec}, because ω_3 yields more utility. But this clearly does not make sense, because our agent has already proposed ω_2 which yielded less utility than ω_{rec}. So, if our agent was willing to propose ω_2 with utility 0.7, then it should certainly be willing to accept ω_{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 ω_{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 ω_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 ω_{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 3.3.3. *The AC_{asp} acceptance strategy accepts if and only if:*

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

where λ 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 3.3.4. *The AC_{low} acceptance strategy accepts if and only if:*

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

where t is the time at which the decision is made and Ω_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 modeled by a parametrized version of AC_{next} [7], which has two parameters α and β and which is denoted by $AC_{next}(\alpha, \beta)$.

Definition 3.3.5. *Let $\alpha, \beta \in \mathbb{R}$ be two real numbers. Then the $AC_{next}(\alpha, \beta)$ acceptance strategy accepts if and only if:*

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

Note that if $\alpha = 1$ and $\beta = 0$, then $AC_{next}(\alpha, \beta)$ is just identical to AC_{next}. Typically, the values of α and β 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

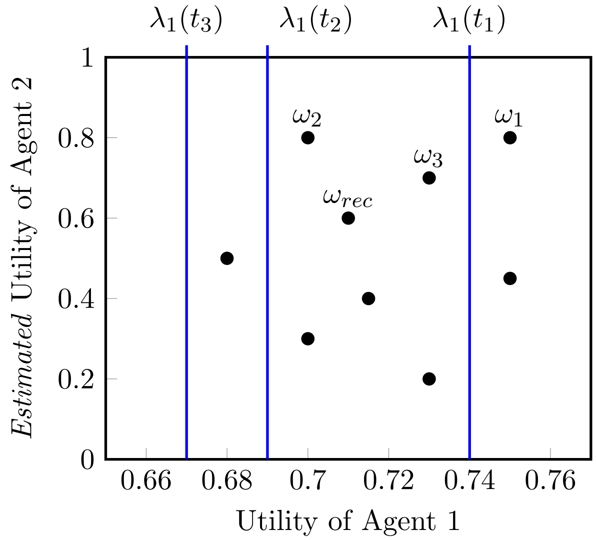


Figure 3.4: The problem with AC_{next}. At t_1 agent 1 proposes ω_1, at t_2 agent 1 proposes ω_2, and at t_3 agent 1 has the choice between proposing ω_3 or accepting ω_{rec}. According to AC_{next}, the agent should reject. However, this does not make sense, since he has already proposed ω_2 which is actually worse than ω_{rec}.

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 α and β, which may be difficult.

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 9. $\mathbf{AC_{low}}$. Adapt the implementation of your Tit-for-Tat agent from Exercise 7 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 [51] and in [17].

Let us explain it with an example. Suppose that we have a negotiation domain with 10 possible offers: $\Omega = \{\omega_1, \omega_2, \ldots, \omega_{10}\}$ 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 ω_{10}. Furthermore, suppose that our agent ag_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 ag_1, the negotiations proceed as follows (see also Figure 3.5):

1.	At $t = 0.0$:	$\lambda_1(t) = 1.0$	ag_1 proposes ω_{10}	
2.	At $t = 0.05$:			ag_2 proposes ω_4
3.	At $t = 0.10$:	$\lambda_1(t) = 0.9$	ag_1 proposes ω_9	
4.	At $t = 0.15$:			ag_2 proposes ω_6
5.	At $t = 0.20$:	$\lambda_1(t) = 0.8$	ag_1 proposes ω_8	
6.	At $t = 0.30$:			ag_2 proposes ω_2
7.	At $t = 0.50$:	$\lambda_1(t) = 0.5$	ag_1 proposes ...	

At time $t = 0.50$, our agent's strategy prescribes that it should propose ω_5. Ideally, however, ag_1 would like to accept ω_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 ω_2. Note that earlier our agent did not accept ω_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 ω_6 instead of ω_5. Since ω_6 was already proposed before by ag_2, it is very likely that ag_2 will now accept it, and therefore it should indeed be better for ag_1 to propose ω_6, than to propose ω_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 3.4.1. *We say an agent ag_i* ***reproposes*** *an offer ω if ag_i proposes it, while it was earlier already proposed by the other agent ag_j and ag_i itself has not yet proposed it since then.*

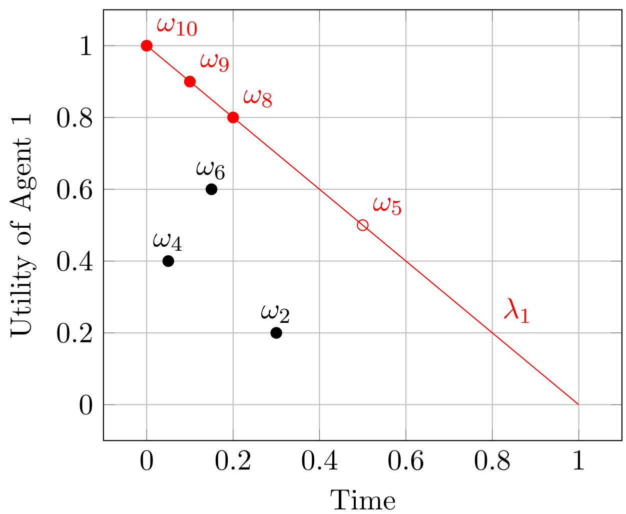


Figure 3.5: The benefit of reproposing. The red dots represent proposals made by ag_1, the red line represents ag_1's aspiration level λ_1 as a function of time, and the black dots represent proposals made by ag_2. At time $t = 0.5$, the bidding strategy of ag_1 suggests to propose ω_5. However, it makes more sense for ag_1 to propose ω_6, which earlier was already proposed by ag_2. Note that at that time ag_1 cannot *accept* ω_6, because the AOP only allows accepting the last received proposal, which was ω_2.

Exercise 10. 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:
Ω ▷ 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_1^o ▷ The observed negotiation history.
ω_{rec} ▷ The offer last proposed by the opponent (if any).

▷ OPPONENT MODELING
$\mathcal{M} \leftarrow \mathit{updateOpponentModel}(\Omega, T, \mathcal{M}, t, \omega_{rec})$

▷ BIDDING STRATEGY
$\omega_{next} \leftarrow \mathit{biddingStrategy}(\Omega, u_1, rv_1, T, \mathcal{M}, t, h_1^o)$

▷ CHECK IF WE CAN FIND A BETTER OFFER TO REPROPOSE
▷ From the negotiation history, extract the set of all offers that have so far been proposed by this agent:
$\Omega^{prop} \leftarrow \mathit{getProposedOffers}(h_1^o)$
▷ From the negotiation history, extract the set of all offers that have so far been proposed by the opponent:
$\Omega^{rec} \leftarrow \mathit{getReceivedOffers}(h_1^o)$
▷ See if we can find any offer that can be reproposed:
if $\Omega^{rec} \setminus \Omega^{prop} \neq \emptyset$ **then**
  $\omega_{rep} \leftarrow \arg\max\{u_1(\omega) \mid \omega \in \Omega^{rec} \setminus \Omega^{prop}\}$
  if $u_1(\omega_{rep}) \geq u_1(\omega_{next})$ **then**
    $\omega_{next} \leftarrow \omega_{rep}$
  end if
end if

▷ ACCEPTANCE STRATEGY
$\mathit{acceptOffer} \leftarrow \mathit{acceptanceStrategy}(\Omega, u_1, T, \mathcal{M}, t, h_1^o, \omega_{rec}, \omega_{next})$

▷ RETURN SELECTED ACTION
▷ Finally, return the selected action (accept or propose):
if *acceptOffer* **then**
  RETURN $(\mathfrak{a}, \omega_{rec})$
else
  RETURN $(\mathfrak{p}, \omega_{next})$
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. Readers who are not interested in the details of such opponent modeling algorithms can safely skip this chapter, since the rest of this book does not depend on it.

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 much 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 each of these techniques assumes 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. (2.5). Therefore, these techniques are not applicable to other types of negotiation domains.

4.1.1 Bayesian Learning

Bayesian learning [30] 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 $\mathcal{U}$ and, based on the proposals $\pi_1, \pi_2, \ldots, \pi_k$ that our agent has so far received from its opponent, we want to calculate the probability, for each function $u \in \mathcal{U}$, that that function u is the actual utility function u_2 of the opponent. That is, for each $u \in \mathcal{U}$ we want to calculate a probability $P(u_2 = u|\pi_1, \pi_2, \ldots, \pi_k)$.

4.1.1.1 Bayesian Learning in General

Bayesian learning is a technique that is much older than automated negotiation and it 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 $\vec{o} = (o_1, o_2, \ldots, o_k)$, 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 $o_1, o_2, \ldots, o_k$.

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: $P(A\spadesuit \mid \spadesuit) = \frac{1}{13}$.

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 $P(y)$.

It is important to understand that we always assume that there is exactly one hypothesis in Y that is true. Therefore, we always have:

$$\sum_{y \in Y} P(y) = 1 \quad \text{and} \quad \sum_{y \in Y} P(y|\vec{o}) = 1$$

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:

$$P(y, o) \quad = \quad P(y \mid o) \cdot P(o) \quad = \quad P(o \mid y) \cdot P(y) \tag{4.1}$$

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

From Equation (4.1) we can directly derive:

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

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

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

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 $o_1, o_2, \ldots, o_k$, then this becomes:

$$P(y \mid o_1, o_2, \ldots, o_k) = \frac{P(o_1, o_2, \ldots, o_k \mid y) \cdot P(y)}{\sum_{y' \in Y} P(o_1, o_2, \ldots, o_k \mid y') \cdot P(y')} \tag{4.3}$$

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

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

Now, suppose that we have already calculated, for each hypothesis $y \in Y$, the probability $P(y|o_1, o_2, \ldots, o_k)$, 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:

$$\tilde{P}(y \mid \vec{o}) \quad := \quad P(y) \cdot P(o_1 \mid y) \cdot P(o_2 \mid y) \cdot \ \ldots \ \cdot P(o_k \mid y) \tag{4.5}$$

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 $P(o_{k+1} \mid y)$. That is:

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

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

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

4.1.1.2 Implementation

We will here discuss how the calculations discussed above 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 $\tilde{P}(y \mid \vec{o}) = P(y)$, 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:

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

2. Calculate the sum:

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

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

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

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

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

3. Calculate the sum:

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

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

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

Note that the fact that in Step 2 each $\tilde{P}(y \mid \vec{o}, o_{k+1})$ 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 $\tilde{P}(y \mid \vec{o})$ as:

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

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

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

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:

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

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

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

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

2. Calculate the sum:

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

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

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

While this looks very similar to our original implementation, the difference is that step 1 now involves $P(y \mid \vec{o})$, rather than $\tilde{P}(y \mid \vec{o})$. 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 $\tilde{P}(y \mid \vec{o}, o_{k+1})$ and then in Step 3 we can simply overwrite it to store the values of $P(y \mid \vec{o}, o_{k+1})$. In our initial implementation this was not possible, because we needed to keep the values of $\tilde{P}(y \mid \vec{o}, o_{k+1})$ 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 $P(o \mid y)$.

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 π is defined as a tuple of the form $(2, \mathfrak{p}, \omega, t)$ for some offer ω and some time t. So we have:

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

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

For the prior probabilities, the simplest approach is to assign all hypotheses the same prior probability. That is: $P(u) = \frac{1}{|\mathcal{U}|}$. 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 $u \in \mathcal{U}$. 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 [30] modeled the opponent's strategy as a linear time-based strategy. So, at any time t they *expect* the opponent to propose an offer ω 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:

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

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

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.

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

$$\hat{u}_2(\omega) = \sum_{u \in \mathcal{U}} P(u|\vec{\pi}) \cdot u(\omega) \tag{4.12}$$

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

Algorithm 7 Opponent modeling algorithm based on Bayesian learning

Parameters:

σ ▷ Standard deviation of the Gaussian distribution.

c ▷ Concession speed of hypothesized opponent strategy.

$\mathcal{U}$ ▷ A set of hypothetical utility functions for the opponent.

Input:

T ▷ The deadline.

t ▷ The current time.

ω_{rec} ▷ The last received offer.

probs ▷ A map that maps each $u \in \mathcal{U}$ to the probability value $P(u \mid \pi_1, \pi_2, \ldots, \pi_k)$ as calculated in the previous call to this algorithm.

▷ Ensure that we initially assign the same probability to each hypothesis:

if this is our first turn **then**
  for $u \in \mathcal{U}$ **do**
    $probs[u] \leftarrow \frac{1}{|\mathcal{U}|}$
  end for
end if

▷ Update all the values in *probs*, given the newly received offer ω_{rec} and simultaneously calculate the sum of all these values:

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

▷ Ensure that all probabilities are normalized:

for $u \in \mathcal{U}$ **do**
  $probs[u] \leftarrow probs[u]/sum$
end for

return *probs*

4.1.1.4 Choosing the Utility Hypotheses

We now know how to apply Bayesian learning for some given set of hypothetical utility functions $\mathcal{U}$. 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. (2.5). 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:

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

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 ag_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:

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

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 $s = 3$, then we could have the following parameters:

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

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 $\{0, \quad 0.1, \quad 0.2, \quad \dots, \quad 0.9, \quad 1.0\}$. Since this set has 11

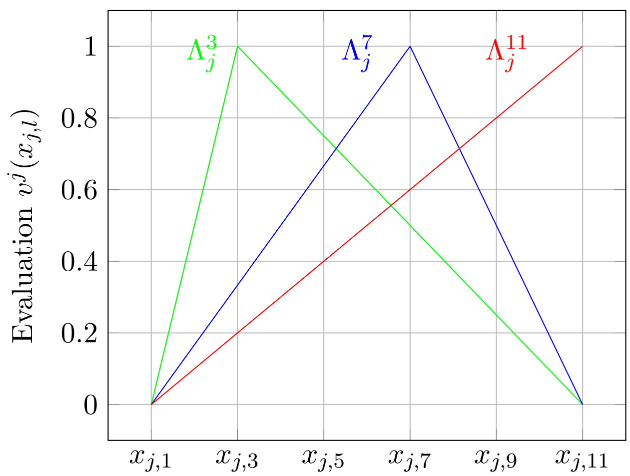


Figure 4.1: Some examples of triangular evaluation functions for an issue I_j with 11 options.

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 $\mathcal{U}$, which is clearly unfeasible for such a large set.

The authors of [30] 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 ag_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 Λ_j^n is defined as:

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

This assumption of triangular evaluation functions greatly reduces the size of the set $\mathcal{U}$ 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 $\mathcal{U}$ becomes small enough to apply Bayesian learning in practice to small domains with just a few issues. However, since the set $\mathcal{U}$ still grows exponentially with the number of issues, this approach is still not feasible for scenarios with many issues. For this reason, luckily, the authors of [30] also proposed a more scalable version of Bayesian opponent modeling, which we will discuss next.

> **Exercise 11. Bayesian Learning.** Implement the Bayesian learning algorithm discussed above. Next, run some negotiations with your agents from Exercises 5, 6, and 7, 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: $y = (y_1, y_2, \ldots, y_m)$, so the hypothesis space can be decomposed as the Cartesian product of a number of sub-hypothesis spaces: $Y = Y^1 \times Y^2 \times \cdots \times Y^m$.

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:

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

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:

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

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:

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

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

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

where s_j is the size of issue I_j.

So a hypothesis y is now a tuple $(w^1, w^2, \ldots w^m, v^1, v^2, \ldots v^m)$, 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:

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

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 ω.

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

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

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:

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

and similarly, for the evaluation functions v^j:

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

Note that these two equations are just special cases of Eq. (4.14), specific to automated negotiation. So, our original question how to calculate $P(o_{k+1} \mid y_j)$ 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 $\overline{w}^j$ and its *expected* evaluation function $\overline{v}^j$ as follows:

$$\overline{w}^j := \sum_{w^j \in Y_w^j} w^j \cdot P(w^j \mid \vec{\pi}) \tag{4.17}$$

$$\overline{v}^j(\omega) := \sum_{v^j \in Y_v^j} v^j(\omega) \cdot P(v^j \mid \vec{\pi}) \tag{4.18}$$

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

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

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

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

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

Similarly, we can define:

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

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 $\overline{u}_{[w^j]}$. That is:

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

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

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

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 $\overline{u}$ 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 σ for the Gaussian distribution.

Exercise 12. 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 5 and 7, but using this new opponent modeling algorithm, instead of the dummy opponent model or the regular Bayesian learning algorithm from Exercise 11.

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

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:
σ ▷ Standard deviation of the Gaussian distribution.
c ▷ Concession speed of hypothesized opponent strategy.
Input:
T ▷ The deadline.
t ▷ The current time.
ω_{rec} ▷ The last received offer.
weight_hyps ▷ 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 ▷ 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 ▷ 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 ▷ 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 $\overline{w}^j$ and $\overline{v}^j(\omega_{rec})$ according to Eqs. (4.17) and (4.18):
for each issue I_j of the domain **do**
  $\overline{w}^j \leftarrow \sum_{l=1}^{|Y_w^j|} weight\_hyps[j][l] \cdot weight\_probs[j][l]$
  $\overline{v}^j \leftarrow \sum_{l=1}^{|Y_v^j|} eval\_hyps[j][l](\omega_{rec}) \cdot eval\_probs[j][l]$
end for

for each issue I_j of the domain **do**

  for $l \in \{0, 1, \dots, |Y_w^j| - 1\}$ **do**
    $\overline{u}_{[w^j]} \leftarrow \sum_{k=1,k\neq j}^{m} \overline{w}^k \cdot \overline{v}^k + weight\_hyps[j][l] \cdot \overline{v}^j$
    $weight\_probs[j][l] \leftarrow$
      $weight\_probs[j][l] \cdot \mathcal{N}(\overline{u}_{[w^j]} \mid 1 - c \cdot \frac{t}{T}, \sigma$ ▷ Eq. (4.15)
  end for
  $normalize(weight\_probs[j])$

  for $l \in \{0, 1, \dots, |Y_v^j| - 1\}$ **do**
    $\overline{u}_{[v^j]} \leftarrow \sum_{k=1,k\neq j}^{m} \overline{w}^k \cdot \overline{v}^k + \overline{w}^j \cdot eval\_hyps[j][l](\omega_{rec})$
    $eval\_probs[j][l] \leftarrow$
      $eval\_probs[j][l] \cdot \mathcal{N}(\overline{u}_{[v^j]} \mid 1 - c \cdot \frac{t}{T}, \sigma)$ ▷ Eq. (4.16)
  end for
  $normalize(eval\_probs[j])$

end for
return $(weight\_probs, eval\_probs)$

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 [48].[1]

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

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

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 $x_{j,l}$:

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

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:

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

and each weight w_2^j can be estimated as:

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

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 13. 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 5 and 7, but using this new opponent modeling algorithm.

[1]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`

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

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

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:

$$z_j \quad := \quad u_1(\omega_j)$$

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

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

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

Of course, 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 [33], 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 [53]), or wavelet decomposition and cubic smoothing splines (OMAC [12]). Here, however, we will only focus on the technique of Gaussian Processes (IAMHaggler2011 [52]).

4.2.1 Gaussian Processes

Due to the technical nature of this topic we cannot discuss Gaussian processes in detail, so we will only give a global idea of how this technique works. For a more detailed discussion we refer to [50] 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 ω with utility $u_1(\omega) = z$ is given by a Gaussian distribution:

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

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 $z_1, z_2, \ldots, z_k$ 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:

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

which can be rewritten using vector-notation:

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

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

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. $z_1 \leq z_2 \leq \cdots \leq z_k$. 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 π_i and π_{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 π_i and π_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:

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

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

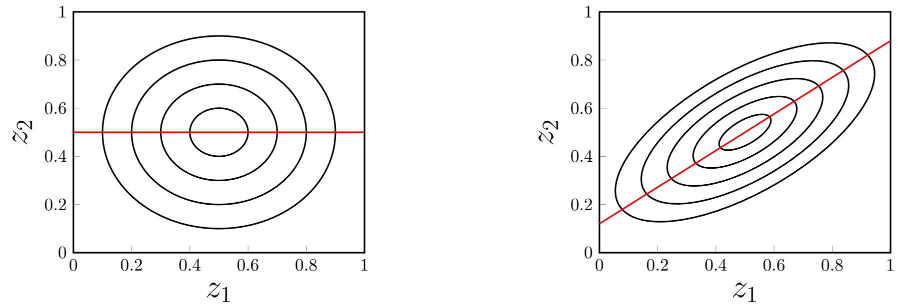
The fact that this this matrix indeed introduces a dependency between each pair of variables z_i and z_j can be seen clearly from Figure 4.2. In this figure we have drawn two contour plots for a Gaussian distribution over just two variables z_1 and z_2. Figure 4.2a shows the case where $\mathbf{K}$ is 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 $\mathbb{E}\Big((z_i-\mu)\cdot(z_j-\mu)\Big)$ 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 $\kappa : \mathbb{R}^2 \to \mathbb{R}$ 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:

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



(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.

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 π_i and π_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. [50] 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 $P(z_{k+1} \mid z_1, z_2, \ldots, z_k)$. 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 $z_1, z_2, \ldots, z_k$.

The expression for $P(z_{k+1} \mid z_1, z_2, \ldots, z_k)$ 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 $P(z_{k+1} \mid z_1, z_2, \ldots, z_k)$ will again be a Gaussian distribution. Therefore, this distribution is determined by just two parameters μ and σ, representing the mean and standard deviation.

Note that, technically, the probability $P(z_{k+1} \mid z_1, z_2, \ldots, z_k)$ also depends on the times $t_1, t_2, \ldots, t_k$, 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 κ, as in Eq. (4.24). We may therefore write this probability more correctly as $P(z_{k+1} \mid \pi_1, \pi_2, \ldots, \pi_k, t_{k+1})$

Finally, let us mention that instead of using *all* received proposals from the opponent as their input, Williams et al. [50] 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 β^* 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 μ and σ, which are the mean and the standard deviation of the Gaussian probability distribution that represents the probability that the opponent will propose an offer ω_{k+1} at time t_{k+1} with utility z_{k+1}:

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

Now, let us suppose for a moment that we know the exact value z_{k+1} of the offer ω_{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 ω for which $u_1(\omega) < z_{k+1}$. After all, if, *for our agent*, the offer ω is worse than the offer ω_{k+1} that the opponent would propose, then by Pareto-optimality, *for the opponent*, the offer ω would be *better* than the offer ω_{k+1} that he would propose. So, it is fair to assume that the opponent would be willing to accept ω.

Of course, in reality we only have a *probability distribution* for z_{k+1}, so we can calculate, for any offer ω 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:

$$P_{\mathfrak{a}}(z) = \int_z^\infty P(z_{k+1} \mid \pi_1, \pi_2, \ldots \pi_k)\; dz_{k+1}$$

where $P_{\mathfrak{a}}(z)$ denotes the probability that ag_2 would accept an offer ω with utility $u_1(\omega) = z$.

Let us now make the pessimistic assumption that if our target value is β, 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 $\beta \cdot P_{\mathfrak{a}}(\beta)$. That is, the utility β 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 β^* as follows:

$$\beta^* = \arg\max_{\beta} \; \beta \cdot P_{\mathfrak{a}}(\beta)$$

4.3 Learning the Opponent's Strategy from Previous Negotiation Sessions

COMING SOON!

Chapter 5

Game Theory

In Chapter 3 we discussed various negotiation strategies. The big question now, is which one is the "best". Unfortunately, it turns out that there is no definitive answer to this question. Nevertheless, we may still want to investigate how close we can get to such an answer, and for that it is absolutely essential to have a basic understanding of the topic of game theory.

Game theory, as the name indicates, deals with the analysis of games. However, it should be understood that the notion of a 'game' here is much more general than what one would normally consider a game in daily life. Specifically, *game theory applies to any scenario that involves multiple agents whose goals are at least partially conflicting, and in which the outcome for each agent also depends on the the actions taken by the other agents.* In particular, this means it applies to automated negotiation.

Game theory is a very large subject and it would go much too far to go into an in-depth discussion in this book. Therefore, we will here only explain the most basic concepts that are relevant for the rest of this book. For a more in-depth study of game theory I recommend the book '*A Course in Game Theory*' by Osborne and Rubinstein [42].

Readers who are not interested in theory may want to skip this chapter, since most of it will not come back in the rest of this book. The only exception is that the theory of normal-form games, discussed in Section 5.2, as well as the notions of a symmetric game and symmetric equilbria discussed in Section 5.3.4 will come back later in Section 6.2.

5.1 Cooperative vs. Non-Cooperative Game Theory

In general, in game theory it is assumed that there are two or more agents, that each agent can perform certain actions, and that each agent chooses its actions so as to maximize its own individual utility function. Furthermore, it is assumed that for each agent, its utility function does not only depend on the agent's own actions, but also on the actions of the other agents.

We can distinguish between two main branches of game theory, namely *cooperative* game theory, and *non-cooperative* game theory. The difference is that in the case of cooperative game theory it is assumed that the agents are able to coordinate their actions, which may allow them to achieve outcomes that are mutually beneficial. In non-cooperative game theory, on the other hand, it is assumed that each agent chooses its actions in an entirely individual way, without any form of explicit coordination with the other agents.

Another way to see it, is to say that non-cooperative game theory purely focuses on the question which actions each agent will take, while cooperative game theory assumes that there is a kind of 'communication layer' superimposed on top of the game, which allows the agents to coordinate or negotiate the actions they will take.

It should be understood however, that even in the case of cooperative game theory, each agent is still assumed to have its own individual utility function and that each agent is still assumed to be purely self-interested. In other words, an agent is only willing to cooperate with the other agents if that yields an individual benefit to that agent. Therefore, cooperative game theory should not be confused with *distributed optimization* in which all agents share the same goal or utility function and are all programmed to work together.

We can summarize the differences as follows.

- **Distributed Optimization**:
  - All agents have the same goals.
  - The agents work together to achieve their common goals.
  - **Example**: A swarm of fire-fighting drones that aim to extinguish a bush fire.
- **Cooperative Game Theory**:
  - Each agent has its own individual goals, which may conflict with the goals of the other agents.

  - Agents may work together, but they only do so if that benefits them individually.
  - **Example**: Political parties that form coalitions to create a government.

- **Non-Cooperative Game Theory:**
  - Each agent has its own individual goals, which may conflict with the goals of the other agents.
  - No cooperation or coordination between the agents at all. Each agent chooses its actions purely individually.
  - **Example**: a game of chess.

Automated negotiation is clearly related to cooperative game theory, since indeed it considers agents that are aiming to find a joint solution, but only if that increases their own individual utilities. In fact, one could see automated negotiation as a sub-field of cooperative game theory, although in practice the literature usually treats them as two distinct fields. A main difference, is that in the field of cooperative game theory one typically assumes that all agents have full knowledge of each others' utility functions, while in automated negotiation we usually assume the agents only have limited or no knowledge about their opponents' utility functions. Furthermore, in automated negotiation we mainly focus on the *process* of how the agents agree on some final outcome (i.e. the negotiation), while in most work on cooperative game theory this process is entirely abstracted away and one only focuses on the *outcome* of such negotiations.

Given the close relationship between automated negotiation and cooperative game theory, it may come as a surprise that in this section and in the rest of this book we are actually more interested in *non-cooperative* game theory, rather than in cooperative game theory. The reason for this, is that in order to determine which negotiation strategies are best, we need to model the process of negotiation *itself* as a game. This contrasts with cooperative game-theory, in which negotiation is considered as a process that is superimposed *on top of* a game. So, if we model negotiation itself as a game, it would be a non-cooperative one.

Within the field of non-cooperative game theory, we can further distinguish between two main types of games:

1. Normal-form games
2. Extensive-form games

Normal-form games are games in which all players simultaneously choose exactly one action and then the game is over. Probably the most well-known example of a normal-form game is 'Paper-Scissors-Rock'. Extensive-form games, on the other hand, are the more common type of games that take place over multiple rounds. Examples are chess, go, and poker. We will first discuss normal-form games in Sections 5.2 and 5.3, and then we will discuss extensive-form games in Section 5.4. Finally, in Sections 5.5 and 5.6 we will apply our knowledge of game theory to the topic of automated negotiation.

5.2 Normal-Form Games

Formally, normal-form games are defined as follows.

Definition 5.2.1. *Let n be a positive integer. Then, an n-player* ***normal-form game*** *consists of:*

- *For each $i \in \{1, 2, \ldots, n\}$ a set of* ***actions*** *A_i (sometimes also referred to as* ***moves****).*
- *For each $i \in \{1, 2, \ldots, n\}$ a* ***utility function*** *u_i that maps the Cartesian product of all action sets to the set of real numbers:*

$$u_i \quad : \quad A_1 \times A_2 \times \cdots \times A_n \to \mathbb{R}$$

Note that in game theory the agents are typically referred to as 'players'. So, we will refer to each set A_i as the set of "actions of player i" and to each utility function u_i as the "utility function of player i". Furthermore, we may use the notation ag_i to refer to player i. In the rest of this section we will mainly focus on 2-player games.

A tuple of actions, consisting of one action for each player is called an **action profile**. In other words, an action profile is an element of the set $A_1 \times A_2 \times \cdots \times A_n$.

Note that for each player, its utility function depends on the actions chosen by *all* players. For example, in the case of Papers-Scissors-Rock (with two players), each player has the same action set $A_1 = A_2 = \{paper, scissors, rock\}$. The utility function u_1 for player 1 could be given by:

$$\begin{aligned}
u_1(paper, paper) = 1, \quad & u_1(paper, scissors) = 0, \quad & u_1(paper, rock) = 2 \\
u_1(scissors, paper) = 2, \quad & u_1(scissors, scissors) = 1, \quad & u_1(scissors, rock) = 0 \\
u_1(rock, paper) = 0, \quad & u_1(rock, scissors) = 2, \quad & u_1(rock, rock) = 1
\end{aligned}$$

That is, player 1 receives 2 utility 'points' if she wins, 0 utility points if she loses, and 1 utility point in case of a draw. Similarly, the utility function for player 2 can then be defined as $u_2(a_1, a_2) = 2 - u_1(a_1, a_2)$, for any pair of actions $(a_1, a_2) \in A_1 \times A_2$.

Two-player normal-form games are typically represented using so-called *pay-off matrices*. That is, a matrix for which each row corresponds to an action $a_1 \in A_1$, and each column corresponds to an action $a_2 \in A_2$, so it's an $|A_1| \times |A_2|$ matrix. Each cell of the matrix therefore corresponds to a pair of actions a_1, a_2 and it contains the corresponding pair of utility values $(u_1(a_1, a_2)\ ,\ u_2(a_1, a_2))$ for the two players. For example, the payoff matrix of Paper-Scissors-Rock is displayed in Table 5.1.

	Paper	Scissors	Rock
Paper	(1 , 1)	(0 , 2)	(2 , 0)
Scissors	(2 , 0)	(1 , 1)	(0 , 2)
Rock	(0 , 2)	(2 , 0)	(1 , 1)

Table 5.1: Payoff-matrix of the game Paper-Scissors-Rock

In this book we will always follow the convention that player 1 is the '*row player*' and that player 2 is the '*column player*'. That is, the rows of the matrix correspond to the actions of player 1, and the columns correspond to the actions of player 2.

For any action profile (a_1, a_2) we may use the notation $\vec{u}(a_1, a_2)$ to denote the utility vector of that profile. That is:

$$\vec{u}(a_1, a_2) \quad := \quad (\ u_1(a_1, a_2)\ ,\ u_2(a_1, a_2)\ )$$

5.2.1 Zero-sum Games

Note that in the game of 2-player Paper-Scissors-Rock, no matter what actions the players choose, the sum of their respective utilities $(u_1 + u_2)$ will always be 2. In other words, the agents' objectives are diametrically opposed. The higher the utility for player ag_1, the lower the utility for player ag_2 and vice versa. Such games are also known as **constant-sum games** or, more commonly, **zero-sum games**. This last name comes from the fact that we can add any arbitrary constant to the utility function of either player, without affecting the essence of the game (as per the principle of 'Invariance under Linear Transformations', see Def. 2.2.6). Therefore, any constant-sum game can be transformed into an equivalent game for which the sum of the players' utility values is always exactly zero. Games in which the sum of the players' utility values is not always the same are called **non-zero-sum games** or **general-sum games**.

Many board games such as chess, checkers, or go, can indeed be seen as zero-sum games because they either end with one player as the winner and the other as the loser, or in a draw. So, we can assign 2 points to the winner, 0 points to the loser, and 1 point to each player in case of a draw. Conversely, for any 2-player zero-sum game we can say that the player that achieved the highest utility is the 'winner' and the other player the 'loser', or that the game ended in a draw if both players achieved the same utility.

However, it is important to understand that when we study *non*-zero-sum games there is not always a clear winner or loser. For example, one could encounter a game that has one action profile for which both players achieve the maximum utility, while it also has one action profile for which both players achieve the minimum utility. Therefore, in such games we cannot say that the goal is to *win* the game. Instead, *the goal for each player is purely to maximize its own utility value.* Especially, we should stress that in non-zero-sum games *it is not the goal of the players to 'beat' the opponent, or to achieve more utility than the opponent.*

For example, if one action profile leads to a utility of 10 for player 1 and a utility of 5 for player 2, while another action profile yields a utility of 100 for player 1 and a utility of 200 for player 2, then player 1 prefers the second action profile, because it yields more utility. In particular, player 1 does *not* care about the fact that with the second action profile player 2 achieves more utility than player 1.

5.2.2 Simultaneous Moves

As we mentioned above, in a normal-form game the players choose their actions simultaneously. What we mean by this, is that each player has to choose his or her action without knowing which actions the other players are choosing. It does *not* mean that the players *literally* have to choose their actions at exactly the same moment. Instead, we can imagine, for example, that each player first secretly writes down his action on a piece of paper and only once all players have written down their chosen actions, those actions are revealed. While in this way the players do not literally choose their actions at exactly the same moment, the point is that each player has to make his choice without knowing the choices of the other players, which, for all intents and purposes, is the same as the situation that all agents really do choose their actions at exactly the same time.

5.2.3 Pure Nash Equilibria

Naturally, the main question any player in any game wants to answer, is the question which action is the best action to choose. In order to study this question we will focus on 2-player games and we will assume that each player has full knowledge of the other player's utility function.

If we knew which action the opponent was choosing, then this question would be easy to answer, because then our best action would simply be the one that maximizes our utility, given the opponent's action. We call this the *best response* to the opponent's action.

Definition 5.2.2. *Let G be some 2-player normal-form game and let $a_1 \in A_1$ be any action for player 1. Then, we say that an action $a_2 \in A_2$ for player 2 is a* ***best response*** *to a_1 if the following holds:*

$$\forall a \in A_2 : \quad u_2(a_1, a) \quad \leq \quad u_2(a_1, a_2)$$

Analogously, an action $a_1 \in A_1$ for player 1 is a ***best response*** *to some action $a_2 \in A_2$ for player 2, if the following holds:*

$$\forall a \in A_1 : \quad u_1(a, a_2) \quad \leq \quad u_1(a_1, a_2)$$

In other words, for any action a_i of player i, a 'best response' for player j is an action that yields highest utility to player j, when player i chooses action a_i.

For example, in the game of 'Paper-Scissors-Rock', if player 1 chooses the action 'scissors' then the best response for player 2 is to choose 'rock'.

Note that the best response may not be unique, because multiple actions may yield the same utility. Therefore, in general, for any action a_i there is a *set* of actions which are all best responses. We will denote this set by $BR_j(a_i)$. That is:

$$\begin{aligned} BR_1(a_2) \quad &:= \quad \arg\max_a \quad \{u_1(a, a_2) \mid a \in A_1\} \\ BR_2(a_1) \quad &:= \quad \arg\max_a \quad \{u_2(a_1, a) \mid a \in A_2\} \end{aligned}$$

So, the phrase "*a_j is a best response to a_i*" can be formally denoted as $a_j \in BR_j(a_i)$.

Of course, the problem is that, in principle, we do not know the opponent's action. However, to solve this, we can assume that the opponent is rational, which may allow us to *reason* about what action the opponent would choose.

In the following we will follow our usual convention that we are implementing agent ag_1 and therefore that ag_2 is our opponent.

The idea is as follows. Before the game starts, we first choose some arbitrary action $a_1 \in A_1$. We then assume that, if there is indeed a good reason for us to pick that action, then the opponent would be able to follow that reasoning and therefore would be able to conclude that we are picking a_1. But that means that if the opponent is rational she would now choose an action a_2 that is a best response against our action (i.e. $a_2 \in BR_2(a_1)$). Now, assuming that the opponent will indeed choose that action, we can change our mind (before the game starts), and instead pick a new action a'_1 that is a best response to *that* action a_2. That is, we choose $a'_1 \in BR_1(a_2)$. Now, again, we can make the assumption that the opponent is able to reason in the same way as us, and therefore is able to anticipate our change of mind, which allows her to also change her mind, and pick a best response to our new choice. That is, we now assume the opponent will actually choose some action $a'_2 \in BR_2(a'_1)$. If we keep reasoning like this, then either of the following two things can happen:

1. The two players keep changing their actions infinitely often.
2. At some point they reach an equilibrium were neither of the two players changes their mind anymore, because they have chosen two actions that are best responses *to each other*.

In the second case, we say the players have reached a *Nash equilibrium*. More precisely, we say the two players have reached a *pure* Nash equilibrium. There also exists a different kind of equilibrium that is called a *mixed* Nash equilibrium, but we will discuss that later on.

It is important to understand that this process of players changing their actions until they reach equilibrium, only describes the *thought process* of the players *before the game has started*. In other words, it only takes place *in their minds*. After all, once the game starts, the player reveal their moves simultaneously, and after that they cannot change their moves anymore.

Formally, a pure Nash equilibrium is a pair of actions, such that each of the two actions is a best response to the other one.

Definition 5.2.3. *Let $(a_1, a_2) \in A_1 \times A_2$ be any pair of actions of a two-player normal-form game. We say it is a **pure Nash equilibrium** iff:*

$$a_1 \in BR_1(a_2) \quad \textit{and} \quad a_2 \in BR_2(a_1)$$

The following observation states the importance of Nash equilibria.

Observation. *If a normal-form game has exactly one Nash equilibrium, then the action profile chosen by two optimal players would be exactly that Nash equilibrium.*

To see this, assume the opposite. Suppose that they choose an action profile (a_1, a_2), that is not a Nash equilibrium. In particular, let us assume that a_1 is not a best response to a_2. That means that player 1 could have achieved more utility if he had chosen a different action a'_1 that *is* a best response to a_2 (i.e. $a'_1 \in BR(a_2)$). So, by choosing a_1 player 1 did not make an optimal choice, which contradicts the assumption that they were playing optimally.

Now, imagine that before they play the game, all players have decided which action they each will play. However, suppose that right before they reveal their chosen actions, one player changes his mind and switches to another action, *while the other player keeps her decisions unchanged.* We then say the first player is making a **unilateral deviation**. With this terminology the notion of a pure Nash equilibrium can be defined alternatively as: "*a strategy profile such that no agent can increase his utility by making a unilateral deviation*".

Unfortunately, not all games have a pure Nash equilibrium. One example is the Paper-Scissors-Rock game. If we apply our reasoning above to this game, it is easy to see that we keep looping forever. For example, if we initially choose 'paper', then our opponent will choose the best response, which is 'scissors'. Then, we can change our mind and choose the best response against 'scissors', which is 'rock'. Next, the opponent will change to the best response against 'rock' which is 'paper', etcetera. Clearly, this will continue forever.

An example of a game that does have a pure Nash equilibrium, is the well-known Prisoner's dilemma, which we will discuss next.

5.2.4 The Prisoner's Dilemma

The prisoner's dilemma is probably the most commonly used example in game theory, because it shows the counter-intuitive result that when every player plays optimally from his own individual point of view, the final outcome may actually turn out to be very bad for each individual player.

The prisoner's dilemma is typically explained as follows: two prisoners are each being questioned separately by the police. They each have two options: to confess that they committed a crime, or to deny that they did it. If they both confess then they both have to stay in prison for 8 years.

If they both deny, then they both only have to stay in prison for 2 years. However, if one of them denies and the other confesses, then the one who confessed will be released from prison immediately and be free, while the other one will have to stay in prison for 10 years.

We should stress that we are discussing this game in the context of non-cooperative game theory, so the prisoners are not able to communicate and each of them has to make his decision in complete isolation from the other.

This game can be displayed as the following payoff matrix.

	Deny	Confess
Deny	(8 , 8)	(0 , 10)
Confess	(10 , 0)	(2 , 2)

Note that the utilities here are given as $10 - x$, where x is the number of years they stay in prison. So, if a prisoner is released immediately he will get a utility of 10. The payoff vector $(8, 8)$ represents that they both go to prison for 2 years, while the payoff vector $(2, 2)$ represents that they both go to prison for 8 years. This is because we follow the standard convention that the matrix displays *utility* values, which the players are aiming to *maximize*.

Now, the question is what the optimal strategy for each of the two prisoners would be. Most people who see this game for the first time would argue that the best strategy is to play 'deny', because if both players choose that action, they will both get a low punishment. However, perhaps surprisingly, we will see that the optimal strategy is actually to play 'confess'.

To see this, let us first imagine that player 1 is choosing to play 'deny'. What is now the best response for player 2? We see from the matrix that if player 2 chooses 'deny' as well, then she receives a utility of 8 (2 years in prison), while if she chooses 'confess' she receives a utility of 10 (immediate freedom). So, 'confess' is the best response.

$$BR_2(deny) = \{confess\}$$

Next, suppose that player 1 chooses to play 'confess'. We now see that if player 2 chooses 'deny' she will get a utility of 0 (i.e. 10 years in prison), while if she chooses 'confess' she will get a utility of 2 (i.e. 8 years in prison). Again, we see that 'confess' is the best option.

$$BR_2(confess) = \{confess\}$$

In other words: *No matter what player 1 chooses, player 2 is always better off if she chooses 'confess'.* Vice versa, the same holds for player 1. Player 1 is always better off by playing 'confess', no matter what player 2 chooses.

We therefore see that the action profile (*confess*, *confess*) is the unique pure Nash equilibrium of this game. From this we conclude that if both players are perfectly rational, they would each choose to play 'confess' and therefore they would each go to prison for 8 years.

The conclusion of our analysis may seem highly counter-intuitive, because if they cooperated they could have ensured to go to prison for only 2 years. The problem with that cooperative solution, however, is that even if the players could somehow make an agreement to each play 'deny', then, *by assumption of non-cooperative game theory*, still neither of the two players could be forced to keep their promise. So, if you agree with your opponent to play 'deny', then the best thing you can do is to break your promise and play 'confess' anyway. Formally speaking, we say that players cannot *commit* to their actions in advance.

The reason this outcome seems so counter-intuitive, is that in real life most situations we encounter do not follow the strict rules of non-cooperative game theory. For example:

- In real life people are social:
  - The prisoners could be friends or family that prefer to help each other rather than to make purely selfish choices.
  - People are hardwired to often be helpful and friendly, even to strangers.
- In real life, people may fear repercussions if they betray others.
- In real life, people *can* commit to their actions:
  - They can sign legally binding contracts.
  - They may feel obliged to keep their promises as a matter of honor.

On the other hand, in non-cooperative game theory we assume:

- that the players are *only* interested in maximizing their own individual utility functions,
- that each game is played in complete isolation, so actions in the current game do not have repercussions in later games,
- that players cannot commit in advance to their actions.

Note that indeed, as per the definition of a Nash equilibrium, neither of the two players can increase their utility by making a *unilateral* deviation. On the other hand, in the prisoner's dilemma it *is* possible for the players to increase their utility if they *both* switch from 'confess' to 'deny'. In other words, if they make a *bilateral* deviation. However, the definition of a Nash equilibrium does not take such bilateral deviations into consideration. The reason for this, again, is that we are talking about *non-cooperative* game theory, which, by definition, assumes the players cannot coordinate their actions. So, whenever a player switches to a different action, he has to assume that this will not affect the opponent, and thus that the opponent's action remains unchanged.

5.2.5 Multiple Pure Nash Equilibria

As discussed above, some games, such as Paper-Scissors-Rock, do not have any pure Nash equilibrium. Other games, on the other hand, have the problem that they actually have *multiple* pure Nash equilibria.

A simple example is the game known as 'Battle of the Sexes'. It can be explained as follows. The two players are a married couple and they want to go out. They each can choose between two options: to go to a football match or to go to a ballet performance. While the husband prefers to see the football match, the wife prefers to go to the ballet performance. However, for both, the most important thing is that they go together. That is, they each prefer to choose the same activity, rather than that they each choose a different activity. This can be summarized in the following payoff matrix:

	Football	Ballet
Football	(2 , 1)	(0 , 0)
Ballet	(0 , 0)	(1 , 2)

Note that no matter what the wife chooses, the best response for the husband is to choose the same option, and similarly, no matter what the husband chooses, the best response for the wife is also to choose the same option:

$$\forall i \in \{1, 2\} : \quad BR_i(\mathit{football}) = \{\mathit{football}\} \;\text{ and }\; BR_i(\mathit{ballet}) = \{\mathit{ballet}\}$$

This means that there are two pure Nash equilibria:

$$(\mathit{football}, \mathit{football}) \quad \text{and} \quad (\mathit{ballet}, \mathit{ballet})$$

5.2.6 Mixed Nash Equilibria

We have seen that the Paper-Scissors-Rock game does not have any pure Nash equilibria. No matter which of the three actions we choose, if the opponent can anticipate our action, then she can choose the best response to that action, and we lose. So, how then do we determine our optimal strategy? The answer is simple: by making sure that the opponent cannot anticipate our action. Specifically, we can do that by picking an action randomly. We call this a *mixed strategy.*

Definition 5.2.4. *Let A_i be the set of actions of player i. Then, a* ***mixed strategy*** *for player i is a probability distribution over the set A_i. That is, a function $m : A_i \to \mathbb{R}$ such that $\sum_{a_i \in A_i} m(a_i) = 1$ and $\forall a_i \in A_i : m(a_i) \geq 0$. We will denote the set of all mixed strategies of player i by $\mathcal{M}_i$.*

The interpretation is that the player selects each action a_i with probability $m(a_i)$. Note that even if the game only has a finite number of actions, each player has an infinite number of possible mixed strategies.

Whenever a player does not choose his action randomly, but instead just chooses one specific action deterministically, then this is also known as a **pure strategy**. Of course, one can say that a pure strategy is actually just a special case of a mixed strategy, for which there is exactly one action a_i with $m(a_i) = 1$ and therefore $m(a_i') = 0$ for all other actions $a_i' \in A_i$.

A tuple $\vec{m} = (m_1, m_2, \dots, m_n)$ consisting of one mixed strategy for each player is called a **strategy profile**.

Previously, we defined the utility function of a player as a function that assigns a utility value to every possible action profile. This can now be extended to profiles of mixed strategies, by defining it as the *expected* utility over all pure action profiles. That is, for games with two players:

$$u_i(m_1, m_2) \quad := \quad \sum_{a_1 \in A_1} \sum_{a_2 \in A_2} m_1(a_1) \cdot m_2(a_2) \cdot u_i(a_1, a_2)$$

We may use the notation $\vec{u}(m_1, m_2)$ or $\vec{u}(\vec{m})$ to denote the utility vector of $\vec{m}$:

$$\vec{u}(\vec{m}) \quad := \quad (\ u_1(\vec{m})\ ,\ u_2(\vec{m})\ )$$

For example, in the game of Paper-Scissors-Rock, suppose that player ag_1 chooses a mixed strategy m_1 in which he plays ‘paper’ with a probability of 40% and ‘scissors’ with a probability of 60%, and suppose that player ag_2 chooses a mixed strategy m_2 in which she plays ‘scissors’ with a probability of 20% and ‘rock’ with a probability of 80%, then, the expected utility of player 1 will be:

$$\begin{aligned} u_1(m_1, m_2) &= 0.4 \cdot 0.2 \cdot u_1(\textit{paper}, \textit{scissors}) + 0.6 \cdot 0.2 \cdot u_1(\textit{scissors}, \textit{scissors}) + \\ &\quad 0.4 \cdot 0.8 \cdot u_1(\textit{paper}, \textit{rock}) + 0.6 \cdot 0.8 \cdot u_1(\textit{scissors}, \textit{rock}) \\ &= 0.4 \cdot 0.2 \cdot 0 + 0.6 \cdot 0.2 \cdot 1 + 0.4 \cdot 0.8 \cdot 2 + 0.6 \cdot 0.8 \cdot 0 \\ &= 0.76 \end{aligned}$$

while for player ag_2 it will be:

$$\begin{aligned} u_2(m_1, m_2) &= 0.4 \cdot 0.2 \cdot u_2(\textit{paper}, \textit{scissors}) + 0.6 \cdot 0.2 \cdot u_2(\textit{scissors}, \textit{scissors}) + \\ &\quad 0.4 \cdot 0.8 \cdot u_2(\textit{paper}, \textit{rock}) + 0.6 \cdot 0.8 \cdot u_2(\textit{scissors}, \textit{rock}) \\ &= 0.4 \cdot 0.2 \cdot 2 + 0.6 \cdot 0.2 \cdot 1 + 0.4 \cdot 0.8 \cdot 0 + 0.6 \cdot 0.8 \cdot 2 \\ &= 1.24 \end{aligned}$$

This, in turn, allows us to extend the definition of 'best response' to mixed strategies.

Definition 5.2.5. *Let G be some two-player normal-form game and let $m_1 \in \mathcal{M}_1$ be a mixed strategy for player 1. Then, we say that a mixed strategy $m_2 \in \mathcal{M}_2$ for player 2 is a* ***best response*** *to m_1 if the following holds:*

$$\forall m \in \mathcal{M}_2 : \quad u_2(m_1, m) \quad \leq \quad u_2(m_1, m_2)$$

Analogously, a mixed strategy $m_1 \in \mathcal{M}_1$ for player 1 is a ***best response*** *to some mixed strategy $m_2 \in \mathcal{M}_2$ for player 2, if the following holds:*

$$\forall m \in \mathcal{M}_1 : \quad u_1(m, m_2) \quad \leq \quad u_1(m_1, m_2)$$

As before, we use the notation $BR_j(m_i)$ to denote the set of best responses to a mixed strategy m_i.

$$\begin{aligned} BR_1(m_2) \quad &:= \quad \arg\max_{m_1} \quad \{u_1(m_1, m_2) \mid m_1 \in \mathcal{M}_1\} \\ BR_2(m_1) \quad &:= \quad \arg\max_{m_2} \quad \{u_2(m_1, m_2) \mid m_2 \in \mathcal{M}_2\} \end{aligned}$$

Finally, we can now also generalize the concept of a pure Nash equilibrium to mixed strategies.

Definition 5.2.6. *Let* (m_1, m_2) *be any pair of mixed strategies of a two-player normal-form game. We say it is a* ***mixed Nash equilibrium*** *if:*

$$m_1 \in BR_1(m_2) \quad \textit{and} \quad m_2 \in BR_2(m_1)$$

It can be shown that every pure Nash equilibrium is also a mixed Nash equilibrium (if we consider a pure strategy to be a special case of a mixed strategy). To prove this, one must show that if a player cannot deviate to a better action, he also cannot deviate to a better mixed strategy. It is not hard to see that this is indeed true, so we will leave this as an exercise to the reader. We refer to [42] for more details.

While we have seen that not every game has a pure Nash equilibrium, it turns out that every finite 2-player normal-form game does have at least one mixed Nash equilibrium. A proof of this theorem can be found in [42].

Theorem 1. *Every finite 2-player normal-form game has at least one mixed Nash equilibrium.*

It is relatively straightforward to determine the pure Nash equilibria of a normal-form game. All it amounts to is to determine for each action of either player which actions are best responses. This can be seen directly from the pay-off matrix. Determining the *mixed* Nash equilibria, on the other hand, is a computationally hard problem that you would typically not do manually. Instead there are various algorithms for this task, such as the Lemke-Howson algorithm [34]. A commonly used software package that implements such algorithms is the Gambit library [46].

5.3 The Equilibrium Selection Problem

As mentioned above, our aim is to determine, for any given normal-form game, what the optimal strategy would be for each of the players. So far, we have only partially answered this question. Namely, we now know that the players should be playing a Nash equilibrium (pure or mixed). Furthermore, we know from Theorem 1 that such a Nash equilibrium always exists. However, that still leaves us with the question *which* Nash equilibrium to choose if the game has *multiple* Nash equilibria. This problem is known as the *equilibrium selection problem* (ESP). This is especially important for us because, as we will see later on (in Sec. 5.5.8), in automated negotiation there are typically indeed many Nash equilibria.

While many solutions to the ESP have been proposed in the literature, none of them is widely accepted as being fully satisfactory for general normal-form games. However, there are a number of solutions to this problem that are applicable to special cases. We will here discuss some of them. But before that, we will first discuss some apparent solutions to the ESP that might seem to make sense at first, but that, upon closer inspection, actually turn out not to be satisfactory.

5.3.1 Wrong Solutions to the Equilibrium Selection Problem

A naive solution to the ESP, would be to assume that a player could simply flip a coin to choose one of the equilibria at random and then play his strategy from that equilibrium. However, we will see that this solution typically does not work.

Suppose that a certain 2-player game has exactly two Nash equilibria: (m_1, m_2) and (m_1', m_2'). Now, suppose that player 1 flips a coin to choose the first equilibrium with probability P and the second equilibrium with probability $1 - P$. The problem, is that this means that essentially, player 1 is playing neither m_1 nor m_1', but in fact is playing an entirely different mixed strategy, namely: $P \cdot m_1 + (1 - P) \cdot m_1'$. And since we assumed there were only two Nash equilibria, this means that player 1 is in fact not playing any equilibrium strategy *at all*. He's playing a different mixed strategy that may not be a best response to the opponent's strategy. Therefore, if player 2 could reason that player 1 is playing that strategy, then player 2 could play a best response against it, which may yield a much better outcome for player 2 (and a much worse outcome for player 1) than if they had played either of the Nash equilibria. Furthermore, it would mean that player 1 could improve by deviating to a different strategy and therefore that it is currently not playing an optimal strategy.

Another idea could be that player 1 chooses a Nash equilibrium based on some entirely different criterion that is not related to his utility function at all. For example, for each of his potential strategies m_1 and m_1', he could look at the *name* of the action that receives the highest probability, and then select the strategy for which this name comes earliest in alphabetical order. However, this solution suffers from essentially the same problem. Since the choice of player 1 is not based on his utility function, player 2 cannot reason which strategy player 1 would choose, and therefore instead has to *guess* it. Therefore, player 2 would reason that there is a 50% chance that player 1 chooses strategy m_1 ad a 50% chance that player 1 chooses strategy m_1'. This means that the optimal strategy for player 2 would be to pick the best response against $0.5 \cdot m_1 + 0.5 \cdot m_1'$. Again, this would typically mean that the players end up playing an entirely different strategy profile, which is not a Nash equilibrium.

5.3.2 Factorizable Sets

The first valid solution to the ESP that we will discuss, applies only to the very special case that all Nash equilibria have exactly the same utility vector and, moreover, the set of all Nash equilibria happens to be *factorizable* or *semi-factorizable.*

Definition 5.3.1. *Let G be a 2-player normal-form game and let $C \subseteq \mathcal{M}_1 \times \mathcal{M}_2$ be a set of strategy profiles of G. We say that C is* ***factorizable,*** *if for each player ag_i there is a set of strategies $S_i \subseteq \mathcal{M}_i$ such that C is the Cartesian product of those two sets: $C = S_1 \times S_2$.*

For example, suppose the set C consists of the following four strategy profiles:

$$C \quad = \quad \{\ (m_1, m_2)\ ,\ (m'_1, m_2)\ ,\ (m_1, m'_2)\ ,\ (m'_1, m'_2)\ \}$$

then C is indeed factorizable, because it can be written as:

$$C = \{m_1, m'_1\} \times \{m_2, m'_2\}$$

That is, every combination of m_1 or m'_1 with either m_2 or m'_2 is contained in C. On the other hand, if we have

$$C = \{\ (m_1, m_2)\ ,\ (m'_1, m'_2)\ \}$$

then C is not factorizable.

Note that any set of strategy profiles C that is a *singleton* set (i.e. it contains exactly one strategy profile), is factorizable.

Now, remember from the previous section that, in general, players cannot just select a Nash equilibrium at random, because if they do try to do that, then the resulting strategy profile may actually turn out not to be a Nash equilibrium. However, if it happens that the players are completely indifferent between all Nash equilibria of the game and on top of that the set of all Nash equilibria happens to be factorizable, then that argument no longer holds and the players can safely choose any random Nash equilibrium.

Observation. *If, for some 2-player normal form game G the set of all Nash equilibria NE happens to satisfy the following two conditions:*

1. *NE is factorizable.*

2. Every element of NE has the same utility vector:

$$\forall \vec{m}, \vec{m}' \in NE : \quad \vec{u}(\vec{m}) = \vec{u}(\vec{m}')$$

then each agent can simply pick a random Nash equilibrium and play his strategy according to that equilibrium.

Note that the second condition is necessary to ensure both agents are completely indifferent between all possible Nash equilibria. After all, if an agent did prefer some equilibria over some other equilibria, then it would not be optimal to just select any arbitrary equilibrium at random.

Next, we will see that this observation can be improved, with the following definition.

Definition 5.3.2. *Let G be a 2-player normal-form game and let $C \subseteq \mathcal{M}_1 \times \mathcal{M}_2$ be a set of strategy profiles of G. Furthermore, let us define F_i to be the set of all strategies of agent ag_i that appear in any of those strategy profiles:*

$$F_1 := \{m_1 \in \mathcal{M}_1 \mid \exists m_2 \in \mathcal{M}_2 : (m_1, m_2) \in C\}$$

$$F_2 := \{m_2 \in \mathcal{M}_2 \mid \exists m_1 \in \mathcal{M}_1 : (m_1, m_2) \in C\}$$

*We say that C is **semi-factorizable**, if for each player ag_i we can find a set $S_i \subseteq \mathcal{M}_i$ such that the following conditions both hold:*

1. $S_1 \times F_2 \subseteq C$
2. $F_1 \times S_2 \subseteq C$

*we will refer to the sets S_1 and S_2 as **safe sets**, and their elements as **safe strategies**.*

Now, if we assume that the two agents will each choose a strategy profile from C and then play their own strategy from that profile, then the first condition guarantees that if ag_1 chooses a strategy from S_1, then the two strategies chosen by the two players will definitely be in C. The second condition says the same, but for ag_2. In other words, each player ag_i can choose any arbitrary safe strategy without having to worry about the choice of the opponent.

Note that each factorizable set is also semi-factorizable (with $S_i = F_i$). An example of a set that is semi-factorizable but not factorizable, is the following:

$$C \quad = \quad \{\ (m_1, m_2)\ ,\ (m_1', m_2)\ ,\ (m_1, m_2')\ \}$$

with $S_1 = \{m_1\}$ and $S_2 = \{m_2\}$. Indeed, we have:

$$S_1 \times F_2 = \{m_1\} \times \{m_2, m_2'\} = \{ (m_1, m_2) , (m_1, m_2') \} \subseteq C$$

$$F_1 \times S_2 = \{m_1, m_1'\} \times \{m_2\} = \{ (m_1, m_2) , (m_1', m_2) \} \subseteq C$$

Observation. *If, for some 2-player normal form game G the set of all Nash equilibria NE happens to satisfy the following two conditions:*

1. *NE is semi-factorizable.*
2. *Every element of NE has the same utility vector:*

$$\forall \vec{m}, \vec{m}' \in NE : \quad \vec{u}(\vec{m}) = \vec{u}(\vec{m}')$$

then each agent can pick a random strategy from its safe set S_i.

While this exact situation may not happen very often, this solution is still very important because we will later see that the same ideas can be used as a refinement on the other solutions that we will discuss next.

5.3.3 Pareto-Optimality among Nash Equilibria

Perhaps the most obvious way to partially resolve the equilibrium selection problem, is to argue that players would never choose a Nash equilibrium that is dominated by some other Nash equilibrium.

In Section 2.3 we gave the definition of 'domination' and 'Pareto-optimality' for offers. The same concepts can also be defined for strategy profiles.

Definition 5.3.3. *We say that a strategy profile $\vec{m}$* ***dominates*** *another strategy profile $\vec{m}'$ if:*

$$\forall i \in \{1, 2\} \ : \ u_i(\vec{m}) \geq u_i(\vec{m}')$$

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

$$\exists i \in \{1, 2\} \ : \ u_i(\vec{m}) > u_i(\vec{m}')$$

We say a strategy profile $\vec{m}'$ ***is dominated*** *by $\vec{m}$, if $\vec{m}$ dominates $\vec{m}'$. A strategy profile $\vec{m}$ is* ***Pareto optimal*** *if it is not dominated by any other strategy profile.*

Clearly, if a game has two Nash equilibria and one of them yields a utility of 10 to each player, while the other one yields a utility of 20 to each player, then both players would choose the second one.

We therefore argue that in a game with multiple Nash equilibria, the players would only consider choosing those that are Pareto-optimal *among the Nash equilbria.*

Definition 5.3.4. *We say a Nash equilibrium $\vec{m}$ is* ***Pareto-optimal among Nash equilibria****, if it is not dominated by any other Nash equilibrium.*

Note that we make a distinction between a Nash equilibrium being '*Pareto-optimal*' and being '*Pareto-optimal among Nash equilibria*'. The first concept means that it is not dominated by any other *action profile.* The second concept is much weaker because it only says that it is not dominated by any other *Nash equilibrium.*

For example, in the prisoner's dilemma, the Nash equilibrium $(confess, confess)$ is dominated by the action profile $(deny, deny)$. Therefore, $(confess, confess)$ is not Pareto-optimal. However, $(deny, deny)$ is not a Nash equilibrium. So, while $(confess, confess)$ is dominated by some other action profile, it is not dominated by any other Nash equilibrium (after all, it is the *only* Nash equilibrium) and therefore we can say that it is Pareto-optimal *among Nash equilbria.*

Unfortunately, however, this solution still does not completely solve the equilibrium selection problem, because it is perfectly possible for a game to have multiple Nash equilibria that are Pareto-optimal among Nash equilibria.

5.3.4 Symmetric Games and Symmetric Equilibria

There is another way to (partially) solve the equilibrium selection problem, but it only applies to so-called *symmetric* games.

A symmetric game is a game for which it does not matter which player you are, because the game looks exactly the same from the point of view of either player. The game of Paper-Scissors-Rock and the prisoner's dilemma are both examples of symmetric games. In each of these games it clearly does not matter whether you are 'player 1' or 'player 2', because those are just labels. If you switch the players' roles, nothing changes.

To keep things simple we will here give a definition of the concept of a 'symmetric game' that is actually somewhat too strict, but easier to understand than the full definition.

Definition 5.3.5. *Let G be a 2-player normal-form game. We say it is a* ***symmetric game*** *if $A_1 = A_2$, and for any $(a_1, a_2) \in A_1 \times A_2$ we have:*

$$u_1(a_1, a_2) \quad = \quad u_2(a_2, a_1) \tag{5.1}$$

It is easy to see that Paper-Scissors-Rock satisfies this definition. For example, suppose that Alice is player 1 and she plays 'scissors', while Bob is player 2 and he plays 'rock'. Then Alice loses so she receives 0 points. That is, we have: $u_1(\mathit{scissors}, \mathit{rock}) = 0$. Now, imagine that the roles are switched, but that the players still play exactly the same actions. That is, Bob is now player 1, but he still plays 'rock' and Alice is now player 2, but she still plays 'scissors'. Clearly, Bob still wins the game and Alice still receives 0 points. However, because we have switched their 'roles', this is now formalized as: $u_2(\mathit{rock}, \mathit{scissors}) = 0$. Indeed, we see that it doesn't matter who is 'player 1' and who is 'player 2' and that we have $u_1(\mathit{scissors}, \mathit{rock}) = u_2(\mathit{rock}, \mathit{scissors})$, which is indeed an instance of Eq. (5.1).

As we mentioned, Def. 5.3.5 is actually too strict in the sense that it requires the two action sets A_1 and A_2 to be *exactly* equal. This means that if we just change the *names* of the actions of one of the two players, then the game will trivially fail Definition 5.3.5. For example, suppose we said that player 1 still has the actions $A_1 = \{\mathit{paper}, \mathit{scissors}, \mathit{rock}\}$, but that player 2 now has the actions $A_2 = \{\mathit{parrot}, \mathit{sizzlers}, \mathit{rack}\}$. The payoff matrix stays exactly the same as in Table 5.1, but the columns are now labeled with these new actions, while the rows are still labeled with the original actions. Since we now have $A_1 \neq A_2$, this game would—according to Def. 5.3.5—no longer be symmetric. Of course, this should not be the case, because the names of the actions should not matter, so there is clearly something wrong with the definition. A similar problem can occur if we multiply the utility function of one of the two players by a fixed constant. Anyway, we will not go into the details of a proper definition of 'symmetric game'. The given definition suffices for our purposes.

Note that to specify the payoff matrix of a symmetric game, it is sufficient to only provide the utilities of the *row*-player. After all, if for some action profile (a_1, a_2), you want to know the corresponding utility value $u_2(a_1, a_2)$ of the column player, then you can just look for $u_1(a_2, a_1)$ in the table. See Table 5.2.

While a negotiation is typically not a symmetric game (because the two agents typically have different utility functions), the topic of symmetric games is still very important for the study of automated negotiation, as we will see later on in this book when we discuss the evaluation of negotiation strategies using 'empirical game-theoretic analysis'.

	Paper	Scissors	Rock
Paper	1	0	2
Scissors	2	1	0
Rock	0	2	1

Table 5.2: Payoff-matrix for the game Paper-Scissors-Rock, with only the utilities for the row-player. Given the knowledge that it is a symmetric game, it is not necessary to explicitly display the utility values of the column player. For example, if you want to know the utility of the column player for the profile (*paper*, *scissors*), then you can just look up the utility of the row player for the profile (*scissors*, *paper*), which we can see is 2.

We can now define the notion of a symmetric Nash equilibrium (for symmetric games).

Definition 5.3.6. *Let G be a symmetric 2-player normal-form game. We say a strategy profile (m_1, m_2) for this game is a **symmetric Nash equilibrium** if it is a Nash equilibrium, and it satisfies $m_1 = m_2$.*

The following theorem is proven in [13].

Theorem 2. *Any finite symmetric game has a symmetric Nash equilibrium.*

We now claim that in a symmetric game, if the players play optimally, they would choose a symmetric Nash equilibrium.

The idea behind this, is that if the game is perfectly symmetrical, and the players are perfectly rational, then, whenever player 1 reasons that some mixed strategy m is the optimal strategy, player 2 would come to exactly the same conclusion, and thus they would always choose the same mixed strategy. Therefore, the only Nash equilibria they could possibly end up choosing, are the symmetric ones.

However, it can still happen that a symmetric game has multiple symmetric equilibria. In that case, we can apply the Pareto-optimality criterion from Section 5.3.3 to make a choice among the symmetric equilibria. Note that for any symmetric equilibrium (m, m) in a symmetric game, the two players will always receive the same utility: $u_1(m, m) = u_2(m, m)$. Therefore, if we have two symmetric equilibria, with different utility vectors, then one will dominate the other. For example, if one symmetric equilibrium yields utility vector $(20, 20)$ and another one yields utility vector $(10, 10)$, then the first one dominates the second one.

Now, a valid question would be what happens if this solution conflicts with the solution we discussed in Section 5.3.3. That is, what happens if a game is symmetric, but every symmetric equilibrium is dominated by a non-symmetric Nash equilibrium. For example, suppose we have a symmetric Nash equilibrium with utility vector $(10, 10)$ and a non-symmetric Nash equilibrium with utility vector $(20, 15)$. On the one hand, our discussion in Section 5.3.3 told us that the players should choose the Pareto-optimal one, but on the other hand, we have just discussed in this section that the players should choose the symmetric one.

We argue that in this situation the players would typically choose the symmetric Nash equilibrium. To see this, note that because the game is symmetric, we know that there must also exist a third Nash equilibrium, with utility vector $(15, 20)$. This means that whenever player 1 reasons that he should choose the equilibrium with outcome $(20, 15)$, by the symmetry of the game, player 2 would reason that she should choose the third equilibrium, with outcome $(15, 20)$. Therefore, just as in Section 5.3.1 they would end up playing an entirely different strategy profile that typically wouldn't be a Nash equilibrium.[1] So, in the end, any Nash equilibrium they could actually end up playing, would have to be a symmetric one.

This, however, still does not solve the equilibrium selection *completely*, even for symmetric games, because it may still happen that some symmetric game has multiple symmetric equilibria with exactly the same utility vector. If that set happens to be semi-factorizable, then we can just pick any random safe strategy. If it is not semi-factorizable, then we have to discard that entire set and pick the next best option, etcetera. This solution to the equilibrium selection problem for symmetric games is displayed in Algorithm 9.

5.3.5 The Assumption of Role-Equifrequency

We will now discuss our last solution to the ESP, which applies not only to symmetric games, but to normal-form games in general. However, instead we do need to make an alternative assumption, called *the Assumption of Role-Equifrequency* (AoRE) which we will explain below. In a nutshell, this solution says that whenever the AoRE holds, one should pick the Nash equilibrium that maximizes the sum of the utilities of the two players [16].

The main idea behind this solution is that in most realistic situations, instead of trying to find an optimal *strategy* for a single game, one actually

[1]We're using the word 'typically' here, because there may exist games in which the resulting strategy profile would actually still be a Nash equilibrium.

Algorithm 9 Algorithm that chooses the optimal strategy for either of the two players of any symmetric 2-player game G.

Input:
G ▷ The game to play (must be a symmetric game).

▷ Determine the set of all Nash equilibria NE of G:
$NE \leftarrow getNashEquilibria(G)$

▷ Determine the set of *symmetric* Nash equilibria SNE:
$SNE \leftarrow \{(m_1, m_2) \in NE \mid m_1 = m_2\}$

while $SNE \neq \emptyset$ **do**
▷ From this, extract the subset of symmetric Nash equilibria for which the utility of the players is maximal.
It doesn't matter if we use u_1 or u_2 for this because for any symmetric game and any mixed strategy m we have $u_1(m, m) = u_2(m, m)$ anyway.
  $MAX \leftarrow \arg\max_m \{u_1(m, m) \mid (m, m) \in SNE\}$

▷ If this set is semi-factorizable (which includes the case that it contains only one element) then we can determine a safe set and pick any random strategy from it.
  if MAX is semi-factorizable **then**
    $S \leftarrow$ getSafeSet(MAX)
    $m \leftarrow$ getRandomElement(S)
    return m
  end if

▷ If not, then we have to discard this set of equilibria, and keep looking for the next best set.
  $SNE \leftarrow SNE \setminus MAX$
end while

▷ If this approach does not yield any solution, then return the empty set.
return $\emptyset$

tries to implement an optimal *algorithm* to select a strategy, that can be applied *multiple times* to one or more different games. After all, you are not going to delete the algorithm directly after playing one game and then implement an entirely new algorithm for the next game (especially if that next game is identical to the previous one).

This means we are essentially solving a kind of 'meta-game' where the possible actions are the possible algorithms that we can implement, and the utility function we are trying to optimize is the agent's expected utility, averaged over some set of 2-player games $\mathcal{G}$ that we expect our algorithm is going to be playing. Furthermore, if we assume the that our agent is going to play each of the two 'roles' of each game equally often (as 'row-player' and as 'column-player'), then this meta-game is symmetric (even if the games in $\mathcal{G}$ themselves are not symmetric), so we can solve it using the approach of Section 5.3.4.

In order to formalize this precisely, we first need to make a clear distinction between two concepts that have until now mostly treated as equal, namely the concept of a '*player*' and the concept of an '*agent*'.

In our terminology, an 'agent' is an entity, such as a computer program, an application, a robot, or even a human being, that is capable of playing a game. For example, if we are talking about the game of chess, then examples of agents are Deep Blue (the first chess program to ever beat the human world champion of chess), Stockfish (one of the strongest chess engines in the world, at the time of writing), or Magnus Carlsen (the highest ranking human chess player in the world, at the time of writing). On the other hand, when we use the term 'player' we are referring to the *role* that the agent is playing in the game. For example, in the game of chess there are exactly two 'roles' or 'players', namely, *black* and *white*.

Many text books and papers on game theory do not make such a clear distinction between 'players' and 'agents', because they typically just study a single instance of a single game, so there are only two agents present in that context and each agent plays exactly one of the two roles. In those cases there is a one-to-one correspondence between the two agents and the two players, and one can use the term *'agent 1'* to refer to *'the agent that plays the role of player 1'*.

However, in our current context the distinction between players and agents is crucial because we will assume that the same agent will be playing the same game multiple times, sometimes in the role of 'player 1' and sometimes in the role of 'player 2'. To avoid confusion, in this section we will refer to our own agent as ag_A and to its opponent as ag_B. So, this means that in some games agent ag_A will play the role of player 1 (a.k.a. the 'row player') while ag_B plays the role of player 2 (a.k.a. the 'column player'), and in other games (or other instances of the same game) the roles will be reversed and ag_A will play the role of player 2 while ag_B plays the role of player 1.

Definition 5.3.7. *Let $\mathcal{G}$ be a set of 2-player normal-form games. Then, a **role-frequency function** $\mathcal{F}_A : \mathcal{G} \times \{1,2\} \to \mathbb{R}^+$ is a function that assigns to each game $G \in \mathcal{G}$ and each player index $i \in \{1,2\}$ a non-negative real number $\mathcal{F}_A(G,i)$.*

The number $\mathcal{F}_A(G,i)$ represents the relative frequency with which our agent ag_A is going to be (or is expected to be) playing game G in the role of player i. For example, if G is the 'Battle of the Sexes' game (Sec. 5.2.5) and if $\mathcal{F}_A(G,1) = 2 \cdot \mathcal{F}_A(G,2)$, then this means that we expect our agent ag_A to be playing this game in the role of the 'husband' twice as often than as in the role of the 'wife'.

There are several ways to interpret $\mathcal{F}_A$ in a more precise manner. For example $\mathcal{F}_A(G,i)$ could literally be the number of times the agent is going to play the game G as player i, or it can be just the *probability* that it will play the game G as player i. The precise interpretation does not matter, however, for the rest of this section.

For any game G, let $m_A(G,i)$ denote the mixed strategy selected by our agent ag_A when playing game G in the role of player i, and similarly, let $m_B(G,i)$ denote the mixed strategy selected by agent ag_B when playing game G in the role of player i. Then, for any set of 2-player normal-form games $\mathcal{G}$ and any role-frequency function $\mathcal{F}_A$ for this set, we can define 'meta-utility' functions $\mathcal{U}_A$ and $\mathcal{U}_B$ as follows:

$$\begin{aligned} \mathcal{U}_A(ag_A, ag_B) &= \sum_{G\in\mathcal{G}} \mathcal{F}_A(G,1)\cdot u_1^G(m_A(G,1), m_B(G,2)) \quad + \\ &\qquad \mathcal{F}_A(G,2)\cdot u_2^G(m_B(G,1), m_A(G,2)) \end{aligned}$$

$$\begin{aligned} \mathcal{U}_B(ag_A, ag_B) &= \sum_{G\in\mathcal{G}} \mathcal{F}_A(G,1)\cdot u_2^G(m_A(G,1), m_B(G,2)) \quad + \\ &\qquad \mathcal{F}_A(G,2)\cdot u_1^G(m_B(G,1), m_A(G,2)) \end{aligned}$$

where u_i^G is the utility function of player i in game G.

Our goal is to implement an agent ag_A that maximizes the meta-utility $\mathcal{U}_A$, which, however, depends on the opponent ag_B that aims to maximize $\mathcal{U}_B$. Note that the only difference between these two expressions is that the places of u_1 and u_2 have been switched. In particular, $\mathcal{U}_B$ is still described in terms of the role-frequency function $\mathcal{F}_A$ of agent ag_A.

We can see this model as if our agent is playing a *tournament*, defined by $\mathcal{G}$ and $\mathcal{F}_A$, rather than just a single game, and its final score in the tournament is given by $\mathcal{U}_A$, which is an aggregation of the utilities it achieved in the individual games of the tournament.

It is important to understand that even though we model our agent's opponent as a single agent that we denote as ag_B, our agent doesn't literally have to be playing against the same opponent in every game. It is perfectly possible that in reality our agent is playing against an entirely different opponent in each game. However, that doesn't change the fact that we can simply model this collection of opponents as a single agent ag_B, which just happens to behave differently in each game. For this reason we also assume that each game is played completely independently from any previous or future games. That is, the agents do not have a memory of what happened in previous games, and do not try to influence each others' actions in future games. After all, it wouldn't make sense to do so, because we assume the agents don't know whether they are playing against the same opponents or against different opponents in every game.

Definition 5.3.8. *Let $\mathcal{G}$ be a set of 2-player normal-form games and $\mathcal{F}_A$ a role-frequency function for this set, then we say the **Assumption of Role-equifrequency** (AoRE) holds if and only if for all $G \in \mathcal{G}$ we have $\mathcal{F}_A(G,1) = \mathcal{F}_A(G,2)$*

In other words, the AoRE holds if for each game in $\mathcal{G}$ we expect our agent to play either of the two roles of that game with the same frequency or with the same probability.

The idea is that the tournament defined by $\mathcal{G}$ and $\mathcal{F}_A$ can be seen as a giant normal-form game for which the set of actions consists of the set of all possible agents that we can implement and the utility functions are the functions $\mathcal{U}_A$ and $\mathcal{U}_B$. Furthermore, the AoRE ensures that this game is symmetrical—even if the games G inside $\mathcal{G}$ are not symmetrical—which means we can apply the solution discussed in Section 5.3.4 to determine the optimal Nash equilibrium.

Definition 5.3.9. *Let $\mathcal{G}$ be a set of 2-player games and $\mathcal{F}_A$ a role-frequency function for this set, then we define the corresponding **meta-game** to be a 2-player normal-form game such that:*

- *For each player, its set of actions is the set of all possible algorithms that can take as their input a description of any 2-player normal-form* *game $G \in \mathcal{G}$ and a number $i \in \{1, 2\}$ and that then output a mixed strategy $\mathscr{m}$ for player i in that game.*

- *The utility functions are the functions $\mathcal{U}_A$ and $\mathcal{U}_B$ as defined above.*

At first it may seem like a very complicated problem to even find any Nash equilibrium at all for this meta-game, since there are infinitely many different algorithms that one could implement, so it's a game with infinitely many actions. However, it turns out that one can show that a pair of agents ag_A, ag_B forms a Nash equilibrium of the meta-game, if and only if for every game $G \in \mathcal{G}$ those agents select a Nash equilibrium of G [16]. Furthermore, writing an algorithm that finds a Nash equilibrium for any arbitrary finite 2-player normal-form game G is not overly complicated. For example, the well-known Lemke-Howson algorithm [34] does exactly that.

Now, since the AoRE ensures the meta-game is symmetrical, we are only interested in the *symmetric* equilibria of the meta-game. Specifically, this means we are interested in implementing a *single* agent ag such that, for any game $G \in \mathcal{G}$ the two strategies $\mathscr{m}(G, 1)$ and $\mathscr{m}(G, 2)$ it would select for the two respective roles, together always form a Nash equilibrium of G. Furthermore, we have to make sure that our agent picks these strategies in such a way that the agent ag, when seen as an action of the meta-game, satisfies the solution concept of Section 5.3.4.

Using this model, it can be shown that under the AoRE, for any 2-player normal-form game G the optimal Nash equilibrium is, in general, the one that maximizes the sum of the utilities of the two players. If there is more than one such Nash equilibrium then, as was argued in [16], one should choose, among those equilibria, one that minimizes the absolute utility difference $|u_1(\vec{\mathscr{m}}) - u_2(\vec{\mathscr{m}})|$. If we then still have multiple possible equilibria, then we need to check if that set of equilibria is semi-factorizable. If yes, then we can randomly choose any of our safe strategies. If not, then we have to discard this set and repeat the process. This procedure is displayed in Algorithm 10. Note that this algorithm is a small improvement with respect to the one presented in [16], because it uses the solution discussed in Section 5.3.2 as a refinement.

To get an intuitive idea of why one should choose the Nash equilibrium that maximizes the utility-sum, imagine that $\mathcal{G}$ consists of just one game G, which our agent will be playing just once. Furthermore, suppose the AoRE holds, which means that there is a 50% chance that our agent will be playing this game as player 1, and a 50% chance that it will be playing it as player 2. Then, this means that for any Nash equilibrium $\vec{\mathscr{m}}$ the expected utility for our agent under that equilibrium would be $0.5 \cdot u_1(\vec{\mathscr{m}}) + 0.5 \cdot u_2(\vec{\mathscr{m}})$. So, in order for our agent to play optimally, he should choose the Nash equilibrium that maximizes this quantity.

It is important to note that the games G in $\mathcal{G}$ do not need to be symmetric. However, if any game G in $\mathcal{G}$ does happen to be symmetric, then for that game, Algorithm 10 is equivalent to Algorithm 9. So there is no conflict between the current solution and the solution from Section 5.3.4.

Now, at first sight, it might look as if something is wrong with this solution, because it seems to disobey the principle of *Invariance under Linear Transformations* (see Def. 2.2.6). For example, suppose we have a game with two Nash equilibria (m_1, m_2) and (m'_1, m'_2), and suppose they have the following utility-vectors:

$$\vec{u}(m_1, m_2) = (10\ ,\ 20)$$

$$\vec{u}(m'_1, m'_2) = (19\ ,\ 10)$$

Then we see that (m_1, m_2) has a utility-sum of $10 + 20 = 30$, which is greater than the utility sum of (m'_1, m'_2), which is $19 + 10 = 29$. So, our agent should choose (m_1, m_2).

However, if we multiply the utility values of player 1 by a factor of 2, then we get:

$$\vec{u}(m_1, m_2) = (20\ ,\ 10)$$

$$\vec{u}(m'_1, m'_2) = (38\ ,\ 20)$$

Now suddenly, (m'_1, m'_2) has the greater utility-sum ($38 + 20 = 58$ vs. $20 + 10 = 30$).

The solution to this paradox lies in the important distinction between *agents* and *players* that we mentioned before. The point is, that the principle of Invariance under Linear Transformations applies to *agents*, while in this example we have erroneously applied it to a *player*. What we mean is that each *agent* is allowed to freely apply a linear transformation to his utilities. However, since we are assuming the AoRE, each agent is going to be playing in both roles of the game and therefore he will have to apply the same linear transformation to *both* utility functions. Importantly, this does not impede the other agent to apply a totally different linear transformation to the utility functions. Let us illustrate this with another example.

Suppose Alice and Bob are going to play a very simple game against each other, in which they can win a monetary prize in euros. For example, suppose this game has the following pay-off matrix:

Algorithm 10 Algorithm that chooses the optimal strategy for player i for any 2-player normal-form game G (not necessarily symmetrical). This is based on the assumption that for any game G this algorithm will be called equally often with $i = 1$ as with $i = 2$.

```
Input:
G        ▷ The game to play.
i        ▷ The role our agent will be playing in this game
           (i.e. 1 if it is the row-player and 2 if it is the column-player.)

▷ Determine the set of all Nash equilibria NE of G.
NE ← getNashEquilibria(G)

while NE ≠ ∅ do
  ▷ Among the candidate Nash equilibria, find the ones that
    maximize the sum of the two players' utility values:
    MaxSum ← arg max_m⃗ {u_1(m) + u_2(m) | m⃗ ∈ NE}

  ▷ Among those Nash equilibria, find the ones that minimizes the
    absolute difference between the utility values of the two players:
    MinAbsDiff ← arg min_m⃗ {|u_1(m) − u_2(m)| | m⃗ ∈ MaxSum}

  ▷ If this set is semi-factorizable (which includes the case that it contains
  only one element) and all its elements have the same utility vector, then
  pick a random safe strategy and return it.
    if MinAbsDiff is semi-factorizable and all equilibria in MinAbsDiff
  have the same utility vector then
        S_i ← getSafeSet(MinAbsDiff, i)
        m ← getRandomElement(S_i)
        return m
    else
        NE ← NE \ MinAbsDiff
    end if
end while
▷ If this did not yield a solution, return the empty set.
return ∅
```

	L	R
T	(€ 40 , € 30)	(€ 0 , € 0)
B	(€ 0 , € 0)	(€ 10 , € 20)

Furthermore, suppose they are going to play this game twice. In the first game Alice will be the row-player and Bob will be the column player, and in the second game their roles are swapped, so the AoRE holds. Note that this game has two Nash equilibria: (T, L) and (B, R), but the first one is clearly the better one, for both players.

Now, suppose that Alice is American, so she would convert her prize money to dollars. Let's say that 1 euro equals 1.10 dollar. Then, from her point of view the payoff matrix would look like:

	L	R
T	($ 44 , $ 33)	($ 0 , $ 0)
B	($ 0 , $ 0)	($ 11 , $ 22)

In other words, she has applied a linear transformation of the form $u' = 1.1 \cdot u$, but since she is going to play the game in both roles, she applies it to both utility functions. Of course, it is clear that this does not make any difference to the question which equilibrium has the highest utility-sum. The equilibrium (T, L) is still the best for both players.

Furthermore, let us suppose that Bob is British, so he will convert his prize money to British pounds, and let's say that 1 euro equals 0.90 pound. Then, from his point of view the payoff matrix would look like:

	L	R
T	(£ 36 , £ 27)	(£ 0 , £ 0)
B	(£ 0 , £ 0)	(£ 9 , £ 18)

Again, this does not make any difference to the question which equilibrium is better.

We see that the two *agents* are each still perfectly allowed to apply a different linear transformation, and that this does not affect the outcome. Therefore, the solution concept discussed in this section is perfectly compatible with the principle of Invariance under Linear Transformations.

Observation. *Under the AoRE, the solution that maximizes the sum of the players' utilities still obeys the principle of Invariance under Linear Transformations (as long as the transformations are applied to agents, rather than players).*

As a final remark, we should stress that the solution that we discussed in this section can be applied even if we don't know exactly which set of games $\mathcal{G}$ our agent is going to play. Moreover, we also don't even need to know the role-frequency function $\mathcal{F}_A$. *The only thing we really need to know, is that it is reasonable to assume that our agent is going to play each role of each game (approximately) equally often.* For example, if we are implementing a chess-bot, then it seems perfectly reasonable to assume that this bot will play as 'black' approximately equally often as as 'white'.

5.4 Turn-taking Games

As explained above, a normal-form game is a game in which each player makes just one move, and then the game is over. However, most games we play in real life are not over after just one action. Typical games like chess or poker involve multiple rounds. Such games are called *extensive-form games*. To keep things simple we here only focus on one specific type of extensive-form game in which in each turn only one player makes a move. Such games are called *turn-taking games*. Again, games like chess and poker fall into this category. On the other hand, the game of *Diplomacy* does not fall into this category because in that game in each round the players choose their moves simultaneously.

5.4.1 Tuples

Before we can formally define the notion of a turn-taking game, we first need to introduce some other mathematical concepts.

First, for any set X and any integer n, let X^n denote the n-fold Cartesian product of X with itself. That is, $X^1 := X, \quad X^2 := X \times X, \quad X^3 := X \times X \times X$, etcetera.

Then, let $X^{\mathbb{N}}$ denote the set of all finite **tuples** over X. That is:

$$X^{\mathbb{N}} \;:=\; \bigcup_{n \in \mathbb{N}} X^n \;=\; X^0 \cup X^1 \cup X^2 \cup X^3 \cup \dots$$

In particular, note that $X^{\mathbb{N}}$ also includes X^0, which is just the singleton set containing only the empty tuple (). In the rest of this book we will use the symbol ε to denote the empty tuple.

For example, if $X = \{a, b, c\}$, then some examples of tuples over X are (b), (a, a), (a, b, c), and (b, c, b, a, a, b, a). Note that tuples can have arbitrary length (as long as they are *finite*), that a tuple may contain the same element multiple times, and that the elements may appear in any arbitrary order. Also note that the order of the elements matters. That is, (a, b, c) is considered a different tuple than (c, b, a).

We use the symbol $\circ$ to denote the **concatenation** of two tuples. For example $(a, b, c) \circ (d, e) = (a, b, c, d, e)$.

Definition 5.4.1. *For any tuple $x \in X^{\mathbb{N}}$ its **length** n, denoted $|x| = n$, is defined as the integer n for which $x \in X^n$.*

For example, the tuple (a, b, c) has length 3.

For any tuple $x \in X^{\mathbb{N}}$ of length n and any integer m with $m \leq n$, we will use the notation $x[m]$ to denote the m-th element of x. For example, if $x = (a, b, c)$ then $x[1] = a$, $x[2] = b$ and $x[3] = c$.

Definition 5.4.2. *Let $x, y \in X^{\mathbb{N}}$ be two tuples with $|x| < |y|$. Then we say that x is a **prefix** of y if there exists some tuple z such that $x \circ z = y$.*

In other words, if x is a tuple of length n (i.e. $|x| = n$), and x is a prefix of y, then that means that x consists of exactly the first n elements of y. For example, the tuple $x = (a, b, c)$ is a prefix of the tuple $y = (a, b, c, d, e)$, because we have $(a, b, c) \circ (d, e) = (a, b, c, d, e)$. In particular, note that the empty tuple is a prefix of every tuple in $X^{\mathbb{N}}$.

Definition 5.4.3. *Let Y be a set of tuples over some set X. That is $Y \subseteq X^{\mathbb{N}}$. Then we say that Y is **prefix closed**, if for any $y \in Y$ and any prefix y' of y we also have $y' \in Y$.*

For example, the set $Y = \{\varepsilon, (a), (b), (a, b), (a, b, c, d)\}$ is *not* prefix closed, because the tuple (a, b, c) is a prefix of (a, b, c, d) but (a, b, c) is not contained in Y, while (a, b, c, d) *is* contained in Y.

On the other hand, the set $Y' = \{\varepsilon, (a), (b), (a, b), (a, b, c), (a, b, c, d)\}$ *is* prefix closed. To verify this, we just need to check for any tuple $y \in Y'$, except the empty tuple, that if we remove the last element of y, then the resulting tuple y' is also contained in Y.

Definition 5.4.4. *Let Y be a set of tuples over some set X. That is $Y \subseteq X^{\mathbb{N}}$. We say a tuple $y \in Y$ is **non-terminal** in Y if there exists another tuple $y' \in Y$ such that y is a prefix of y'. On the other hand, if there is no such tuple y' then we say that y is **terminal**. The set of all terminal tuples in Y is denoted as Y^T.*

For example, again let $Y = \{\varepsilon, (a), (b), (a, b), (a, b, c, d)\}$. The tuple (a) is non-terminal in Y, because it is a prefix of (a, b). Similarly, (a, b) is

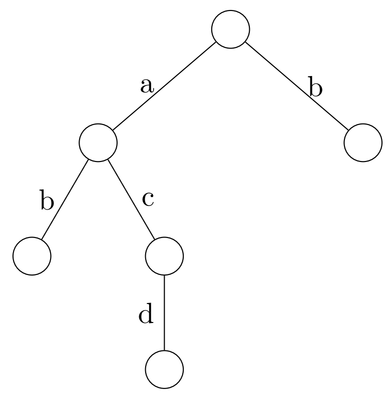


Figure 5.1: Example of a tree corresponding to a set of tuples $\{\varepsilon, (a), (b), (a, b), (a, c), (a, c, d)\}$. The root corresponds to the empty tuple ε. The two children of the root correspond to the tuples (a) and (b) respectively. The two 'grand children' of the root correspond to the tuples (a, b) and (a, c) respectively. Finally, the last node corresponds to the tuple (a, c, d).

non-terminal because it is a prefix of (a, b, c, d). On the other hand, (b) is terminal, because there is no other tuple in Y that starts with b. Similarly, (a, b, c, d) is also terminal. The empty tuple $\varepsilon = ()$ is of course always non-terminal, except in the case that it is the only tuple in the entire set.

5.4.2 Tree Diagrams

Any finite set of tuples that is prefix closed can be visually displayed as a tree. The easiest way to see this, is to simply look at Figure 5.1, which displays the tree corresponding to the set of tuples $\{\varepsilon, (a), (b), (a, b), (a, c), (a, c, d)\}$.

Formally, a **tree** is a connected acyclic graph, for which one of the nodes is marked as the **root**. The **depth** of a node is the length of the unique path from the root to that node. For any node ν with depth d, its **children** are those neigbors of ν that have depth $d + 1$. Furthermore, its **parent** is its unique neighbor with depth $d - 1$. A **leaf node** is a node that does not have any children.

Formally, for any set X and any prefix closed set of tuples $Y \subset X^{\mathbb{N}}$, the tree-diagram of Y is a tree such that the following holds:

- There is a one-to-one correspondence between Y and the set of nodes of the tree. We will use the notation $y(\nu)$ to denote the tuple corresponding to node ν.

- The root node corresponds to the empty tuple.
- For any pair of nodes ν and ν' such that ν' is a child of ν, there exists an element of X such that $y(\nu')$ can be obtained from $y(\nu)$ by concatenating it with a single element of X:

$$\exists x \in X \quad : \quad y(\nu') = y(\nu) \circ (x)$$

and the edge (ν, ν') is labeled with x.

5.4.3 Definition of a Turn-taking Game

We are now ready to define the notion of a turn-taking game. However, before we give the formal definition, let us first explain it informally, using the game of Tic-Tac-Toe as an example.

The game of Tic-Tac-Toe is a turn-taking game, which means that in each turn one of the players chooses an action to play. So, in order to define the rules of this game, we first need to specify the set of actions that the players can choose from. In Tic-Tac-Toe choosing an action consists in marking a symbol **X** or **O** in a 3×3 grid. We can formalize such an action as a tuple (r, c, s) where $r \in \{1, 2, 3\}$ is the row in which the symbol is marked, $c \in \{1, 2, 3\}$ is the column, and $s \in \{\mathbf{X}, \mathbf{O}\}$ is the symbol itself. For example, when a player puts the symbol **X** in the center of the grid, this action is denoted by $(2, 2, \mathbf{X})$. So, we have a set of *actions* $A = \{1, 2, 3\} \times \{1, 2, 3\} \times \{\mathbf{X}, \mathbf{O}\}$.

Every time a player makes a move, the state of the game changes. Therefore, any state of the game can be identified with the sequence of of actions that have already been played. In other words, the set of all possible states of the game is a subset of $A^{\mathbb{N}}$.

For example, suppose that in the first turn player 1 plays $(2, 2, \mathbf{X})$. Then, in the second turn player 2 plays $(1, 1, \mathbf{O})$, and then, in the third turn player 1 plays $(1, 2, \mathbf{X})$. At that point, the state of the game is the tuple:

$$\Big((2, 2, \mathbf{X}), (1, 1, \mathbf{O}), (1, 2, \mathbf{X})\Big)$$

Of course, not every action in A is legal in every state of the game. For example, after the first player has played $(2, 2, \mathbf{X})$, the second player is not allowed to play $(2, 2, \mathbf{O})$, because the cell $(2, 2)$ is already filled. Therefore, the set of *legal* sequences of actions is only a *subset* of $A^{\mathbb{N}}$. We will refer to such legal sequences as *action histories* and we will denote the set of all such histories by $\mathcal{H}$.

In particular, note that $\mathcal{H}$ must be prefix closed. After all, the state $\big((2,2,\mathbf{X}),(1,1,\mathbf{O}),(1,2,\mathbf{X})\big)$ can only be reached if the previous state was $\big((2,2,\mathbf{X}),(1,1,\mathbf{O})\big)$. In other words, if $\big((2,2,\mathbf{X}),(1,1,\mathbf{O}),(1,2,\mathbf{X})\big)$ is legal, then $\big((2,2,\mathbf{X}),(1,1,\mathbf{O})\big)$ must also be legal.

Furthermore, to fully define the game of Tic-Tac-Toe, we have to specify the goals of the respective players. This can be formalized by defining a utility function for each player. For example, we can assign a value of 2 to the winner, a value of 0 to the loser, and in case of a draw we can assign a utility value of 1 to each of the players. Of course, the notion of a winner or loser is only defined at the end of the game. Therefore, the utility functions are defined over the set of all *terminal* histories.

Finally, we have to specify which player can can choose an action when. We call the player who's turn it is, the *active player*. Formally, we need a function that maps each non-terminal history to the index of the active player:

$$pl \quad : \quad \mathcal{H} \setminus \mathcal{H}^T \to \{1, 2, \ldots, n\}$$

where n is the number of players.

In Tic-Tac-Toe, just as in most other turn-taking games, the active player simply alternates each turn. So, in each odd turn, player 1 is the active player and in each even turn player 2 is the active player. That is: $pl(h) = |h| \pmod 2 + 1$

Whenever the current state of the game is a non-terminal history h and it's the turn of player i, then this player can choose any action $a \in A$ such that the concatenation $h \circ (a)$ is legal (i.e. $h \circ (a) \in \mathcal{H}$). Such an action certainly exists, because we assumed h was non-terminal. On the other hand, if h is terminal, then, by definition, the game is over and the utility function determines the outcome of the game.

In summary, a turn-taking game is formally defined as follows.

Definition 5.4.5. *A **turn-taking** game for n players, consists of the following components:*

- *A set A, which we call the set of **actions**.*
- *A set $\mathcal{H}$, called the set of all legal **action histories**, which is a subset of the set of all finite tuples over A (i.e. $\mathcal{H} \subseteq A^{\mathbb{N}}$), such that $\mathcal{H}$ is prefix closed.*
- *A function pl called the **active player map**, that maps each non-terminal history $h \in \mathcal{H} \setminus \mathcal{H}^T$ to the index of the player whose turn it is:*

$$pl \quad : \quad \mathcal{H} \setminus \mathcal{H}^T \to \{1, 2, \ldots, n\}$$

- *For each $i \in \{1, 2, \ldots, n\}$ a **utility function** u_i that assigns a utility value for player i to each terminal history in $\mathcal{H}$:*

$$u_i \quad : \quad \mathcal{H}^T \to \mathbb{R}$$

5.4.4 Game Trees

Since a turn-taking game is essentially a set of tuples that is prefix closed, together with utility functions and an active player function, we can visually display it as a tree. See for example the game displayed in Figure 5.2.

Note that in this case the nodes corresponding to the non-terminal histories are labeled with the index of the active player, and that the nodes corresponding to the terminal histories are labeled with the utility values of the respective players. We will call such diagrams **game trees**.

In the game of Figure 5.2, each player has just one turn. In the first turn, player 1 can choose between actions a and b. If player 1 chooses a then in next player 2 can choose between actions c and d. Otherwise, if player 1 chooses to play b, then next player 2 can choose between actions e and f.

If the two players choose a and c respectively, then each of them will receive a utility of 0. On the other hand, if they choose actions b and f respectively, then player 1 will receive a utility of 5, while player 2 will receive a utility of 30.

5.4.5 Strategies

We will now define the notion of a 'strategy' for a turn-taking game.

Let $\mathcal{H}_i$ denote the set of all non-terminal histories after which player i is the active player. That is:

$$\mathcal{H}_i \quad := \quad \{h \in \mathcal{H} \setminus \mathcal{H}^T \mid pl(h) = i\}$$

Furthermore, for any non-terminal history h, let A_h denote the set of *legal actions* that the active player is allowed to choose after history h. More formally, it is the set of actions that yield a legal history when concatenated with h.

$$A_h \quad := \quad \{a \in A \mid h \circ (a) \in \mathcal{H}\}$$

Definition 5.4.6. *For any turn-taking game, a **strategy** σ for player i is a map that assigns to each history h after which ag_i is the active player, a legal action for ag_i.*

$$\sigma \quad : \quad \mathcal{H}_i \to A \quad \textit{such that} \quad \forall h \in \mathcal{H}_i : \quad \sigma(h) \in A_h$$

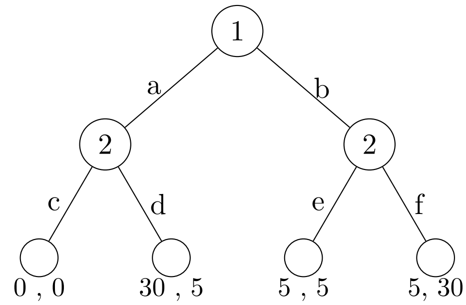


Figure 5.2: A game tree that visualizes a very simple 2-player turn-taking game that only lasts for two rounds. Each edge is labeled with an action from the game, and therefore each node corresponds to the history consisting of all actions along the path from the root to that node. Furthermore, each non-terminal node is labeled with the index of the active player after that history and each terminal node is labeled with the utility values of the two respective players.

In line with our earlier definitions, we refer to a tuple of strategies $(\sigma_1, \sigma_2, \ldots, \sigma_n)$, one for each agent, as a **strategy profile**.

In the example game of Figure 5.2, there is only one history after which player 1 is the active player, namely the empty history (i.e. at the beginning of the game). Therefore his strategy is entirely determined by the action he chooses at the start of the game. Since he can choose between two actions, a and b, he also only has a total of two strategies, defined by $\sigma(\varepsilon) = a$ and $\sigma(\varepsilon) = b$, respectively.

On the other hand, for player 2 there are two possible histories after which she needs to choose an action. Namely after the history (a) and after the history (b). So, to choose a strategy, she has to make two choices: what to do after history (a) and what to do after history (b). For each of these two histories she has two actions to choose from, so she has $2^2 = 4$ possible strategies:

1. $\sigma(a) = c$, $\sigma(b) = e$
2. $\sigma(a) = c$, $\sigma(b) = f$
3. $\sigma(a) = d$, $\sigma(b) = e$
4. $\sigma(a) = d$, $\sigma(b) = f$

In general, if there are m histories after which it is your turn, and after each of these histories you have exactly n possible actions, then you have

n^m possible strategies (although in general the number of legal actions may be different after each history).

Note that once every player has chosen a strategy, and each player follows his chosen strategy throughout the game, then the evolution of the game is completely fixed, so the terminal state in which the game will end will be fixed.

For example, in the case of Tic-Tac-Toe, in the first round player 1 will play the action given by $\sigma_1(\varepsilon)$. Let's say he chooses the center square, so we have: $\sigma_1(\varepsilon) = (2, 2, \mathbf{X})$. Next, player 2 plays the action given by $\sigma_2\Big((2, 2, \mathbf{X})\Big)$. Let's say that this is $\sigma_2\Big((2, 2, \mathbf{X})\Big) = (1, 1, \mathbf{O})$. This continues until a terminal history is reached.

1. Player 1 chooses action $\sigma_1(\varepsilon) = (2, 2, \mathbf{X})$.
2. Player 2 chooses action $\sigma_2((2, 2, \mathbf{X})) = (1, 1, \mathbf{O})$.
3. Player 1 chooses action $\sigma_1((2, 2, \mathbf{X}), (1, 1, \mathbf{O})) = (1, 2, \mathbf{X})$.
4. Player 2 chooses action $\sigma_2((2, 2, \mathbf{X}), (1, 1, \mathbf{O}), (1, 2, \mathbf{X})) = (3, 2, \mathbf{O})$.
5. etcetera...

Let $\vec{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ be a strategy profile. Then we use the notation $h_{\vec{\sigma}}$ to denote the unique terminal history generated by this strategy profile. Formally, it is defined as the unique terminal history that satisfies:

$$h_{\vec{\sigma}} \quad := \quad \Big(a_1 \ , \ a_2 \ , \ a_3 \ , \ \ldots \ , \ a_k\Big)$$

where:

$$a_j \quad := \quad \begin{cases} \sigma_1(\epsilon) & \text{if } j = 1 \\ \sigma_i(a_1 \ , \ a_2 \ , \ \ldots \ , \ a_{j-1}) & \text{if } j > 1 \end{cases}$$

$$\text{with } i \quad := \quad pl(a_1, a_2, \ldots, a_{j-1})$$

Furthermore, we may use the notation $u_i(\sigma_1, \sigma_2, \ldots, \sigma_n)$ or $u_i(\vec{\sigma})$ as a shorthand for $u_i(h_{\vec{\sigma}})$.

5.4.6 Non-credible Threats

Just like in our section about normal-form games, the main question we aim to answer is how to find the optimal strategy for each player. We will first explore a naive potential solution to this problem, which will turn out to be wrong.

The idea behind this wrong solution is as follows: for any given turn-taking game Γ we can consider the set of all possible strategies for player i, which we will denote by $\mathcal{S}_i$. As mentioned before, if each player chooses a strategy, this will uniquely determine a terminal history $h_{\vec{\sigma}}$, and therefore it will uniquely determine a tuple of utility values $(u_1(h_{\vec{\sigma}}), u_2(h_{\vec{\sigma}}), \ldots, u_n(h_{\vec{\sigma}}))$, which we may denote as $(u_1(\vec{\sigma}), u_2(\vec{\sigma}), \ldots, u_n(\vec{\sigma}))$. We can then define the notion of a pure Nash equilibrium for a turn-taking game in an analagous manner as for normal-form games.

Definition 5.4.7. *Let Γ denote a two-player turn-taking game and let σ_1 and σ_2 denote two strategies for player 1 and player 2, respectively. Then, we say that σ_1 is a best response against σ_2 if:*

$$\forall \sigma \in \mathcal{S}_1 : \quad u_1(\sigma, \sigma_2) \leq u_1(\sigma_1, \sigma_2)$$

and similarly, we say that σ_2 is a best response against σ_1 if:

$$\forall \sigma \in \mathcal{S}_2 : \quad u_2(\sigma_1, \sigma) \leq u_2(\sigma_1, \sigma_2).$$

We say a pair of strategies σ_1, σ_2 is a ***pure Nash equilibrium*** *of the turn-taking game Γ, if σ_1 is a best response against σ_2 and σ_2 is a best response against σ_1.*

Another way to look at this, is to say that each turn-taking game corresponds to a normal-form game. That is, given the n-player turn-taking game Γ we can define a corresponding n-player normal-form game G as follows:

- For each $i \in \{1, 2, \ldots, n\}$ the set of *actions* A_i^G of player i in G is exactly the set of *strategies* $\mathcal{S}_i$ for player i in Γ. That is:

$$A_i^G \quad := \quad \mathcal{S}_i$$

- For each $i \in \{1, 2, \ldots, n\}$ the utility function u_i^G of player i in G is defined as

$$u_i^G(\sigma_1, \sigma_2, \ldots, \sigma_n) \quad := \quad u_i(h_{(\sigma_1, \sigma_2, \ldots, \sigma_n)})$$

  where the utility functions u_i on the right-hand side are the utility functions of Γ.

It should now be clear that the pure Nash equilibria of the turn-taking game Γ conincide exactly with the corresponding pure Nash equilibria of the normal-form game G.

In principle, we could now also define a *mixed* Nash equilibrum of Γ to be exactly mixed Nash equilibrum of G. However, there is no reason to

	ce	*cf*	*de*	*df*
a	(0 , 0)	(0 , 0)	(30 , 5)	(30 , 5)
b	(5 , 5)	(5 , 30)	(5 , 5)	(5 , 30)

Table 5.3: The pay-off matrix corresponding tot the game of Figure 5.2

consider such mixed equilibria, because, as we recall from Section 5.2.6, the purpose of a mixed strategy is to be unpredictable to your opponent. But that doesn't work in a turn-taking game, because in each turn, the active player already knows what action the opponent has chosen in the previous turn, anyway.

Now that we have re-interpreted the turn-taking game Γ as a normal-form game G, one might think that the optimal solution for each player is to choose a strategy σ_i such that the strategy profile $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ forms a Nash equilibrium of G. However, we will show that this solution is not satisfactory. This is demonstrated with the game displayed in Figure 5.2. As explained above, in this game player 1 has two possible strategies, corresponding to the actions a and b, and player 2 has four possible strategies, which we will here denote as *ce*, *cf*, *de* and *df* respectively. So, we can model this game as a 2×4 normal-form game, of which the payoff matrix is displayed in Table 5.3.

Note that this game has three pure Nash equilibria:

1. Actions: $(a\ ,\ de)$ utilities: $(30\ ,\ 5)$
2. Actions: $(a\ ,\ df)$ utilities: $(30\ ,\ 5)$
3. Actions: $(b\ ,\ cf)$ utilities: $(5\ ,\ 30)$

We will argue that the third Nash equilibrium is, in a certain sense, unrealistic. To see that it is a Nash equilibrium indeed, first note that in this strategy profile player 2 receives the maximum utility she can possibly achieve, so indeed she cannot benefit from any deviation. Furthermore, note that if player 1 were to deviate to action a, then the resulting action profile would be (a, cf), which means that player 1 would play action a, followed by player 2 playing action c. The resulting utility vector would then be $(0\ ,\ 0)$, so player 1 does not benefit from any deviation either.

However, this all depends on the assumption that player 1 indeed makes a *unilateral* deviation. The problem, is that if player 1 would indeed switch to action a, then it would be highly unlikely that player 2 would still stick with strategy *cf*. After all, playing c after a is essentially a form of 'suicide' by player 2. In principle, player 2 could play d and obtain 5 points, but instead she plays c yielding 0 points to herself.

This problem occurs because, as explained in Section 5.2.4, the definition of a Nash equilibrium only takes *unilateral* deviations into consideration. This makes sense if the game was truly a normal-form game in which each player has to fully commit to its own strategy without observing the actions of the opponent. But in this case we are playing a turn-taking game. This means that if player 1 deviates to strategy a, then player 2 will *observe* that player 1 plays action a, which means that player 2 now has the possibility to also change her strategy, based on that observation. Indeed, if she is rational, she would also deviate and choose action d instead of action c. Therefore, in turn-taking games it is not enough to only consider unilateral deviations, and thus the concept of a Nash equilibrium is too weak.

We say that the third Nash-equilbrium in our example is based on a so-called *non-credible threat*. It is as if player 2 is saying to player 1: "*If you play action a then I will play action c and you will end up with 0 utility. Therefore, you'd better play action b*". This threat is not credible, because playing action c does not only hurt player 1, but also player 2 herself. Therefore, player 1 could simply ignore this threat and play action a anyway, knowing that player 2 is rational and therefore would not follow through with her threat but play action d instead.

From this, we conclude that the concept of a Nash equilibrium is not satisfactory for turn-taking games, because some Nash equilibria may be based on non-credible threats. Therefore, we need a refined solution concept that only considers those Nash equilibria that do not involve such non-credible threats.

5.4.7 Subgame Perfect Equilibria

We will now discuss an alternative solution concept, known as the the '*subgame perfect equilibrium*', which is widely regarded as the 'correct' solution concept for turn-taking games.

To explain this concept, we first need to define the notion of a subgame. Informally, for any turn-taking game Γ and any given non-terminal history h of that game, the subgame of Γ at h is exactly the same as Γ, except that it doesn't start from the same initial state as Γ, but rather it starts from some non-empty history h of Γ. In other words, it is as if we start somewhere in the middle of the game.

For example, let Γ be the game of Tic-Tac-Toe, and let h be the history given by:

$$h = \Big((2, 2, \mathbf{X}), (1, 1, \mathbf{O}), (1, 2, \mathbf{X})\Big)$$

Then the subgame of Γ at h follows the same rules as ordinary Tic-Tac-Toe, except that the game does not start from an empty grid, but rather starts from the state:

O	**X**	
	X	

This can be formalized as follows.

Definition 5.4.8. *Let Γ be a turn-taking game and let $\mathcal{H}$ denote the set of histories of that game. Furthermore, let $h \in \mathcal{H} \setminus \mathcal{H}^T$ be any non-terminal history of G. Then the* ***subgame*** *of Γ at h is a turn-taking game, denoted Γ_h, such that its histories (denoted $\mathcal{H}_h$) are exactly those histories in $\mathcal{H}$ that have h as a prefix.*

$$\mathcal{H}_h = \{h' \in \mathcal{H} \mid h \textit{ is a prefix of } h'\}$$

The active player function and the utility functions of Γ_h are just the same as those of Γ, but restricted to the set $\mathcal{H}_h$.

Note that any strategy for the game Γ can naturally be interpreted as a strategy for the game Γ_h as well, simply by restricting it to the histories $\mathcal{H}_h$ of Γ_h.

Definition 5.4.9. *Let Γ be an n-player turn-taking game and $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ a strategy profile for this game. We say that this strategy profile is a* ***subgame-perfect equilibrium*** *if it is a Nash equilibrium on all subgames of Γ.*

The proof of the following theorem can be found in [42].

Theorem 3. *Every finite turn-taking game has a subgame perfect equilibrium.*

Let us now try to find the subgame-perfect equilibria of our example game from Figure 5.2. First note that that Definition 5.4.9 implies that every subgame-perfect equilibrium of a turn-taking game Γ is also a Nash equilibrium of Γ. After all, by definition it has to be a Nash equilibrium on *all* subgames of Γ, which includes Γ itself. Since we already know the

Nash equilibria of Γ, namely (a, de), (a, df) and (b, cf), we can restrict our attention to those three strategy profiles.

Next, let us look at the subgame $\Gamma_{(a)}$ defined by the history (a). In this subgame there is only one player, namely player 2, who can choose between actions c and d. Action c will yield a utility of 0 to player 2 and action d will yield her a utility of 5, so she would choose action d. Therefore, the strategy profile (b, cf), is not a Nash equilibrium on the subgame $\Gamma_{(a)}$, since it prescribes that player 2 would choose action c instead of d.

Finally, let us look at the subgame $\Gamma_{(b)}$ defined by the history (b). Again, in this subgame player 2 is the only player, and this time she can choose between actions e and f. Action e will yield a utility of 5 to player 2 and action f will yield her a utility of 30, so she would choose action f. Therefore, the strategy profile (a, de), is not a Nash equilibrium on the subgame $\Gamma_{(b)}$, since it prescribes that player 2 would choose action e instead of f.

In conclusion, we see that the strategy profile (a, df) is the only subgame-perfect equilibrium of our example game, because indeed it forms a Nash equilibrium on all three subgames of Γ (that is, $\Gamma_{(a)}$, $\Gamma_{(b)}$, and Γ itself).

5.4.8 Non-deterministic Turn-taking Games

Games like chess or Tic-Tac-Toe are completely deterministic. However, many other games, such as backgammon or poker involve randomness because players need to throw dice or shuffle cards.

A common way to formally model non-deterministic games is to introduce an extra player to the game, which is often called 'nature'. The idea is that, unlike the other players, nature does not have a utility function and always selects its actions randomly. For example, whenever a 6-sided die is thrown, we say it is nature's turn and that nature will randomly choose an action $a \in \{1, 2, 3, 4, 5, 6\}$.

Definition 5.4.10. *A* ***non-deterministic turn-taking*** *game for n players, consists of the following components:*

- *A set A, which we call the set of* ***actions****.*
- *A set $\mathcal{H}$, called the set of all* ***histories****, which is a subset of the set of all finite tuples over A (i.e. $\mathcal{H} \subseteq A^{\mathbb{N}}$), such that $\mathcal{H}$ is prefix closed.*
- *A function pl called the* ***active player map****, that maps each non-terminal history $h \in \mathcal{H} \setminus \mathcal{H}^T$ to the index of the player whose turn it is, or to 0, representing 'nature':*

$$pl \quad : \quad \mathcal{H} \setminus \mathcal{H}^T \to \{0, 1, 2, \ldots, n\}$$

- *For each $i \in \{1, 2, \ldots, n\}$ a **utility function** u_i that assigns a utility value for player i to each terminal history in $\mathcal{H}$:*

$$u_i \quad : \quad \mathcal{H}^T \to \mathbb{R}$$

- *For each history h such that $pl(h) = 0$, a probability distribution P_h over the set A_h of legal actions after h.*

Note that we still refer to this game as an n-player game, even though it technically has $n+1$ players, including nature. This is of course because we don't want to count 'nature' as a real player.

In an n-player non-deterministic turn-taking game, it no longer holds that any n-tuple of strategies $\vec{\sigma}$ yields a unique terminal history, because the terminal history now also depends on the random choices made by nature. Instead, however, each n-tuple of strategies $\vec{\sigma}$ leads to a probability distribution $P(h \mid \vec{\sigma})$ over the set of all terminal histories $h \in \mathcal{H}^T$. This means that, for any player i and any strategy profile $\vec{\sigma}$, we can only calculate an *expected* utility $\overline{u}_i(\vec{\sigma})$:

$$\overline{u}_i(\vec{\sigma}) \quad := \quad \sum_{h \in \mathcal{H}^T} P(h \mid \vec{\sigma}) \cdot u_i(h)$$

In order to define the notion of an 'optimal' strategy, we can now follow the same procedure as for deterministic turn-taking games, except that we need to define everything in terms of the *expected* utility functions. That is, a non-deterministic turn-taking game Γ corresponds to a normal-form game G, where the actions of G are exactly the strategies of Γ and the utility functions of G are exactly the *expected* utility functions of Γ. Then, the pure Nash equilibria of Γ are defined as the pure Nash equilibria of G and a subgame perfect equilibrium of Γ is defined as a strategy profile that forms a Nash equilibrium on every subgame of Γ.

5.4.9 Turn-taking Games with Imperfect Information

Another property that many games satisfy, but that we haven't discussed yet, is the property of *imperfect information.* This means that during the game the players do not have full knowledge of the state of the game, or of the actions played by the other players. Typical examples of such games are card games, such as poker, where each player can only see his own cards but not the cards in the hands of the other players.

To model the notion of a turn-taking game with imperfect information, we assume that whenever a player plays an action, this action is not seen by the other players. Instead, every player receives a signal that may or may not reveal some (limited) information about which action was played. For example, imagine the players are playing a card game, and imagine that player 1 discards one of his cards, say, his ace of spades. So, while player 1 is playing the action $(discard, Ace, \spadesuit)$, the other players will only observe the signal $(discard)$. From this signal, the other players will understand that player 1 discarded a card, but they will not be able to tell *which* card player 1 was discarding.

In order to formalize this, we will assume that the game has a predefined set of possible **observations** (or 'signals') O and that each player has a so-called **observation function** $f_i^{obs} : \mathcal{H} \to O^{\mathbb{N}}$ that maps each legal action history to a sequence of observations for that player.

For example, suppose the current state of some game is given by a history (a_1, a_2, a_3), but player 1 has received the following sequence of observations: $f_1^{obs}(a_1, a_2, a_3) = (o_1, o_2, o_3)$. Then, after player 2 plays action a_4, player 1 will receive some observation o_4, so we have $f_1^{obs}(a_1, a_2, a_3, a_4) = (o_1, o_2, o_3, a_4)$. Typically, f_1^{obs} would be a non-invertible function, so just from the observations (o_1, o_2, o_3, o_4) the player would not be able to deduce the actual actions (a_1, a_2, a_3, a_4) that have been played. In other words, at any point during the game, a player will, in general, not be aware of the history of actions that have so far been played, but instead will only be aware of the sequence of observations he or she has so far received. Also note that each player has its own individual observation function, so each player may receive different observations.

Definition 5.4.11. *Let $\mathcal{H}$ be some set of action histories and O be some set of observations, then an **observation function** $f_i^{obs} : \mathcal{H} \to O^{\mathbb{N}}$ is a function that maps every possible history to a tuple of observations, such that for any pair of histories $h, h' \in \mathcal{H}$ where h is a prefix of h', we also have that $f_i^{obs}(h)$ is a prefix of $f_i^{obs}(h')$.*

We will refer to $f_i^{obs}(h)$ as the **observed history** of agent i and we may sometimes use the notation h_i^o as a shorthand for $f_i^{obs}(h)$.

Note that this definition allows for the possibility that a player sometimes may not receive any observation at all, when another player plays an action. For example, we could have something like: $f_1^{obs}(a_1, a_2, a_3) = f_1^{obs}(a_1, a_2, a_3, a_4)$. This means that when player 2 plays action a_4, player 1 will not even be aware that player 2 played any action at all.

With these definitions we can now formally define the notion of a turn-taking game with imperfect information.

Definition 5.4.12. *A* ***turn-taking game with imperfect information*** *(for n players) is a turn-taking game together with a set of possible* ***observations*** *O and for each player ag_i an* ***observation function*** *$f_i^{obs} : \mathcal{H} \to O^{\mathbb{N}}$. Furthermore, apart from the active-player function, pl, each player ag_i also has its own individual active-player function $pl_i : O^{\mathbb{N}} \to \{1, 2, \ldots, n, ?\}$ which must satisfy:*

$$\forall h \in \mathcal{H}\ \forall i \in \{1, 2, \ldots, n\} : \quad pl_i(f_i^{obs}(h)) = i \quad \textit{if and only if} \quad pl(h) = i$$

The last constraint in this definition ensures that, even though the players do not have full information about the current state of the game, each player is still able to correctly determine whether or not it is his turn to make a move, based only on his own observations. Technically, we should also include similar constraints to ensure the players always have full knowledge of their legal actions and their utility functions. However, we will skip that to avoid overcomplicating things.

Furthermore, note that we have included the symbol '?' in the codomain of the functions pl_i. This symbol represents the case that player i does not know whose turn it is.

Now, a strategy for a turn-taking game with imperfect information can be defined as a function that maps observation histories to actions.

Definition 5.4.13. *Let Γ be a turn-taking game with imperfect information. Furthermore, let O_i denote the set of all possible observed histories after which it is player i's turn:*

$$O_i := \{\vec{o} \in O^{\mathbb{N}} \mid pl_i(\vec{o}) = i\}$$

Then, a ***strategy*** *for player i is a map that assigns to each observed history $\vec{o}$ after which ag_i is the active player, a legal action for ag_i.*

$$\sigma \quad : \quad O_i \to A \quad \textit{such that} \quad \forall h \in \mathcal{H}_i : \quad \sigma(f_i^{obs}(h)) \in A_h$$

This definition implies that a player can only choose his actions based on the observations that he has seen, rather than on the actual actions that have been played. This represents the fact that in general the player doesn't know exactly which actions have been played, and that the 'observations' are indeed the only thing the player observes.

Of course, in most games a player would at least be able to fully observe his *own* actions. This means the observation made by the active player would typically simply be the action itself.

Furthermore, note that a turn-taking game with *perfect* information (i.e. a game such as chess or go where all the players do have a full view of all the players' actions), can be seen as a special case of a game with imperfect information, where each observed history is just the full history itself:

$$\forall h \in \mathcal{H}\ \forall i \in \{1, 2, \ldots, n\} : \quad f_i^{obs}(h) \ = \ h$$

The question how to determine the optimal strategy profile for games with imperfect information is, however, a lot more difficult to answer than for ordinary turn-taking games. We will just comment that the commonly accepted solution concept for such games is known as the *sequential equilibrium*, without going into detail about how it is defined. For more information about this topic we refer to [42].

5.4.10 Turn-taking Games with Incomplete Information

In game theory, the notion of *incomplete* information is similar to, but not exactly the same as, the notion of *imperfect* information.

As we discussed in the previous section, *imperfect* information refers to the lack of knowledge about the *current state* of the game. In other words, at any moment *during* the game the players may not know exactly at which node of the game tree they are currently situated.

On the other hand, a game with **incomplete** information, is a game for which the players do not even have full information about the *structure* of the game itself. For example, the players may not know each others' utility functions or may not know exactly which actions are available to the other players. Note that this lack of information already exists even *before* the game has started.

So, in a game of *complete*, but *imperfect* information, the players initially have full knowledge of the structure of the game, but after the game has started some of the actions played by some of the players may be invisible to other players, yielding uncertainty about the current state. Most card games, such as poker, are examples of such games.

On the other hand, in a game of *incomplete* information, the players already lack knowledge of some aspects of the game, even before the game has started. Of course, it is also possible that a game has both incomplete *and* imperfect information.

In automated negotiation one often assumes that the negotiating agents do not know each others' utility functions. So, under that assumption, automated negotiation is indeed an example of a game with incomplete information.

5.5 Automated Negotiation as a Game

Now that we have discussed the basic principles of game theory, we can finally come back to the topic of automated negotiation, and discuss in what sense it is a game.

The basic idea is simple: each negotiating agent is a player of the game and the actions they can play are exactly the negotiation actions as defined in Definition 2.2.1. However, since each action is followed by a small unpredictable delay due to network latency, it is a non-deterministic game and since this delay itself cannot be observed, it is also a game of imperfect information. Furthermore, in case we assume the agents do not no each others' utility functions, then it is also a game of incomplete information.

So, in this section we will formally define, for any negotiation domain D, a corresponding non-deterministic turn-taking game with imperfect information for 2 players, denoted Γ_D. Note that essentially we are just repeating the definition of a bilateral negotiation under the alternating offers protocol that we already gave in Chapter 2, but this time we are using game-theoretical terminology.

5.5.1 Actions

The actions of the two players in the game Γ_D are exactly the negotiation actions as defined in Def. 2.2.1. We use the notation A_i^D to refer to the set of negotiation actions for player i. That is:

$$A_1^D := \{1\} \times \{\mathfrak{p}, \mathfrak{a}\} \times \Omega \times \mathbb{R}^+$$
$$A_2^D := \{2\} \times \{\mathfrak{p}, \mathfrak{a}\} \times \Omega \times \mathbb{R}^+$$

However, since it is a non-deterministic game, we also need an extra player called 'nature', as explained in Section 5.4.8. Every time after one of the two real players has submitted a negotiation action, it is nature's turn to "choose" a random delay for the message to arrive at the other agent. This delay can be any positive real number, so the set of actions for nature is the set of positive real numbers $\mathbb{R}^+$.

So, in total, the set of actions A of the game Γ_D is:

$$A \quad = \quad A_1^D \cup A_2^D \cup \mathbb{R}^+$$

5.5.2 The Active Player Map

Since we are modeling the alternating offers protocol, the agents' turn to make a proposal will alternate between players 1 and 2. However, since each negotiation action is followed by a random 'delay', every turn in which one of the two players chooses an action has to be followed by a turn for nature to choose the delay. Therefore, the game has the following turn-taking structure:

Player 1, nature, player 2, nature, player 1, nature, player 2, nature, etc...

Formally, we can define this as follows:

$$pl(h) = \begin{cases} 1 & \text{if } |h| (\text{mod } 4) = 0 \\ 2 & \text{if } |h| (\text{mod } 4) = 2 \\ 0 & \text{otherwise (i.e. } |h| \text{ is odd).} \end{cases}$$

where h is any tuple over the set of actions A, i.e. $h \in A^{\mathbb{N}}$.

Note that we here follow the convention that it is always player 1 that starts the negotiation (unlike in some of the previous sections in which we followed the convention that player 1 is 'our' agent).

5.5.3 The Set of Legal Histories

The set of legal histories of Γ_D is exactly the set of negotiation histories as defined by Definitions 2.2.2 and 2.2.3.

We can define it recursively. That is, let h' be any legal history, then we can define the criteria that an action $a \in A$ would need to satisfy in order for the history $h := h' \circ (a)$ to be legal as well. Then, given that the empty history ε is legal, we can construct all other legal histories.

Suppose the current state of the game is given by some history h'. If, in this state, it is player 1's turn (i.e. $pl(h') = 1$), then she can either propose an offer or accept an offer. That is, she can play an action of the form $(1, \mathfrak{p}, \omega, t)$ or $(1, \mathfrak{a}, \omega, t)$. In other words, she can choose an action a from the set A_1^D. And analogously for player 2. On the other hand, when it is nature's turn (i.e. $pl(h') = 0$), nature can select any positive number $a \in \mathbb{R}^+$.

Formally, this means that an action a is only legal in state h' if the following conditions hold:

- if $pl(h') = 1$ then $a \in A_1^D$
- if $pl(h') = 2$ then $a \in A_2^D$
- if $pl(h') = 0$ then $a \in \mathbb{R}^+$

Furthermore, there are a number of other constraints that must be satisfied as well.

Specifically, in order for the numbers t_j and ϵ_j to be interpretable as *times* we have to impose the condition that, for any index j the number t_{j+1} must be larger than $t_j + \epsilon_j$. That is, if $(i_k, \eta_k, \omega_k, t_k)$ and ϵ_k are the last two actions of the history h', and $a = (i_{k+1}, \eta_{k+1}, \omega_{k+1}, t_{k+1})$, then we must have:

$$t_k + \epsilon_k < t_{k+1}$$

In addition, recall that the definition of the AOP specifies that an agent can only accept the *last* offer proposed by the other agent. That is, we must have:

$$\text{if} \quad \eta_{k+1} = \mathfrak{a} \quad \text{then} \quad \omega_k = \omega_{k+1}$$

Finally, the history h' is terminal (meaning that there is no action a such that $h' \circ (a)$ is legal), if and only if its length is an even number (i.e. $|h'| (\text{mod } 2) = 0$) and at least one of the following holds:

- $t_k + \epsilon_k \geq T$
- $k = N$
- $\eta_k = \mathfrak{a}$

The condition that the length has to be an even number, means that the negotiations have finished only after 'nature' has made its move, which means that the last propose- or accept-message must have arrived at its recipient.

5.5.4 The Observation Functions

Suppose we have the following history:

$$h = \Big((1, \mathfrak{p}, \omega_1, t_1), \epsilon_1, (2, \mathfrak{p}, \omega_2, t_2), \epsilon_2, (1, \mathfrak{p}, \omega_3, t_3), \epsilon_3, (2, \mathfrak{p}, \omega_4, t_4), \epsilon_4, \dots\Big)$$

As explained in Section 2.2.2, whenever player 1 proposes an offer, he will only be aware of the time t at which he proposed it, but he will not know how much time ϵ it takes for that message to arrive at player 2, and therefore he will not know the time $t+\epsilon$ at which player 2 receives it. Similarly, player 2 will not be able to observe the time t at which the message was sent, nor the delay ϵ, but will only observe the time $t + \epsilon$ at which she receives the message.

Therefore, the observed history for player 1 looks as follows:

$$f_1^{obs}(h) = \Big((1, \mathfrak{p}, \omega_1, t_1), (2, \mathfrak{p}, \omega_2, t_2 + \epsilon_2), (1, \mathfrak{p}, \omega_3, t_3), (2, \mathfrak{p}, \omega_4, t_4 + \epsilon_4), \dots\Big)$$

and for player 2:

$$f_2^{obs}(h) = \Big((1, \mathfrak{p}, \omega_1, t_1 + \epsilon_1), (2, \mathfrak{p}, \omega_2, t_2), (1, \mathfrak{p}, \omega_3, t_3 + \epsilon_3), (2, \mathfrak{p}, \omega_4, t_4), \dots\Big)$$

That is, the set of observations of Γ_D is just the set of negotiation actions:

$$O = A_1^D \cup A_2^D$$

Formally, let o_j^i denote the j-th observation received by player i, so we have:

$$f_i^{obs}(h) = (o_1^i, o_2^i, o_3^i, \dots, o_k^i)$$

Then, if $(i_j, \eta_j, \omega_j, t_j)$ denotes the j-th negotiation action of h, each o_j^i must satisfy:

$$o_j^i = \begin{cases} (i_j, \eta_j, \omega_j, t_j) & \text{if } i = i_j \\ (i_j, \eta_j, \omega_j, t_j + \epsilon_j) & \text{if } i \neq i_j \end{cases}$$

5.5.5 The Individual Active-Player functions

Recall that for a game of imperfect information, besides the active player function pl, we also need to define an *individual* active-player function pl_i, representing each player's *knowledge* about whose turn it is.

Note that when player 1 proposes an offer, then directly after this action, he knows that it is now the turn of 'nature', until the message has arrived, after which it will be player 2's turn. However, since player 1 cannot observe the duration of the delay, he will not know when exactly it stops being nature's turn and when it starts being player 2's turn. In other words, player 1 will typically not know whose turn it is, until it is his own turn. And the same holds of course for player 2.

So, if h_i^o denotes the observed history of player i (i.e. $h_i^o := f_i^{obs}(h)$), then:

$$pl_i(h_i^o) = \begin{cases} i & \text{if } pl(h_i^o) = i \\ ? & \text{otherwise} \end{cases}$$

5.5.6 The Utility Functions

The utility functions of the game Γ_D are defined in terms of the utility functions of the negotiation domain D. However, the utility functions of the game are defined over the set of terminal histories.

If the negotiation ended with an acceptance that arrived before the deadline, then each player receives their respective utility value $u_i(\omega_k)$ corresponding to the accepted offer ω_k. Otherwise, each player i receives his reservation value rv_i.

Formally, let h be a terminal history, and let $(i_k, \eta_k, \omega_k, t_k)$ denote the last negotiation action of h. Then:

$$u_i(h) = \begin{cases} u_i(\omega_k) & \text{if } \eta_k = \mathfrak{a} \text{ and } t_k + \epsilon_k < T. \\ rv_i & \text{otherwise} \end{cases}$$

where the u_i on the left-hand side is a utility function of the game Γ_D and the u_i on the right-hand side is a utility function of the negotiation domain D. Furthermore rv_i is the reservation value of player i of negotiation domain D.

5.5.7 Formal Definition

We can now put all this together into the formal definition of the game Γ_D.

Definition 5.5.1. *Let D be a bilateral negotiation domain with offer space Ω. Then a negotiation over this domain, according to the alternating offers protocol, with deadline T and maximum number of rounds N, can be modeled as a non-deterministic turn-taking game with imperfect information Γ_D, defined as follows:*

- *The set of actions A of the game Γ_D is defined as in Section 5.5.1.*
- *The active player map pl of Γ_D is defined as in Section 5.5.2.*
- *The set of legal histories $\mathcal{H}$ of Γ_D is defined as in Section 5.5.3.*
- *The observation functions f_i^{obs} of Γ_D are defined as in Section 5.5.4.*
- *The individual active-player functions pl_i of Γ_D are defined as in Section 5.5.5*
- *The utility functions u_i of Γ_D are defined as in Section 5.5.6.*

Now that we have formalized negotiation using the terminology of game-theory, we would like to apply techniques from game theory to determine the optimal negotiation strategy. Unfortunately, however, this turns out extremely difficult for several reasons.

The first reason, is that most techniques from game theory assume that the players have full information about each others' utility functions. An assumption that often does not hold in automated negotiation.

A second reason, is that we had to model negotiation as a turn-taking game with *imperfect* information. This means that to find the optimal strategy profile, we would need to determine the sequential equilibria of Γ_D, which is known to be an extremely hard problem to solve, even for very simple games. We therefore have to lower our expectations, and ignore the fact that the agents are not able to observe the delay times. If we pretend that they do know this information, then we can treat the game as if it was as a turn-taking game with perfect information, so we can try to determine its subgame-perfect equilibria.

A third reason, is that even if we assume that the agents could somehow observe the delays of *past* messages and therefore treat the game as a turn-taking game with *perfect* information, it would still be very difficult to find its subgame-perfect equilibria. This is because in order to play optimally, the agents would also have to be able to deal with the randomness of the delays of *future* messages. That is, the agents would still have to deal with the fact that it is a *non-deterministic* game. While in general there are techniques to deal with this, the problem is that in our case the random choices ϵ_j of nature can take an infinite number of possible values, which makes it hard to apply any well-known techniques.

For these reasons, the best result we can expect to obtain here, is to find the ordinary Nash equilibria of Γ_D, when regarded as a game of perfect information, and under the assumption that we know the utility functions and reservation values of both players.

While the assumption of full knowledge of both agents' utility functions and reservation values may be unrealistic in many real-life negotiation scenarios, it does allow us to determine a theoretical upper bound to what an agent could achieve in the ideal case that it had a perfect opponent modeling algorithm. In other words, it can be used in a laboratory setting to compare a real negotiation algorithm with a theoretically optimal one.

5.5.8 Nash Equilibria of a Negotiation

In this section we will show that the game Γ_D typically has many pure Nash equilibria.

As explained above, ideally, we would like to find the subgame-perfect equilibria, or even the sequential equilibria, of the game Γ_D. However, since this is very hard, we will instead just try to determine its ordinary Nash equilibria. We could then hope to find that this game has only one Nash equilibrium, which would then automatically also have to be its subgame-perfect equilibrium. Unfortunately, however, it turns out that this is typically not the case. In fact, the following theorem shows that the game Γ_D typically has *many* Nash equilibria: at least one for every offer that is Pareto-optimal and individually rational. An important consequence of this, is that if we want to determine an optimal negotiation strategy, we would need to apply some of the techniques discussed in Section 5.3.

Theorem 4. *Let D be a bilateral negotiation domain with a finite offer space Ω and let T be the deadline for the negotiations. If T is sufficiently large then for every offer $\omega \in \Omega$ that is Pareto-optimal and individually rational, there exists a pair of negotiation strategies (σ_1, σ_2) that forms a Nash equilibrium and that leads to ω as the final agreement (or another offer $\hat{\omega}$ with exactly the same utility values).*

Proof. Let ω be any arbitrary Pareto-optimal and individually rational offer. Given ω, let σ_1 be a time-based strategy based on Eq. (3.1) or Eq. (3.3) and with an aspiration function defined by Eq. (3.5) with target value $\beta_1 = u_1(\omega)$, in combination with the AC_{asp} acceptance strategy (Def. 3.3.3). Similarly, let σ_2 be a time-based strategy, defined by the same equations, and with target value $\beta_2 = u_2(\omega)$.

Now, to prove the theorem, we will prove the following three claims one by one:

1. If these two strategies come to an agreement, then it must either be the offer ω, or some other offer $\hat{\omega}$ with exactly the same utility vector.
2. These two strategies will indeed come to an agreement.
3. Neither of the two agents can improve by making a unilateral deviation.

To prove the first claim, note that for any arbitrary offer ω' one of the following must hold:

1. ω' dominates ω.
2. $u_1(\omega') < u_1(\omega)$
3. $u_2(\omega') < u_2(\omega)$
4. $u_1(\omega') = u_1(\omega)$ and $u_2(\omega') = u_2(\omega)$.

However, the first case is impossible, because we assumed that ω was Pareto-optimal. In the second case we would have that $u_1(\omega') < \beta_1$ which means, by definition of β_1, that ag_1 would never propose or accept ω'. Similarly, in the third case we would have that $u_2(\omega') < \beta_2$ which means that ag_2 would never propose or accept ω'. This means that in the second or third case, ω' could not be the final agreement of the negotiations. Therefore, the only case in which ω' could be the final agreement is the fourth case, which is what we wanted to prove.

Now, to prove the second claim, note that if T is large enough, then sooner or later either of the two agents will have proposed all other offers that are better for him than ω, so that agent will eventually propose ω. Furthermore, sooner or later the other agent's aspiration level $\lambda_i(t)$ will become equal to her utility value $u_i(\omega)$ and thus she will eventually accept the offer ω.

Finally, to prove the third claim, we will show that agent ag_2 cannot deviate unilaterally to a better strategy (we should also show the same for ag_1, but that goes analogously). To do this, note that if ag_2 does deviate to any alternative strategy σ'_2, then this must yield one of the following outcomes:

1. The negotiations end without agreement.
2. The negotiations end with the same agreement ω.
3. The negotiations end with a different agreement ω' such that $u_2(\omega') \leq u_2(\omega)$.
4. The negotiations end with a different agreement ω' such that $u_2(\omega') > u_2(\omega)$.

In the first case, the deviation did not improve the outcome for agent ag_2, because she ends up with her reservation value rv_2. Note that we assumed that ω was individually rational, and therefore we have $rv_2 < u_2(\omega)$, so indeed she would have been better off if she didn't deviate.

In the second case the deviation did not improve the outcome for ag_2 either, because the outcome is the same as for the original strategy profile.

In the third case, again, the deviation did not improve her outcome, because agent ag_2 ends up with less or equal utility than in the original situation.

In the fourth case agent ag_2 does improve, but we will show that this case cannot happen. The reason for this, is that we assumed that ω was Pareto-optimal. This means that if $u_2(\omega') > u_2(\omega)$, we must necessarily have $u_1(\omega') < u_1(\omega)$, otherwise ω' would dominate ω and therefore ω would not be Pareto-optimal. However, since we assumed that ag_1 applies a time-based strategy with target value $\beta_1 = u_1(\omega)$, we know that ag_1 would never accept or propose any offer with utility lower than $u_1(\omega)$, so in particular she would never propose or accept ω', which means that ω' could never become an agreement.

We have therefore proved that ag_2 cannot make a unilateral deviation that increases her utility. The fact that this also holds for ag_1 can be proved in exactly the same way. □

5.5.9 Non-credible Threats in a Negotiation

Now that we have determined the Nash equilibria of a negotiation, the question we will investigate next, is whether or not any of them are based on non-credible threats. It turns out that indeed, such non-credible threats do appear when either of the agents blindly follows a Nash equilibrium.

Imagine we are very close the the deadline and agent 1 has proposed some Pareto-optimal and individually rational offer ω. Furthermore, suppose that for the remainder of the negotiations, agent 1 has chosen the following strategy: "reject any counter offer from agent 2 that is worse for me than ω, and do not make any further concessions, no matter what". Now, it is easy to see that for agent 2 the best response to this strategy would be to accept the offer ω. However, let us assume that agent 2 does not play this best response (that is, player 2 'deviates') and instead makes a counter offer ω' with slightly less utility for agent 1. Furthermore, suppose that there is not enough time left for agent 1 to propose any new offer. So, agent 1 can only accept ω' or accept that the negotiations will fail. Assuming ω' is also individually rational, it would be sub-optimal for agent 1 to stick to his strategy (which would cause the negotiations to fail) because he would be better off by accepting ω'. Therefore, he would be forced to also deviate, which means that his original strategy was indeed based on a non-credible threat.

5.6 Bargaining Solutions

Now that we have developed a basic understanding of game theory and modeled negotiations as a game, we can start investigating how game theory can help us finding an optimal negotiation strategy. This question is also known as **the bargaining problem**.

Despite the fact that this is a very old research topic, dating back to the 1950's, however, there is still no definitive general solution to this problem. However, just as for the equilibrium selection problem, many different solutions have been proposed that are each applicable to different special cases. Such solutions are often referred to as **bargaining solutions**. We will here discuss just a few of them.

Note that these bargaining solutions do not actually find any optimal negotiation strategies *directly*. Instead, they merely determine the offer that two 'optimal' negotiation strategies would agree upon (depending on the definition of 'optimal', which differs for each such bargaining solution). However, once we have found such an offer ω^*, we know that our agent shouldn't be proposing or accepting anything worse than that, so we can implement a strategy that concedes towards, but no further than, that offer. For example, it could be a time-based strategy that uses the value $u_1(\omega^*)$ as its target value. We then know from the proof of Theorem 4 that a pair of such strategies would form a Nash equilibrium.

We should point out, however, that each of these bargaining solutions is based on the assumption that we know the utility functions of *both* agents. This means we typically cannot apply them directly in practice, because a negotiating agent typically only knows his own utility function, but not his opponent's. Nevertheless, these bargaining solutions can still be very useful, because they can serve as a theoretical upper bound to what a real negotiation algorithm could achieve. That is, we can evaluate a real negotiation algorithm (that does not have knowledge of its opponent's utility function) by comparing it with the theoretically optimal solution that could be achieved if one does have knowledge of the opponent's utility function. For example, it has been shown [17] that if two MiCRO agents negotiate against each other, then they often come remarkably close to to the *Nash Bargaining Solution* as well as the *Max-Sum solution* (we will discuss these solutions below).

Another important simplification made by most bargaining solutions, is that they model the the negotiations as a normal-form game, even though we have seen that negotiation should technically be seen as an extensive-form game.

5.6.1 The Nash Bargaining Solution

The solution to the bargaining problem that is by far the best-known, is the so-called *Nash Bargaining Solution* (NBS), which was invented by John Nash in 1950 [40] (the same person that also invented the concept of the Nash equilibrium).

So far, we have always considered negotiation domains in this book that have *finite* offer spaces. However, the NBS applies to negotiation domains with an *infinite* number of possible agreements. In particular, Nash assumed that the offer space formed a subset of an m-dimensional vector space $\mathbb{R}^m$, for some positive integer m. That is, $\Omega \subseteq \mathbb{R}^m$. This means that we can now talk about 'linear combinations' of offers. That is, given two offers ω and ω' and two real numbers a and b, the linear combination $a \cdot \omega + b \cdot \omega'$ is also a vector, which may or may not lie inside the agreement space. Moreover, it means we can calculate the Euclidean distance $d(\omega, \omega')$ between those two offers. More specifically, Nash worked under the assumption that the offer space is a *closed*, *bounded* and *convex* subset of $\mathbb{R}^m$. We will now explain these terms.

Definition 5.6.1. *Let S be some subset of $\mathbb{R}^m$, i.e. $S \subseteq \mathbb{R}^m$. Then, we say that S is **bounded** iff there exists a number M such that for any two vectors $v, w \in S$ their distance is smaller than M:*

$$\exists M \in \mathbb{R} : \quad \forall v, w \in S : \quad d(v, w) \leq M$$

where $d(v, w)$ denotes the Euclidean distance between v and w.

In other words, there is a maximum distance M between any two points in S.

Definition 5.6.2. *Let S be some subset of $\mathbb{R}^m$, i.e. $S \subseteq \mathbb{R}^m$. Then, we say that S is **closed** iff for any convergent sequence of vectors in S, the limit of that sequence is also contained in S.*

For example, the following set is *not* closed:

$$S \quad = \quad \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 < 1\}$$

because the following sequence:

$$(0,0) \quad , \quad (0, \frac{1}{2}) \quad , \quad (0, \frac{3}{4}) \quad , \quad (0, \frac{7}{8}) \quad , \quad (0, \frac{15}{16}) \quad , \quad \ldots$$

converges to the point $(0, 1)$, but that point does not lie inside S. On the other hand, the following set *is* closed:

$$S' \quad = \quad \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$$

Roughly speaking, one could say that a set is closed if the "border" of that set is also part of the set itself.

Definition 5.6.3. *A subset S of $\mathbb{R}^m$ is called **compact** if it is both bounded and closed.*

Definition 5.6.4. *Let S be some subset of $\mathbb{R}^m$, i.e. $S \subseteq \mathbb{R}^m$. Then, we say that S is **convex** iff for any two vectors $v, w \in S$, and any real number $x \in [0, 1]$, it holds that the vector $x \cdot v + (1 - x) \cdot w$ is also contained in S.*

Intuitively, the term 'convex' means that if you take any two points v and w in S, then the line between v and w must be completely contained inside S. See Figure 5.3 for a visualization of this concept. Note that a

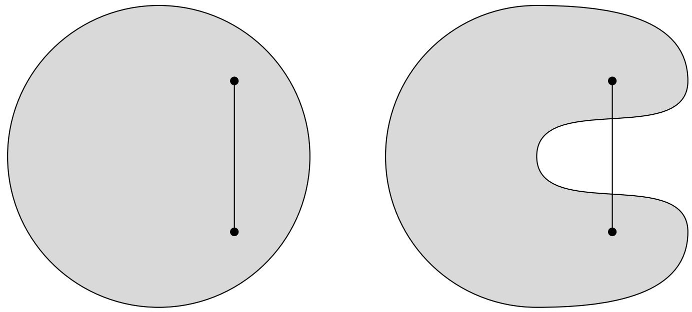


Figure 5.3: Left: a convex set. For any pair of points inside this set, the line between them will also lie completely inside the set. Right: a non-convex set. We can find two points that are both inside the set, but the line between them is not completely contained inside the set.

convex set must necessarily contain *infinitely* many elements (because there are infinitely many real numbers x in the interval $[0, 1]$).

The offer spaces that Nash discusses (compact and convex) are very different from the offer spaces we have so far discussed in this book. An example of a compact and convex negotiation domain, is the scenario that one agent aims to sells a certain commodity, such as oil or gold, to another agent, and they negotiate the quantity to sell and the price, which are both constrained to some finite domain. For example, Alice has 1000 liters of crude oil for sale, and Bob has a budget of $800 that he can spend on oil. In that case the offer space is given by the set $\Omega = [0, 1000] \times [0, 800]$, which is indeed a compact and convex[2] subset of $\mathbb{R}^2$.

Furthermore, Nash himself argued that even if there are really only a finite number of possible offers, then we could still consider the offer space to be compact and convex, if we also allow the agents to propose and accept *lotteries* over offers. For example, they could agree that they will flip a coin. If the coin comes up 'heads' then they will execute solution ω, while if the coin comes up 'tails' then they will execute solution ω'. If we furthermore allow the agents to agree that the coin can be biased with any probability P for heads or tails, then the set of lotteries is indeed compact and convex.

[2]Technically, this is not entirely true, because the price can only be specified as a finite number of dollar cents. However, this is such a small step-size that we can ignore this and pretend that money is a continuous commodity.

> ***Opinion.*** Personally, I have always found the possibility of negotiating over lotteries rather far-fetched. I just can't imagine many real-world situations in which two negotiators would jointly agree to flip a coin.

Apart from these assumptions about the agreement space, Nash also assumed that the two agents have full knowledge of each others' utility functions, that the offer space contains at least one individually rational offer, and that the two utility functions are '*distributive*' over linear combinations of offers. That is:

$$\forall i \in \{1,2\}: \ \forall \omega, \omega' \in \Omega: \ \forall a, b \in \mathbb{R}: \quad u_i(a\omega + b\omega') = au_i(\omega) + bu_i(\omega')$$

Note that this is a stronger assumption than merely assuming the functions are linear in the sense of Eq. (2.5), because this implies that every *evaluation* function v_i^j must also be a linear function itself (from $\mathbb{R}$ to $\mathbb{R}$).

In summary, the Nash Bargaining Solution applies to negotiations that satisfy the following conditions:

- C1: The negotiations are bilateral.
- C2: The agreement space Ω is a compact and convex subset of $\mathbb{R}^m$ (for some positive integer m).
- C3: The agreement space contains at least one individually rational offer (see Def. 2.3.1).
- C4: The utility functions are distributive over the offers.
- C5: The two agents have full knowledge of each others' utility functions.

For any negotiation domain D, let us define the **utility space** $\Upsilon_D \subseteq \mathbb{R}^2$ to be the set of all utility vectors. That is:

$$\Upsilon_D := \{\big(u_1(\omega), u_2(\omega)\big) \mid \omega \in \Omega\}$$

Note that if Ω is convex, and the two utility functions are distributive, then Υ_D is also a convex set.

Definition 5.6.5. *Let D be a negotiation domain. We say that D is* ***symmetric*** *if for any $(x, y) \in \Upsilon_D$ we also have $(y, x) \in \Upsilon_D$.*

If D and D' are two negotiation domains, then we say that D' is a linear transformation of D if they both have the same offer space, and the utility functions and reservation values of D' are just linear transformations of the utility functions and reservation values of D. More precisely:

Definition 5.6.6. *Let* $D = (\Omega, u_1, u_2, rv_1, rv_2)$ *and* $D' = (\Omega', u_1', u_2', rv_1', rv_2')$ *be two negotiation domains. We say that* D' *is a* ***linear transformation*** *of* D *iff* $\Omega = \Omega'$ *and there exist four real numbers* $a_1, b_1, a_2, b_2 \in \mathbb{R}$*, with* $a_1 > 0$ *and* $a_2 > 0$*, such that all of the following conditions are satisfied:*

$$\begin{array}{rrcl} \forall \omega \in \Omega : & u_1'(\omega) & = & a_1 \cdot u_1(\omega) + b_1 \\ & rv_1' & = & a_1 \cdot rv_1 + b_1 \\ \forall \omega \in \Omega : & u_2'(\omega) & = & a_2 \cdot u_2(\omega) + b_2 \\ & rv_2' & = & a_2 \cdot rv_2 + b_2 \end{array}$$

We say that a domain D' is an *extension* of another domain D, if D can be obtained by removing some offers from the offer space of D'. More precisely:

Definition 5.6.7. *Let* $D = (\Omega, u_1, u_2, rv_1, rv_2)$ *and* $D' = (\Omega', u_1', u_2', rv_1', rv_2')$ *be two negotiation domains. We say that* D' *is an* ***extension*** *of* D *iff* $\Omega \subseteq \Omega'$ *and the following conditions are all satisfied:*

$$\begin{array}{rrcl} \forall \omega \in \Omega : & u_1'(\omega) & = & u_1(\omega) \\ & rv_1' & = & rv_1 \\ \forall \omega \in \Omega : & u_2'(\omega) & = & u_2(\omega) \\ & rv_2' & = & rv_2 \end{array}$$

Nash argued that, for any negotiation domain D, if the conditions C1–C5 hold, then the agreement between two optimal negotiators would be an offer ω_D^* that satisfies all of the following axioms:

- A1 *Pareto-optimality*:
  The agreement ω_D^* should be Pareto-optimal (see Def. 2.3.3).
- A2 *Symmetry*:
  If Υ_D is symmetric, then $u_1(\omega_D^*) = u_2(\omega_D^*)$
- A3 *Invariance under linear transformations*:
  If D' is a linear transformation of D, then $\omega_D^* = \omega_{D'}^*$.
- A4 *Independence of irrelevant alternatives*:
  If D' is an extension of D, then either $\omega_{D'}^* = \omega_D^*$, or $\omega_{D'}^* \in \Omega' \setminus \Omega$.

The first two of these axioms speak for themselves. The third axiom follows directly from the principle of von Neumann-Morgenstern utilities, which we discussed in Section 2.2.3.1. That is, since the application of a linear transformation to either of the two utility functions of D does not essentially change the domain, we can say that D' is essentially the same as D, and therefore the outcome of the negotiations should be the same.

The fourth axiom is probably less obvious, and is easiest to explain for the case that Ω' contains just one offer more than Ω. That is, suppose that in domain D, two optimal negotiators would agree on offer $\omega^*_D \in \Omega$. Now, suppose that we add a new offer ω' to the offer space, so we have $\Omega' = \Omega \cup \{\omega'\}$, and we repeat the negotiations. Then, it may or may not happen that the two negotiators will now agree on that new offer ω'. However, in the case that this does not happen, then this new offer can be considered 'irrelevant' to the negotiations and should not affect the outcome. Therefore, in that case the agents should come to exactly the same agreement ω^*_D as in the original negotiations. After all, it would be strange if the addition of some new offer ω' would lead the negotiators to accept a *different* offer ω''. This is exactly what axiom A4 says, except that it allows for any arbitrary number of 'irrelevant' new offers.

Nash proved the following theorem [40]:

Theorem 5. *Under conditions C1, C2, C3, C4 and C5, any offer ω^*_D that satisfies axioms A1, A2, A3, and A4, satisfies:*

$$\omega^*_D \quad \in \quad \arg\max_{\omega\in\Omega} \quad \{ \ (u_1(\omega) - rv_1) \cdot (u_2(\omega) - rv_2) \ \}$$

Furthermore, all such offers have exactly the same utility vector.

In other words, Nash argued that under the given conditions two optimal negotiators would agree on the offer that maximizes the product of the two agents' utility values (minus reservation values). This implies that an optimal negotiation strategy, according to Nash, would be one that aims to achieve that offer as the final agreement. For example, this could be a time-based strategy with $\beta_i = u_i(\omega^*_D)$.

Definition 5.6.8. *Let D be a bilateral negotiation domain. Then, for any offer $\omega \in \Omega$ of this domain, the product $(u_1(\omega) - rv_1) \cdot (u_2(\omega) - rv_2)$ is called the **Nash product**. Furthermore, the offer that maximizes the Nash product is called the **Nash Bargaining Solution**.*

The essence behind the NBS, is that it models negotiation as a normal-form game. To simplify it a bit, we can restrict the game to only allow the players to choose *time-based* strategies. We know from Theorem 4 that such a game has many Nash equilibria. Then, if D is symmetric, this means that the game is a symmetric game, so we can use the solution of Section 5.3.4 to pick an optimal Nash equilibrium, and it can be shown that this will indeed yield an agreement that maximizes the Nash product (there are many such pairs of time-based strategies, but this is a factorizable set). Furthermore, for those cases that the negotiation domain D is not symmetric, Nash used his various conditions and axioms to show that there must exist some other domain D' that *is* symmetric and that should yield the same outcome as D.

Now, we should warn the reader that the NBS is sometimes discussed by authors as if it was the one and only correct solution to the bargaining problem. However, one should not forget that the NBS in principle only applies to those negotiations where the (rather strong) conditions C1–C5 hold, and even then it only applies under the (controversial) assumption that the axioms A1–A5 are indeed a 'correct' description of an optimal solution. Authors sometimes tend to forget or ignore these restrictions.

Finally, we should warn that some authors erroneously define the NBS as the offer that directly maximizes the product of the utilities $u_1(\omega) \cdot u_2(\omega)$, without subtracting the reservation values. However, this is wrong, because it would not satisfy the axiom of Invariance under Linear Transformations. Unless, of course, the reservation values both happen to be zero.

5.6.2 The Kalai-Smorodinsky Solution

In 1975, two researchers by the names of Kalai and Smorodinsky [32] argued that there is a problem with the NBS, which they demonstrated with the following example.

Suppose we have two bilateral negotiation domains D_1 and D_2 that satisfy the conditions C1–C5 from Section 5.6.1. Furthermore, suppose that D_2 is an extension of D_1. To be specific, suppose that their utility spaces are given as follows:

- Υ_1 is the interior of the following four points:

$$\{ \ (0,0) \ , \ (1,0) \ , \ (0,1) \ , \ (0.75, 0.75) \ \}$$

- Υ_2 is the interior of the following four points:

$$\{ \ (0,0) \ , \ (1,0) \ , \ (0,1) \ , \ (1, 0.7) \ \}$$

See Figure 5.4 for an illustration.

Now, note that for any utility value $x \in [0,1]$ there exists a $y \in [0,1]$ such that $(x,y) \in \Upsilon_2$ and such that for all $(x,y') \in \Upsilon_1$ we have $y' < y$. So,

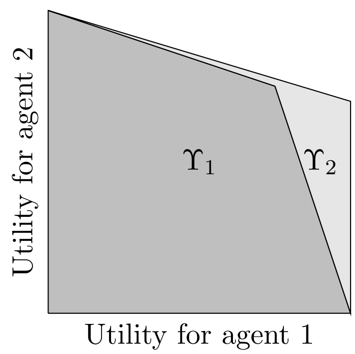


Figure 5.4: The dark grey area represents all Υ_1, while the light grey area represents the difference between Υ_1 and Υ_2, i.e. $\Upsilon_2 \setminus \Upsilon_1$. The horizontal axis represents the utility of agent ag_1 while the vertical axis represents the utility of agent ag_2.

in words: no matter what utility value ag_1 would receive from the optimal solution of domain D_1, there always exists some offer ω in Ω_2 that gives the same utility to ag_1, but that is strictly better for ag_2. It therefore makes sense to expect that ag_2 would at least receive the same utility from D_2 as from D_1, and possibly even better.

However, if we follow the NBS, then we would conclude that the outcomes $\omega^*_{D_1}$ and $\omega^*_{D_2}$ for the two respective domains should be given by:

$$\begin{aligned} \vec{u}(\omega^*_{D_1}) &= (0.75\ ,\ 0.75) \\ \vec{u}(\omega^*_{D_2}) &= (1\ ,\ 0.7) \end{aligned}$$

That is, even though we have extended domain D_1 with offers that are better for ag_2, the final outcome of the negotiation would, according to the NBS, actually be *worse* for ag_2 (i.e. 0.7 instead of 0.75).

Kalai and Smordinsky argued that this result is highly unsatisfactory and that it is caused by the axiom of independence of irrelevant alternatives. They therefore proposed an alternative axiom, called *The Axiom of Monotonicity.*

In order to define this axiom we must first introduce some more notation. For any negotiation domain D, let Υ^p denote its pareto-frontier (see Def. 2.3.4):

$$\Upsilon^p \quad := \quad \{\ (u_1(\omega), u_2(\omega)) \mid \omega \in \Omega^p\ \}$$

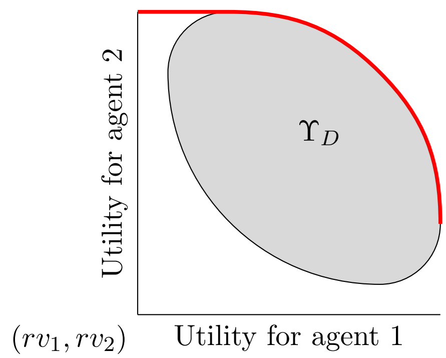


Figure 5.5: The red line indicates the function ks_D as defined by Eq. (5.2)

where Ω^p is the Pareto-set of D. Furthermore, let ks_D be a function from the interval $[rv_1, u_1^{max}]$ to $\mathbb{R}$, defined as follows:

$$ks_D(x) = \begin{cases} y & \text{if } \exists y : (x,y) \in \Upsilon^p \\ u_2^{max} & \text{otherwise} \end{cases} \tag{5.2}$$

This function is visualized in Figure 5.5. Finally, for any two given negotiation domains D and D', let u_1^{max} and $u_1^{max\prime}$ denote the highest possible utility values that agent 1 can achieve in these two respective domains (as per Eq. (2.3)).

The Axiom of Monotonicity is then defined as follows.

- A5 *The Axiom of Monotonicity*: If D and D' are two negotiation domains such that $u_1^{max} = u_1^{max\prime}$ and such that

  $$\forall x \in [rv_1, u_1^{max}] : \quad ks_D(x) \leq ks_{D'}(x)$$

  then we should have:

  $$u_2(\omega_D^*) \quad \leq \quad u_2(\omega_{D'}^*).$$

Kalai and Smorodinsky themselves described this axiom informally as follows: *"If, for every utility level that agent 1 may demand, the maximum feasible utility level that agent 2 can simultaneously reach is increased, then the utility level assigned to agent 2 according to the solution should also be increased."*

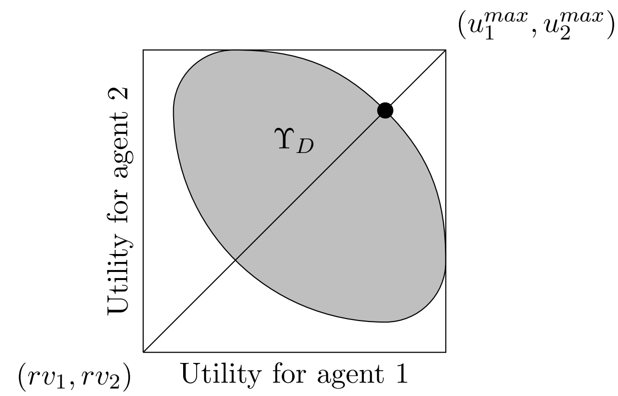


Figure 5.6: The black dot represents utility vector of the Kalai-Smorodinsky solution. It lies, by definition, on the intersection of Υ_D and the diagonal between (rv_1, rv_2) and (u_1^{max}, u_2^{max}). Specifically, it is the point on this intersection that lies closest to (u_1^{max}, u_2^{max}).

So, they argue that under the same conditions C1–C5 as Nash, the agreement made by two optimal negotiators should actually satisfy axioms A1, A2, A3 and A5 (instead of A4).

They then prove the following theorem:

Theorem 6. *Under conditions C1, C2, C3, C4 and C5, any offer ω_D^* that satisfies axioms A1, A2, A3, and A5, is given by:*

$$\omega_D^* \quad = \quad \arg\max_{\omega \in \Omega} \{u_1(\omega) \mid (u_1(\omega), u_2(\omega)) \in L(D)\}$$

where $L(D)$ is the line from the point (rv_1, rv_2) to the 'utopian' point (u_1^{max}, u_2^{max}). Furthermore, all such offers have exactly the same utility vector.

In other words, Kalai and Smorodinsky argued that, under the given conditions, two optimal negotiators would agree on the offer for which the utility vector lies on the intersection of the utility space and the diagonal between the reservation values and the 'utopian' point (u_1^{max}, u_2^{max}), and such that it is closest to that utopian point. This is known as the **Kalai-Smorodinsky Solution** (KSS). See Figure 5.6 for an illustration of this concept.

5.6.3 The Max-Sum Solution

The NBS and the KSS are widely used as a reference to compare to what extent real negotiation algorithms are able to negotiate optimally. This is quite striking, however, given that both solutions technically only apply to negotiations over convex offer spaces, and with distributive utility functions, while many negotiation scenarios that are studied in the literature do not satisfy these conditions.

For this reason, in [16] a new bargaining solution was proposed specifically for negotiations over *finite* offer spaces, rather than convex ones. Just as for the NBS, the idea behind this solution is to model negotiations as a normal-form game, which, as discussed in Section 5.5.8, typically has many Nash equilibria. However, this time, to choose the optimal equilibrium we use the solution discussed in Section 5.3.5. This means that this bargaining solution only applies to those situations in which the Assumption of Role Equifrequency (AoRE) holds (Def. 5.3.8). That is, given any domain D, our agent assumes that it will be negotiating over that domain with each of the two utility functions equally often or with equal probability. Note that for most studies in the literature, this assumption indeed holds. See for example the various ANAC competitions [9, 3].

From this it follows that two optimal negotiators would (typically) agree on the offer that maximizes the *sum* of the utilities of the two agents. We will therefore call this the **Max-Sum** solution.[3]

More precisely, he max-sum solution works as follows:

1. Select all offers that are Pareto-optimal and individually rational.
2. Among those offers, select the subset that maximizes the utility-sum $u_1(\omega) + u_2(\omega)$.
3. Among those offers, select the subset that minimizes the absolute utility difference $|u_1(\omega) - u_2(\omega)|$.
4. If these offers all have the same utility vector, then return any of those offers.
5. Otherwise, discard this set of offers and go back to step 2 with the remaining Pareto-optimal and individually rational offers.

Note that this is essentially just Algorithm 10, where we identify the set of Nash equilibria with the set of Pareto-optimal and individually rational offers (as per Theorem 4).

[3] In the literature this bargaining solution has also been called the *maximum social welfare solution*, but this name may be a bit misleading, because it actually has nothing to do with maximizing social welfare.

The main advantage of this bargaining solution is that we can drop the assumption that the offer space has to be compact and convex, or that the utility functions are distributive. Furthermore, we no longer need the controversial axioms of Independence of Irrelevant Alternatives, or of Monotonicity.

Regarding to the axiom of Invariance under Linear Transformations, however, we have to be a bit careful. Nash's axiom states that we can apply two different linear transformations to the two utility functions of the domain. However, as we also discussed in Section 5.3.5, under the AoRE, this is actually too strong. That is, we have to make a careful distinction between 'players' and 'agents', and while it makes perfect sense to allow each *agent* to apply any arbitrary linear transformation, we cannot say the same about *players*.

Now, to demonstrate that, under the AoRE and on finite domains, the Max-Sum solution is indeed better than the NBS, we will now give an explicit example of such a negotiation domain and show that a strategy based on the Max-Sum solution indeed outperforms a strategy based on the NBS.

Suppose Alice and Bob are negotiating against each other over some negotiation domain D. We will assume that they will negotiate twice. Once with Alice having utility u_1 and Bob having utility u_2, and once with the utility functions assigned in the opposite way. Therefore, the AoRE holds. Furthermore, let us assume that Alice and Bob are each only considering two possible negotiation strategies, which we will call 'sum-maximizer' (denoted S) and 'product-maximizer' (denoted P). The sum-maximizer strategy only accepts the offer that maximizes the utility-sum or anything with higher utility, while the product-maximizer only accepts the offer that maximizes the Nash product, or anything with higher utility.

Now, let us assume that the negotiation domain D only has two offers: ω_s and ω_p. These offers have respective utility vectors $\vec{u}(\omega_s) = (5, 1)$ and $\vec{u}(\omega_p) = (2, 3)$. Furthermore, both reservation values are 0. Note that ω_s has the highest utility-sum ($5 + 1 = 6$ vs. $2 + 3 = 5$), while ω_p has the highest Nash product ($(5 - 0) \cdot (1 - 0) = 5$ vs. $(2 - 0) \cdot (3 - 0) = 6$).

We now observe the following facts:

- If both agents choose strategy S, then the only possible agreement is ω_s.
- If both agents choose strategy P, then the only possible agreement is ω_p.
- If the agent with utility u_1 chooses S and the agent with utility u_2 chooses P, then neither of the two offers can become an agreement

(because ag_1 demands a utility of at least $u_1(\omega_s) = 5$, while ag_2 demands a utility of at least $u_2(\omega_p) = 3$ and there is no offer that satisfies both constraints).

- If the agent with utility u_1 chooses P and the agent with utility u_2 chooses S, then either of the two offers can become an agreement (because ag_1 demands a utility of at least $u_1(\omega_p) = 2$, while ag_2 demands a utility of at least $u_2(\omega_s) = 1$ and both offers satisfy both constraints).

  Whenever this happens we will assume there is a 50% chance that the agreement will be ω_s and a 50% chance that the agreement will be ω_p.

From this, we conclude that, depending on which strategies Alice and Bob choose, we get the following:

- If Alice and Bob both choose strategy S, then in both negotiations the agreement will be ω_s, and thus they will each obtain a utility of 6 (in the first negotiation Alice will receive 5 and Bob will receive 1, and in the second negotiation vice versa).
- If Alice and Bob both choose strategy P, then in both negotiations the agreement will be ω_p, and thus they will each obtain a utility of 5 (in the first negotiation Alice will receive 3 and Bob will receive 2, and in the second negotiation vice versa).
- If Alice chooses strategy S and Bob chooses P, then the first negotiation will end without agreement, so both will receive a utility of 0. The second negotiation will end with either of the two offers. Therefore, Alice will have an expected utility of $0.5 \cdot 1 + 0.5 \cdot 3 = 2$, while Bob will have an expected utility of $0.5 \cdot 5 + 0.5 \cdot 2 = 3.5$.
- If Alice chooses strategy P and Bob chooses S, then the situation is reversed, so now Alice has an expected utility of 3.5 and Bob has an expected utility of 2.

This can now be modeled as a normal-form game with the following (symmetric) payoff matrix:

	S	P
S	(6 , 6)	(2 , 3.5)
P	(3.5 , 2)	(5 , 5)

While this game has two Nash equilibria (S, S) and (P, P), it is clear that (S, S) is the better choice.

Chapter 6

Evaluation of Negotiation Algorithms

In Chapter 3 we discussed various types of negotiation strategies. Then, in Chapter 5 we discussed how one could try to use game theory to determine which strategy is the best. Unfortunately, however, the 'bargaining solutions' we discussed there were based on two major simplifications. Firstly, we had to model negotiations as a normal-form game, rather than as an extensive-form game. Secondly, they all depended on having full knowledge of both the agents' utility functions. For these reasons, game theory is only of limited use when implementing an actual negotiation strategy.

Therefore, in practice, whenever we implement a negotiation strategy we have to resort to experimental methods to assess its strength. In this chapter we will discuss how to do that. In fact, we will describe three different methods to evaluate agents:

1. Tournament Evaluation
2. Empirical Game-Theoretical Analysis (EGTA)
3. Sequential Elimination Ranking

Throughout this chapter we will assume we have some set of agents, denoted $Ag = \{ag_{\underline{1}}, ag_{\underline{2}}, \dots, ag_{\underline{n}}\}$, that we want to compare to each other and that are all developed to negotiate under some bilateral negotiation protocol.

Note that we here underline the indices of the agents. This is to make a clear distinction between the $\underline{i}$-th agent from our collection of agents Ag, and the i-th agent in a given bilateral negotiation. More precisely, for any negotiation over some given bilateral domain D, the notation ag_1 refers to the agent that aims to maximize the utility function u_1 from that domain, and ag_2 refers to the agent that aims to maximize utility u_2. On the other hand $ag_{\underline{1}}$ refers to the first agent from our collection of agents Ag, and $ag_{\underline{2}}$ refers to the second agent from that collection.

For example, suppose that the domain D represents a car sale, in which u_1 is the utility function of the buyer and u_2 is the utility function of the seller. Then, the expression $ag_{\underline{1}} = ag_2$ would mean that the first agent from our set of agents Ag is acting as the 'seller' in the negotiation, while the expression $ag_{\underline{4}} = ag_1$ would mean that the fourth agent from our set of agents is acting as the buyer in this negotiation.

In general, any underlined index always refers to the index of an agent within the set Ag, while a regular index refers to the role the agent is playing within a given negotiation.

6.1 Tournament Evaluation

The most basic way to evaluate a negotiating agent, is to compare it with a number of other benchmark agents, by means of a tournament. That is, we first pick a number of well-known existing benchmark agents, plus a number of negotiation domains, and then let all agents (i.e. all benchmark agents plus our own agent) negotiate against each other, in every domain. We may repeat this several times, in order to obtain enough data to get statistically significant results, and then finally we calculate the average utility obtained by each agent, over all the negotiations it was involved in, and rank all agents based on that average utility. Hopefully, our agent then ends in first place in that tournament.

6.1.1 Tournament Score

Suppose we have an agent $ag_{\underline{1}}$ and we want to compare it to a number of benchmark agents $ag_{\underline{2}}, ag_{\underline{3}}, \dots, ag_{\underline{n}}$, all developed to negotiate under some given bilateral negotiation protocol Π. Furthermore, suppose we have a number of different bilateral negotiation domains $D_1, D_2, \dots, D_m$. Since all domains are bilateral we can create $m \times n \times n$ different negotiation *scenarios*.

Definition 6.1.1. *We define a bilateral negotiation* ***scenario*** *sc as a tuple* $sc = (\Pi, D, ag, ag')$*, where* Π *is a negotiation protocol,* D *is a negotiation domain, and* ag *and* ag' *are two negotiating agents.*

A negotiation scenario represents a negotiation between agents ag and ag' over domain D, under negotiation protocol Π. It is important to realize that if ag and ag' are two different agents, then (Π, D, ag, ag') and (Π, D, ag', ag) are two different scenarios. This is because in the first scenario ag has utility function u_1 from domain D and ag' has utility function u_2 from that same domain. On the other hand, in the second scenario ag' has utility function u_1 and ag has utility function u_2.

For example, if D represents a negotiation between a car seller and a buyer, then in the first scenario ag plays the role of the seller and ag' plays the role of the buyer, while in the second scenario the roles are reversed, so now ag' is the seller and ag is the buyer.

Furthermore, it is also important to realize that we allow a scenario to be of the form (Π, D, ag, ag) where both roles are played by the same agent ag. Of course, it doesn't make sense to literally have a *single* agent negotiating against itself, but what we mean is that we may have two agents that are each applying exactly the same negotiation algorithm. Or, in other words, we may have two identical *copies* of the same agent negotiating against each other. We should stress that in that case, even though the two agents are identical, they should still be seen as two separate agents that have opposing interests. That is, the first copy is purely interested in maximizing utility function u_1, while the second copy is purely interested in maximizing utility function u_2. The fact that both agents have identical implementations doesn't change that. There is no reason to assume that an agent would treat his opponent any differently from other opponents if that opponent happens to be an identical copy of himself.

For example, imagine again the case of a seller and a customer that are negotiating the price of a car. This time, assume that they are humans, but that they each download a negotiation algorithm from the Internet that will do the negotiations for them. It is then perfectly conceivable that they each happen to download exactly the same algorithm, so there will be two copies of the same agent negotiating against each other. The agent downloaded by the seller will then try to negotiate the highest possible price, while the agent downloaded by the buyer will try to negotiate the lowest possible price.

Definition 6.1.2. *A bilateral* ***negotiation tournament*** *is a tuple* $(\Pi, \mathcal{D}, Ag, \mathcal{R})$ *where:*

- Π *is a bilateral negotiation protocol.*
- $\mathcal{D} = \{D_1, D_2, \ldots, D_m\}$ *is a set of bilateral negotiation domains.*
- $Ag = \{ag_{\underline{1}}, ag_{\underline{2}}, \ldots, ag_{\underline{n}}\}$ *is a set of agents.*
- $\mathcal{R} : \mathcal{D} \times Ag \times Ag \rightarrow \mathbb{N}$ *is a function that maps each possible negotiation scenario to a non-negative integer.*

The function $\mathcal{R}$ indicates how often each negotiation scenario is repeated. We will use the notation $R^{d,\underline{i},\underline{j}}$ as a shorthand for $\mathcal{R}(D_d, ag_{\underline{i}}, ag_{\underline{j}})$. So, $R^{d,\underline{i},\underline{j}}$ is the number of times the agents $ag_{\underline{i}}$ and $ag_{\underline{j}}$ will be negotiating (with utility functions u_1 and u_2 respectively) over domain D_d. In general, we would initially choose the number of repetitions to be the same for every scenario. However, it may turn out that some scenarios yield a lot more variance in their outcomes than others, so for those scenarios we might afterwards choose to increase the number of repetitions to get more accurate data.

The goal of running the tournament, is to gather data. Specifically, for each repetition of any given scenario, we obtain two data points, namely the utility values obtained by the two respective agents in that negotiation. Let us use the notation $u_1^{d,\underline{i},\underline{j},r}$ to denote the utility obtained by the *first* agent in the r-th repetition of the scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$. So, it is the utility obtained by $ag_{\underline{i}}$. Similarly, let $u_2^{d,\underline{i},\underline{j},r}$ denote the the utility obtained by the *second* agent in the r-th repetition of the scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$. So, it is the utility obtained by $ag_{\underline{j}}$. For example, $u_1^{7,\underline{3},\underline{4},9}$ represents the utility obtained by agent $ag_{\underline{3}}$ in the ninth repetition of the scenario $(\Pi, D_7, ag_{\underline{3}}, ag_{\underline{4}})$.

Specifically:

$$u_l^{d,\underline{i},\underline{j},r} := \begin{cases} u_l(\omega) & \text{If the } r\text{-th repetition of scenario } (\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}}) \\ & \text{ended with agreement } \omega. \\ rv_l & \text{If the } r\text{-th repetition of scenario } (\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}}) \\ & \text{ended without agreement.} \end{cases}$$

for $l \in \{1, 2\}$, where u_1, u_2, rv_1, and rv_2 are the utility functions and reservation values of domain D_d.

Pay attention to the fact that the numbers $u_1^{d,\underline{i},\underline{j},r}$, $u_2^{d,\underline{i},\underline{j},r}$, $u_1^{d,\underline{j},\underline{i},r}$, and $u_2^{d,\underline{j},\underline{i},r}$ are, in general, all different numbers. For example, suppose we have agent $ag_{\underline{3}}$ and $ag_{\underline{4}}$ negotiating over domain D_7 and that in this domain the two agents are referred to as the 'buyer' and the 'seller' respectively. Then we can distinguish between two different scenarios:

- The scenario $(\Pi, D_7, ag_{\underline{3}}, ag_{\underline{4}})$ in which agent $ag_{\underline{3}}$ is the buyer and $ag_{\underline{4}}$ is the seller.
- The scenario $(\Pi, D_7, ag_{\underline{4}}, ag_{\underline{3}})$ in which agent $ag_{\underline{4}}$ is the buyer and $ag_{\underline{3}}$ is the seller.

If we assume that each of these scenarios is repeated only once, then we obtain four numbers: the two utility values obtained by the two respective agents from the negotiation in the first scenario ($u_1^{7,\underline{3},\underline{4},1}$ and $u_2^{7,\underline{3},\underline{4},1}$), and the two utility values obtained by the two agents from the negotiation in the second scenario ($u_1^{7,\underline{4},\underline{3},1}$ and $u_2^{7,\underline{4},\underline{3},1}$). That is:

- $u_1^{7,\underline{3},\underline{4},1}$ is the utility obtained by agent $\boldsymbol{ag_{\underline{3}}}$ when it acted as the **buyer** in domain D_7 against agent $ag_{\underline{4}}$.
- $u_2^{7,\underline{3},\underline{4},1}$ is the utility obtained by agent $\boldsymbol{ag_{\underline{4}}}$ when it acted as the **seller** in domain D_7 against agent $ag_{\underline{3}}$.
- $u_1^{7,\underline{4},\underline{3},1}$ is the utility obtained by agent $\boldsymbol{ag_{\underline{4}}}$ when it acted as the **buyer** in domain D_7 against agent $ag_{\underline{3}}$.
- $u_2^{7,\underline{4},\underline{3},1}$ is the utility obtained by agent $\boldsymbol{ag_{\underline{3}}}$ when it acted as the **seller** in domain D_7 against agent $ag_{\underline{4}}$.

For any pair of agents $ag_{\underline{i}}$, $ag_{\underline{j}}$ and any domain D_d we can now calculate the average utility $U_{\underline{i}}^{d,\underline{j}}$ obtained by agent $ag_{\underline{i}}$ against opponent $ag_{\underline{j}}$ in domain D_d (i.e. averaged over all repetitions of the scenarios $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$ and $(\Pi, D_d, ag_{\underline{j}}, ag_{\underline{i}})$):

$$U_{\underline{i}}^{d,\underline{j}} \quad := \quad \frac{1}{2R^{d,\underline{i},\underline{j}}} \sum_{r=1}^{R^{d,\underline{i},\underline{j}}} u_1^{d,\underline{i},\underline{j},r} + \frac{1}{2R^{d,\underline{j},\underline{i}}} \sum_{r=1}^{R^{d,\underline{j},\underline{i}}} u_2^{d,\underline{j},\underline{i},r} \tag{6.1}$$

Note that this formula can also be applied to the case that $i = j$.

Then, for each agent $ag_{\underline{i}}$, we can calculate its **tournament score** $U_{\underline{i}}$ by averaging over all domains and all opponents (including itself):

$$U_{\underline{i}} \quad := \quad \frac{1}{|\mathcal{D}| \cdot |Ag|} \sum_{d=1}^{|\mathcal{D}|} \sum_{\underline{j}=1}^{|Ag|} U_{\underline{i}}^{d,\underline{j}} \tag{6.2}$$

We can now rank all agents in the tournament based on their tournament scores. The agent with the highest tournament score can be considered the best.

For example, suppose that we have implemented an agent called *Il-Padrino* and we have tested it against 3 benchmark agents, called *CrazyAgent*, *RandomAgent*, and *MegaBarter3000*. Then we can display the results of a tournament between these agents in a table such as Table 6.1.

Agent	**Tournament Score** ($\underline{U_i}$)
MegaBarter3000	0.729
IlPadrino	0.665
CrazyAgent	0.604
RandomAgent	0.488

Table 6.1: Results of a fictional tournament between four fictional agents.

Exercise 14. Run a Tournament. Implement code that runs a tournament among the agents that you implemented in the previous exercises (if you didn't do those exercises you can use the agents from the folder 'Solutions to Exercises'). Also, make sure the tournament involves both example domains that are given with the framework, as well as the negotiation domain that you created yourself in Exercise 2 (if you did that exercise).
Hint: the NegoSimulator framework already contains a function to run a single negotiation. So, all you have to do is create a loop that iterates over all possible negotiation scenarios, and that calls this function in each iteration.
Make sure the results of all the individual negotiations in this tournament are stored in a text file (or a .csv file) on your hard disk. Each line should correspond to the result of one negotiation, and should contain the following information:

- The names of the two agents in that negotiation.
- The name of the domain used in that negotiation.
- Whether or not the agents came to an agreement.
- The utility values obtained by the two respective agents (their reservation values in case there was no agreement).

For example, this text file could look as follows:

```
Agent 1;    Agent2;           Domain;   Agree?;   util1;   util2;
IlPadrino;  IlPadrino;        CarSale;  YES;      0.8;     0.54;
IlPadrino;  CrazyAgent;       CarSale;  NO;       0.0;     0.0;
IlPadrino;  RandomAgent;      CarSale;  NO;       0.0;     0.0;
IlPadrino;  MegaBarter3000;   CarSale;  YES;      0.74;    0.67;
...
```

Exercise 15. Calculate Tournament Scores. Implement a program that does the following:

1. Read the text file from Exercise 14.
2. Based on the contents of that file, calculate the tournament score of each agent.
3. Display a table such as Table 6.1, showing the names and scores of the agents, sorted in order of decreasing tournament score.

6.1.2 Storing your Data

Note that in Exercises 14 and 15 we asked you to implement two *separate* programs. One for running the tournament and one for calculating the tournament scores. There are very good reasons for this.

It is a common mistake among beginning researchers to just write one single program that runs the tournament and then immediately calculates the tournament scores, without storing the results of the individual negotiations. This is a big mistake, because after running your first tournament it often turns out that you need to increase the number of repetitions, or that you need to add more agents or domains to your experiment, or that you may want to analyze one particular aspect of the data that you did not think of before. If you didn't store the individual results, it means that you would then have to run the entire tournament all over again, which may cost a lot of time. On the other hand, if you did store the results, you can simply run a number of extra negotiations and add the results of those negotiations to the data that you already had.

6.1.3 Agreement Rate and Utility-Under-Agreement

While the tournament score $U_{\underline{i}}$ can be used to determine *which* agents performed well and which performed poorly, we typically also want to know *why* some agent did or did not perform well. Recall from Chapter 3 that negotiating well comes down to striking the right balance between being hardheaded and being conceding. An agent that is *too* hardheaded will not make many agreements, while an agent that is *too* conceding will only make agreements with low utility. To measure whether our agent is too hardheaded or too conceding, we can use the following two quantities:

- **Agreement Rate**: the percentage of all negotiations that our agent was involved in that ended in agreement.

- **Utility-under-Agreement**: the average utility obtained by our agent among only those negotiations that ended in agreement.

A hardheaded agent would typically score a low agreement rate, but a high utility-under-agreement. Reversely, a conceding agent would typically score a high agreement rate, but a low utility-under-agreement.

Let us make this precise. Let $\mathbb{1}_{agr(d,\underline{i},\underline{j},r)}$ be the 'indicator function' that has value 1 if the r-th repetition of scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$ ended with agreement, and 0 otherwise.

$$\mathbb{1}_{agr(d,\underline{i},\underline{j},r)} := \begin{cases} 1 & \text{If the } r\text{-th repetition of scenario } (\Pi, D_d, ag_{\underline{i}}, ag_j) \\ & \text{ended with agreement.} \\ 0 & \text{If the } r\text{-th repetition of scnario } (\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}}) \\ & \text{ended without agreement.} \end{cases}$$

Furthermore, let $NA^{d,\underline{i},\underline{j}}$ represent the number of times a negotiation in the scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$ ended with agreement:

$$NA^{d,\underline{i},\underline{j}} := \sum_{r=1}^{R^{d,\underline{i},\underline{j}}} \mathbb{1}_{agr(d,\underline{i},\underline{j},r)}$$

Then we can calculate the agreement rate of agent $ag_{\underline{i}}$, for a given opponent and a given domain as:

$$AR_{\underline{i}}^{d,\underline{j}} := \frac{1}{2} \cdot \frac{NA^{d,\underline{i},\underline{j}}}{R^{d,\underline{i},\underline{j}}} + \frac{1}{2} \cdot \frac{NA^{d,\underline{j},\underline{i}}}{R^{d,\underline{j},\underline{i}}}$$

and the agreement rate of agent $ag_{\underline{i}}$ for the entire tournament is obtained by averaging this quantity over all domains and opponents:

$$AR_{\underline{i}} := \frac{1}{|\mathcal{D}| \cdot |Ag|} \sum_{d=1}^{|\mathcal{D}|} \sum_{\underline{j}=1}^{|Ag|} AR_{\underline{i}}^{d,\underline{j}}$$

This quantity will of course always be a number between 0 and 1, but it is custom to present it as a percentage, so 0.65 would be presented as 65%.

Similarly, we can define the average *utility-under-agreement* $UA_{\underline{i}}$ using the following two equations:

$$UA_{\underline{i}}^{d,\underline{j}} := \frac{1}{2 \cdot NA^{d,\underline{i},\underline{j}}} \sum_{r=1}^{R^{d,\underline{i},\underline{j}}} u_1^{d,\underline{i},\underline{j},r} \mathbb{1}_{agr(d,\underline{i},\underline{j},r)} + \frac{1}{2 \cdot NA^{d,\underline{j},\underline{i}}} \sum_{r=1}^{R^{d,\underline{j},\underline{i}}} u_1^{d,\underline{j},\underline{i},r} \mathbb{1}_{agr(d,\underline{i},\underline{j},r)}$$

$$UA_{\underline{i}} \quad := \quad \frac{1}{|\mathcal{D}| \cdot |Ag|} \sum_{d=1}^{|\mathcal{D}|} \sum_{\underline{j}=1}^{|Ag|} UA_{\underline{i}}^{d,\underline{j}}$$

Now, we can present the results of our experiment including these values, such as in Table 6.2. Note that this table is much more informative than Table 6.1. While the first table only showed us that our agent 'IlPadrino' was second-best, after MegaBarter3000, we can now also see *why* it performed worse than MegaBarter3000. To see this, let us analyze each agent one by one.

- **IlPadrino**: We see that IlPadrino scores the lowest utility-under-agreement of all agents, but at the same time it scores the highest agreement rate. This clearly indicates that it is too conceding.
- **CrazyAgent**: On the other hand, CrazyAgent suffers from the opposite problem. It has a very low agreement rate, but a very high utility-under-agreement, indicating that it is a very hardheaded agent.
- **MegaBarter3000**: We see that MegaBarter3000 scores neither the highest nor the lowest utility-under-agreement, and the same holds for its agreement rate. However, it does score the highest overall tournament score, which is the score that actually matters. Therefore, we can say that it is the best of the four agents, because it strikes the best balance between being hardheaded and being conceding.
- **RandomAgent**: Finally, we note that RandomAgent scores low on all quantities. This means that it suffers from more fundamental problems rather than just being too hardheaded or too conceding. One possible explanation is that it mainly proposes offers that are really bad for both himself and his opponent at the same time.

Exercise 16. Calculate Agreement Rates and Utility-under-Agreement. Modify your code of Exercise 15 so that it also calculates the $UA_{\underline{i}}$ and $AR_{\underline{i}}$ of every agent, and so that it displays a table such as Table 6.2.

6.1.4 Abuse of $UA_{\underline{i}}$ and $AR_{\underline{i}}$

We should stress the fact that the utility-under-agreement $UA_{\underline{i}}$ and agreement rate $AR_{\underline{i}}$ should only be used for *diagnostic* purposes. That is, they should only be used to answer the question *why* a particular agent did or

Agent	**Tournament Score** ($U_{\underline{i}}$)	**Utility-under-Agreement** ($UA_{\underline{i}}$)	**Agreement Rate** ($AR_{\underline{i}}$)
MegaBarter3000	0.729	0.81	90%
IlPadrino	0.665	0.70	95%
CrazyAgent	0.604	0.93	65%
RandomAgent	0.488	0.75	65%

Table 6.2: Results of a fictional tournament between four fictional agents. This time presented together with the Utility-under-Agreement and the Agreement Rate of each agent.

did not perform well. These quantities, however, should *not* be used to determine *whether* or not it performed well. Instead, the tournament score is the only of the three measures that should be used for that.

In fact, it is a common mistake among beginning researchers to argue that they implemented a strong agent based only on the observation that it scored a high utility-under-agreement in their experiments. It is, however, easy to see that a high value of $UA_{\underline{i}}$ by itself does not say anything about the quality of the agent, because implementing an agent that scores high $UA_{\underline{i}}$ is entirely trivial. For example, we can simply implement an agent that only proposes or accepts any offers with utility value $u_1(\omega) \geq 0.99$ (assuming the maximum utility is 1.0) and that rejects any other offers. Clearly, when such an agent comes to an agreement, it will always be one with a very high utility for himself, and this agent would be guaranteed to achieve a $UA_{\underline{i}}$ of at least 0.99. But that is completely useless because this agent would probably almost never make any agreements at all, and therefore would likely end up with a very low average utility overall.

For the same reason, we cannot use the agreement rate as a performance measure for our agent. After all, it is also trivial to implement an agent that simply always accepts every proposal, and which is therefore guaranteed to achieve an agreement rate of 100%.

Observation. *The agreement rate and the utility-under-agreement should never be used as measures of performance of an agent. They should only be used as an aid to understand why the agent performed well or not.*

6.1.5 The Importance of Self-Play

One aspect that is often overlooked in experiments, is the importance of agents negotiating not only against other agents, but also against *themselves*. Many experiments described in the literature do not involve such self-play (that is, they set $R^{d,\underline{i},\underline{j}} = 0$ whenever $\underline{i} = \underline{j}$). However, we argue that it is in fact extremely important that any new negotiation strategy is not only tested against a number of benchmark agents, but also against itself.

The reasoning behind this is as follows. Imagine a world in which many people use negotiation algorithms to do their negotiations for them. Obviously, those people would want to use the best such algorithms available. Now, suppose that we have invented a new algorithm that is very strong against most other existing algorithms, but that performs poorly when pitted against itself. Then initially, when we just start using our new algorithm, it may obtain very good results. However, sooner or later other people will also discover our algorithm and will start using it as well. But then it also becomes more and more likely that our agent will encounter opponents that use exactly the same algorithm, or a very similar one. And since our algorithm performs poorly against such opponents, eventually, its results will deteriorate.

Observation. *The stronger a negotiation algorithm performs, the more likely it is that, in a real-world setting, it would encounter itself as its opponent. Therefore, any 'strong' algorithm must necessarily also be strong against itself.*

A good example of a strategy that may perform well against other agents, but not against itself, is a very hardheaded time-based agent.

Another important reason for self-play, is that it is a requirement for a so-called *empirical game-theoretical analysis*, which we will discuss later on.

6.1.6 The Importance of Benchmark vs. Benchmark Negotiations

Another mistake that beginning researchers sometimes tend to make, is that, when testing a new agent, they only run negotiations between their new agent and some existing benchmark agents, without running any negotiations between the benchmark agents themselves. We will here argue that it is extremely important that you also include the results of mutual negotiations among the benchmark agents in your data.

The reason for this is the same as the reason why we argued against the use of the utility-under-agreement as a performance measure, in Section

	$U_{\underline{i}}$	$UA_{\underline{i}}$	$AR_{\underline{i}}$
$ag_{\underline{2}}$	0.410	0.75	55%
$ag_{\underline{3}}$	0.401	0.73	55%
$ag_{\underline{1}}$	0.093	0.93	10%

Table 6.3: Results of a fictional tournament in which $ag_{\underline{1}}$ is a very hardheaded agent.

6.1.4. To see this, suppose again that we implement a very hardheaded agent that only proposes or accepts any offers with utility value $u_1(\omega) \geq 0.90$ (assuming the maximum utility is 1.0) and that rejects any other offers. Clearly, whenever this agent makes an agreement, it will be one with very high utility for itself, and therefore most likely with very low utility for its opponent. However, it is then also very likely that the negotiations will fail very often, so the tournament score of our agent would actually be very low.

For example, say that when our agent $ag_{\underline{1}}$ negotiates against $ag_{\underline{2}}$, then on average, if there is an agreement, $ag_{\underline{1}}$ scores 0.92 and $ag_{\underline{2}}$ scores 0.2. However, they only come to an agreement in 10% of all negotiations, so their overall average utilities would be 0.092 and 0.02 (assuming they both have a reservation value of 0). Similarly, when our agent $ag_{\underline{1}}$ negotiates against ag_3, then on average, if there is an agreement, $ag_{\underline{1}}$ scores 0.94 and $ag_{\underline{2}}$ scores 0.22. However, they only come to an agreement in 10% of all negotiations, so their overall average utilities would be 0.094 and 0.022.

Now, if we stop here, we might falsely conclude that $ag_{\underline{1}}$ is the best among the three agents, since it clearly scored higher utilities than the other two. However, if we also let the other two agents negotiate against each other, they might reach an agreement in 100% of the negotiations, and they might on average achieve utility values of 0.8 and 0.78 respectively. If we now calculate the tournament scores of each agent, we get the results as displayed in Table 6.3. We see that our agent scores, by far, the lowest utility. Now, if you look at this table, ask yourself: which agent would you choose to do your negotiations?

6.1.7 Standard Error

We have argued above that we can assess the strength of an agent by means of its tournament score $U_{\underline{i}}$. However, just like in any other experimental science, we have to keep in mind that the value of $U_{\underline{i}}$ that we measure may be heavily affected by statistical noise, especially if the agents are, to some extent, non-deterministic. This means that if we repeat the entire tournament a second time, then the agents obtain a different tournament score than in the first tournament. It is therefore extremely important to also measure how strongly $U_{\underline{i}}$ is affected by statistical fluctuations. A commonly used quantity to measure this, is the *standard error*, which we will discuss in this section.

Before we continue, we should first note that we actually have to deal with two different types of statistical noise:

1. Noise due to the variation of opponents and domains. That is, our agent may perform very well against one specific opponent, or on one specific domain, but may perform poorly against another opponent or on another domain.
2. Purely random noise due to the non-determinism of one or both of the agents. That is, even if we repeat exactly the same scenario, our agent may still achieve different utility values in each negotiation if some of its actions, or some of its opponent's actions, are randomized.

We will first discuss the second type of noise. That is, in Section 6.1.7.2 we will assume we are measuring the performance of our agent only on one specific scenario. Next, in Section 6.1.7.3 we will show how to generalize this to a *fixed* set of multiple scenarios (meaning we are still only including the second type of noise). And then in Section 6.1.7.4 we will generalize this further to include possibility that we may use a different set of scenarios every time we repeat the experiment, hence including the first type of noise.

In the rest of this section we will present several mathematical equations. However, we will not go into a full derivation of those equations, because that would require a much more in-depth study of the topic of statistics, which is a vast field in itself. For more details we therefore refer to the many textbooks that have been written on this topic, such as [23] or [26].

6.1.7.1 Random Variables

In statistics, any process that returns random outcomes is called a '*random variable*'. A typical example of a random variable is a standard six-sided die, which can return any integer between 1 and 6, each with an equal probability of $\frac{1}{6}$. Every time a random variable returns a new outcome (e.g. every time we throw a die), we say we **draw an observation** from that variable.

Formally, we can define a **finite real-valued random variable** $\mathcal{X}$ as a pair $(S_{\mathcal{X}}, P_{\mathcal{X}})$ where $S_{\mathcal{X}}$ is some finite set of real numbers $S_{\mathcal{X}} =$

$\{x_1, x_2, \ldots, x_K\} \subset \mathbb{R}$, and $P_{\mathcal{X}}$ is a probability distribution over this set. That is, $P_{\mathcal{X}} : S_{\mathcal{X}} \rightarrow \mathbb{R}$ such that:

$$\forall x \in S_{\mathcal{X}}: \quad P_{\mathcal{X}}(x) \geq 0 \quad \text{and} \quad \sum_{x \in S_{\mathcal{X}}} P_{\mathcal{X}}(x) = 1$$

Then, the **mean** of $\mathcal{X}$, denoted $\mu_{\mathcal{X}}$, is defined as:

$$\mu_{\mathcal{X}} \quad := \quad \sum_{x \in S_{\mathcal{X}}} P_{\mathcal{X}}(x) \cdot x$$

the **variance** of $\mathcal{X}$, denoted $Var_{\mathcal{X}}$, is defined as:

$$Var_{\mathcal{X}} \quad := \quad \sum_{x \in S_{\mathcal{X}}} P_{\mathcal{X}}(x) \cdot (\mu_{\mathcal{X}} - x)^2$$

and the **standard deviation** of $\mathcal{X}$, denoted $\sigma_{\mathcal{X}}$, is defined as:

$$\sigma_{\mathcal{X}} \quad := \quad \sqrt{Var_{\mathcal{X}}}$$

In the context of a negotiation tournament, for any negotiation over some negotiation scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$ we can see the utility $u_1^{d,\underline{i},\underline{j},r}$ obtained by agent $ag_{\underline{i}}$, as an observation from a random variable. Similarly, the utility $u_2^{d,\underline{i},\underline{j},r}$ obtained by the other agent $ag_{\underline{j}}$ can also be seen as an observation drawn from a (different) random variable. Therefore, any negotiation scenario is associated with two random variables, which we will denote as $\mathcal{X}_1^{d,\underline{i},\underline{j}}$ and $\mathcal{X}_2^{d,\underline{i},\underline{j}}$.

In general, whenever we have two real-valued random variables, $\mathcal{X}_1$ and $\mathcal{X}_2$, then we can define a new random variable, denoted as $\mathcal{X}_1 + \mathcal{X}_2$, which corresponds to the process that we draw one observation from $\mathcal{X}_1$ and one observation from $\mathcal{X}_2$, and then return the sum of the two observations. For example, if $\mathcal{X}_1$ and $\mathcal{X}_2$ each represent an ordinary 6-sided die, then an observation of the variable $\mathcal{X}_1 + \mathcal{X}_2$ represents the process of throwing two dice at the same time, and recording the sum of the two numbers shown by the two dice. This means, an observation of the variable $\mathcal{X}_1 + \mathcal{X}_2$ could be any integer between 2 and 12, so we have $S_{\mathcal{X}_1+\mathcal{X}_2} = \{2, 3, 4, \ldots, 11, 12\}$

Similarly, if $\mathcal{X}$ is a real-valued random variable and c is any arbitrary real number, then we can define a new random variable $c\mathcal{X}$. An observation from $c\mathcal{X}$ is defined as an observation from $\mathcal{X}$, multiplied by c. For example, if $c = 2$ and $\mathcal{X}$ represents a 6-sided die, then $c\mathcal{X}$ can be any *even* number between 2 and 12, so we have $S_{2\mathcal{X}} = \{2, 4, 6, 8, 10, 12\}$

Note, therefore, that $\mathcal{X} + \mathcal{X}$ is *not* the same as $2\mathcal{X}$, because in the first case we are drawing *two* observations from $\mathcal{X}$, which may be different from each other, while in the second case we just draw one observation from $\mathcal{X}$ which is then multiplied by 2. For example, if $\mathcal{X}$ is a 6-sided die, then in the first case we could throw a 3 and then a 6, so our observation is the number $3 + 6 = 9$, while in the second case we can only observe even numbers.

6.1.7.2 Standard Error for a Single Scenario

Now, suppose that have a random variable $\mathcal{X}$, but we do not know its probability distribution $P_{\mathcal{X}}$. Furthermore, we also don't know its mean, variance or standard deviation. In that case, we can estimate these quantities, by drawing a sequence of observations $(o_1, o_2, \ldots, o_R)$ from $\mathcal{X}$, where each o_i is an element of $S_{\mathcal{X}}$ that was randomly selected (with replacement) with probability $P_{\mathcal{X}}(o_i)$. Such a sequence of observations is called a **sample** and we will denote its size as R.

We can then calculate an *estimated* mean $\hat{\mu}_{\mathcal{X}}$, *estimated* variance $\hat{Var}_{\mathcal{X}}$ and *estimated* standard deviation $\hat{\sigma}_{\mathcal{X}}$ of $\mathcal{X}$ as follows:

$$\hat{\mu}_{\mathcal{X}} = \frac{1}{R}\sum_{i=1}^{R} o_i \tag{6.3}$$

$$\hat{Var}_{\mathcal{X}} = \frac{1}{R-1}\sum_{i=1}^{R}(\hat{\mu}_{\mathcal{X}} - o_i)^2 \tag{6.4}$$

$$\hat{\sigma}_{\mathcal{X}} = \sqrt{\hat{Var}_{\mathcal{X}}} = \frac{\sqrt{\sum_{i=1}^{R}(\hat{\mu}_{\mathcal{X}} - o_i)^2}}{\sqrt{R-1}} \tag{6.5}$$

The estimated variance $\hat{Var}_{\mathcal{X}}$ and standard deviation $\hat{\sigma}_{\mathcal{X}}$ are also known as the **sample variance** and **sample standard deviation**. On the other hand, the actual variance $Var_{\mathcal{X}}$ and standard deviation $\sigma_{\mathcal{X}}$ are known as the **population variance** and **population standard deviation**.

Note that the expression for the sample variance has the number $R - 1$ in the denominator instead of R. This is a correction for the fact that the expression does not use the true mean $\mu_{\mathcal{X}}$, but the estimated mean $\hat{\mu}_{\mathcal{X}}$. For a mathematical derivation of this fact we refer to any standard text book on statistics.

Of course, the estimated mean $\hat{\mu}_{\mathcal{X}}$ is only an approximation of the real value of $\mu_{\mathcal{X}}$ and will typically not be exactly equal to $\mu_{\mathcal{X}}$. In fact, every time we draw a new sample from $\mathcal{X}$ and re-calculate $\hat{\mu}_{\mathcal{X}}$ we will likely obtain a different value. Now, a key insight, is that the value $\hat{\mu}_{\mathcal{X}}$ that we obtain from these calculations can itself also be seen as an observation from a random variable. Let us denote this random variable by $\mathcal{Y}$. That is:

$$\mathcal{Y} = \frac{1}{R}\sum_{r=1}^{R} \mathcal{X}$$

So, a *single* observation from $\mathcal{Y}$ is, by definition, obtained from drawing R observations from $\mathcal{X}$ and then calculating their average.

The point of this, is that one can mathematically prove that the mean of $\mathcal{Y}$ equals the mean of $\mathcal{X}$ (i.e. $\mu_{\mathcal{X}} = \mu_{\mathcal{Y}}$), but the standard deviation of $\mathcal{Y}$ is smaller than that of $\mathcal{X}$. In particular: $\sigma_{\mathcal{Y}} = \frac{\sigma_{\mathcal{X}}}{\sqrt{R}}$. The random variable $\mathcal{Y}$ is therefore said to be an *estimator* for $\mu_{\mathcal{X}}$.

Note that the larger we choose the sample size R the smaller $\sigma_{\mathcal{Y}}$ and therefore the more likely it will be that $\hat{\mu}_{\mathcal{X}}$ will be close to $\mu_{\mathcal{X}}$. The standard deviation of $\mathcal{Y}$ can therefore be seen as a measure of accuracy of our estimation of $\mu_{\mathcal{X}}$ and is known as the **standard error** on our estimation of $\mu_{\mathcal{X}}$:

$$se_{\mu_{\mathcal{X}}} \quad := \quad \sigma_{\mathcal{Y}} \quad = \quad \frac{\sigma_{\mathcal{X}}}{\sqrt{R}}$$

Roughly speaking, the interpretation of the standard error is that there is a 95% probability that the true mean $\mu_{\mathcal{X}}$ lies somewhere between $\hat{\mu}_{\mathcal{X}} - 2 \cdot se_{\mu_{\mathcal{X}}}$ and $\hat{\mu}_{\mathcal{X}} + 2 \cdot se_{\mu_{\mathcal{X}}}$. For example, if $\hat{\mu}_{\mathcal{X}} = 10$ and $se_{\mu_{\mathcal{X}}} = 1$ then there is a 95% probability that $\mu_{\mathcal{X}}$ lies in the interval [8 , 12].

In practice, however, we often can't directly calculate $se_{\mu_{\mathcal{X}}}$ because we do not know the exact value of $\sigma_{\mathcal{X}}$. Instead, we can only calculate the *estimated* standard error for $\mu_{\mathcal{X}}$, denoted $\hat{se}_{\mu_{\mathcal{X}}}$, by using the estimated standard deviation $\hat{\sigma}_{\mathcal{X}}$ as defined by Equation (6.5):

$$\hat{se}_{\mu_{\mathcal{X}}} = \frac{\hat{\sigma}_{\mathcal{X}}}{\sqrt{R}}$$

It is important to understand that, for any given random variable $\mathcal{X}$, its standard deviation $\sigma_{\mathcal{X}}$ is fixed. It is a property of that random variable, so we can *estimate* it, but it is not something that we can *change*. On the other hand, the standard error on $\mu_{\mathcal{X}}$ is not a property of $\mathcal{X}$, but rather, it is a property of our experiment. It is a measure of how accurately we have estimated the mean of $\mathcal{X}$, which depends on how many observations we have drawn from $\mathcal{X}$. The larger the number of observations, the smaller the standard error. This means we can make the standard error as small as we like, as long as we draw enough observations from $\mathcal{X}$.

Another very important thing to remember is not to confuse the standard *error* on $\mu_{\mathcal{X}}$ with the standard *deviation* on $\mathcal{X}$. This is a very common mistake. Many beginning authors erroneously report their estimated mean $\hat{\mu}_{\mathcal{X}}$ of $\mathcal{X}$ together with the standard deviation $\sigma_{\mathcal{X}}$ of $\mathcal{X}$ as a measure of accuracy. However, this is incorrect because, as mentioned above, this standard deviation is a fixed value and therefore does not say anything about the accuracy of the measurement. Instead, they were supposed to mention the standard *error*, which is the standard deviation of $\mathcal{Y}$, and which decreases with the number of observations.

6.1.7.3 Standard Error for Multiple Scenarios

As mentioned above, every negotiation scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$ is associated with two random variables $\mathcal{X}_1^{d,\underline{i},\underline{j}}$ and $\mathcal{X}_2^{d,\underline{i},\underline{j}}$. However, whenever we run a tournament we are not just interested in one single negotiation scenario, but rather we want to calculate the average utility of our agent over many different scenarios. So, if our agent is $ag_{\underline{1}}$, then we are interested in all random variables of the form $\mathcal{X}_1^{d,\underline{1},\underline{j}}$ or $\mathcal{X}_2^{d,\underline{j},\underline{1}}$.

Specifically, if there are m domains and n agents (including our own agent) then the calculation of our agent's tournament score would involve a total $2mn$ of these variables. To see this, note that there are $n-1$ opponents, and for each domain and each opponent there are 2 possible scenarios (one in which our agent has utility function u_1 and one in which it has utility function u_2), yielding $2m(n-1)$ possible scenarios involving our agent. However, on top of that there are also m scenarios (one for each domain) in which our agent negotiates against itself. In such scenarios there are two outcomes for our agent (one for the copy of our agent that had utility function u_1 and one for the copy that had utility function u_2). Yielding a total of $2m \cdot (n-1) + 2m = 2mn$ random variables for the entire tournament.

For example, if there are 2 domains and 3 agents, and we want to calculate the tournament score of agent $ag_{\underline{1}}$ then this involves the following 12 random variables:

$$\mathcal{X}_1^{1,\underline{1},\underline{1}}, \mathcal{X}_2^{1,\underline{1},\underline{1}}, \mathcal{X}_1^{1,\underline{1},\underline{2}}, \mathcal{X}_1^{1,\underline{1},\underline{3}}, \mathcal{X}_2^{1,\underline{2},\underline{1}}, \mathcal{X}_2^{1,\underline{3},\underline{1}}$$

$$\mathcal{X}_1^{2,\underline{1},\underline{1}}, \mathcal{X}_2^{2,\underline{1},\underline{1}}, \mathcal{X}_1^{2,\underline{1},\underline{2}}, \mathcal{X}_1^{2,\underline{1},\underline{3}}, \mathcal{X}_2^{2,\underline{2},\underline{1}}, \mathcal{X}_2^{2,\underline{3},\underline{1}}$$

Note that these are indeed all the variables of the form $\mathcal{X}_1^{d,\underline{1},\underline{j}}$ or $\mathcal{X}_2^{d,\underline{j},\underline{1}}$, for some domain D_d and opponent $ag_{\underline{j}}$, and that they all return utility values for agent $ag_{\underline{1}}$.

In the rest of this section we will assume we are calculating the tournament score of some arbitrary agent $ag_{\underline{i}}$, and to simplify the notation we will just denote the corresponding variables as:

$$\mathcal{X}_{\underline{i},1}, \quad \mathcal{X}_{\underline{i},2}, \quad \ldots, \quad \mathcal{X}_{\underline{i},k}$$

where $k = 2mn$. Each random variable will, in general, have a different mean and a different standard deviation, so we have k different unknown means $\mu_{\underline{i},1}, \mu_{\underline{i},2}, \ldots, \mu_{\underline{i},k}$ and k different unknown standard deviations $\sigma_{\underline{i},1}, \sigma_{\underline{i},2}, \ldots, \sigma_{\underline{i},k}$.

Now, when we run an experiment and we calculate the tournament score of agent $ag_{\underline{i}}$, we first calculate for each scenario the average utility the agent achieved on that scenario, and then we calculate the average that over all scenarios. This can be modeled by means of the following two random variables:

$$\mathcal{Y}_{\underline{i},s} := \frac{1}{R_s} \sum_{r=1}^{R_s} \mathcal{X}_{\underline{i},s} \tag{6.6}$$

$$\mathcal{Z}_{\underline{i}} := \frac{1}{k} \sum_{s=1}^{k} \mathcal{Y}_{\underline{i},s} \tag{6.7}$$

Here, an observation of $\mathcal{Y}_{\underline{i},s}$ represents the average utility obtained by agent $ag_{\underline{i}}$ in the scenario corresponding to variable $\mathcal{X}_{\underline{i},s}$, and a single observation from $\mathcal{Z}_{\underline{i}}$ represents the average utility obtained by $ag_{\underline{i}}$ in the entire tournament. In other words, the tournament score $U_{\underline{i}}$ of agent $ag_{\underline{i}}$ is, by definition, a single observation from the variable $\mathcal{Z}_{\underline{i}}$.

Note that the means of $\mathcal{X}_{\underline{i},s}$ and $\mathcal{Y}_{\underline{i},s}$ are equal:

$$\mu_{\mathcal{X}_{\underline{i},s}} = \mu_{\mathcal{Y}_{\underline{i},s}}$$

we will therefore use the notation $\mu_{\underline{i},s}$ as a shorthand for $\mu_{\mathcal{X}_{\underline{i},s}}$, and $\mu_{\mathcal{Y}_{\underline{i},s}}$. Furthermore, the mean $\mu_{\mathcal{Z}_{\underline{i}}}$ of $\mathcal{Z}_{\underline{i}}$ is given by:

$$\mu_{\mathcal{Z}_{\underline{i}}} = \frac{1}{k} \sum_{s=1}^{k} \mu_{\underline{i},s} \tag{6.8}$$

Of course, every time we repeat the experiment, we may obtain a slightly different tournament score for $ag_{\underline{i}}$, but the higher the number of repetitions R_s for each scenario, the closer the observed tournament scores will be to the mean of $\mathcal{Z}_{\underline{i}}$, which can therefore be seen as an measure of the true strength of the agent.

Observation. *The mean of $\mathcal{Z}_{\underline{i}}$ represents the* <u>*true*</u> *strength of agent $ag_{\underline{i}}$, while the tournament score $U_{\underline{i}}$ is only a noisy approximation of the agent's strength.*

This means that the (estimated) standard deviation of $\mathcal{Z}_{\underline{i}}$ can be used as a measure of how accurately we have estimated the true strength of $ag_{\underline{i}}$. This is, therefore, the (estimated) standard error of our experiment.

$$\hat{se}_{\mu_{\mathcal{Z}_{\underline{i}}}} \quad := \quad \hat{\sigma}_{\mathcal{Z}_{\underline{i}}} \quad = \quad \sqrt{\hat{Var}_{\mathcal{Z}_{\underline{i}}}}$$

So, to calculate the standard error we need to estimate $Var_{\mathcal{Z}_{\underline{i}}}$. To do this, we first estimate the variance of each individual variable $\mathcal{X}_{\underline{i},s}$, using Eq. (6.4). Then, the estimated variance of $\mathcal{Y}_{\underline{i},s}$ is given by $\frac{1}{R_s} \cdot \hat{Var}_{\mathcal{X}_{\underline{i},s}}$, so we get:

$$\hat{Var}_{\mathcal{Y}_{\underline{i},s}} \quad = \quad \frac{1}{R_s \cdot (R_s - 1)} \sum_{r=1}^{R_s} (\hat{\mu}_{\underline{i},s} - u_{\underline{i},s,r})^2$$

where each $u_{\underline{i},s,r}$ is the r-th observation from variable $\mathcal{X}_{\underline{i},s}$, and $\hat{\mu}_{\underline{i},s}$ is calculated as:

$$\hat{\mu}_{\underline{i},s} = \frac{1}{R_s} \sum_{r=1}^{R_s} u_{\underline{i},s,r} \tag{6.9}$$

Note that we are now using the symbol u for observations from $\mathcal{X}_{\underline{i},s}$, to stress the fact that they are indeed utility values obtained by our agent in the respective negotiations it is participating in.

Next, it can be shown mathematically that, if all random variables are mutually independent, then the estimated variance of $\mathcal{Z}_{\underline{i}}$ can then be calculated as:

$$\hat{Var}_{\mathcal{Z}_{\underline{i}}} \quad = \quad \frac{1}{k^2} \sum_{s=1}^{k} \hat{Var}_{\mathcal{Y}_{\underline{i},s}} \tag{6.10}$$

$$\quad = \quad \frac{1}{k^2} \sum_{s=1}^{k} \frac{1}{R_s \cdot (R_s - 1)} \sum_{r=1}^{R_s} (\hat{\mu}_{\underline{i},s} - u_{\underline{i},s,r})^2 \tag{6.11}$$

Unfortunately, however, the assumption that all variables $\mathcal{X}_k^{d,\underline{i},\underline{j}}$ are mutually independent does not hold in our context. Actually, it does hold for most variables, but not for those that correspond to the self-play scenarios. Specifically, for any domain D_d and any agent $ag_{\underline{i}}$, the variables $\mathcal{X}_1^{d,\underline{i},\underline{i}}$ and $\mathcal{X}_2^{d,\underline{i},\underline{i}}$ are not independent from each other, because their values are drawn from the same negotiation scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{i}})$. To correct for this, we need to add, for every domain D_d, a term involving the *covariance* between these two variables. This yields the following expression:

$$\hat{Var}_{\mathcal{Z}_{\underline{i}}} = \frac{1}{k^2}\Big(\sum_{s=1}^{k} \frac{1}{R_s \cdot (R_s - 1)} \sum_{r=1}^{R_s} (\hat{\mu}_{\underline{i},s} - u_{\underline{i},s,r})^2 \quad + \\ \frac{2}{R^{d,\underline{i},\underline{i}} \cdot (R^{d,\underline{i},\underline{i}} - 1)} \sum_{d=1}^{|\mathcal{D}|} \sum_{r=1}^{R^{d,\underline{i},\underline{i}}} (\hat{\mu}_1^{d,\underline{i},\underline{i}} - u_1^{d,\underline{i},\underline{i},r})(\hat{\mu}_2^{d,\underline{i},\underline{i}} - u_2^{d,\underline{i},\underline{i},r})\Big) \quad (6.12)$$

where $R^{d,\underline{i},\underline{i}}$ denotes the number of repetitions of the self-play scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{i}})$.

If we take the square root of this expression, then we obtain the estimated standard error $\hat{\sigma}_{\mathcal{Z}_{\underline{i}}}$, which is a measure of how close the agent's tournament score $U_{\underline{i}}$ is to the true mean $\mu_{\mathcal{Z}_{\underline{i}}}$. If $\hat{\sigma}_{\mathcal{Z}_{\underline{i}}}$ is very low, it means that if we repeat the entire tournament again, then we will probably obtain a tournament score that is very close to the one we obtained the first time.

However, there is still a problem, namely that our calculations above only include the second type of noise: the noise due to non-determinism in the behavior of the agents. This would be fine, if we were only interested in the performance of our agent *on one specific set of scenarios.* In reality, however, we are normally interested in the performance of our agent, *in general.* The set of domains that we have used for our experiments should be seen as just a randomly selected subset from the set of *all* possible negotiation domains that we are interested in, and similarly, the set of opponents used in our experiments should only be considered as a randomly selected subset of all the possible opponents our agent might encounter in reality. We therefore still need to adapt our calculation of the standard error to include the first type of noise.

6.1.7.4 Standard Error with Between-Scenario Noise

In order to include the noise caused by the fact that our agent may behave differently in different scenarios, we can model our calculation of the tournament score $U_{\underline{i}}$ as a two-step process.

That is, we can imagine that there exists a very large set of possible scenarios, Sc. This set may contain billions of possible scenarios, or may even be infinite, so we cannot test our agent on all of them. Therefore, the first step of our experiment consists of randomly choosing some limited subset $\{sc_1, sc_2, sc_3, \dots\}$ of scenarios from Sc. Then, in the second step, we run a number of negotiations for each such scenario sc_s.

Now, when we calculate the standard error, we aim to answer the question how likely it would be that we would find the same tournament score for our agent if we repeated the *entire* experiment, including the first step in which we randomly selected the negotiation scenarios.

As before, the utility values obtained by $ag_{\underline{i}}$ in these negotiations can be seen as observations drawn from corresponding random variables, and we will again denote these variables as $\mathcal{X}_{\underline{i},1}, \mathcal{X}_{\underline{i},2}, \ldots, \mathcal{X}_{\underline{i},k}$ with $k = 2mn$.

Now, to calculate the standard error, we should not only take into account the variance of each individual variable $\mathcal{X}_{\underline{i},s}$, but also the variance among the means of the various random variables $\mathcal{X}_{\underline{i},s}$. After all, if we were to repeat this entire experiment, we would get different outcomes, not only because for each scenario the negotiations on that scenario might yield different outcomes, but also because we may be picking entirely different scenarios in the first step of our experiment.

For now, let us ignore again the fact that some variables may be mutually dependent. It can be shown mathematically that the correct formula for the variance of $\mathcal{Z}_{\underline{i}}$ is then given by:

$$Var_{\mathcal{Z}_{\underline{i}}} \quad = \quad \frac{1}{k^2}\sum_{s=1}^{k} Var_{\mathcal{Y}_{\underline{i}}} + \frac{1}{k^2}\sum_{s=1}^{k}(\mu_{\mathcal{Z}_{\underline{i}}} - \mu_{\underline{i},s})^2 \tag{6.13}$$

The first sum in this expression represents the second type of noise (variance due to the non-determinism of the agents), while the second sum represents the first type of noise (variance among the possible domains and opponents).

The problem with this equation, however, is that it requires knowledge of the *exact* means $\mu_{\underline{i},s}$ and $\mu_{\mathcal{Z}_{\underline{i}}}$. Of course, in reality we don't know these numbers, because if we did know them, then we wouldn't need to perform this experiment in the first place.

A naive idea to solve this, would be to simply replace the values of $\mu_{\underline{i},s}$ and $\mu_{\mathcal{Z}_{\underline{i}}}$ by their estimations $\hat{\mu}_{\underline{i},s}$ and $\hat{\mu}_{\mathcal{Z}_{\underline{i}}}$. While this idea is in principle correct, it turns out that if we do that, then we have to remove the first sum $\frac{1}{k^2}\sum_{s=1}^{k} Var_{\mathcal{Y}_{\underline{i}}}$ from the equation. As usual it would go too far to give a mathematical proof of this claim here, but the basic idea is that the statistical noise from the randomness of the agents is already represented in the fact that we are using *estimates* $\hat{\mu}_{\underline{i},s}$ of the means $\mu_{\underline{i},s}$, rather than the true means $\mu_{\underline{i},s}$ themselves. To see this, note that if all agents were purely deterministic, then every $\hat{\mu}_{\underline{i},s}$ would be exactly equal to $\mu_{\underline{i},s}$. Therefore, the difference between each $\hat{\mu}_{\underline{i},s}$ and each $\mu_{\underline{i},s}$ indeed represents the non-determinism of the agents.

So, the correct mathematical equation for the *estimated* variance of $\mathcal{Z}_{\underline{i}}$ (ignoring possible dependency between variables) is:

$$\hat{Var}_{\mathcal{Z}_{\underline{i}}} \quad = \quad \frac{1}{k \cdot (k-1)} \cdot \sum_{s=1}^{k} (\hat{\mu}_{Z_{\underline{i}}} - \hat{\mu}_{\underline{i},s})^2 \tag{6.14}$$

where, as before, each $\hat{\mu}_{\underline{i},s}$ is given by Eq. (6.9), and $\hat{\mu}_{\mathcal{Z}_{\underline{i}}}$ by:

$$\hat{\mu}_{\mathcal{Z}_{\underline{i}}} = \frac{1}{k} \sum_{s=1}^{k} \hat{\mu}_{\underline{i},s}$$

Finally, we still need to take into account that for each domain D_d, the means $\hat{\mu}_1^{d,\underline{i},\underline{i}}$ and $\hat{\mu}_2^{d,\underline{i},\underline{i}}$ are not independent. The final correct formula to calculate the standard error is therefore given by the square root of the following expression:

$$\hat{Var}_{\mathcal{Z}_{\underline{i}}} \quad = \quad \frac{1}{k \cdot (k-1)} \cdot \Big(\sum_{s=1}^{k} (\hat{\mu}_{\mathcal{Z}_{\underline{i}}} - \hat{\mu}_{\underline{i},s})^2 + 2 \cdot \sum_{d=1}^{|\mathcal{D}|} (\hat{\mu}_{\mathcal{Z}_{\underline{i}}} - \hat{\mu}_1^{d,\underline{i},\underline{i}})(\hat{\mu}_{\mathcal{Z}_{\underline{i}}} - \hat{\mu}_2^{d,\underline{i},\underline{i}}) \Big) \tag{6.15}$$

All of the above calculations were based on the assumption that in the first step of the experiment we picked the scenarios *randomly.* Of course, in reality when we run an experiment, we often do not *really* pick the scenarios randomly. Instead, we often just use some given set of agents and domains that happen to be available to us, or we manually pick subsets of those agents and domains. Nevertheless, that does not change the fact that we should model this selection of scenarios *as if* it was done randomly, because only in that way we can properly take into account that the results can be different if the agent encounters different domains or opponents. The fact that our choice is not truly random doesn't really matter, as long as we make sure that the domains and agents we pick are representative for the set of all agents and domains that we are interested in.

6.1.7.5 Calculating the Standard Error – an Example

Now let us look at an example. Suppose we have a very small tournament with only $m = 2$ domains and $n = 3$ agents, so we have $m \times n^2 = 2 \times 3 \times 3 = 18$ scenarios, and suppose that each scenario is repeated three times, so there are $3 \times 18 = 54$ negotiations. The three agents are called *IlPadrino*, *MegaBarter3000*, and *CrazyAgent*, which we may alternatively refer to as

$ag_{\underline{1}}$, $ag_{\underline{2}}$, and $ag_{\underline{3}}$, respectively. Our goal is to calculate the tournament score and standard error of $ag_{\underline{1}}$, a.k.a. *IlPadrino*.

While there are 18 scenarios in this tournament, we are for now only interested in the tournament score of *IlPadrino*, so we are only interested in the scenarios involving that agent. There are 10 such scenarios and, as explained above, these correspond to $2mn = 2 \times 2 \times 3 = 12$ different 'variables' that involve *IlPadrino* (there are 2 scenarios in which *both* roles are assumed by *IlPadrino*, and 8 scenarios in which *IlPadrino* negotiates against a different agent, so that makes $2 \times 2 + 8 \times 1 = 12$ random variables).

Now suppose the outcomes of our tournament are as given in Table 6.4. Since we are for now only interested in the 10 scenarios that involve *IlPadrino* we have omitted the other scenarios from this table. Furthermore, since we only need the utilities obtained by *IlPadrino*, we have highlighted those in boldface.

To calculate the tournament score and standard error of *IlPadrino*, we first start by calculating the average utility of each instance of *IlPadrino* in each scenario (i.e. the estimated mean of each random variable), according to:

$$\hat{\mu}_l^{d,\underline{i},\underline{j}} \quad := \quad \frac{1}{R^{d,\underline{i},\underline{j}}} \sum_{r=1}^{R^{d,\underline{i},\underline{j}}} u_l^{d,\underline{i},\underline{j},r} \tag{6.16}$$

Agent 1	**Agent 2**	**Domain**	$\mathbf{u_1}$	$\mathbf{u_2}$
IlPadrino	IlPadrino	CarSale	**0.724**	**0.600**
IlPadrino	IlPadrino	CarSale	**0.714**	**0.487**
IlPadrino	IlPadrino	CarSale	**0.791**	**0.541**
IlPadrino	MegaBarter3000	CarSale	**0.539**	0.485
IlPadrino	MegaBarter3000	CarSale	**0.520**	0.468
IlPadrino	MegaBarter3000	CarSale	**0.521**	0.498
IlPadrino	CrazyAgent	CarSale	**0.751**	0.682
IlPadrino	CrazyAgent	CarSale	**0.787**	0.697
IlPadrino	CrazyAgent	CarSale	**0.762**	0.703
MegaBarter3000	IlPadrino	CarSale	0.586	**0.746**
MegaBarter3000	IlPadrino	CarSale	0.671	**0.693**
MegaBarter3000	IlPadrino	CarSale	0.646	**0.631**
CrazyAgent	IlPadrino	CarSale	0.770	**0.422**
CrazyAgent	IlPadrino	CarSale	0.776	**0.426**
CrazyAgent	IlPadrino	CarSale	0.704	**0.407**
IlPadrino	IlPadrino	Cinema	**0.366**	**0.445**
IlPadrino	IlPadrino	Cinema	**0.342**	**0.432**
IlPadrino	IlPadrino	Cinema	**0.320**	**0.513**
IlPadrino	MegaBarter3000	Cinema	**0.604**	0.754
IlPadrino	MegaBarter3000	Cinema	**0.608**	0.733
IlPadrino	MegaBarter3000	Cinema	**0.517**	0.642
IlPadrino	CrazyAgent	Cinema	**0.532**	0.444
IlPadrino	CrazyAgent	Cinema	**0.532**	0.444
IlPadrino	CrazyAgent	Cinema	**0.532**	0.444
MegaBarter3000	IlPadrino	Cinema	0.461	**0.730**
MegaBarter3000	IlPadrino	Cinema	0.357	**0.758**
MegaBarter3000	IlPadrino	Cinema	0.534	**0.701**
CrazyAgent	IlPadrino	Cinema	0.643	**0.602**
CrazyAgent	IlPadrino	Cinema	0.630	**0.602**
CrazyAgent	IlPadrino	Cinema	0.535	**0.537**

Table 6.4: Outcomes of all negotiations involving IlPadrino from a fictional tournament. The results are grouped by scenario, with 3 repetitions for each scenario. The utility values obtained by IlPadrino are indicated in bold.

Using the numbers from Table 6.4 we get:

$$
\begin{aligned}
\hat{\mu}_1^{1,\underline{1},\underline{1}} &= \frac{1}{3}(0.724 + 0.714 + 0.791) = 0.743 \\
\hat{\mu}_2^{1,\underline{1},\underline{1}} &= \frac{1}{3}(0.600 + 0.487 + 0.541) = 0.543 \\
\hat{\mu}_1^{1,\underline{1},\underline{2}} &= \frac{1}{3}(0.539 + 0.520 + 0.521) = 0.527 \\
\hat{\mu}_1^{1,\underline{1},\underline{3}} &= \frac{1}{3}(0.751 + 0.787 + 0.762) = 0.767 \\
\hat{\mu}_2^{1,\underline{2},\underline{1}} &= \frac{1}{3}(0.746 + 0.693 + 0.631) = 0.690 \\
\hat{\mu}_2^{1,\underline{3},\underline{1}} &= \frac{1}{3}(0.422 + 0.426 + 0.407) = 0.418 \\
\hat{\mu}_1^{2,\underline{1},\underline{1}} &= \frac{1}{3}(0.366 + 0.342 + 0.320) = 0.343 \\
\hat{\mu}_2^{2,\underline{1},\underline{1}} &= \frac{1}{3}(0.445 + 0.432 + 0.513) = 0.463 \\
\hat{\mu}_1^{2,\underline{1},\underline{2}} &= \frac{1}{3}(0.604 + 0.608 + 0.517) = 0.576 \\
\hat{\mu}_1^{2,\underline{1},\underline{3}} &= \frac{1}{3}(0.532 + 0.532 + 0.532) = 0.532 \\
\hat{\mu}_2^{2,\underline{2},\underline{1}} &= \frac{1}{3}(0.730 + 0.758 + 0.701) = 0.730 \\
\hat{\mu}_2^{2,\underline{3},\underline{1}} &= \frac{1}{3}(0.602 + 0.602 + 0.537) = 0.580
\end{aligned}
$$

Next, we calculate the tournament score of $ag_{\underline{1}}$ as the total estimated mean:

$$
\begin{aligned}
U_{\underline{1}} = \hat{\mu}_{\mathcal{Z}_{\underline{1}}} &= \frac{1}{2mn}\sum_{d=1}^{m}\sum_{\underline{j}=1}^{n}\hat{\mu}_1^{d,\underline{1},\underline{j}} + \hat{\mu}_2^{d,\underline{j},\underline{1}} \\
&= \frac{1}{2\cdot 2\cdot 3}\sum_{d=1}^{2}\sum_{\underline{j}=1}^{3}\hat{\mu}_1^{d,\underline{1},\underline{j}} + \hat{\mu}_2^{d,\underline{j},\underline{1}} \\
&= \frac{1}{12}\cdot\Big(\hat{\mu}_1^{1,\underline{1},\underline{1}} + \hat{\mu}_2^{1,\underline{1},\underline{1}} + \hat{\mu}_1^{1,\underline{1},\underline{2}} + \hat{\mu}_1^{1,\underline{1},\underline{3}} + \hat{\mu}_2^{1,\underline{2},\underline{1}} + \hat{\mu}_2^{1,\underline{3},\underline{1}} \\
&\qquad + \hat{\mu}_1^{2,\underline{1},\underline{1}} + \hat{\mu}_2^{2,\underline{1},\underline{1}} + \hat{\mu}_1^{2,\underline{1},\underline{2}} + \hat{\mu}_1^{2,\underline{1},\underline{3}} + \hat{\mu}_2^{2,\underline{2},\underline{1}} + \hat{\mu}_2^{2,\underline{3},\underline{1}}\Big) \\
&= \frac{1}{12}\cdot\Big(0.743 + 0.543 + 0.527 + 0.767 + 0.690 + 0.418 \\
&\qquad + 0.343 + 0.463 + 0.576 + 0.532 + 0.730 + 0.580\Big) \\
&= 0.576
\end{aligned}
$$

Next, using Eq. (6.14) we get:

$$
\begin{aligned}
\hat{Var}_{\mathcal{Z}_{\underline{1}}} &= \frac{1}{12\cdot(12-1)}\cdot\Big((0.576-0.743)^2 + (0.576-0.543)^2 + (0.576-0.527)^2 + \\
&\quad (0.576-0.767)^2 + (0.576-0.690)^2 + (0.576-0.418)^2 + \\
&\quad (0.576-0.343)^2 + (0.576-0.463)^2 + (0.576-0.576)^2 + \\
&\quad (0.576-0.532)^2 + (0.576-0.730)^2 + (0.576-0.580)^2 + \\
&\quad 2\cdot(0.576-0.743)\cdot(0.576-0.543) + \\
&\quad 2\cdot(0.576-0.343)\cdot(0.576-0.463)\Big) \\
&= 1.817\times 10^{-3}
\end{aligned}
$$

And from this it finally follows that the total estimated standard error of our agent is:

$$
\hat{\sigma}_{\mathcal{Z}_{\underline{1}}} = \sqrt{\hat{Var}_{\mathcal{Z}_{\underline{1}}}} = \sqrt{1.817\times 10^{-3}} = 0.0426
$$

Now, we still need to repeat these calculations for every other agent as well, and then finally we can present the results of the entire tournament as in Table 6.5. Note that we have multiplied the utility values and the standard errors by a factor of 1,000. This is purely for the purpose of readability, because 576 ± 42.6 is a bit easier to read than 0.576 ± 0.0426

Agent	**Tournament Score** ($U_{\underline{i}}$)	**Utility-under-Agreement** ($UA_{\underline{i}}$)	**Agreement Rate** ($AR_{\underline{i}}$)
MegaBarter3000	717 ± 30.0	815	88%
IlPadrino	576 ± 42.6	619	93%
CrazyAgent	511 ± 23.7	824	62%

Table 6.5: Results of a fictional tournament between three fictional agents. This time presented together with the standard errors. We have multiplied the utility values and standard errors by a factor of $1,000$ purely for readability.

Exercise 17. Standard Error Modify your code of Exercise 15 so that it also calculates, for each agent, the standard error on its tournament score, according to Equation (6.14) and then displays a table such Table 6.5.

6.1.7.6 Decreasing the Standard Error

A common situation that we may encounter when doing experiments, is that although our agent's tournament score is higher than that of other agents, the difference is not large enough compared to the standard error to conclude that our agent is indeed truly better.

A naive solution would be to simply increase the number of repetitions, in order to achieve more accurate results. However, this can only decrease the standard error to a limited extent.

To see this, note that in Equation (6.14) or Equation (6.15) this would allow the estimated means $\hat{\mu}_{\underline{i},s}$ and $\hat{\mu}_{\mathcal{Z}_{\underline{i}}}$ to get very close to the true means $\mu_{\underline{i},s}$ and $\mu_{\mathcal{Z}_{\underline{i}}}$, but this would still not reduce the total standard error to zero, because the $\hat{\mu}_{\underline{i},s}$ would still be different from $\hat{\mu}_{\mathcal{Z}_{\underline{i}}}$. Therefore, a much more effective way to reduce statistical noise, is to increase the number of negotiation scenarios, instead (note that this increases the value of k and therefore decreases the standard error).

Of course, in some cases it may happen that we simply can't increase the number of scenarios, because we only have a limited number of domains and agents available to us. In that case, increasing the number of repetitions is the only solution. If this does not decrease the standard errors enough to get statistically significant results, then, as an alternative, we can still use Equation (6.12) instead. Using this equation is not *wrong* per se, but rather, it changes the *interpretation* of our results.

Specifically, if the standard errors of the agents are very small according to Equation (6.12), it means that if we repeat the experiment we will likely get the same tournament scores, as long as we use *exactly the same set of scenarios*. On the other hand, if the standard errors are very small according to Equation (6.15), it means that we will likely get the same tournament scores *even if* we repeat the experiment *on an entirely different set of scenarios*, as long as those scenarios are still similar to the ones used in the first experiment. This is of course a much stronger statement, and therefore it makes scientifically more sense to use Equation (6.15).

6.1.8 Statistical Significance

Calculating the standard error on the tournament score is a useful way to get an idea of how accurately we have determined an agent's strength. However, this is not sufficient to determine whether or not any difference between two agents is statistically significant. In this section we will briefly explain how to perform a proper statistical test.

We should note again, however, that this is a vast topic, so we can't go into full detail here. For a full understanding of statistical tests we therefore refer the reader to a more specialized textbook on statistics, such as [23] or [26].

In the rest of this section we will assume that we have calculated the tournament scores of agents $ag_{\underline{i}}$ and $ag_{\underline{j}}$ and that this was higher for $ag_{\underline{i}}$ than for $ag_{\underline{j}}$, so $U_{\underline{i}} > U_{\underline{j}}$. We now want to determine whether that difference is statistically significant. In order to do this we need to perform a so-called *paired t-test*.

6.1.8.1 Formulating the Problem

We have previously seen that $U_{\underline{i}}$ can be seen as an observation from a random variable $\mathcal{Z}_{\underline{i}}$ and moreover that the tournament score can also be seen as an approximation $\hat{\mu}_{\mathcal{Z}_{\underline{i}}}$ of the mean $\mu_{\mathcal{Z}_{\underline{i}}}$ of $\mathcal{Z}_{\underline{i}}$.

The idea in this section, is that we will now regard the *difference* $U_{\underline{i}} - U_{\underline{j}}$ as an observation from a random variable that we will denote $\mathcal{Z}_{\underline{i},\underline{j}}$. This observation can be seen as an estimation $\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}}$ of the true mean $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}}$ of this random variable, which represents the 'true' difference in strength between the two agents. So, we have $\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}} = U_{\underline{i}} - U_{\underline{j}}$, and since we have $U_{\underline{i}} > U_{\underline{j}}$ it follows that $\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}} > 0$.

Now, the fact that our **observation** $\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}}$ is greater than 0 *suggests* that

$ag_{\underline{i}}$ is stronger than $ag_{\underline{j}}$. However, we can only say that that is really true if in fact we have $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}} > 0$. This is therefore our **hypothesis**.

Now, ideally, *we would like to prove that, given our observation, we can conclude that our hypothesis is true.*

Unfortunately, however, in the world of statistics we can never really prove anything with 100% certainty, so instead we could reformulate our goal as follows:

"We would like to prove that, given our observation, there is a very high probability that our hypothesis is true."

Once again, however, it turns out that, in general, this question is impossible to answer. To understand why, it may be helpful to look at the following two problems:

1. If we flip a coin 10 times, and we know that it is a *fair* coin (i.e. for each flip the probability of getting 'heads' is exactly 50%), then what is the probability that we will observe 'heads' 7 times and 'tails' 3 times?
2. If we flip a coin 10 times and we observe 'heads' 7 times and 'tails' 3 times, then what is the probability that the coin is fair?

The first of these two problems is easy to solve. It can be calculated in a straightforward manner using only basic probability theory. The second problem, however, is impossible to answer without any further information. In principle it can be solved with Bayes' rule (Section 4.1.1.1), but that would require knowing a prior probability for the 'fairness' of the coin. Without such extra information, however, it is not even a mathematically well-defined problem.

The difference between these two problems is that in the first case we know the parameters of the probability distribution, and we use them to calculate the probability of getting a certain observation, while in the second case we are given the observation, and with that we are hoping to figure out the probability that the parameters have a certain value.

Our goal that we described above was also formulated as a problem of the second type. Luckily, however, we can reformulate it again, so that it becomes a problem of the first type:

"We would like to prove that, if our hypothesis is <u>not</u> true, then the probability of getting our observation is very <u>small</u>."

This probability is called the ***p*-value**. The lower this value, the more confident we can be that our hypothesis is true. In order to determine if it is low *enough*, it is common in the scientific literature to apply a threshold of 0.05. So, if the p-value is below 0.05, then we can say that our conclusion is statistically significant. This threshold is also known as the **α-value**.

The *negation* of our hypothesis is usually called the **null hypothesis**, and the hypothesis that we are actually trying to prove is also known as the **alternative hypothesis**. So, in our example, the null hypothesis can be formulated as "*agent $ag_{\underline{i}}$ is not better than agent $ag_{\underline{j}}$*" or, mathematically, as $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}} \leq 0$.

If the p-value is below the α-value, we say the **null hypothesis is rejected**. This statement may at first sound a bit confusing, because it's a kind of double negation. It means the *null hypothesis* can be considered *false*, which in turn means the *alternative hypothesis* (which we wanted to prove) can be considered to be *true*.

In order to calculate the p-value, we need to have some probability distribution for our random variable $\mathcal{Z}_{\underline{i},\underline{j}}$. The p-value is then defined as the probability that we draw an observation from $\mathcal{Z}_{\underline{i},\underline{j}}$ that is equal to, or higher than, the actual observation $\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}}$ that we made.

We will assume that $\mathcal{Z}_{\underline{i},\underline{j}}$ has a Gaussian probability distribution. This assumption is safe, since any observation from $\mathcal{Z}_{\underline{i},\underline{j}}$ is obtained by calculating and subtracting averages over many (mostly independent) observations from random variables $\mathcal{X}_l^{d,\underline{i},\underline{j}}$. It is well-known that, no matter what kind of probability distribution those variables have, as long as there are enough of them, the distribution of $\mathcal{Z}_{\underline{i},\underline{j}}$ will indeed approximate the Gaussian distribution.

Next, we need to know the mean $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}}$ of that Gaussian distribution. Recall that we are trying to prove the hypothesis that $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}} > 0$, and as explained above, to do this we have to calculate the p-value under the assumption that this hypothesis is *not* true. So, we have to assume that $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}} \leq 0$. Now, in order to be absolutely sure that our conclusions are valid, we have to make the worst-case assumption, which means we have to assume $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}} = 0$. Indeed, note that the higher the mean $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}}$ the more likely it is that we make the an observation $\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}} > 0$, and thus the higher the p-value. So, if our p-value stays below the α-value *even if* we chose the highest value of $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}}$, then we are sure that it is also below the α-value for any other value of $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}}$.

The final ingredient we need, is the standard deviation $\sigma_{\mathcal{Z}_{\underline{i},\underline{j}}}$ of $\mathcal{Z}_{\underline{i},\underline{j}}$. We will discuss how to estimate $\sigma_{\mathcal{Z}_{\underline{i},\underline{j}}}$ next, but for now, let us assume that we already know its exact value. In that case, it follows directly that we can calculate the p-value as follows.

$$p \quad = \quad \int_{\hat{\mu}}^{\infty} \mathcal{N}(x|\mu,\sigma)dx \quad = \quad \frac{1}{\sqrt{2\pi}\sigma}\int_{\hat{\mu}}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx \qquad (6.17)$$

where $\mathcal{N}$ denotes the Gaussian distribution, and we used μ, $\hat{\mu}$ and σ as shorthands for $\mu_{\mathcal{Z}_{\underline{i},\underline{j}}}$, $\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}}$ and $\sigma_{\mathcal{Z}_{\underline{i},\underline{j}}}$

Then, with our assumption that $\mu = 0$, and with a change of variables $x' := \frac{x}{\sigma}$ and defining the ***z*-statistic**: $z := \frac{\hat{\mu}}{\sigma}$ this can be rewritten as:

$$p \quad = \quad \frac{1}{\sqrt{2\pi}} \int_{z}^{\infty} e^{-\frac{x'^2}{2}} dx' \tag{6.18}$$

The advantage of this expression is that it contains only one parameter: the z-statistic. So, the p-value only depends on the value of z.

6.1.8.2 Estimating the Standard Deviation

In reality, however, we typically wouldn't know the standard deviation of $\mathcal{Z}_{\underline{i},\underline{j}}$, so instead we have to estimate it, which we will explain here.

In Section 6.1.7 we have seen how the standard deviation of $\mathcal{Z}_{\underline{i}}$ could be calculated by modeling $\mathcal{Z}_{\underline{i}}$ as a sum over variables $\mathcal{Y}_{\underline{i},s}$ and then using the observations of those variables to calculate $\hat{\sigma}_{\mathcal{Z}_{\underline{i}}}$. So, here we will do something similar. That is, we define:

$$\mathcal{Y}_{\underline{i},\underline{j},s} := \mathcal{Y}_{\underline{i},s} - \mathcal{Y}_{\underline{j},s}$$

$$\mathcal{Z}_{\underline{i},\underline{j}} := \frac{1}{k} \sum_{s=1}^{k} \mathcal{Y}_{\underline{i},\underline{j},s}$$

with $\mathcal{Y}_{\underline{i},s}$ and $\mathcal{Y}_{\underline{j},s}$ defined as before (See Eq. (6.6)). That is, an observation of $\mathcal{Y}_{\underline{i},s}$ is defined as the average utility obtained by agent $ag_{\underline{i}}$ over some number of repetitions R_s of some scenario sc_s.

However, we have to be careful here, because the two agents $ag_{\underline{i}}$ and $ag_{\underline{j}}$ do not participate in exactly the same scenarios. That is, for any given integer s the variables $\mathcal{X}_{\underline{i},s}$ and $\mathcal{X}_{\underline{j},s}$ may correspond to two entirely different scenarios. While we mentioned that each variable of agent $ag_{\underline{i}}$ can be denoted as $\mathcal{X}_{\underline{i},s}$ for some integer s, we never specified how to decide which variable gets assigned which integer s. So, if we don't do this carefully, the two variables $\mathcal{X}_{\underline{i},s}$ and $\mathcal{X}_{\underline{j},s}$ may correspond to two entirely different and unrelated scenarios. For example, they could be representing negotiations on two different negotiation domains. But it doesn't make any sense to compare the utility of $ag_{\underline{i}}$ on one domain with the utility of $ag_{\underline{j}}$ on a totally different domain. We therefore have to make sure that for each integer s, the variables $\mathcal{X}_{\underline{i},s}$ and $\mathcal{X}_{\underline{j},s}$ can be compared to each other.

To do this, we first have to make a careful distinction between three different types of scenario:

1. Scenarios in which agent $ag_{\underline{i}}$ or agent $ag_{\underline{j}}$ was negotiating against a *third* agent $ag_{\underline{l}}$ (with $ag_{\underline{i}} \neq ag_{\underline{l}}$ and $ag_{\underline{j}} \neq ag_{\underline{l}}$).
   That is, scenarios of the form $(\Pi, D, ag_{\underline{i}}, ag_{\underline{l}})$, $(\Pi, D, ag_{\underline{j}}, ag_{\underline{l}})$, $(\Pi, D, ag_{\underline{l}}, ag_{\underline{i}})$, or $(\Pi, D, ag_{\underline{l}}, ag_{\underline{j}})$.
2. Scenarios in which the two agents $ag_{\underline{i}}$ and $ag_{\underline{j}}$ were negotiating directly against *each other*.
   That is, scenarios of the form $(\Pi, D, ag_{\underline{i}}, ag_{\underline{j}})$ or $(\Pi, D, ag_{\underline{j}}, ag_{\underline{i}})$.
3. Scenarios in which either agent $ag_{\underline{i}}$ or $ag_{\underline{j}}$ was negotiating against *itself*.
   That is, scenarios of the form $(\Pi, D, ag_{\underline{i}}, ag_{\underline{i}})$ or $(\Pi, D, ag_{\underline{j}}, ag_{\underline{j}})$.

For the first type of scenario we calculate, for each domain D_d and each opponent $ag_{\underline{l}}$, the average utility obtained by $ag_{\underline{i}}$ against that opponent and the average utility obtained by $ag_{\underline{j}}$ against that same opponent, and we then calculate the difference. In fact, we do that twice, once for those negotiations in which $ag_{\underline{l}}$ had utility u_1 and once for those negotiations in which $ag_{\underline{l}}$ had utility u_2. In other words, we calculate the following two quantities:

$$\hat{\mu}_{\underline{i},\underline{j}}^{d,\cdot,\underline{l}} \quad := \quad \hat{\mu}_1^{d,\underline{i},\underline{l}} - \hat{\mu}_1^{d,\underline{j},\underline{l}}$$

$$\hat{\mu}_{\underline{i},\underline{j}}^{d,\underline{l},\cdot} \quad := \quad \hat{\mu}_2^{d,\underline{l},\underline{i}} - \hat{\mu}_2^{d,\underline{l},\underline{j}}$$

where each $\hat{\mu}$ on the right-hand side is calculated with Equation (6.16).

Next, for the second type of scenario, we calculate, for each domain $\mathcal{D}_d$, the difference between the average utility obtained by $ag_{\underline{i}}$ and the average utility obtained by $ag_{\underline{j}}$. This is also done twice: once for the case that $ag_{\underline{i}}$ had utility u_1 and one for the case that $ag_{\underline{i}}$ had utility u_2:

$$\hat{\mu}_{\underline{i},\underline{j}}^{d,\underline{i},\underline{j}} \quad := \quad \hat{\mu}_1^{d,\underline{i},\underline{j}} - \hat{\mu}_2^{d,\underline{i},\underline{j}}$$

$$\hat{\mu}_{\underline{i},\underline{j}}^{d,\underline{j},\underline{i}} \quad := \quad \hat{\mu}_2^{d,\underline{j},\underline{i}} - \hat{\mu}_1^{d,\underline{j},\underline{i}}$$

Finally, we calculate, for each domain $\mathcal{D}_d$ the difference between the agents' average utilities in the 'self-play' scenarios:

$$\hat{\mu}_{\underline{i},\underline{j}}^{d,sp_1} \quad := \quad \hat{\mu}_1^{d,\underline{i},\underline{i}} - \hat{\mu}_1^{d,\underline{j},\underline{j}}$$

$$\hat{\mu}_{\underline{i},\underline{j}}^{d,sp_2} \quad := \quad \hat{\mu}_2^{d,\underline{i},\underline{i}} - \hat{\mu}_2^{d,\underline{j},\underline{j}}$$

To better understand these expressions, note that each of them consists of an average utility of $ag_{\underline{i}}$ minus an average utility of $ag_{\underline{j}}$. The expressions just differ in the question for which scenarios those averages are calculated. Furthermore, it may help to note that for each of these quantities the subscript indices $\underline{i}$ and $\underline{j}$ always refer to the two agents $ag_{\underline{i}}$ and $ag_{\underline{j}}$ that we are comparing, while the superscript indices refer to the scenarios for which each quantity is calculated.

The idea is that, for each of these equations, the two means on the right-hand side can be seen as observations from two variables $\mathcal{Y}_{\underline{i},s}$ and $\mathcal{Y}_{\underline{j},s}$ respectively, and therefore the number on the left-hand side can be seen as an observation from a variable $\mathcal{Y}_{\underline{i},\underline{j},s}$.

We now have a set of in total $2mn$ different numbers, with $m = |\mathcal{D}|$ and $n = |Ag|$. For example, if we have 2 domains and 4 agents then we have to calculate are 16 different numbers. Specifically, if we are comparing the scores of agents $ag_{\underline{1}}$ and $ag_{\underline{2}}$, then we have $\underline{i} = 1$, $\underline{j} = 2$. Furthermore, since there are 2 other agents, $ag_{\underline{3}}$ and $ag_{\underline{4}}$, we then have $\underline{l} \in \{3, 4\}$ and since there are two domains we have $d \in \{1, 2\}$. Therefore, we have to calculate the following 16 numbers:

$$
\begin{array}{cccc}
\hat{\mu}^{1,\cdot,\underline{3}}_{\underline{1},\underline{2}} & \hat{\mu}^{1,\cdot,\underline{4}}_{\underline{1},\underline{2}} & \hat{\mu}^{2,\cdot,\underline{3}}_{\underline{1},\underline{2}} & \hat{\mu}^{2,\cdot,\underline{4}}_{\underline{1},\underline{2}} \\
\hat{\mu}^{1,\underline{3},\cdot}_{\underline{1},\underline{2}} & \hat{\mu}^{1,\underline{4},\cdot}_{\underline{1},\underline{2}} & \hat{\mu}^{2,\underline{3},\cdot}_{\underline{1},\underline{2}} & \hat{\mu}^{2,\underline{4},\cdot}_{\underline{1},\underline{2}} \\
\hat{\mu}^{1,\underline{1},\underline{2}}_{\underline{1},\underline{2}} & \hat{\mu}^{1,\underline{2},\underline{1}}_{\underline{1},\underline{2}} & \hat{\mu}^{2,\underline{1},\underline{2}}_{\underline{1},\underline{2}} & \hat{\mu}^{2,\underline{2},\underline{1}}_{\underline{1},\underline{2}} \\
\hat{\mu}^{1,sp_1}_{\underline{1},\underline{2}} & \hat{\mu}^{1,sp_2}_{\underline{1},\underline{2}} & \hat{\mu}^{2,sp_1}_{\underline{1},\underline{2}} & \hat{\mu}^{2,sp_2}_{\underline{1},\underline{2}}
\end{array}
$$

To simplify notation we will now instead denote these numbers as:

$$\hat{\mu}_{\underline{i},\underline{j},1}, \quad \hat{\mu}_{\underline{i},\underline{j},2}, \quad \ldots, \quad \hat{\mu}_{\underline{i},\underline{j},16}$$

Each of these numbers $\hat{\mu}_{\underline{i},\underline{j},s}$ can be seen as an observation from a different random variable $\mathcal{Y}_{\underline{i},\underline{j},s}$.

It is easy to check that if we calculate the estimated mean of $\mathcal{Z}_{\underline{i},\underline{j}}$, i.e. the average of these 16 observations, we just get the difference between the two agents' tournament scores:

$$\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}} \quad = \quad \frac{1}{2mn}\sum_{s=1}^{2mn} \hat{\mu}_{\underline{i},\underline{j},s} \quad = \quad U_{\underline{i}} - U_{\underline{j}}$$

We are now ready to calculate the estimated standard deviation of $\mathcal{Z}_{\underline{i},\underline{j}}$. As before, however, we have to take into account that not all of these observations are independent. Specifically, for each domain D_d the values of $\hat{\mu}^{d,sp_1}_{\underline{i},\underline{j}}$

and $\hat{\mu}_{\underline{i},\underline{j}}^{d,sp_2}$ are mutually dependent, because they are both calculated from the same two scenarios $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{i}})$ and $(\Pi, D_d, ag_{\underline{j}}, ag_{\underline{j}})$. We therefore use an expression similar to Eq. (6.15):

$$\hat{Var}_{\mathcal{Z}_{\underline{i},\underline{j}}} = \frac{1}{k \cdot (k-1)} \cdot \Big(\sum_{s=1}^{k} (\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}} - \hat{\mu}_{\underline{i},\underline{j},s})^2 + 2 \cdot \sum_{d=1}^{|\mathcal{D}|} (\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}} - \hat{\mu}_{\underline{i},\underline{j}}^{d,sp_1}) \cdot (\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}} - \hat{\mu}_{\underline{i},\underline{j}}^{d,sp_2}) \Big) \tag{6.19}$$

with $k = 2mn = 2 \cdot |\mathcal{D}| \cdot |Ag|$

Finally, the estimated standard deviation of $\mathcal{Z}_{\underline{i},\underline{j}}$ can be calculated by taking the square root:

$$\hat{\sigma}_{\mathcal{Z}_{\underline{i},\underline{j}}} = \sqrt{\hat{Var}_{\mathcal{Z}_{\underline{i},\underline{j}}}}$$

6.1.8.3 Performing the Test

We are now finally ready to calculate the p-value of our hypothesis. Essentially, we are going to do the same as in Equations (6.17) and (6.18), except that this time we will be using the *estimated* standard deviation $\hat{\sigma}_{\mathcal{Z}_{\underline{i},\underline{j}}}$ instead of the true standard deviation $\sigma_{\mathcal{Z}_{\underline{i},\underline{j}}}$. This means that instead of the z-statistic we are now calculating what is called the '**t-statistic**':

$$\mathsf{t} = \frac{\hat{\mu}_{\mathcal{Z}_{\underline{i},\underline{j}}}}{\hat{\sigma}_{\mathcal{Z}_{\underline{i},\underline{j}}}}$$

A major consequence of this difference, is that we can now no longer assume that this statistic is drawn from a Gaussian distribution. Instead, the t-statistic is drawn from what is known as the '**t-distribution with** $2mn-1$ **degrees of freedom**' (denoted t_{2mn-1}). This is because $\hat{\sigma}_{\mathcal{Z}_{\underline{i},\underline{j}}}$ is not a constant, but rather a quantity that was calculated from observations of a number of random variables. As usual, we refer to more specialized text books on statistics, such as [23] and [26], for more details on this.

The t-distribution is very similar to the Gaussian distribution but has 'heavier' tails. So, we now need to calculate the p-value as follows:

$$p = \int_{\mathsf{t}}^{\infty} t_{2mn-1}(x)dx$$

Now, we have good news and bad news. The bad news is that the t-distribution has a very complicated expression, so we can't expect to calculate this integral analytically. The good news, however, is that we don't have to care about that, because most programming languages and statistics programs have this function built-in. For example, if $t = 3.1$ and $2nm - 1 = 15$, then, in Excel, we can calculate the p-value with the following formula:

=T.DIST.RT(3.1, 15)

and in Python we can calculate it as follows (after installing the scipy package):

```
from scipy import stats
p_value = 1 - stats.t.cdf(3.1, 15)
```

Now, if our p-value is smaller than or equal to the α-value (which is typically set to 0.05), then we say that the null hypothesis is rejected, which is another way of saying that our conclusion that agent $ag_{\underline{i}}$ is better than agent $ag_{\underline{j}}$, is statistically significant.

On the other hand, if $p > \alpha$, this does *not* mean that we can draw the opposite conclusion that $ag_{\underline{j}}$ is better than $ag_{\underline{i}}$. It just means that we do not have sufficient evidence to say confidently that $ag_{\underline{i}}$ is better than $ag_{\underline{j}}$.

Exercise 18. Paired t-test. Modify your code of the previous exercises so that it also calculates, for each pair of agents, the p-value for the hypothesis that the agent with the higher tournament score is indeed stronger than the other, as explained in this section.

6.1.8.4 Combining Multiple Tests

In the previous sections we have compared the scores of two agents, and determined whether the difference was statistically significant. Of course, 'statistically significant' is still no guarantee that $ag_{\underline{i}}$ is truly better than $ag_{\underline{j}}$. After all, as we mentioned above, in statistics we can never be 100% sure of anything. The best we can do is to say that it is *unlikely* that $ag_{\underline{i}}$ is *not* better than $ag_{\underline{j}}$, because if that were the case, then the probability of obtaining the data we observed would have been smaller than 5%.

This is fine if we are just comparing two agents, but this becomes problematic when we are comparing our agent against multiple benchmark agents (which we would normally do). The problem is that if we compare our agent $ag_{\underline{1}}$ against multiple other agents, then even though for each *individual* opponent there is only a small chance that we draw a false conclusion, these probabilities add up, yielding a relatively large possibility that *at least one* of our conclusions is false.

For example, suppose that we test our agent against four different opponents, so for each of these four opponents we test the null hypothesis that that opponent is better than or equal to our agent $a\underline{g}_1$, and suppose that for each of these hypotheses we apply an α-value of 0.05. Furthermore assume the worst case-scenario that all null hypotheses are in reality true (so in reality our agent does not outperform any of the benchmark agents). Then the probability that we will falsely reject at least one null hypothesis would be:

$$P(\text{at least one of the null hypotheses is rejected}) = 1 - (1-\alpha)^4 \approx 4 \cdot \alpha$$

In other words, the fact that we are testing *multiple* hypotheses increases the chance that we will draw a false conclusion from 5% to 20%, which is too high. In order to compensate for this, we therefore need to apply a correction to our acceptance threshold. To this end, we define the **global** α-value, which we'll denote as α_G, to be the maximum probability of error that we tolerate for the *entire* experiment. That is, the maximum probability that we draw at least one false conclusion. This value would typically be set to 0.05. Furthermore, we define for every single null hypothesis y_i a **local** α-value, which we'll denote as α_i, which represents the maximum p-value for which we reject that individual null hypothesis.

A commonly used and simple method to determine the local α-values is the **Bonferroni correction**.

Definition 6.1.3. *Suppose we are testing n hypotheses and that we are given a global α-value α_G. Then, we say that we have applied a **Bonferroni correction** if the local α-values α_i are set as:*

$$\alpha_i = \frac{\alpha_G}{n}$$

For example, suppose if we are testing 4 different null hypotheses y_1, y_2, y_3, and y_4, and for each hypothesis y_i we have calculated a corresponding p-value p_i. Furthermore, suppose we have a global α-level of 0.05. Then, applying a Bonferroni correction means that any individual null hypothesis y_i is rejected if and only if $p_i \leq \frac{0.05}{4} = 0.0125$.

While this method is simple, it turns out that it is actually overly strict, in the sense that it reduces the individual α-levels too much, which means that we might throw away some results that are actually statistically significant. A better but somewhat more complicated solution is the so-called **Holm-Bonferroni procedure**.

Definition 6.1.4. *Suppose we have n null hypotheses, denoted $y_1, y_2, \dots y_n$, and suppose that for each null hypothesis y_i we have calculated a corresponding p-value p_i. Furthermore, assume that these hypotheses are sorted in order of increasing p-value, so we have: $p_1 \leq p_2 \leq \cdots \leq p_{n-1} \leq p_n$. Finally, assume we are given some global α-value α_G. Then, according to the **Holm-Bonferroni procedure**:*

- *Hypothesis y_1 is rejected iff $p_1 \leq \frac{\alpha_G}{n}$.*
- *Hypothesis y_2 is rejected iff y_1 is rejected and $p_2 \leq \frac{\alpha_G}{n-1}$.*
- *Hypothesis y_3 is rejected iff y_1 and y_2 are both rejected and $p_3 \leq \frac{\alpha_G}{n-2}$.*
- *etcetera...*

So, in general, the Holm-Bonferroni procedure rejects a null hypothesis y_i iff all null hypotheses y_j with $j < i$ have been rejected and $p_i \leq \frac{\alpha_G}{n+1-i}$.

6.2 Empirical Game-Theoretical Analysis

In the previous section we discussed how we can compare agents experimentally, through the use of a 'tournament evaluation'. While this is by far the most commonly used method, it suffers from one important problem. Namely, that it is very sensitive to the presence of weak agents. We therefore here present an alternative evaluation method that avoids this problem.

To explain this, imagine a tournament between an extremely hardheaded agent, a very conceding agent, and an intermediate agent. In this tournament the hardheaded agent might obtain a very high score because it is able to fully exploit the conceding agent, while the conceding agent will achieve a very low score. This may make it *seem* that the hardheaded agent is very strong. However, one could argue that this outcome is unrealistic, because in a real-world situation it would unlikely to encounter a very poorly performing conceding agent, and without the presence of the conceding agent the hardheaded agent would achieve a lot less utility.

This problem can be partially mitigated by organizing the tournament over multiple rounds. That is, after we have obtained the results, we remove the worst performing agents, and run another tournament, but this time with only the top performing agents from the first round.

However, there exists a more principled approach to this, known as '*empirical game-theoretical analysis*' (EGTA) [49]. In the following subsection we will first explain this through an example.

6.2.1 Example

Suppose we have again the same four agents as before: *MegaBarter3000*, *IlPadrino*, *CrazyAgent* and *RandomAgent*. Furthermore, assume that we have run a tournament in which all agents negotiate against each other (including against themselves), as in the previous section, and that we have stored the results of all the negotiations in this tournament in a database.

Now, imagine that there are two people, called Alice and Bob, that have to negotiate about something, but they will not do these negotiations themselves, but instead they each choose one of those four agents to do the negotiations on their behalf. Note that it is perfectly possible that they each choose the same agent, so in that case there would be two copies of the same agent negotiating against each other.

Now, the question is, which agents should Alice and Bob choose, assuming they have access to the full database with all the results of the tournament?

Let's assume that, initially, Alice is a bit naive and simply chooses the agent that achieved the highest tournament score in our tournament. Let's say that that is *MegaBarter3000*. Bob, however, is not so naive. He also sees that *MegaBarter3000* scored the highest tournament score, but when he looks a bit closer to the data, he realizes that *MegaBarter3000* mainly achieved high utility against *CrazyAgent* and *RandomAgent*, which each achieved a very low tournament score. On the other hand, in the negotiations between *MegaBarter3000* and *IlPadrino*, it was actually *IlPadrino* that performed much better. Therefore, assuming that Alice chooses *MegaBarter3000*, Bob is smart and chooses *IlPadrino*. If they now let these two agents negotiate each other, then they will likely achieve a deal that is much better for Bob than for Alice.

Luckily for Alice, however, she also realizes this, just in time before they start their negotiations. So, knowing that Bob will choose *IlPadrino*, she now carefully examines the data to determine which of the four agents performs best against *IlPadrino*. She learns from this that the agent that scores the highest utility against *IlPadrino*, is *IlPadrino* itself. Therefore, she also chooses *IlPadrino*. So, in the end, Alice and Bob both end up choosing *IlPadrino*.

We learn from this example that the agent that scores the highest average utility in the tournament is not necessarily the best choice.

You may have noticed that Alice and Bob were following the same pattern of reasoning as what we discussed in Section 5.2.3. That is, Alice and Bob are essentially playing a normal-form game, in which their 'actions' are the 4 respective agents, and their utility functions are given by the scores that the agents obtained in the tournament. In the example, *IlPadrino* was a best response against *MegaBarter3000*, and *IlPadrino* was also a best response against itself. Therefore, the action profile (*IlPadrino*, *IlPadrino*) was a pure Nash equilibrium of the game. So, the essence of an empirical game-theoretical evaluation, is to model the choice of agents as a normal-form game and then determine its Nash equilibria.

6.2.2 Formal Definition

We will now formally define the procedure described in the example above.

Suppose we have a finite set of agents $Ag = \{ag_{\underline{1}}, ag_{\underline{2}}, \dots ag_{\underline{n}}\}$, and some finite set of domains $\mathcal{D}$, so we have $|\mathcal{D}| \times |Ag| \times |Ag|$ possible scenarios. Furthermore, suppose that each scenario $(\Pi, D_d, ag_{\underline{i}}, ag_{\underline{j}})$ is repeated a certain number of times, denoted $R^{d,\underline{i},\underline{j}}$.

Then, after running the tournament, we calculate for each pair of agents $(ag_{\underline{i}}, ag_{\underline{j}})$ the average utility $U_{\underline{i}}^{\underline{j}}$ achieved by agent $ag_{\underline{i}}$ against agent $ag_{\underline{j}}$, and the average utility $U_{\underline{j}}^{\underline{i}}$ achieved by agent $ag_{\underline{j}}$ against agent $ag_{\underline{i}}$.

$$U_{\underline{i}}^{\underline{j}} := \frac{1}{|\mathcal{D}|} \sum_{d=1}^{|\mathcal{D}|} U_{\underline{i}}^{d,\underline{j}} \tag{6.20}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{d=1}^{|\mathcal{D}|} \Big(\frac{1}{2R^{d,\underline{i},\underline{j}}} \sum_{r=1}^{R^{d,\underline{i},\underline{j}}} u_1^{d,\underline{i},\underline{j},r} + \frac{1}{2R^{d,\underline{j},\underline{i}}} \sum_{r=1}^{R^{d,\underline{j},\underline{i}}} u_2^{d,\underline{j},\underline{i},r} \Big) \tag{6.21}$$

This yields $|Ag| \times |Ag|$ numbers which can be organized in a square matrix, where the cell in row i and column j contains the number $U_{\underline{i}}^{\underline{j}}$. This can be seen as the payoff matrix of a symmetric normal-form game (recall from Section 5.3.4 that for symmetric games the payoff matrix only needs to contain one number in each cell, which is the utility of the row-player).

An example of such a payoff matrix is presented in Table 6.6. Each number represents the average utility obtained by the agent in the row-header, averaged over all scenarios in which it negotiated against the agent indicated in the column-header. For example, we see that *IlPadrino* obtained an average utility of 0.91 over all the sessions in which it negotiated against *MegaBarter300*. On the other hand, the average utility that *MegaBarter300* obtained in those same negotiations (against *IlPadrino*), was only 0.23.

We then need to determine the pure symmetric Nash equilibria of this game (recall that in Section 5.3.4 we argued that for symmetric games we are

	MegaBarter3000	IlPadrino	CrazyAgent	RandomAgent
MegaBarter3000	0.53	0.23	**0.80**	**0.90**
IlPadrino	**0.91**	**0.82**	0.20	0.10
CrazyAgent	0.40	0.23	0.30	0.35
RandomAgent	0.30	0.25	0.27	0.28

Table 6.6: Each cell contains the average score obtained by the agent in the row-header, when negotiating against the agent in the column-header. For example, IlPadrino scores an average utility of 0.91 when negotiating against MegaBarter3000. In each column, the highest value is highlighted in bold, indicating the best response against the agent in the column header.

only interested in *symmetric* equilibria), and among those equilibria we then determine which one achieves the highest utility (since we are talking about *symmetric* equilibria, both players achieve the same utility). Note that the fact that this equilibrium is pure, means that in this equilibrium each player selects exactly one agent. Furthermore, the fact that it is symmetric, means that both players choose the same agent. Therefore, a pure symmetric Nash equilibrium consists of exactly one agent. So, according to the empirical game-theoretical analysis, the best agent is the one that forms the pure symmetric Nash equilibrium with highest utility.

In order to find, for any given agent, the best response against that agent, we need to look at the column corresponding to that agent and then find the highest utility in that column. For example, in Table 6.6, we see that in the column representing *MegaBarter3000*, the highest utility is 0.91, which is achieved by *IlPadrino*. Therefore, *IlPadrino* is the best response against *MegaBarter3000*. Similarly, to find the best response against *CrazyAgent*, we see that the highest utility in that column is 0.80, which is obtained by *MegaBarter3000*, so therefore we conclude that *MegaBarter3000* is the best response against *CrazyAgent*. To make this easier to see the highest value in each column is highlighted in bold.

To find the pure symmetric Nash equilibria, we just have to see for which agents the best response falls on the diagonal of the matrix. In Table 6.6 we see that this is the case for IlPadrino: the highest value in that column is 0.82, which is achieved by IlPadrino itself. Therefore, the profile (*IlPadrino*, *IlPadrino*) forms a pure symmetric Nash equilibrium.

Unfortunately, however, empirical game-theoretical evaluation does have a number of disadvantages. Namely:

1. It allows us only to find the 'best' agent, but it does not allow us to

determine a further ranking among the agents. That is, it doesn't allow us to tell which agent is the second best or the third best.
2. The optimal symmetric Nash equilibrium may not be unique.
3. The optimal pure symmetric Nash equilibrium may actually be dominated by *mixed* symmetric Nash equilibrium.
4. EGTA requires a lot more data, compared to tournament evaluation, to get statistically significant results.
5. It is based on the assumption that the players are fully rational and have full knowledge of the payoff matrix which, one can argue, is not entirely realistic.

The first of these points is not a big issue if we are only interested in proving that our agent is the best. However, if, for example, we want to run a competition that awards a prize for the second and third-best agents, then EGTA cannot help us with that. Also, if our agent is not the best, then EGTA cannot really tell us how great the difference between our agent and the best agent is.

The second issue happens when we have two agents that each score exactly the same utility when negotiating against themselves. This is not a very big problem, because the chance of that happening is not very big, and moreover, the same problem can occur in a tournament evaluation just as well. In fact, this can happen in literally any evaluation method, because it is always possible that two agents are simply equally strong.

The third and fourth issues are more problematic ones.

Suppose that (*IlPadrino*, *IlPadrino*) is the optimal *pure* symmetric Nash equilibrium, but that there also exists a *mixed* symmetric Nash equilibrium, for which the two players achieve even higher utilities. For example, a strategy profile in which both players select *MegaBarter3000* with 60% probability and *CrazyAgent* with 40% probability. It is then questionable to say that *IlPadrino* is the game-theoretically best agent. After all, each player would be better off selecting the mixed strategy. But then there is no longer one unique agent that can be considered the best.

Ideally, we should therefore also check if there are any such *mixed* symmetric Nash equilibria and, if yes, check that they do not dominate the optimal pure symmetric equilibrium. This can be done using software libraries such as Gambit [46], but unfortunately this gets computationally very costly if there are many agents.

If it happens that the game does have one or more *mixed* symmetric Nash equilibria that dominate the pure ones, then we cannot really conclude anything from the evaluation. In this case, the EGTA does not consider any single agent to be the best, and instead the optimal strategy for Alice and Bob would be to flip a coin to choose between the agents that make up the optimal mixed symmetric Nash equilibrium.

The fourth issue is caused by the fact that for a tournament evaluation we only need to determine $|Ag|$ different numbers (one tournament score $U_{\underline{i}}$ for each agent), while an EGTA requires $|Ag| \times |Ag|$ different numbers (one $U_{\underline{i}}^{\underline{j}}$ for every pair of agents $(ag_{\underline{i}}, ag_{\underline{j}})$) and for each of these numbers the standard error needs to be sufficiently low. This means that we need to run a lot more negotiations for EGTA than for tournament evaluation, which can be very time-consuming.

Finally, the fifth issue is more of theoretical problem rather than a practical one. The problem here, is that if humans are not fully rational, or they do not have full knowledge of the payoff matrix, then they may not make optimal choices, and it may therefore be that the agent that forms a symmetric Nash equilibrium is actually a sub-optimal choice if our opponent is irrational. For example, if Bob is irrational and chooses *CrazyAgent*, then Alice would actually be better of with her original choice *MegaBarter3000* than with *IlPadrino*.

Exercise 19. Perform EGTA. Implement a program that, similarly to Exercise 15, does the following:

1. Read the text file from Exercise 14.
2. Based on the contents of that file, calculate the values $U_{\underline{i}}^{\underline{j}}$ for every pair of agents $(ag_{\underline{i}}, ag_{\underline{j}})$.
3. Display them in a table such as Table 6.5.
4. Determines and outputs the pure symmetric Nash equilibrium with highest utility values.

6.2.3 EGTA vs. Tournament Evaluation

In a certain sense, game-theoretical evaluation and tournament evaluation can be seen as two extreme ends of a spectrum. While game-theoretical evaluation is based on the assumption that all people are fully rational and have full knowledge of the experimental results, tournament evaluation is based on the opposite assumption that other people are completely irrational, or have absolutely no knowledge at all about the payoff matrix.

To see this, assume that Alice indeed does not apply any form of rationality. That is, she picks her agent completely at random. That means that for each of the four agents there is a 25% chance that Alice will pick that agent. Therefore, the optimal choice for Bob is to pick the agent that maximizes his expected utility against Alice's 'uniform' mixed strategy. For example, if Bob picks an agent $ag_{\underline{i}}$, then his expected utility would be:

$$E(u_{\underline{i}}) \quad = \quad \frac{1}{4}U_{\underline{i}}^{1} + \frac{1}{4}U_{\underline{i}}^{2} + \frac{1}{4}U_{\underline{i}}^{3} + \frac{1}{4}U_{\underline{i}}^{4}$$

but it is easy to see that this is exactly the tournament score $U_{\underline{i}}$ of agent $ag_{\underline{i}}$. In other words, Bob should pick the agent with the highest tournament score. So, we see that tournament evaluation indeed selects the agent that is optimal under the assumption that our opponent is not rational at all, or has no knowledge whatsoever about the payoff matrix.

Opinion. In my opinion, neither of the two methods is based on entirely realistic assumptions. On the one hand, people are often not perfectly rational and in a real-world situation they would probably not have perfect knowledge about how well every possible agent performs against every possible other agent.

On the other hand, however, I also don't think it's realistic to assume people would have absolutely no knowledge at all about how well each agent performs. Typically, if one agent clearly under-performs compared to other agents, then sooner or later people would stop using that agent. This is demonstrated by the fact that for many real-world applications their popularity follows a '*power law distribution*', which means that just a few apps are significantly more popular than others. For example, if you search the Internet for data about the most popular messaging apps, dating apps, web browsers, or any other type of application, then in almost all cases you will find a histogram that follows such a pattern. So, people clearly do not select their applications purely at random.

In other words, neither tournament evaluation, nor empirical game-theoretical evaluation gives us the definitive answer to the question which agent is the 'best'. I therefore recommend to always perform *both* evaluations. In the ideal case both methods yield the same result. If they give different results, however, then the answer to the question which agent is the best just depends on which set of assumptions you would consider more realistic in a specific domain of application.

6.3 Sequential Elimination Ranking

Previously, we have discussed the two most common methods to evaluate negotiation algorithms. We have argued that neither of the two are clearly the best, and that they are respectively based on two strictly opposing sets of assumptions, neither of which are entirely realistic.

Therefore, we would like to end this chapter by proposing a third, alternative method that lies somewhere in between tournament evaluation and EGTA. We will call it 'Sequential Elimination Ranking'. It should be noted that, to the best of our knowledge, this method has never actually been applied in the literature on automated negotiation. However, similar methods have been studied and applied extensively in other research topics, such as *computational social choice*. .

This method is based on the idea is that in a real-world situation people would be less likely to choose weaker agents. So, the final score of an agent should be weighted more heavily by the utility it obtained against the stronger opponents than by the utility it obtained against weaker opponents.

As usual, we assume we have some set of negotiation domains $\mathcal{D}$, and some set of agents Ag of size n (i.e. $n := |Ag|$). However, for reasons that will become clearly imminently, we will here denote the set of agents as Ag_n instead of Ag. Furthermore, for any $i \in \{1, 2, \ldots, n\}$ we will use the notation $ag_{[i]}$ to denote the agent that ranks in i-th place according to the Sequential Elimination Ranking.

Sequential Elimination Ranking works as follows.

1. First, we run a tournament, just like for tournament evaluation, in which all agents $ag \in Ag_n$ negotiate against each other (including against themselves), over all domains $D \in \mathcal{D}$.
2. We then calculate, for each pair of agents $(ag_{\underline{i}}, ag_{\underline{j}})$ the average utility obtained by agent $ag_{\underline{i}}$ against agent $ag_{\underline{j}}$, which we denote as $U(ag_{\underline{i}}, ag_{\underline{j}})$. That is:

$$U(ag_{\underline{i}}, ag_{\underline{j}}) = \frac{1}{|\mathcal{D}|} \sum_{d=1}^{|\mathcal{D}|} \Big(\frac{1}{2R^{d,\underline{i},\underline{j}}} \sum_{r=1}^{R^{d,\underline{i},\underline{j}}} u_1^{d,\underline{i},\underline{j},r} + \frac{1}{2R^{d,\underline{j},\underline{i}}} \sum_{r=1}^{R^{d,\underline{j},\underline{i}}} u_2^{d,\underline{j},\underline{i},r} \Big)$$

   Note that this is the same as Equation (6.21), except that we here use the notation $U(ag_{\underline{i}}, ag_{\underline{j}})$ instead of $U_{\underline{i}}^{\underline{j}}$.
3. Next, we can use these values to calculate for each agent $ag \in Ag_n$ its

tournament score, which we here denote as $U^n(ag)$:

$$U^n(ag) \quad := \quad \frac{1}{n} \sum_{ag' \in Ag_n} U(ag, ag')$$

4. We then determine the agent with the lowest tournament score, and we set its **rank** equal to n. In other words, this agent ends the tournament in last place. We denote this agent by $ag_{[n]}$.

$$ag_{[n]} \quad := \quad \arg\min\{U^n(ag) \mid ag \in Ag_n\}$$

5. We now regard $ag_{[n]}$ as 'eliminated' from the tournament, and we define the set Ag_{n-1} as the set of all remaining agents:

$$Ag_{n-1} \quad := \quad Ag_n \setminus \{ag_{[n]}\}$$

6. Now, we recalculate the scores of all the remaining agents $ag \in Ag_{n-1}$, by calculating the average of all the utilities they obtained against the other remaining agents (i.e. we are no longer counting the utilities they obtained against the eliminated agent). We will denote this new score as $U^{n-1}(ag)$:

$$U^{n-1}(ag) \quad := \quad \frac{1}{n-1} \sum_{ag' \in Ag_{n-1}} U(ag, ag')$$

7. We now again determine the agent with the lowest score (among the remaining agents), which we will denote as $ag_{[n-1]}$. This agent ends the tournament in second-last place.

$$ag_{[n-1]} \quad := \quad \arg\min\{U^{n-1}(ag) \mid ag \in Ag_{n-1}\}$$

8. This agent can now also be considered eliminated and we remove it from the set of remaining agents.

$$Ag_{n-2} \quad := \quad Ag_n \setminus \{ag_{[n]}, ag_{[n-1]}\}$$

9. We keep doing this over and over until only one agent is left. That is, in each iteration we calculate the scores of all remaining agents, by averaging only over the utilities obtained against those same remaining agents themselves and then determine the agent with the lowest score, which will be next to be eliminated.

10. Finally, the winner of the tournament is $ag_{[1]}$, the second-best agent is $ag_{[2]}$, etcetera.

Formally, for any $i \in \{1, 2, \ldots, n\}$ the agent $ag_{[i]}$ is defined by the following recursive set of equations:

$$\begin{aligned} ag_{[i]} &:= \arg\min\{U^i(ag) \mid ag \in Ag_i\} \\ U^i(ag) &:= \frac{1}{i} \sum_{ag' \in Ag_i} U(ag, ag') \\ Ag_i &:= Ag \setminus \{ag_{[i+1]}, ag_{[i+2]}, \ldots, ag_{[n]}\} \end{aligned}$$

The motivation for this method is as follows. Imagine a society in which many people use negotiation algorithms to do their negotiations for them. Initially there are, say, 10 different algorithms available, and initially each of them is equally popular. However, as time passes and people start using them more and more often, it quickly becomes obvious that some of these algorithms are stronger than others. Therefore, people will quickly abandon the weakest ones. As those weaker algorithms start disappearing, this will influence the results obtained by the other algorithms. That is, some algorithms that are very good at exploiting the weaker ones and that initially obtained good results, may now slowly start performing worse, because of the increasing lack of weaker opponents to exploit. So, again, as time passes, the weaker algorithms among those that are left will now also start declining in popularity. This process will continue until slowly but surely only one algorithm is left at the end, which can therefore be considered the 'best'.

Note that, on the one hand, this system does not assume that people simply choose their agents randomly, but on the other hand it also does not assume that all people have perfect knowledge of the full payoff matrix. Instead, regarding knowledge, it merely assumes that people know, at any moment, which agent is at that moment the weakest one available. Furthermore, regarding to rationality, it only assumes that nobody wants to use the worst agent available. So, indeed, we see that the assumptions underlying this method lie somewhere in between the assumptions underlying tournament evaluation and the assumptions underlying EGTA.

Moreover, this system is very robust against the presence of unrealistically weak agents. For example, if we add one extremely weak agent to the set of agents, then that agent will probably be eliminated in the first iteration and after that will no longer have any influence on the other agents' scores. Thus, the rankings of the other agents remain unaffected.

Exercise 20. Sequential Elimination Ranking. Implement a program that does the following:

1. Read the text file from Exercise 14.
2. Based on the contents of that file, calculate the value of $U(ag_{\underline{i}}, ag_{\underline{j}})$ for every pair of agents $(ag_{\underline{i}}, ag_{\underline{j}})$.
3. Calculate for each agent its rank according to the sequential elimination ranking, as explained in this section.
4. Display the results in a table.

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 Aydogan, 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] Shih-Fen Cheng, Daniel M Reeves, Yevgeniy Vorobeychik, and Michael P Wellman. Notes on equilibria in symmetric games. In Simon Parsons and Piotr Gmytrasiewicz, editors, *Proceedings of the 6th International Workshop On Game Theoretic And Decision Theoretic Agents GTDT*, pages 71–78, 7 2004.

[14] 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.

[15] 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.

[16] Dave de Jonge. A new bargaining solution for finite offer spaces. *Applied Intelligence*, 53(23):28310–28332, 2023.

[17] Dave de Jonge. Theoretical properties of the MiCRO negotiation strategy. *Autonomous Agents and Multi-Agent Systems*, 38(46), 2024.

[18] 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.

[19] 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, *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 404–412. ACM, 2021.

[20] Dave de Jonge, Filippo Bistaffa, and Jordi Levy. Multi-objective vehicle routing with automated negotiation. *Applied Intelligence*, 52(14):16916–16939, Nov 2022.

[21] 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.

[22] Dave de Jonge and Carles Sierra. D-Brane: a diplomacy playing agent for automated negotiations research. *Applied Intelligence*, 47(1):158–177, 2017.

[23] David M Diez, Christopher D Barr, and Mine Cetinkaya-Rundel. *OpenIntro statistics*, volume 4. OpenIntro Boston, MA, USA, 2012.

[24] 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.

[25] 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.

[26] David Freedman, Robert Pisani, and Roger Purves. *Statistics, 4th éd.* New York, WW Norton, 2007.

[27] 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.

[28] Katsuhide Fujita, Reyhan Aydogan, 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.

[29] Maria Jose Herrero. The nash program: non-convex bargaining problems. *Journal of Economic Theory*, 49(2):266–277, 1989.

[30] 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.

[31] 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.

[32] Ehud Kalai and Meir Smorodinsky. Other solutions to nash's bargaining problem. *"Econometrica"*, "43"(3):513–518, 1975.

[33] 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.

[34] C. E. Lemke and J. T. Howson, Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] J.F. Nash. The bargaining problem. *"Econometrica"*, "18":155–162, 1950.

[41] 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.

[42] Martin J Osborne and Ariel Rubinstein. *A course in game theory.* MIT press, 1994.

[43] Bram Renting, Dave de Jonge, Holger Hoos, and Catholijn Jonker. Analysis of learning agents in automated negotiation. Under review.

[44] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter.* The MIT Press, Cambridge, USA, 1994.

[45] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica: Journal of the Econometric Society*, pages 97–109, 1982.

[46] Rahul Savani and Theodore L. Turocy. Gambit: The package for computation in game theory, 2025.

[47] 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, *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 1163–1172. ACM, 2021.

[48] 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.

[49] Michael P. Wellman, Karl Tuyls, and Amy Greenwald. Empirical game theoretic analysis: A survey. *J. Artif. Intell. Res.*, 82:1017–1076, 2025.

[50] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[51] 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, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 432–438. IJCAI/AAAI, 2011.

[52] 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.

[53] 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.

[54] 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.