Strategy Configuration and Selection for Automated Negotiation Agents

by

Bram M. Renting

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday May 22, 2019 at 13:00.

Student number:	4081137	
Project duration:	March 2, 2018 – May 22, 2019	
Thesis committee:	Prof. Dr. ir. C.M. Jonker,	TU Delft, supervisor
	Prof. Dr. H.H. Hoos,	Leiden University
	Dr. M.M. de Weerdt,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Preface

In front of you lies the epitome and long-awaited end point of my studies. After my Bachelor's in Marine Technology, I can now say that I have successfully converted into a Computer Scientist. The topic of this thesis originates from my interests in multi-agent systems that I developed during my Master studies. The concept of autonomous multi-agent systems collaborating to achieve emergence fascinates me and I hope to see more of it in the near future.

Although this is merely a small step into the direction of that concept, I hope that my work will contribute to the automated negotiation research community, by providing new insights into the development of strategies and by lowering the threshold of developing dynamic strategies.

My gratitude goes to Catholijn for her guidance during my project, teaching me the ways of scientific research and letting me attend the International Joint Conference on Artificial Intelligence (IJCAI). Her ability to put the results of my work into perspective always boosted my motivation. My gratitude also goes to Holger, his insights in algorithm configuration and selection often left me mind blown on the train back home from Leiden University.

Finally, I would like to thank my parents for their patience and support during my studies. The unrestricted freedom they gave me to develop myself made me the person I am today. This thesis marks a milestone as, after all these years, it heralds the end of my life as a student.

Bram Renting
Delft, May 2019

Contents

Chapter 1

Introduction

Negotiation occurs in every aspect of our lives, to the extent that we are not always aware of it. Commercial negotiations are easily identifiable in human society. For example, buying a car, merging companies, or negotiating your salary. Non-commercial negotiation is much more invisible and embedded into our daily lives. For example, setting an appointment with another person, discussing where to go on a holiday, or participating in traffic. All problems that need to be solved while involving multiple people with colliding interests are a form of negotiation. Since negotiation is such a big part of everyday live, researchers have studied the topic of negotiation from different perspectives, as well as a science perspective [1], [2].

Since the 1980s researches have tried to design computer negotiators that can replace or assist humans in negotiation. Early adopters in this field are Smith, Sycara, Robinson, Rosenschein and Klein [3]–[8]. Many publications followed, improving the capabilities of computer negotiators with success. Computer negotiators are capable of reaching more optimal solutions compared to a negotiation between human negotiators [9]. However, there was a lack of a standardised testbed for negotiation, so researchers had to create their own. It made it difficult to compare the state of the art in automated negotiation, as testbeds were not compatible and negotiators were tested against a limited pool of other strategies. With current knowledge, we know that the success of a negotiator also depends on the strategy of the opponent [10]. In 2010 General Environment for Negotiation with Intelligent multi-purpose Usage Simulation [11] (GENIUS) was created to resolve the issue of incompatible negotiators and to provide a testbed for testing new developments in the field of automated negotiation. Alongside, the Automated Negotiating Agents Competition [12] (ANAC) was organised to stimulate negotiator development in the academic field. Today, the combined effort of GENIUS and ANAC resulted in a standardised testbed with more than one hundred opponents and negotiation problems to use for research [13].

Negotiators are still being developed every year for the ANAC and every year a new challenge is added that contestants have to cope with. The negotiators are generally hard coded pieces of software based on a strategy with parameters that must be set to characterise its behaviour. Developing a basic negotiator is not that difficult, however, developing a well performing negotiator is. Strategy and parameter configuration can be time consuming depending on the negotiation

settings. Especially when negotiation budgets or deadlines are bound to real time[1], which is the case in stock exchange negotiations and auctions, making computational speed-up unprofitable. Strategy tuning via trial and error on a large and diverse testbed is time consuming and impractical, so agents in the literature are optimised on smaller testbeds [14]. Some attempts were made to automate this process, often using genetic programming [15], but again only on much more specific and simplified environments. For instance, agents were only tested in one or two scenarios or only optimised against itself [16], [17], which is partly due to the lack of a general testbed. To our knowledge, no attempts have been made at automating this configuration problem on significantly sized training sets with a large range of scenarios as well as opponent strategies.

In this work, we make a first attempt at solving this automated configuration challenge for negotiation settings and the difficulties it brings. With as most important difficulty, the computational expense of testing a single configuration on a larger training set of negotiation settings. We recreate a negotiation agent from literature that is configured by hand, combine it with existing opponent learning techniques and create a configuration space of its parameters and strategic decisions. This negotiator is then configured automatically by an algorithm that is capable of training on subsets of training sets by extruding the results over the entire set based on similarities. This algorithm was successfully applied to configure solvers for Boolean Satisfiability (SAT) and Mixed Integer Programming (MIP) problems, which are known to be computationally expensive. The difference is that these problems are single-agent problems, while negotiation is a multi-agent problem. Our method overcomes this incompatibility by considering the opponent negotiators as part of the problem, making the problem instances a combined problem. The aim of our attempt is to create an agent that achieves good results on a wide variety of scenarios and opponents.

We distinguish between optimising a configuration to create a strategy that works well in general, and one that works well for specific settings. The latter is done in the configuration procedures using genetic programming as described earlier, resulting in strategies that are outperformed easily when used outside of their "comfort zone". In contrast, we attempt to configure a strategy that works well in general, but might still be outperformed in specific settings, as it is widely observed that no single strategy is optimal for all negotiation settings [11], [18]. Having a fixed strategy that performs well in general is beneficial when switching strategies is difficult (e.g. due to a lack of information), but if the aim is to improve our pay-off even further, then switching between strategies is a must. This introduces the problem of algorithm selection [19] into automated negotiation. Attempts on applying algorithm selection in automated negotiation have been made by Ilany et al. [18], [20], but they only selected a strategy based on the negotiation scenario without considering the opponent.

We introduce a method that learns from opponents while repeatedly encountering them in negotiation, using this additional information for algorithm selection. The problem that remains, is that of constructing a portfolio of algorithms to select from. Ilany et al. solved this by selecting from a pool of agents that competed in the ANAC 2012, depending on negotiation strategies developed by other people. This, yet again, introduces the question if these strategies are configured properly. So finally, we use HYDRA [21] to combine algorithm selection and configuration and fully automate the process of creating a portfolio of negotiation strategies that can be used for algorithm selection in automated negotiation. Only a single highly parameterised strategy is required to build an optimised and dynamic negotiation agent.

[1]The ANAC maintains a 3 minute deadline for a single negotiation session.

This thesis is structured as follows. In Chapter 2 we elaborate upon current literature in related topics. Chapter 3 describes the basics of negotiation theory and the structure of negotiation sessions. We end with a formal problem definition to introduce the goal of the thesis. In Chapter 4, we introduce the agent with a highly parameterised strategy that we used for configuration. For both configuration and algorithm selection we need a feature model of the negotiation settings. The features that we used are described in Chapter 5. The method of configuration and the results are presented in Chapter 6. Chapter 7 explains the algorithm selection method that we used, as well as the process of building a portfolio and the results of using this portfolio in negotiation. Finally we conclude with a discussion (Chapter 8) and a conclusion (Chapter 9).

Chapter 2

Related work

In this chapter we discuss related work in the field on algorithm configuration and selection, as well as some past applications in the research area of automated negotiations.

2.1 Algorithm configuration

In literature, algorithm configuration is also referred to as parameter tuning or hyperparameter optimisation. I can be formally described as follows: given a parameterised algorithm A, a set of problem instances I and a cost metric c, find parameter settings of A that minimise c on I [22]. The configuration problem occurs for example in solvers for MIP problems [23], neural networks, classification pipelines, and every other algorithm that contains performance-relevant.

These configuration problems can be solved by basic approaches such as manual search, random search, and grid search, but over the years researchers developed more intelligent methods to obtain the best possible configuration for an algorithm. Two separate part within these methods can be identified: how new configurations are selected for evaluation and how a set of configurations is evaluated.

F-Race [24] races a set of configurations against each other on an incremental set of target instances and drops low performing configurations in the process. This saves computational budget as not all configurations have to be tested on the full target instance set. The set of configurations to test can be selected either manually, as a grid search, or at random. Balaprakash et al. [25] extended upon F-Race by implementing it as a model-based search [26], which iteratively models and samples the configuration space in search of promising candidate configurations.

ParamILS [27] does not use a model, but instead performs a local tree search operation to iteratively find better configurations. Like F-Race, ParamILS is capable of eliminating low performing configurations without evaluating them on the full set of instances.

Finally, a last popular method of algorithm configuration is GGA [28], which makes use of evolutionary programming to find configurations that perform well. This method does not model the configuration space and has no method to eliminate low performing configurations early.

2.2 Algorithm configuration in negotiation

Earlier attempts for solving the algorithm configuration problem in automated negotiations mostly used basic approaches, such as manual and grid search. To the best of our knowledge, the only advanced method used to automatically set negotiation strategies is the genetic algorithm.

Matos et al. [14] encoded a mix of baseline tactics as an chromosome and optimised it using a genetic algorithm. They assumed perfect knowledge of opposing party's preferences and tested against itself on a single negotiation domain. Eymann [16] encoded a more complex strategy as a chromosome with 6 parameters, again only testing its performance on itself and using the same domain. Dworman et al. [17] implement the genetic algorithm in a coalition game with 3 players, with a strategy in the form of a hard coded if-then-else rule. The parameters of the rule are implemented as a chromosome. The strategy is tested against itself on a coalition game with varying coalition values.

Lau et al. [29] use a genetic algorithm to explore the outcome space during a negotiation session, but do not use it to change the strategy.

2.3 Algorithm selection in negotiation

Not all algorithms are well suited to solve all target problem, which is widely observed in classification problems. To exploit differences between individual problem instances, algorithm selection is applied [19]. Algorithm selection is a classification problem itself: What algorithm to choose given the faced problem instance (features). A popular application of algorithm selection methods is the selection of SAT, Answer Set Programming (ASP), and Constraint Satisfaction Problem (CSP) solvers [30]–[32].

To the best of our knowledge, only one attempt is made to apply algorithm selection methods in automated negotiation. Ilany et al. [18], [20] used a set of past ANAC strategies and predicted which strategy would perform best given the negotiation scenario. They would then enter that strategy into the negotiation session. Although they were capable of improving the pay-off of the agent, they were unable to win the ANAC.

Chapter 3

Preliminaries

This chapter provides an overview of the terminology and symbols that are used throughout the thesis. The second part provides a formal problem definition.

3.1 Terminology

The idea behind automated negotiation is to replace human negotiators by computers. Here, humans are replaced by software agents that act on their behalf. We call these software agents *parties*, *negotiation agents* or simply *agents*. Agents that represent opposing parties in negotiation are also referred to as *opponents*.

We refer to a *multilateral* negotiation as a negotiation between three or more parties. However, this thesis focusses solely on negotiations between two parties, which is known as *bilateral* negotiation. The software platform that we use for agent construction and testing is GENIUS [11], which contains all the necessary components to setup a negotiation, allowing us to focus only on agent construction.

There are three component types that form a *negotiation setting* [33]. Their relations are illustrated in Figure 3.1:

- Negotiation scenario, we denote a set of scenarios by S with $s \in S$ as variables
- Negotiation protocol
- Negotiation agents, we denote a set of agents by A with $a \in A$ as variables

3.1.1 Negotiation scenario

The protocol defines the rules, the agents perform the interactions, the last component that is required is the negotiation problem itself along with details on the preferences of every party. This

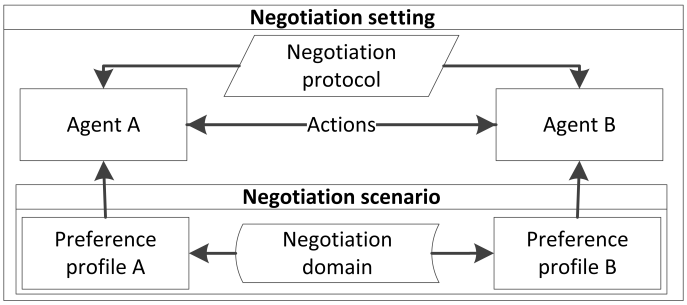


Figure 3.1: Overview of the components of a negotiation setting (bilateral) [33]

is called the negotiation scenario, which is discussed in this section. The first part focusses on the *negotiation domain* that defines all the possible outcomes of the negotiation. The second part focusses on the *preference profiles*, which describes the preferences of both agents.

Negotiation domain

The negotiation domain is a description of the negotiation problem that has to be solved. A negotiation domain is built up by one or more *issues* that form the issue set I. For each issue, a solution must be agreed upon by both agents. Every continuous or integer issue can be interpreted as a discreet issue, which we do in this thesis. All the possible solutions within an issue are called *values* and the set of values within an issue is denoted by V.

All possible solutions in the negotiation domain are called *outcomes* and the agreed outcome at the end of a negotiation is called the *agreement*. The total set of possible outcomes of the negotiation domain is called the *outcome space* and is denoted by Ω, where a single possible outcome is denoted by $\omega \in \Omega$. The total amount of possible outcomes in a negotiation domain is given by Equation 3.1.

$$|\Omega| = \prod_{i \in I} |V_i| \tag{3.1}$$

where:

V_i : The set of values V of issue i

Preference profiles

The negotiations are performed under incomplete information, meaning that an agent does not know the preferences of an opponent. The negotiation domain is common knowledge for both agents in the negotiation. The preferences of a party are defined in a preference profile and thus private information for both parties. The profile contains all agent specific details regarding pay-off, which we call *utility*. The achieved utility of a party calculated is after the negotiation has ended.

There are three types of components that form a preference profile:

1. Utility function
2. Discount factor
3. Reservation utility

Utility function A utility function is defined in the preference profile by specifying a *evaluation function* and *issue weight* for every issue in the domain. This type of utility function is also know as the *linear additive utility function* and is commonly used in the literature. It provides information to map any outcome $\omega \in \Omega$ to a utility following Equation 3.2. In this thesis, we will limit ourselves to this type of utility function.

$$u(\omega) = \sum_{i \in I} w_i * e_i(\omega_i) \tag{3.2}$$

where:

u : Utility function, where $u : \Omega \rightarrow [0, 1]$
w_i : *Issue weight* as a measure of issue importance, where $\sum_{i \in I} w_i = 1$ and $w_i \in [0, 1]$
e_i : *Evaluation function* of an issue i, where $e_i : \omega_i \rightarrow [0, 1]$
ω_i : Sub-outcome for issue i in the outcome ω

We use $u(\omega)$ to indicate our utility function and $u_o(\omega)$ to indicate the opponents utility function. As the opponents utility function is not common knowledge, we add a hat to our notation to indicate predictions ($\hat{u}_o(\omega)$).

Discount factor The discount factor is used in negotiation scenarios with time pressure. It reduces the utility of an agent when time passes by Equation 3.3, putting pressure on reaching an agreement quickly.

$$u^d(\omega) = u(\omega) * d^t \tag{3.3}$$

where:

u^d : Discounted utility function
d : Discount factor, where $d \in [0, 1]$
t : Time, where $t \in [0, 1]$

Reservation utility The reservation utility, which is also referred to as reservation value, is set to implement the Best Alternative To a Negotiation Agreement [34] (BATNA). In GENIUS it is handled as the achieved utility when no agreement is reached between agents. Typically, an agent should not consider any outcome below the reservation utility. If a discount factor is applied to the negotiation scenario, it also reduces the reservation utility over time similar to Equation 3.3.

In this thesis, we simplify by setting both the discount factor and reservation utility to zero for all scenarios.

Visualising the outcome space

The outcome space is multidimensional, which makes it difficult to visualise. However, using the preference profiles, every outcome can be mapped to a utility pay-off for both party. This allows us to map the outcome space to two values in bilateral negotiation, a utility for our agent $u(\omega)$ and a utility for the opponent $u_o(\omega)$. We can now visualise the outcome space in two dimensions based on the utilities of both parties. The outcome space is plotted in Figure 3.2 along with some points of interests.

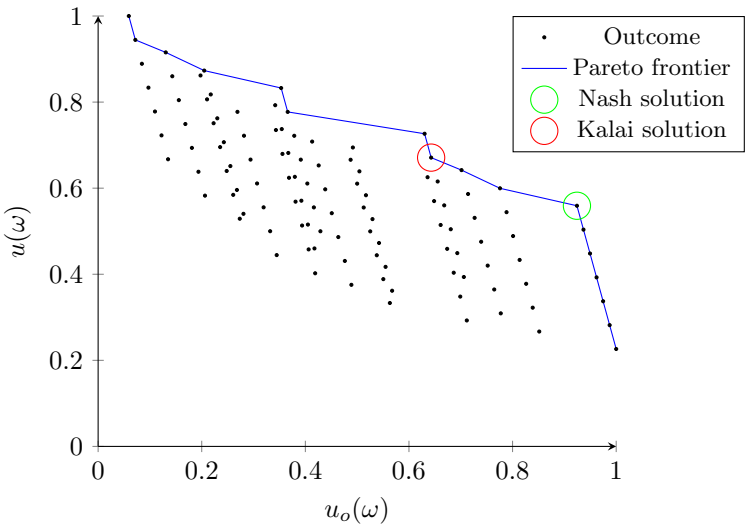


Figure 3.2: Outcome space plotted by utility

Pareto frontier The Pareto frontier is the set of Pareto optimal outcomes Ω_p. It is the set of outcomes that are not strictly dominated by another outcome, i.e. it is the set of outcomes for which there does not exist another outcome that improves the utility for at least one party without making the other party worse off [1]. A formal description of the Pareto set in Figure 3.2 is given in Equation 3.4 [33].

$$\begin{aligned}\Omega_p = \{\omega \in \Omega \mid \neg\exists\omega' \in \Omega : (u(\omega') > u(\omega) \wedge u_o(\omega') \geq u_o(\omega)) \\ \vee \\ (u(\omega') \geq u(\omega) \wedge u_o(\omega') > u_o(\omega))\}\end{aligned} \tag{3.4}$$

Nash solution The Nash solution [35] of a negotiation problem balances between individual utility and social welfare. In our bilateral negotiation case it is defined by Equation 3.5.

$$\omega_{Nash} = \arg\max_{\omega\in\Omega} (u(\omega) * u_o(\omega)) \tag{3.5}$$

Kalai solution The Kalai-Smorondinsky solution [36] maintains the ratio between utilities of parties while being Pareto optimal. As utility is normalised by default within GENIUS, we can define the Kalai solution by Equation 3.6.

$$\omega_{Kalai} = \arg\min_{\omega \in \Omega_p} |u(\omega) - u_o(\omega)| \tag{3.6}$$

3.1.2 Negotiation protocol

The negotiation protocol defines the manner of negotiation by setting the rules. These rules restrict for example the actions that can be performed by the agents or the deadline for the negotiation. A popular protocol that is widely used in automated negotiation literature is the *Alternating Offers Protocol*. In this thesis, we will use a generalised version of this protocol called Stacked Alternating Offers Protocol [37] (SAOP), which is an extension of the Alternating Offers Protocol that can also be used for multilateral negotiation. SAOP puts agents around a virtual table and lets them take clock-wise turns in rounds. At every turn, an agent can perform one of three actions:

1. Make a(n) (counter) *Offer*
2. *Accept* an offer (not possible if the agent must make the opening offer)
3. *End* negotiation (walk away)

An action is visible to all agents that are sitting at the table. Every time an agent makes an offer, the last offer is wiped of the table and replaced by the new offer. This process is continued until either, all agents accept an offer, an agent ends the negotiation, or if the deadline is reached (if applicable). The last two cases result in a failure of reaching an agreement and cause the agents to receive no pay-off (we do not set a reservation utility).

We can set a deadline that is based on real time, or based on played rounds around the table. We will follow the line of the ANAC by setting a real time based deadline. Note that this introduces a dependency on calculation efficiency of the agents. We represent time by $t' \in [0, t_d]$, where t_d is the real time deadline. Within GENIUS, time is normalised simplifying it to $t = \frac{t'}{t_d} \in [0, 1]$. A pseudo code description of SAOP is given in Algorithm 1.

We introduce the list offered outcomes by the opponent H^t at time t, which is a simplified version of the *negotiation thread* [33] or *negotiation dance* [1]. It is an incremental list that grows as offers are received. We present the formulation in Equation 3.7.

$$H^t = \left[x^{t_1}, x^{t_2}, x^{t_3}, \ldots, x^{t_n}\right] \tag{3.7}$$

where:

x^{t_k} : An offer made by the opponent, it represents an outcome $\omega \in \Omega$
t_k : Timestamp of the offers, where $t_k \geq t_l$ for $k \geq l$, and $t_k \leq t$

Algorithm 1 Stacked Alternating Offers Protocol [37] (SAOP)

Input	A	Set of agents
Variables	a	Agent
	ω	Outcome that lies on the table (last offer)
	A_{agree}	Set of agents that agreed with ω

```
possibleActions ← {Offer, End}
loop until EndNegotiation
    for a ∈ A do
        Action ← a.chooseAction(possibleActions)
        messageAgents(A, Action)
        if Action == Offer then
            possibleActions ← possibleActions ∪ {Accept}      ▷ Accept possible after first offer
            ω ← Action.getOutcome()
            A_agree ← ∅
        else if Action == Accept then
            A_agree ← A_agree ∪ {a}
            if |A_agree| == |A| − 1 then
                EndNegotiation(ω)                             ▷ End with agreement ω
                break
        else if Action == End then
            EndNegotiation(null)                              ▷ End without agreement
            break
        if getTime() ≥ 1 then
            EndNegotiation(null)                              ▷ End without agreement
            break
```

3.1.3 Negotiation agents

Negotiation agents are autonomous pieces of software that are capable of negotiating conform the style that is imposed by the negotiation protocol and the negotiation scenario. They produce the negotiation activity and represent a person/company/entity who's interests they try to serve. They make the decision whether to concede towards the opponent, maintain their ground or walk away from the negotiation. Baarslag separates negotiation agents in three components, also referred to as BOA-components [33]:

- *Bidding strategy*
- *Acceptance strategy*
- *(optionally) Opponent model*

Bidding strategy

The bidding strategy is the most important part of the agent in terms of agent performance [38], as it determines what to offer, when, and why. There are two common tactics implemented in literature based on time or opponent behaviour called: *time-dependent tactics* and *behaviour-dependent tactics*.

Time-dependent tactics Time-dependent tactics aim to offer outcomes near a utility target that varies over time $u_t(t)$. Agents usually start with a higher utility target at the start of a negotiation ($t = 0$) and decrease this target as time approaches the deadline ($t = 1$) due to the risk of reaching no agreement at all. A popular utility target function is presented in Equation 3.8 [39], which can be set to various behaviours depending on its parameters. We present some examples of utility target curves with a variety of parameter settings in Figure 3.3.

$$u_t(t) = P_{min} + (P_{max} - P_{min}) * (1 - F(t)) \tag{3.8}$$

$$F(t) = k + (1 - k) * t^{\frac{1}{e}} \tag{3.9}$$

where:

P_{min} : Lower bound of utility target, where $P_{min} \in [0, 1]$
P_{max} : Upper bound of utility target, where $P_{max} \in [0, 1]$
k : Factor of initial utility target between P_{max} and P_{min}, where $k \in [0, 1]$
e : Factor to control the shape of the utility target curve, where $e \in (0.\infty)$. Boulware if $0 < e < 1$, linear conceder if $e = 1$, conceder if $e > 1$.

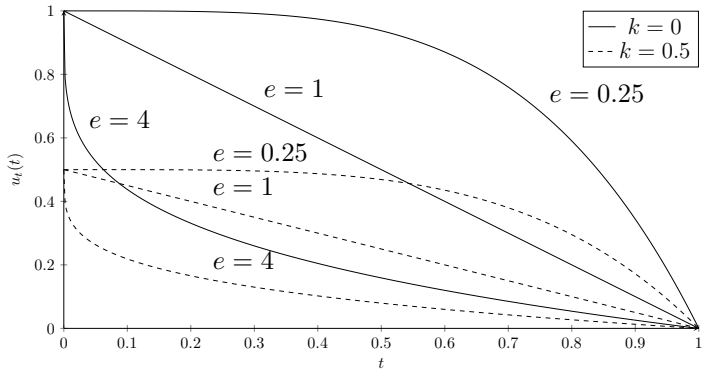


Figure 3.3: Examples of utility target curves with varying k and e. ($P_{max} = 1 \quad P_{min} = 0$)

Behaviour-dependent tactics Behaviour-dependent tactics respond to the behaviour of the opponent by mirroring their concessions. One of the best known behaviour-dependent tactic is *Tit-For-Tat*, which can be separated into three families [39]:

1. *Relative Tit-For-Tat*, where the agent mimics the opponents concessions based on percentages. e.g. The opponent conceded 5% between the last two offers, so we also concede by 5%.
2. *Random Absolute Tit-For-Tat*, where the agent mimics the opponent by absolute concessions and adds a random amount. e.g. The opponent conceded by 2$ between the last two offers, so we concede by 2 ± 0.5\$.
3. *Averaged Tit-For-Tat*, which is similar to the *Relative Tit-For-Tat* tactic, but calculates the average percentage of concession over a window in the past.

Baseline tactics There is a number of very simplistic baseline tactics that are often used as baselines in agent performance analytics. These include basic time-dependent tactics with no other intelligent functionality that are classified according to their settings of the target utility curve Equation 3.8: *Boulware* if $0 < e < 1$, *Linear conceder* if $e = 1$, *Conceder* if $e > 1$. There is also the *Hardliner* strategy (or Hardball strategy [40]), which persistently offers its own best outcome and does not concede at all. The final baseline tactic is the *Random walker* (or Zero Intelligence strategy [41]), that only makes random offers.

Acceptance strategy

Negotiating does not only require an agent to make offers, but also requires agents to accept offers of opponents. After all, a negotiation without parties that accept will always result in failure to reach an agreement. Within its turn, an agent has to decide whether to accept an outcome that is offered by the opponent, or to make a counter offer. Analysis of the ANAC indicated four regularly used basic acceptance conditions [42] described below. Out of these four conditions, most agents use either one of the conditions, or a combination of the conditions.

1. The agent compares the last offer of the opponent against his upcoming offer and decides whether to accept or not (known as AC_{next}).
2. The agent compares the last offer of the opponent against his own last offer and decides whether to accept or not (known as AC_{prev}).
3. The agent accepts based on a fixed pay-off constant (known as AC_{const}).
4. The agent accepts after a fixed amount of time has passed (known as AC_{time}).

3.1.4 Special outcomes

We will define a few special outcomes that will be used throughout this thesis. A visualisation of the special outcomes is presented in Figure 3.4, which uses the same example as Figure 3.2.

The best outcome for our agent ω^+ that maximises the opponent's utility:

$$\begin{aligned} \Omega^+ &= \arg\max_{\omega \in \Omega} u(\omega) \\ \omega^+ &= \arg\max_{\omega \in \Omega^+} u_o(\omega) \end{aligned} \tag{3.10}$$

The worst outcome for our agent ω^- that maximises the opponent's utility:

$$\begin{aligned} \Omega^- &= \arg\min_{\omega \in \Omega} u(\omega) \\ \omega^- &= \arg\max_{\omega \in \Omega^-} u_o(\omega) \end{aligned} \tag{3.11}$$

The best outcome for the opponent ω_o^+ that maximises our utility:

$$\begin{aligned} \Omega_o^+ &= \arg\max_{\omega \in \Omega} u_o(\omega) \\ \omega_o^+ &= \arg\max_{\omega \in \Omega_o^+} u(\omega) \end{aligned} \tag{3.12}$$

The worst outcome for the opponent ω_o^- that maximises our utility:

$$\begin{aligned} \Omega_o^- &= \arg\min_{\omega \in \Omega} u_o(\omega) \\ \omega_o^- &= \arg\max_{\omega \in \Omega_o^-} u(\omega) \end{aligned} \tag{3.13}$$

We define the same principles for the *offered* outcomes by the opponent by replacing the outcome space Ω with the negotiation thread H^t:

$$x^+ = \arg\max_{x \in H^{t+}} u_o(x) \qquad H^{t+} = \arg\max_{x \in H^t} u(x) \tag{3.14}$$

$$x^- = \arg\max_{x \in H^{t-}} u_o(x) \qquad H^{t-} = \arg\min_{x \in H^t} u(x) \tag{3.15}$$

$$x_o^+ = \arg\max_{x \in H^{t+}_o} u(x) \qquad H^{t+}_o = \arg\max_{x \in H^t} u_o(x) \tag{3.16}$$

$$x_o^- = \arg\max_{x \in H^{t-}_o} u(x) \qquad H^{t-}_o = \arg\min_{x \in H^t} u_o(x) \tag{3.17}$$

The last offered outcome by the opponent:

$$x^{last} = \left\{ x^{t_n} \in H^t : n = |H^t| \right\} \tag{3.18}$$

We like to introduce a *fictional* average outcome $\bar{\omega}$ and expand the utility function u with this outcome. This outcome generally does not exist, but we define it as an outcome that would result in the average utility of the entire outcome set for both parties:

$$\begin{aligned} u(\bar{\omega}) &= \frac{1}{|\Omega|} \sum_{\omega \in \Omega} u(\omega) \\ u_o(\bar{\omega}) &= \frac{1}{|\Omega|} \sum_{\omega \in \Omega} u_o(\omega) \end{aligned} \tag{3.19}$$

Finally we introduce the similar *fictional* average offer $\bar{x}$, such that:

$$\begin{aligned} u(\bar{x}) &= \frac{1}{|H^t|} \sum_{x \in H^t} u(x) \\ u_o(\bar{x}) &= \frac{1}{|H^t|} \sum_{x \in H^t} u_o(x) \end{aligned} \tag{3.20}$$

3.2 Problem definition

In this section, we describe the problems that we try to solve in this thesis. In the first part, we will present the two main problems and elaborate on them in a top-down approach. In the last section, we will formally define the problems that lie at the basis of the overarching goal.

3.2.1 Algorithm configuration

The negotiation agents in the GENIUS environment are mostly based on a fixed strategy, hand designed by competitors in ANAC. These agents almost always contain parameters that are set

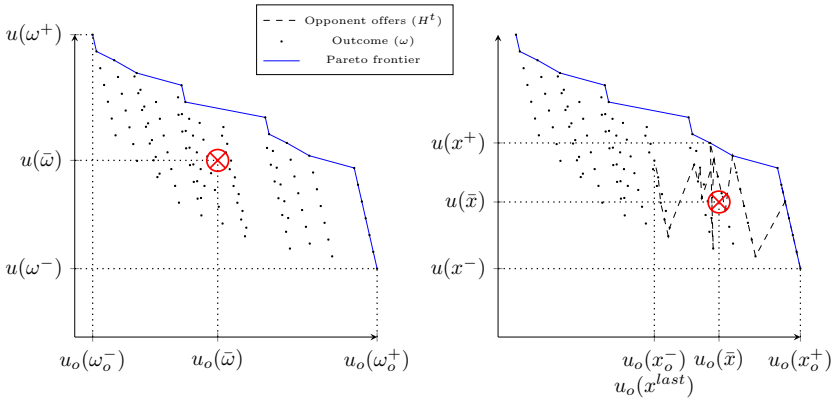


Figure 3.4: Special outcomes visualised

by trial and error, despite the abundance of automated algorithm configuration techniques (e.g. Genetic Algorithm [15]). Manually exploring configuration spaces rarely leads to optimal configurations, so automatic configuration seems a better solution. A few attempts were made in the past to automate this process as discussed in Chapter 2, but on very specific problem sets. Tuning an agent on a larger representative training set of opponents and scenarios is much more time consuming, as many automatic configuration algorithms require to evaluate a challenging configuration on the full training set. To illustrate, obtaining the performance of a configuration on the training set that we use in this thesis Table 3.1 would already take 18.5 hours, regardless of the hardware due to the real-time deadline. These conventional methods of algorithm configuration are therefore impractical.

Description	**Value**
agents	20
scenarios	28
deadline	60 $[s]$ (ANAC = 180 $[s]$)

Table 3.1: Training set for agent configuration

3.2.2 Algorithm selection

Assuming we are able to tackle the problem of automated algorithm configuration, we are still far from producing an optimal negotiation agent. So far, it has been observed that there is no single best strategy optimal for all negotiation settings [11]. Our agent needs to select a strategy based on the negotiation setting if we are to increase its performance further. This introduces the problem of algorithm selection[19] to our agent. In [43], four conditions are formulated that a problem should hold to be amenable to algorithm selection.

1. There are multiple instances of the problem with diverse complexities.
2. There exists a portfolio of algorithms for solving the problems with diverse complexity and performance.
3. There exist general and well defined metrics for measuring algorithm performance on the problem.
4. There is a set of known features that characterise problem instances that can be computed off-line and that correlate with the computational hardness/complexity of the problem.

Multiple instances

Every negotiation setting, which is the combination of an opponent and a scenario, is considered to be a problem instance.

Portfolio of algorithms

A portfolio of strategies can be constructed by selecting agents that already exist within GENIUS, which is the approach used (successfully) by Ilany et al. [18]. However, this relies again on manually tuned agents that have been submitted in the ANAC. Combining algorithm configuration with selection can be a solution to this problem by configuring the strategy on a subset of the negotiation instances.

Performance measure

The performance measure is well defined in the negotiation problem. It is the utility obtained by the agent at the end of the negotiation.

Instance features

To actually perform the selection, we need a feature model of the negotiation setting that has to be performed. In bilateral negotiation there are two components that impact the negotiation, the opponent and the scenario. Modelling the scenario by features is already done in previous work [18], which we will follow. Modelling opponents by features is done on a smaller scale for opponent classification (e.g. Concession Rate [33]), but not for the sole purpose of strategy selection. We like to point out that the opponents and scenarios are independent of each other and can be combined in every possible manner to create a new negotiation problem. This implies that their features should also be independent, i.e. no scenario information should be hidden in the opponent features and vice verse. Failing to achieve that introduces noise in the feature model of the negotiation setting.

3.2.3 Formal problem definitions

Problem 1: Dynamic Agent

Before we can optimise a configuration, we need something to configure. For the negotiation problem, we must construct a Dynamic Agent $DA(\theta)$ that varies in strategy based on its configuration θ. The range of strategies must be as broad as possible to ensure that there are well performing strategies within the configuration space. We deem the Dynamic Agent successful if there is a configuration $\theta \in \Theta$ such that our agent $DA(\theta)$ outperforms the test set of opponent agents A_{test} on the test set of scenarios S_{test}. We define outperforming as achieving a higher average utility in a bilateral ANAC-like bilateral tournament.

Problem 2: Feature construction

Suppose we have a set of opponent agents A and a set of scenarios S, such that combining a single agent $a \in A$ and a single scenario $s \in S$ creates a new negotiation setting or instance $\pi \in \Pi$. Can we derive a set of features for both the opponent agent and the scenario, that are independent of each other and provide insight in the complexity of the negotiation instance.

As this is difficult to prove, we want to test this by analysing if the following two statements are true:

1. The feature set helps the automated algorithm configuration method in converging to a better configuration while maintaining the computational budget.
2. The feature set contains sufficient information, such that algorithm selection improves the utility pay-off of our agent.

Problem 2.1: Preference estimation model accuracy Knowing the preferences profile of the opponent helps us in modelling the opponents behaviour by features. However, the negotiation happens under incomplete information, making such information unavailable. We try to estimate opponent preferences by taking an opponent preference estimation method from literature and use it for opponent feature construction. Inaccuracies in the model will lead to noise in these features, so we aim to improve the accuracy.

We take a preference estimation model from literature that performs well and use two accuracy measures that correlate with agent performance: *Pearson Correlation of outcomes* and *difference in Pareto frontier surface* [44]. Can we improve the preference estimation model based on these two measures?

Problem 3: Automated configuration

Suppose we have an agent with a dynamic strategy called Dynamic Agent $DA(\theta)$, with a parameter setting θ. We want to configure this agent, such that it performs generally well, using automated algorithm configuration methods. More specifically, we want the agent to perform generally well in bilateral negotiations with a real time deadline of $60[s]$. To do so, we take a diverse and large

set of both agents A_{train} of size $|A_{train}| = 20$ and scenarios S_{train} of size $|S_{train}| = 56$ that we use for training, making the total amount of training instances $|\Pi_{train}| = |A_{train}| * |S_{train}| = 1120$. Running all negotiation settings in the training set would take 1120 minutes or $\sim$ 18.5 hours, regardless of the hardware as we use real time deadlines.

Now suppose we have a setting for the Dynamic Agent based on the original literature θ_{lit} and a setting that is hand tuned based on intuition, modern literature and trial-and-error θ_{hand} that we consider baselines. Can we automatically tune a parameter configuration θ_{opt} that outperforms the baselines on average on a never before seen test set of negotiation instances?

Problem 4: Portfolio creation

Suppose we have an agent with a dynamic strategy called Dynamic Agent $DA(\theta)$, with a parameter configuration $\theta \in \Theta$. We use this agent in a negotiation instance π and obtain a utility of $o(DA(\theta), \pi)$. Since we know that no single strategy is optimal for all instances and that we must revert to algorithm selection, we need more than a single configuration.

How do we create a portfolio of configurations $\vec{\Theta}$ consisting of configurations that outperform each other on specific subsets of negotiation instances $\Pi' \subset \Pi$. i.e. Assuming that we are capable of selecting the best algorithm for every negotiation instance (oracle selector $\theta_\pi = OR(\vec{\Theta}, \pi)$), $DA(OR(\vec{\Theta}, \pi))$ must outperform $DA(\theta_i)$ on average over the testing set of negotiation instances Π_{test}, for all $\theta_i \in \vec{\Theta}$.

Problem 5: Algorithm selection

Suppose we have an agent with a dynamic strategy called Dynamic Agent $DA(\theta)$, with a portfolio of configurations $\vec{\Theta} = \{\theta_1, \theta_2, \ldots, \theta_n\}$ and θ_1 is the single best performing algorithm. Can we apply an algorithm selection method $AS(\vec{\Theta}, \pi) = \theta_\pi$ that selects a configuration θ_π from $\vec{\Theta}$ based on the negotiation instance π, such that $DA(AS(\vec{\Theta}, \pi))$ outperforms the single best strategy $DA(\theta_1)$ in an ANAC-like bilateral tournament?

Chapter 4

Dynamic agent

In this section, the Dynamic Agent that is used for optimisation is discussed. We define the following main requirements:

- The agents bidding behaviour must be as flexible as possible on the spectrum of competitiveness and conceding behaviour depending on a parameter configuration.
- The agents accepting behaviour must be flexible.
- Negotiation domains can be vast, containing many possible outcomes. To prevent every negotiation agent from needing a powerful computer, the agent should be capable of exploring a large outcome space without relying of brute force methods.
- The agent needs to build an opponent model during negotiations. This opponent model can be used to estimate opponent preferences in an attempt to approach Pareto optimal agreements, and is also required for opponent feature extraction for future encounters.

In Section 4.1 we will discuss the bidding strategy that solves the first requirement, the statements we make are based on the assumption of a perfect opponent model. In Section 4.2 we explain the implemented acceptance strategy. Finally, the required opponent model is discussed in Section 4.3.

4.1 Bidding strategy

Lau et al. [29] proposed a bidding strategy that uses a heuristic to assign a fitness value to every outcome. When the agent needs to choose an action, the outcome with the highest fitness at that time is selected as potential offer. Before making the actual offer, the agent checks if the opponents offer is acceptable via its acceptance strategy. In this section we first discuss the fitness function. The second part of this section elaborates on the process of exploring the outcome space without relying on brute force.

4.1.1 Fitness function

The main difference between this fitness and our utility is that it also considers opponent preference. The paper was published before the GENIUS framework existed, so some modifications are in place. We will start by presenting the original fitness function from [29] and expand upon it until we reach the current implementation.

$$\begin{aligned} f(\omega, t) &= \delta * TP(t) * \frac{u(\omega)}{\max_{\omega \in \Omega} u(\omega)} + (1 - \delta * TP(t)) * \left(1 - \frac{dist(\omega, x^{last})}{MaxDist(\Omega)}\right) \\ TP(t) &= 1 - \left(\frac{\min(t, t^d)}{t^d}\right)^{\frac{1}{e}} \end{aligned} \tag{4.1}$$

where:

δ	: Trade-off factor to control the relative importance of optimising one's own pay-off or reaching a deal, where $\delta \in [0, 1]$
TP	: Time-pressure function
x^{last}	: Last offered outcome by opponent (Equation 3.18)
$MaxDist(\Omega)$	: Maximum distance in outcome space
t	: Time
t^d	: Time of deadline
e	: Factor to control an agents eagerness to concede relative to time, where $e \in (0.\infty)$. Boulware if $0 < e < 1$, linear conceder if $e = 1$, conceder if $e > 1$.

Both the time in the time pressure function and our utility in the fitness function are normalised. This is a default setting in GENIUS and can be simplified by using the normalised values. The last part in the fitness function tries to minimise the normalised distance between the considered outcome and the opponents last offered outcome in an attempt to cater to the opponent. Such a distance measure will, however, make less sense in a discrete non-linear scenario as changing the value of a single issue can make a large difference in utility. Instead of using a distance measure, we will use the preference estimation (Section 4.3.1) to minimise the difference in terms of estimated opponent utility. It resembles the original idea, but we consider it to be a more robust approach. The fitness function now becomes:

$$\begin{aligned} f(\omega, t) &= \underbrace{\delta * TP(t) * u(\omega)}_{f_{my}(\omega,t)} + \underbrace{(1 - \delta * TP(t)) * (1 - |\hat{u}_o(\omega) - \hat{u}_o(x^{last})|)}_{f_o(\omega,t)} \\ TP(t) &= 1 - t^{\frac{1}{e}} \end{aligned} \tag{4.2}$$

where:

$\hat{u}_o$	: Estimated opponent utility function (Equation 4.30)
t	: Time, where $t \in [0, 1]$

We will defined the fittest outcome that should be selected for offering by Equation 4.3.

$$\omega_{fit}(t) = \arg\max_{\omega \in \Omega} f(\omega, t) \tag{4.3}$$

Limitations

We consider Equation 4.2 to be the baseline fitness function from the literature, as it resembles the original fitness function most. For convenience we split up the fitness function into an opponent part, and a part for our own agent as displayed in Equation 4.2. The first two rows of Figure 4.1 represent a visualisation of the fitness function. There are two points of criticism regarding the opponent part of this function:

- By taking the absolute difference in utility between the opponents last offered outcome and the considered outcome, an improvement of the opponents utility beyond the last offered outcome is penalised. This can lead to outcomes not being selected for offering, because they improve the opponents utility to much, even tough they strictly dominate the highest scoring outcome in terms of utility for both parties. This behaviour can be observed in Figure 4.1 and in Table 4.1, where a specific example is elaborated. This behaviour leads to less Pareto efficient bidding behaviour.
- The choice of comparing the considered outcome to the last offered outcome is slightly restrictive. It can lead to less optimal behaviour when facing a less intelligent agent or a Random Walker [41]. A less intelligent agent might fail to propose Pareto efficient outcomes, while thinking it is. It might be willing to concede more than it is actually doing because it is incapable of estimating our preferences. Facing a Random Walker causes the fitness values of outcomes to fluctuate heavily in between rounds, as the utility of the opponents last offered outcome is randomised.

Outcome	$u(\omega)$	$\hat{u}_o(\omega)$	δ	e	t	$\hat{u}_o(x^{last})$	$f(\omega,t)$
ω_1	0.57	0.65	1	0.5	0.7	0.52	0.717
ω_2	0.60	0.78	1	0.5	0.7	0.52	0.669

Table 4.1: Comparison of fitness values between outcomes (Equation 4.2)

To solve the points of criticism, we propose four alternative fitness functions. We then add the selection of a fitness function to the configuration space of the agent, providing more flexibility to the strategy. Out of this set, one of the functions must be picked during configuration:

$$f_n(\omega,t) = f_{my}(\omega,t) + f_{o,n}(\omega,t) \tag{4.4}$$
$$f_{o,1}(\omega,t) = (1-\delta * TP(t)) * (1 - |\hat{u}_o(\omega) - \hat{u}_o(x^{last})|) \tag{4.5}$$
$$f_{o,2}(\omega,t) = (1-\delta * TP(t)) * \min(1 + \hat{u}_o(\omega) - \hat{u}_o(x^{last}), 1) \tag{4.6}$$
$$f_{o,3}(\omega,t) = (1-\delta * TP(t)) * (1 - |\hat{u}_o(\omega) - \hat{u}_o(x^{+})|) \tag{4.7}$$
$$f_{o,4}(\omega,t) = (1-\delta * TP(t)) * \min(1 + \hat{u}_o(\omega) - \hat{u}_o(x^{+}), 1) \tag{4.8}$$
$$f_{o,5}(\omega,t) = (1-\delta * TP(t)) * \hat{u}_o(\omega) \tag{4.9}$$
$$TP(t) = 1 - t^{\frac{1}{e}}$$

The behaviour of every fitness function is illustrated in Figure 4.1. An intuitive description is given below.

- Equation 4.5 converges to the outcome where $\hat{u}_o(\omega) = \hat{u}_o(x^{last})$
- Equation 4.6 converges to the outcome where $\hat{u}_o(\omega) >= \hat{u}_o(x^{last})$
- Equation 4.7 converges to the outcome where $\hat{u}_o(\omega) = \hat{u}_o(x^{+})$
- Equation 4.8 converges to the outcome where $\hat{u}_o(\omega) >= \hat{u}_o(x^{+})$
- Equation 4.9 converges to the outcome where $\hat{u}_o(\omega) = 1$

4.1.2 Outcome space exploration

The last requirement to fulfil concerns outcome space exploration without brute-force methods. Every round, the agent has to find or approach the outcome with the highest fitness value as defined in Section 4.1.1 within a possibly large outcome space. Lau et al. [29] solved this problem by using a Genetic Algorithm [15]. As outcome space exploration is not a main topic of interest for this thesis, we will follow Lau and implement the algorithm as given in pseudocode in Algorithm 2. We do add the genetic algorithm parameters to the configuration space, as these must also be tuned. Standard Genetic Algorithm procedures are applied, which will not be elaborated upon in this section. However, they are included in Appendix B for reference.

4.1.3 Configuration space

We present the final configuration space hyperparameters for the bidding strategy in Table 4.2.

Symbol	**Description**	**Use**	**Type**	**Value range**
n_{fit}	Fitness function selection	Fitness function	Categorical	$\{1, 2, 3, 4, 5\}$
δ	Trade-off factor	Fitness function	Real	$[0, 1]$
e	Conceding factor	Fitness function	Real	$(0, 1]$
N_{pop}	Population size	Space exploration	Integer	$[50, 400]$
N_{tour}	Tournament size	Space exploration	Integer	$[1, 10]$
E	Evolutions	Space exploration	Integer	$[1, 5]$
R_c	Crossover rate	Space exploration	Real	$[0.1, 0.5]$
R_m	Mutation rate	Space exploration	Real	$[0, 0.2]$
R_e	Elitism rate	Space exploration	Real	$[0, 0.2]$

Table 4.2: Configuration space of bidding strategy

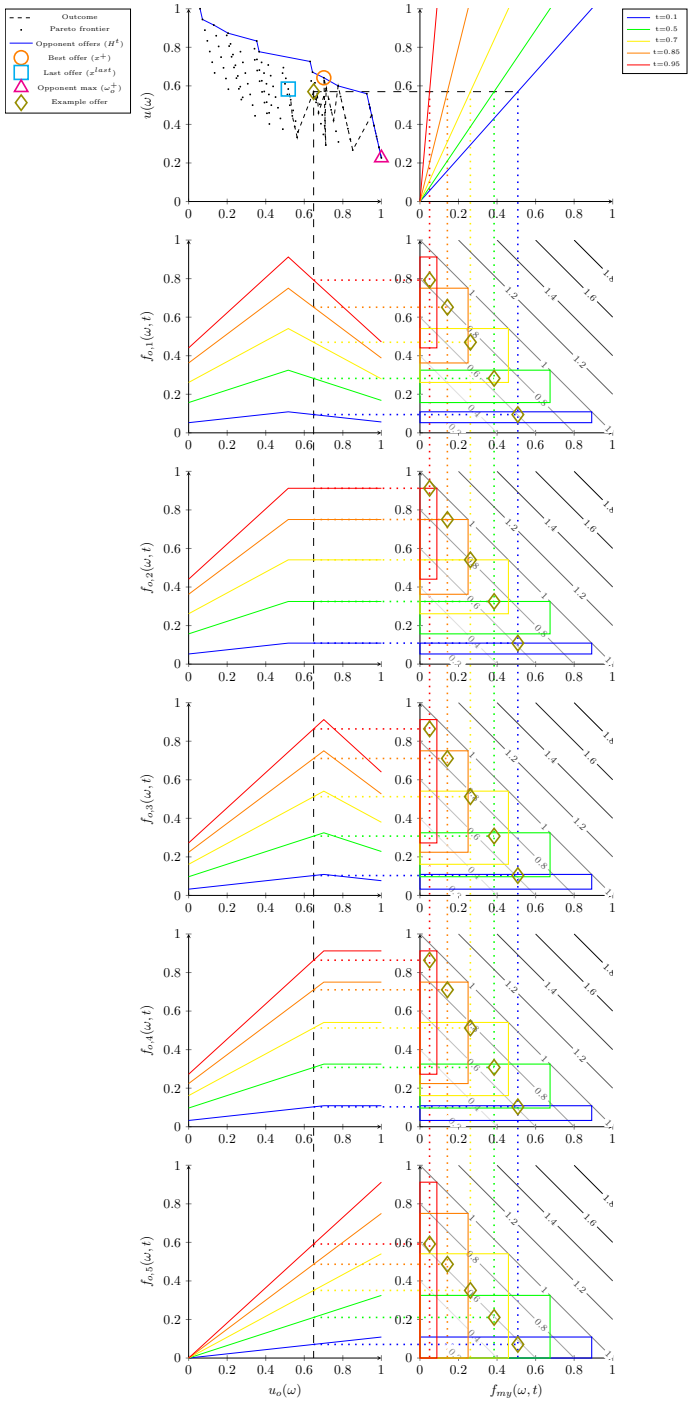


Figure 4.1: Behaviour of fitness functions ($\delta = 0.9, e = 0.5$)

Algorithm 2 Genetic Algorithm for outcome space exploration

Input	Ω	Outcome Space
	$f(\omega, t)$	Fitness function
	N_{pop}	Population size
	N_{tour}	Tournament size
	E	Evolutions
	R_c	Crossover rate
	R_m	Mutation rate
	R_e	Elitism rate
Variables	Pop	Population, where $Pop = [Ind_1, \ldots, Ind_{Size(Pop)}]$
	Ind	Individual, where $Ind = \langle fitness, \omega \rangle$
Output	ω_{fit}	Highest scoring outcome

$Pop \leftarrow RandomInitialise(\Omega, N_{pop})$
$t \leftarrow GetTime()$
$Pop \leftarrow CalculateFitness(Pop, f(\omega, t))$
for $i := 1, \ldots, E$ **do**
    $Pop_{new} \leftarrow GetElites(Pop, R_e)$
    while $Size(Pop_{new}) < N_{pop}$ **do**
        $Ind_a \leftarrow TournamentSelect(Pop, N_{tour})$
        $Ind_b \leftarrow TournamentSelect(Pop, N_{tour})$
        $Ind_{new} \leftarrow Crossover(Ind_a, Ind_b, R_c)$
        $Ind_{new} \leftarrow Mutate(Ind_{new}, R_m)$
        $Pop_{new} \leftarrow Pop_{new} + [Ind_{new}]$
    $Pop \leftarrow Pop_{new}$
    $Pop \leftarrow CalculateFitness(Pop, f(\omega, t))$
return $GetFittest(Pop)$

4.2 Acceptance strategy

In Section 3.1.3, we introduced a base set of acceptance conditions that are often used in negotiation agent as a single condition or a combination of conditions. Baarslag et al. [45] analysed the performance of the acceptance condition and concluded the following:

1. There is not single constant that makes AC_{const} an effective condition, as it is very domain dependent.
2. AC_{next} will always outperform AC_{prev}.
3. AC_{time} always reaches an agreement, but of relatively low utility.
4. A combination of conditions, such as AC_{time} and AC_{next}, outperforms other acceptance conditions.

We aim to follow Baarslag et al. by implementing a combination AC_{combi} as described in the paper, though we relax the parameter space a bit further. We will now describe the acceptance condition.

Suppose we are at time t', which is beyond the time of the last offer by the opponent $t = t_n$. We consider the following negotiation thread (Section 3.1.2):

$$H^t = \left[x^{t_1}, x^{t_2}, \ldots, x^{t_{n-1}}, x^{t_n}\right] \tag{4.10}$$

At this point, our agent has to decide whether to accept or not and the remaining time is $r = 1 - t'$. We define a time window of equal size to the remaining time, but in the past $W = [t' - r, t'] \subseteq T$. All the offers made by the opponent within this window are denoted by:

$$H^W = \left\{x^s \in H^t | s \in W\right\} \tag{4.11}$$

We now define a maximum and average utility for our agent over this window:

$$MAX^W = \max_{x \in H^W} u(x) \tag{4.12}$$

$$AVG^W = \frac{1}{|H^W|} \sum_{x \in H^W} u(x) \tag{4.13}$$

Following Baarslag et al. [45], we define the combined acceptance condition AC_{combi}:

$$AC_{combi}(\alpha, \beta, t_{acc}, \gamma) \iff AC_{next}(\alpha, \beta) \vee AC_{time}(t_{acc}) \wedge (u(x^t) \geq \gamma) \tag{4.14}$$

$$AC_{next}(\alpha, \beta) \iff \alpha * u(x^t) + \beta \geq u(\omega_{fit}(t')) \tag{4.15}$$

$$AC_{time}(t_{acc}) \iff t' \geq t_{acc} \tag{4.16}$$

where:

α	: Scale factor by which the opponents offer utility is multiplied
β	: Minimal utility gap to accept
t_{acc}	: Time after which AC_{time} accepts
γ	: Lower utility to accept, where $\gamma \in \{MAX^W, AVG^W\}$
x^t	: Offer of opponent at time t
$\omega_{fit}(t')$	: Offer that our agent wants to make at time t'

4.2.1 Configuration space

We present the final configuration space hyperparameters for the acceptance strategy in Table 4.3.

Symbol	Description	Type	Value range
α	AC_{next} scale factor	Real	$[1, 1.1]$
β	AC_{next} utility gap	Real	$[0, 0.2]$
t_{acc}	AC_{time} accepting time	Real	$[0.9, 1]$
γ	Lower utility to accept	Categorical	$\{MAX^W, AVG^W\}$

Table 4.3: Configuration space of acceptance strategy

4.3 Opponent model

An opponent model is an abstract representation of the opponent that is learning during interaction with the opponent and revolves around three questions according to Baarslag [33]:

- Preference estimation. What does the opponent want?
- Strategy prediction. What will the opponent do, and when? *Not considered in this thesis*
- Opponent classification. What kind of player is the opponent, and how should we act accordingly?

4.3.1 Preference estimation

The negotiations are performed under incomplete information, this means without knowledge of the opponents preference profiles. It is beneficial to make offers that lie close to the Pareto frontier as this can improve agreement utility for all parties. However, in order to be able to propose Pareto optimal bids, we need the preference profile of the opponent, which we do not have. The preference profile of the opponent can be estimated based on the offers that the opponent makes. In the ANAC, estimating opponent preference profiles is applied by a large part of the participants. A comparison of the methods used by the participants showed that *frequency models* and *value models* perform well [44]. From this comparison, we picked the Smith Frequency Model [46] (SFM) as our preference estimation model, due to it's performance and simplicity. This model keeps track of how many times a value within an issue is offered by the opponent and calculates issue weight and value weight based on this count. The estimated utility function of the SFM is presented in Equation 4.17.

$$\hat{u}(\omega) = \left(\sum_{i \in I} \hat{w}_i * \hat{e}_i(\omega_i) \right) * \frac{1}{\sum_{i \in I} \hat{w}_i} \tag{4.17}$$

$$\hat{w}_i = \frac{C_i^+}{C_i^{sum}} \tag{4.18}$$

$$\hat{e}_i(\omega_i) = \frac{C_{\omega_i}}{C_i^+} \tag{4.19}$$

$$C_i^+ = \max_{v \in V_i} C_v \tag{4.20}$$

$$C_i^{sum} = \sum_{v \in V_i} C_v \tag{4.21}$$

where:

$\hat{u}$	: estimated utility function
ω_i	: Sub-outcome for issue i
I	: Set of issues
$\hat{w}_i$	: estimated weight of issue (i)
$\hat{e}_i$	: estimated evaluation function of issue (i)
v_i	: Value for issue (i) in the outcome (ω)
C_{v_i}	: Count of value ω_i in received offers
C_i^+	: Max value count within issue (i)
C_i^{sum}	: Sum of value counts of issue (i)
V_i	: Set of values in issue (i)

Modifications

We performed a few negotiations sessions with the SFM and gathered offer trajectories of opponents. We used these trajectories to analyse the performance of the SFM by hand, which led to the following observations:

1. The weight of an issue is typically overestimated when the issue has a small set of values.
2. The weight of sub-optimal values is typically underestimated when the issue weight is high.

We make an attempt to improve the preference estimation by analysing the observations. Improving the preference profile estimation directly influences the quality of the opponent model, as behavioural features are extracted using opponent utility estimations.

Issue weight calculation In the SFM, the weight of an issue is calculated based on the offer distribution over the values in the issue. It simply finds the maximum value count in an issue and divides it by the total value count (Equation 4.18). A maximum weight of $\hat{w}_i = 1$ is assigned if the opponent exclusively offers a single value within an issue, as it is likely to be an important issue

for the opponent. Intuitively, if an opponent does not care at all about an issue, the offered values will spread out over the set of values within the issue. We could say that perfect distribution over the set of values implies little importance for the issue, so the issue weight must be set to $\hat{w}_i = 0$. Doing so, every issue has a predicted weight assigned of $\hat{w}_i \in [0, 1]$.

However, this is not the case with Equation 4.18, as issue weight is lower bounded in relation to the size of the value set. In order for $\hat{w}_i$ to be zero, C_i^+ must be zero, but at perfect distribution:

$$C_i^+ = \frac{C_i^{sum}}{|V_i|} \tag{4.22}$$

And thus:

$$\hat{w}_i = \frac{C_i^+}{|V_i|} = \frac{\frac{C_i^{sum}}{|V_i|}}{C_i^{sum}} = \frac{1}{|V_i|} \tag{4.23}$$

This lower bound approaches zero when an issue has a large value set, but in the extreme case of $|V_i| = 2$ the predicted issue weight is lower bounded by $\hat{w}_i \in [0.5, 1]$. This lower bound causes overestimation of issue weights with a smaller set of values. We illustrated this overestimation in an example in Figure 4.2 where we created a fictional distribution of 100 offers over two different issues. Although both issues have the same estimated weight, intuitively, Figure 4.2 (a) seems to be more important to the opponent.

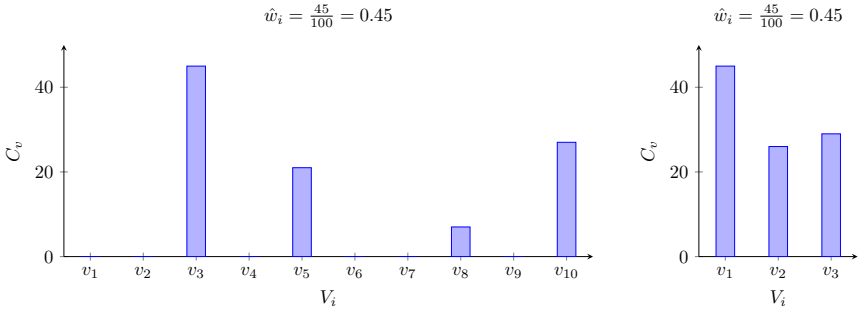


Figure 4.2: Comparison of issue weights for original SFM

We propose a correction to the issue weight estimation such that $\hat{w}_i \in [0, 1]$, despite the size of the issue value set. The new proposal is given in Equation 4.24. We compare the new method in Figure 4.3 using the same example as in Figure 4.2.

$$\hat{w}_i = \frac{C_i^+ - C_i^{base}}{C_i^{sum} - C_i^{base}} \tag{4.24}$$

$$C_i^+ = \max_{v \in V_i} C_v \tag{4.25}$$

$$C_i^{sum} = \sum_{v \in V_i} C_v \tag{4.26}$$

$$C_i^{base} = \frac{C_i^{sum}}{|V_i|} \tag{4.27}$$

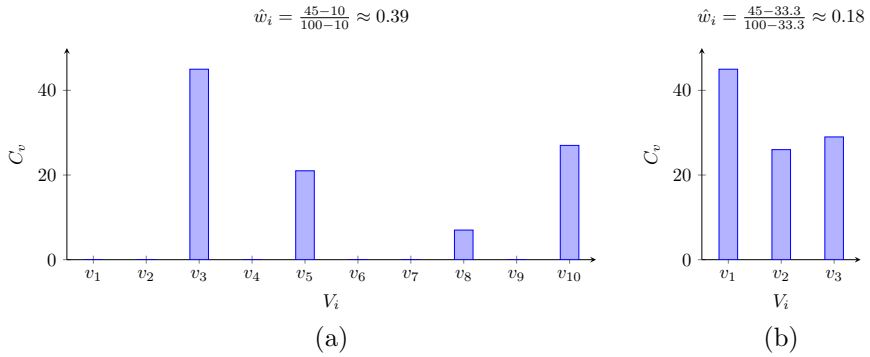


Figure 4.3: Comparison of issue weights for modified SFM

Evalutation function Frequency models estimate both issue weight and value weight separately [44]. This is also the case for the SFM, where value weight is estimated according to Equation 4.18. We observed that value weights are systematically underestimated for issues with a higher issue weight. This applies to all values that are sub-optimal ($\hat{e}_i(v) < 1$). We argue that the value weight cannot be estimated separately from the issue weight for frequency models, but that estimated issue weight must be included in value weight calculation.

We again present a fictional example to illustrate the problem in Figure 4.4. Suppose we receive the illustrated distributions over two equally sized issues for which the opponent has different issue weights. The estimated evaluation of value v_2 is displayed on top for both cases. The question is whether the estimation is realistic or not. According to this estimation:

1. In case (a), offering v_2 over v_1 leads to an estimated decrease in utility of $(1-0.1)*0.9 = 0.81$.
2. In case (b), offering v_2 over v_1 leads to an estimated decrease in utility of $(1-0.1)*0.3 = 0.27$.

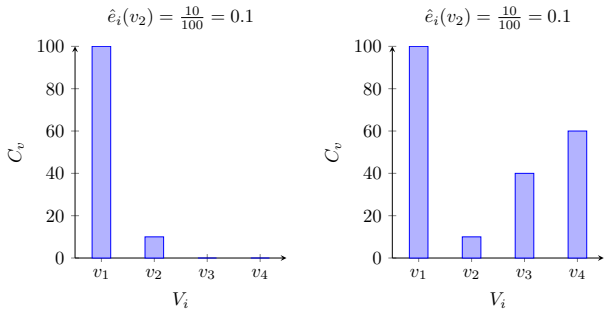


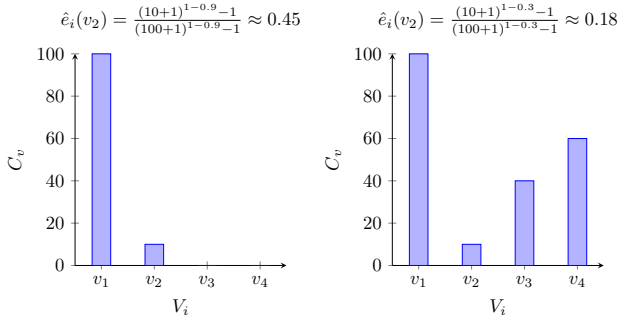
Figure 4.4: Comparison of value weights for original SFM

The second case is realistic, conceding by 0.27 utility is a lot, but an opponent might still consider it. The first case is less realistic, as it is unlikely that an opponent will concede by 0.81 utility,

especially not 10 times. The fact that an opponent considers another value within a very important issue at all, implies that the value also has a reasonable pay-off. To correct this observation we introduce a new method of value weight estimation, that also includes estimated weight of the issue in Equation 4.28. The proposal is identical to the SFM at $\hat{w}_i = 0$, but increasingly scales up estimated value weights for sub-optimal values when $\hat{w}_i > 0$. We present the same example with updated value weight estimation in Figure 4.5.

$$\hat{e}_i(\omega_i) = \frac{(C_{\omega_i} + 1)^{1-\hat{w}_i} - 1}{(C_i^+ + 1)^{1-\hat{w}_i} - 1} \tag{4.28}$$

$$C_i^+ = \max_{v \in V_i} C_v \tag{4.29}$$



(a) $\hat{w}_i \approx 0.9$ (b) $\hat{w}_i \approx 0.3$

Figure 4.5: Comparison of value weights for modified SFM

Computational expense SFM has a high accuracy according to [44]. It performs best in negotiations with round based deadlines, but performs less in negotiations with time based deadlines due to its computational expensiveness [47], decreasing the amount of negotiation rounds that can be played. This makes SFM not the best choice as preference estimation model in our time based deadline setup, despite being very accurate. However, for future opponent feature extraction, an accurate preference estimation model reduces the amount of noise. We aim to decrease the computational expensiveness of the SFM by recoding the original model to a more efficient version. To do so, we look at the two main functionalities of an preference estimation model. The first is registration of an offer made by an opponent to update the model. The second is using the model to estimate the opponents utility of an outcome.

The model keeps a list of offered values per issue during negotiation, at every received offer it registers a value for every issue resulting in $\mathcal{O}(|I|)$ time complexity ($|I|$ = size of issue set). At utility estimation, the SFM cycles the list of registered values per issue to estimate both the issue weight and issue weight resulting in $\mathcal{O}(|I| * n)$ time complexity (n = amount of offers received). Note that this means that utility estimation becomes increasingly computational expensive as the amount of rounds progresses. In bilateral negotiation, only one offer is received every round and,

generally, many utility estimations are performed. The computational complexity is therefore on on the wrong side of the balance.

We made two modifications that aim to improve the balance of computational expense. First, instead of keeping a list of values offered per issue, we keep a counter per value that we increment. This improves the time complexity for utility estimation to $\mathcal{O}(|I| * |V_i|)$, where $|V_i|$ is the size of the value set for issue i. It also decreases memory usage and removes the increasing computational expensiveness as the negotiation progresses through rounds. Secondly, we moved the calculation of issue and value weights at utility estimation to the point of offer registration as this is much less frequent. The resulting time complexities are presented in Table 4.4, which provides an overview of the time complexities for the original SFM and the modified SFM.

Operation	**Original**	**Modified**	**Frequency per round**
estimate utility	$\mathcal{O}(\|I\| * n)$	$\mathcal{O}(\|I\|)$	many
register offer	$\mathcal{O}(\|I\|)$	$\mathcal{O}(\|I\| * \|V_i\|)$	once

Table 4.4: Time complexity of original and modified SFM operations. $|I|$ = size of the issue set, n = amount of received offers, $|V_i|$ = size of value set for issue i.

Final version

The final version of the utility estimation equation after combining all that is discussed is displayed in Equation 4.30.

$$\hat{u}(\omega) = \left(\sum_{i \in I} \hat{w}_i * \hat{e}_i(\omega_i) \right) * \frac{1}{\sum_{i \in I} \hat{w}_i} \tag{4.30}$$

$$\hat{w}_i = \frac{C_i^+ - C_i^{base}}{C_i^{sum} - C_i^{base}} \tag{4.31}$$

$$\hat{e}_i(\omega_i) = \frac{(C_{\omega_i} + 1)^{1-\hat{w}_i} - 1}{(C_i^+ + 1)^{1-\hat{w}_i} - 1} \tag{4.32}$$

$$C_i^+ = \max_{v \in V_i} C_v \tag{4.33}$$

$$C_i^{sum} = \sum_{v \in V_i} C_v \tag{4.34}$$

$$C_i^{base} = \frac{C_i^{sum}}{|V_i|} \tag{4.35}$$

Performance measure

In this section, we proposed two modifications the original SFM based on observed shortcomings. The next step is to validate if both modifications can be justified. The overarching goal of the agent in this thesis is to improve utility of the agreements. Baarslag et al. [44] proposed a number

of accuracy measures that measure differences between the predicted outcome space and the true outcome space (Figure 3.2). He also showed how the accuracy measures correlate with actual agent performance in terms of utility. We follow Baarslag et al. [44] and use two of his proposed accuracy measures that correlate best with agent performance. Should we be able to improve on these accuracy measures by modifying SFM, then we can justify the modifications made to SFM.

The first accuracy measure is the *Pearson Correlation of outcomes* between the estimated opponent utility and the opponent utility:

$$PC(u_o, \hat{u}_o) = \frac{\sum_{\omega \in \Omega} (u_o(\omega) - u_o(\bar{\omega}))\,(\hat{u}_o(\omega) - \hat{u}_o(\bar{\omega}))}{\sqrt{\sum_{\omega \in \Omega} (u_o(\omega) - u_o(\bar{\omega}))^2 \sum_{\omega \in \Omega} (\hat{u}_o(\omega) - \hat{u}_o(\bar{\omega}))^2}} \tag{4.36}$$

The second accuracy measure is the *difference in Pareto frontier surface*, which is a measure of difference in predicted Pareto frontier $\hat{\Omega}_p$ and true Pareto frontier Ω_p. Both pareto frontiers are mapped on the true outcome space and the area below the frontier is calculated. The absolute difference between the two surfaces is the *difference in Pareto frontier surface* measure and is illustrated in Figure 4.6.

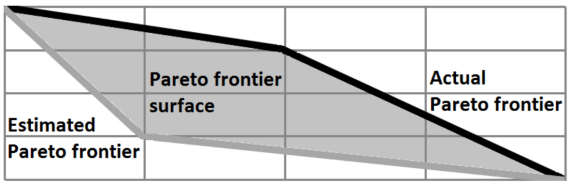


Figure 4.6: Visualisation of the difference in Pareto frontier surface [44]

4.3.2 Opponent classification

In this thesis, we will not adapt our negotiation strategy during a negotiation session, so classification of opponents is not necessary online. Instead, the opponents are classified between negotiation sessions and effective counter strategies are selected at the start of the negotiation session. This is possible since we assume repeated encountering of opponents. We want to revert the reader to Chapter 5 and Chapter 7 for, respectively, the modelling of opponents and the adaptation to opponents.

4.4 Results

In this section, we will already show results related to problem 2.1 in Section 3.2 concerning preference estimation accuracy. To do so, we take the Dynamic Agent as discussed in this chapter and lock its strategy by setting a parameter configuration. The configuration that is set is found in a later stage of this thesis (Chapter 6) and is considered to be the single best strategy. Since we are interested in preference estimation accuracy, the strategy of our own agent is of less importance.

We take the training set of negotiation instances that we use throughout this thesis (Appendix A) and run 10 negotiation sessions with our Dynamic Agent on each of the instances with a 60 second deadline for a total of 11200 negotiation sessions. The difference in Pareto frontier surface and the Pearson correlation of outcomes (Section 4.3.1) are calculated at the end of every negotiation session. We compare a total of 4 preference estimation methods by exact value and increase ratio relative to the original SFM:

1. The original SFM, which is the baseline
2. SFM with modified $\hat{w}_i$ and original $\hat{e}_i(\omega_i)$
3. SFM with original $\hat{w}_i$ and modified $\hat{e}_i(\omega_i)$
4. SFM with modified $\hat{w}_i$ and modified $\hat{e}_i(\omega_i)$

We present the results for the Pearson correlation of outcomes measure and for the difference in Pareto surface measure in Table 4.5. As can be seen, both modifications result in an improvement for both accuracy measures. The largest improvement is achieved with the modified value evaluation function in Equation 4.28, causing a 6.43% improvement of the Pearson correlation measure and a 3.64% decrease of the difference in Pareto surface. We further discuss the results in Chapter 8.

Modifications		**Pearson correlation**	
$\hat{w}_i$	$\hat{e}_i(\omega_i)$	value	increase ratio
original	original	0.7301	0
modified	original	0.7312	0.0015
original	modified	0.7771	0.0643
modified	modified	0.7779	0.0655

Modifications		**Pareto surface**	
$\hat{w}_i$	$\hat{e}_i(\omega_i)$	value	increase ratio
original	original	0.0772	0
modified	original	0.0770	-0.0036
original	modified	0.0744	-0.0364
modified	modified	0.0742	-0.0391

Table 4.5: Preference estimation accuracy measures comparing modified SFM to original SFM

Chapter 5

Features

In this chapter, we will attempt to define a feature description of a negotiation setting that stipulate differences between negotiation settings. We will use this feature description for both configuration and selection of our Dynamic Agent as presented in Chapter 4. As described in Section 3.1, a negotiation setting can be separated into three types of components. Should we engage in bilateral negotiation, then there are three components that define the negotiation environment for our agent: a protocol, an opponent, and a scenario. The features are meant to indicate differences between negotiation, so we do not have to model the protocol as it remains fixed throughout this thesis.

This leaves us with two components that we want to model by features:

1. The negotiation scenario
2. The opponent

5.1 Scenario features

The negotiation scenario consists of a domain and a preference profile for all agents involved (Section 3.1.1). Only the domain and our own preference profile are known in advance and can be used for feature extraction. Ilany et al. [18] specified a list of scenario features that they used for agent selection in bilateral negotiation, which we will follow. The features are calculated before the negotiation starts and are fully independent of the opponent faced. An overview of the scenario features is provided in Table 5.1.

5.2 Opponent features

In contrast to the scenario features, opponent features cannot be calculated in advance of a negotiation session. Since this thesis revolves around repeatedly encountering opponents, we can extract features after a negotiation session and store them for future opponent encounters. For

Feature type	Description	Equation	Notes
Domain	Number of issues	$\|I\|$	
Domain	Average number of values per issue	$\frac{1}{\|I\|}\sum_{i\in I}\|V_i\|$	
Domain	Number of possible outcomes	$\|\Omega\|$	
Preference	Standard deviation of issue weights	$\sqrt{\frac{1}{\|I\|}\sum_{i\in I}(w_i-\frac{1}{\|I\|})^2}$	
Preference	Average utility of all possible outcomes	$\frac{1}{\|\Omega\|}\sum_{\omega\in\Omega}u(\omega)$	denoted by $u(\bar{\omega})$
Preference	Standard deviation utility of all possible outcomes	$\sqrt{\frac{1}{\|\Omega\|}\sum_{\omega\in\Omega}(u(\omega)-u(\bar{\omega}))^2}$	

Table 5.1: Scenario features

each opponent, we store both the *mean* and the *Coefficient of Variance (CoV)* of all features and update them after every negotiation round with the opponent.

Storing opponent features for future encounters does introduce a difficulty, as opponent behaviour is influenced by the scenario. Using behavioural features that are influenced by the scenario blurs our model when we encounter the opponent in a different scenario. We therefore aim to construct opponent features that are disassociated from the scenario. In this section, we will describe every opponent feature.

5.2.1 Normalised time

The time $t \in [0, 1]$ it takes to reach an agreement with the opponent.

5.2.2 Concession rate

To measure how much an opponent is willing to concede towards our agent, we use the notion of Concession Rate (CR) introduced by Baarslag et al. [48]. The CR is a normalised ratio $CR \in [0, 1]$, where $CR = 1$ means that the opponent fully conceded and $CR = 0$ means that the opponent did not concede at all. By using a ratio instead of an absolute value (utility), the feature is disassociated from the scenario. The original paper on CR defined a value for every time point in the negotiation session, but we simplify by only calculating the CR at the end of the negotiation session.

To calculate the CR, Baarslag et al. [48] used two constants. The minimum utility an opponent has demanded during the negotiation session $u_o(x_o^-)$ (Section 3.1.4) and the Full Yield Utility (FYU), which is the utility that the opponent receives at our maximum outcome $u_o(\omega^+)$. Baarslag et al. [48] analysed the CR under perfect knowledge of the opponent's utility function, which is impossible in our application, so we use the estimated utility function $\hat{u}_o(\omega)$ instead.

We present a formal description of the CR in Equation 5.1 and a visualisation in Figure 5.1. As this CR is based on an estimated opponent utility, we compared the quality of the approximation along with two alternative methods of approximating the CR. As this comparison is a detail with relation to the main goal of this thesis, we moved it to Appendix C for reference.

$$CR(x_o^-) = \begin{cases} 1 & \text{if } \hat{u}_o(x_o^-) \leq \hat{u}_o(\omega^+), \\ \frac{1-\hat{u}_o(x_o^-)}{1-\hat{u}_o(\omega^+)} & \text{otherwise.} \end{cases} \tag{5.1}$$

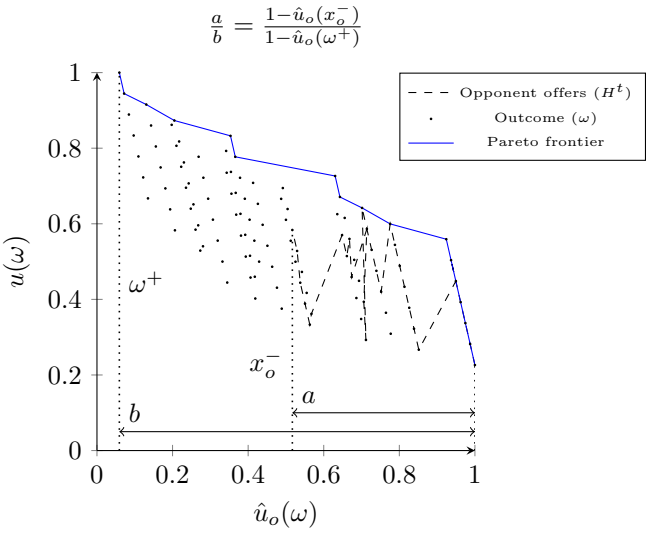


Figure 5.1: Visualisation of Concession Rate (CR)

5.2.3 Average rate

We introduce the Average Rate (AR) that indicates the average utility an opponent has demanded as a ratio depending on the scenario. The two constants needed are the FYU ($u_o(\omega^+)$) as described in the previous section and the average utility an opponent demanded ($u_o(\bar{x})$). The AR is a normalised ratio $AR \in [0, 1]$, where $AR = 0$ means that the opponent only offered his maximum outcome and $AR = 1$ means that the average utility the opponent demanded is less than or equal to the FYU. As with the CR, we use the estimated opponent utility function $\hat{u}_o(\omega)$ to approximate the true function. We present a definition of the AR in Equation 5.2 and a visualisation in Figure 5.2.

$$AR(\bar{x}) = \begin{cases} 1 & \text{if } \hat{u}_o(\bar{x}) \leq \hat{u}_o(\omega^+), \\ \frac{1-\hat{u}_o(\bar{x})}{1-\hat{u}_o(\omega^+)} & \text{otherwise.} \end{cases} \tag{5.2}$$

The AR is an indication of competitiveness of the opponent based on average utility demanded instead of minimum demanded utility as the CR is. A combination of the AR and the CR can classify time-dependent opponent tactics.

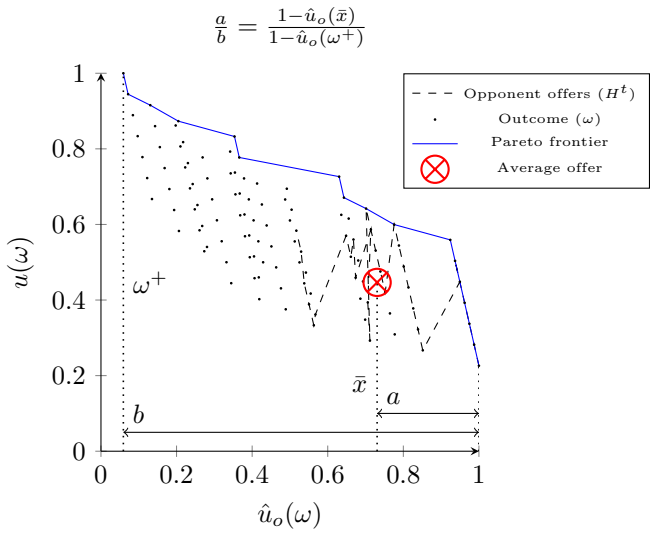


Figure 5.2: Visualisation of Average Rate (AR)

Lemma 5.2.1 *For time-dependent opponent tactics that start at maximum utility, $AR < \frac{1}{2} * CR$ indicates a Boulware strategy, while $AR > \frac{1}{2} * CR$ indicates a conceding strategy.*

To prove this, we take both *otherwise* cases from Equation 5.1 and Equation 5.2 and expand the Boulware indicator from Theorem 5.2.1 in Equation 5.3. Assume that the opponent is a linear conceding strategy, in that case its average demanded utility $\hat{u}_o(\bar{x})$ lies exactly halfway in between its maximum demanded utility 1 and its minimum demanded utility $\hat{u}_o(x_o^-)$, so $\hat{u}_o(\bar{x}) = \frac{1}{2}(1 + \hat{u}_o(x_o^-))$. In case of a Boulware opponent, the average demanded utility is greater than the average demanded utility of a linear conceding opponent $\hat{u}_o(\bar{x}) > \frac{1}{2}(1 + \hat{u}_o(x_o^-))$, which is what we found in Equation 5.3. The opposite is also true for a conceding opponent.

$$\begin{aligned} AR &< \frac{1}{2} * CR \\ \frac{1-\hat{u}_o(\bar{x})}{1-\hat{u}_o(\omega^+)} &< \frac{1}{2} * \frac{1-\hat{u}_o(x_o^-)}{1-\hat{u}_o(\omega^+)} \\ 1-\hat{u}_o(\bar{x}) &< \frac{1}{2}(1-\hat{u}_o(x_o^-)) \\ \hat{u}_o(\bar{x}) &> \frac{1}{2} + \frac{1}{2}\hat{u}_o(x_o^-) \end{aligned} \tag{5.3}$$

As with the CR, we calculate the AR under incomplete information to approximate the true AR. We compared the quality of this approximation, as well as an alternative method of AR approximation, with the true AR. As this comparison is a detail with relation to the main goal of this thesis, we moved it to Appendix D for reference.

5.2.4 Default algorithm performance

According to Hutter et al. [22], the performance of any default algorithm on a problem works surprisingly well as a feature for that specific problem. For negotiation, this translates to the obtained utility of a hand-picked default strategy on a negotiation instance. The obtained utility is normalised and can be used as a feature for that negotiation instance.

We implement this concept as an opponent feature by selecting a default strategy and use it to obtain an agreement ω_{agree} and a utility $u(\omega_{agree})$ against an opponent. We then normalise this obtained utility an rename it as our Default Algorithm Performance (DAP) feature. We present the formal definition of this feature in Equation 5.4 and a visualisation in Figure 5.3.

$$DAP(\omega_{agree}) = \begin{cases} 0 & \text{if } u(\omega_{agree}) \leq u(\omega^-), \\ \frac{u(\omega_{agree})-u(\omega^-)}{1-u(\omega^-)} & \text{otherwise.} \end{cases} \tag{5.4}$$

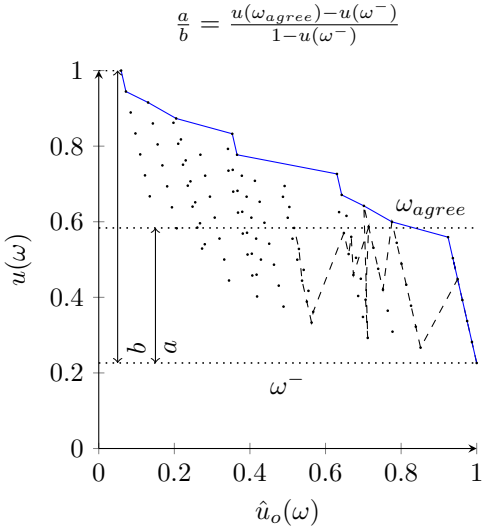


Figure 5.3: Visualisation of Default Algorithm Performance (DAP)

5.2.5 Disassociation of opponent features with scenario

We now demonstrate to what extent the opponent features, as described in this section, are disassociated with the features from the scenario component. As we store opponent features for future encounters in new scenarios, we should avoid hidden scenario information embedded in the opponent features, as the new scenario is different. In general, it is unlikely that we succeed in completely removing the influence of the scenario on the opponent features, especially since the negotiation is performed under incomplete information. Therefore, it is prudent to aim for features that provide more informational value on the opponent, than on the scenario.

As stated before, we store the mean and the CoV of the opponent features per opponent after every negotiation session. We use these features at future encounters with the same opponent. For the sake of this demonstration, we also store the mean and the CoV of the opponent features per scenario. To prove that the opponent features contain more informational value on the opponents than on the scenarios, we must show that the standard deviation of the features is higher over the opponents than over the scenarios.

To do so, we take the Dynamic Agent as described in Chapter 4 and fix its strategy by setting the single best strategy configuration that we find in Chapter 6. The single best strategy is also the default strategy, for which we can extract the DAP. In our demonstration we use the training set of negotiation instances (Appendix A) and run every negotiation session 10 times with the Dynamic Agent. We calculate the mean and the CoV of the opponent features per opponent and per scenario. Per feature, we calculate the standard deviation over the opponent set and over the scenario set. The results are provided in Table 5.2.

Feature	**StdDev over opponents**	**StdDev over scenarios**
$mean(t)$	0.232	0.105
$CoV(t)$	0.101	0.063
$mean(AR)$	0.206	0.127
$CoV(AR)$	0.139	0.121
$mean(CR)$	0.217	0.064
$CoV(CR)$	0.080	0.042
$mean(DAP)$	0.236	0.102
$CoV(DAP)$	0.057	0.051

Table 5.2: Standard deviation of opponent features over opponents & scenarios

The results in Table 5.2 show that we managed to construct opponent features that satisfy the requirement. Especially the mean of the opponent features show a much higher standard deviation over the opponents as compared to the standard deviation over the scenarios.

Chapter 6

Algorithm configuration for negotiation

We created an agent in Chapter 4 with a dynamic strategy depending on a parameter configuration. So far, we defined every parameter as a hyperparameter for which a value must be selected. The total hyperparameter set or configuration space Θ is summarised in Table 6.1. Setting a parameter configuration $\theta \in \Theta$ within this hyperparameter space "locks" the strategy of our *Dynamic Agent*. We will denote our Dynamic Agent set by a parameter configuration as $DA(\theta)$. We now want to obtain a parameter configuration θ that results in a well performing negotiation strategy. In this chapter, we will address this hyperparameter optimisation problem for negotiation.

Symbol	**Description**	**Use**	**Type**	**Value range**
α	AC_{next} scale factor	Accepting	Real	$[1, 1.1]$
β	AC_{next} utility gap	Accepting	Real	$[0, 0.2]$
t_{acc}	AC_{time} accepting time	Accepting	Real	$[0.9, 1]$
γ	Lower utility to accept	Accepting	Categorical	$\{MAX^W, AVG^W\}$
n_{fit}	Fitness function selection	Fitness function	Categorical	$\{1, 2, 3, 4, 5\}$
δ	Trade-off factor	Fitness function	Real	$[0, 1]$
e	Conceding factor	Fitness function	Real	$(0, 1]$
N_{pop}	Population size	Space exploration	Integer	$[50, 400]$
N_{tour}	Tournament size	Space exploration	Integer	$[1, 10]$
E	Evolutions	Space exploration	Integer	$[1, 5]$
R_c	Crossover rate	Space exploration	Real	$[0.1, 0.5]$
R_m	Mutation rate	Space exploration	Real	$[0, 0.2]$
R_e	Elitism rate	Space exploration	Real	$[0, 0.2]$

Table 6.1: Configuration space Θ

6.1 Problem definition

The problem definition is already briefly described in Section 3.2, but in this section we describe the problem in more detail. We defined both a set of agents (opponents) and scenarios in Appendix A. As is standard procedure in optimisation problems, we split the set of agents and scenarios in a training set used for optimisation and a test set used for testing to ensure unbiased results. Combining a training opponent $a \in A_{train}$ with a training scenario $s \in S_{train}$ gives us a single training instance $\langle a, s \rangle = \pi \in \Pi_{train}$, for a total of $|\Pi_{train}| = 1120$ training instances.

> The subscript "*train*" is dropped throughout the rest of the chapter to improve readability.

Hyperparameter optimisation algorithms evaluate configurations on the considered problem to observe a performance and use this information to obtain new promising configurations. As stated before, the goal is to achieve a configuration that results in a well performing agent. This requires us to define what well performing is. A logical value to measure performance is the obtained utility $o(\theta, \pi)$ by playing strategy θ on instance π. As we are interested in optimising on the full set of training instances and not on a single instance, we define the performance of a configuration on an instance set as the average obtained utility (Equation 6.1).

$$O(\theta, \Pi) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} o(\theta, \pi) \tag{6.1}$$

where:

- o : Obtained utility of configuration θ on instance π
- O : Average obtained utility of configuration θ on instance set Π
- θ : Parameter configuration, where $\theta \in \Theta$
- π : Single negotiation instance consisting of opponent agent $a \in A$ and scenario $s \in S$, where $\pi = \langle a, s \rangle \in \Pi$
- Π : Set of instances

Generally, many configurations are evaluated to let the optimisation converge to a (local) optimum, which is not a problem if calculating the performance is computational inexpensive. For example, the exploration of the outcome space as described in Section 4.1.2 is also an optimisation problem and its performance (the fitness function) is calculated within a millisecond.

However, obtaining the performance on the training set as defined in Equation 6.1 requires us to test a configuration on the full training set of negotiation instances. A single negotiation round has a deadline of 60 seconds, so in the worst case scenario, obtaining the performance of a configuration would take 1120 minutes or 18.5 hours. We conclude that optimisation algorithms that require configurations to be evaluated on the full training set of instances, thus requiring Equation 6.1 to be calculated, are impractical for our application.

Sequential Model-based optimization for general Algorithm Configuration [22] (SMAC) uses a different approach to the automated configuration problem by using a approach that races promising configurations against each other based on single instances. As a result, SMAC generally does not need to evaluate a challenging configuration on the full set of instances before making a decision about its performance.

6.2 SMAC

To solve the described problem, we bring SMAC into the research area of automated negotiation. SMAC is a hyperparameter optimisation algorithm that is well suited to solve our optimisation problem for multiple reasons:

1. It can handle hyperparameters that are real, integers, or categorical, which is required (Table 6.1).
2. It can optimise using only subsets of the training instance set, reducing the computational expense.
3. It has a mechanism to terminate poorly performing configurations early, saving additional computation time. If it detects that a configuration is performing very poor on a small set of instances (e.g. a very eager conceder), it stops evaluating and drops the configuration.
4. The optimisation can be run in parallel, significantly reducing execution time.

6.2.1 Structure

In Algorithm 3, the basic structure of a parallel Sequential Model-Based Optimisation [22] (SMBO) procedure is provided, which also forms the body of SMAC. SMAC keeps a runhistory (Equation 6.3) consisting of a parameter configuration θ_i with its obtained utility o_i on a negotiation instance. This runhistory is combined with runhistories of parallel pools and a random forest regression model is fitted to the full runhistory, mapping the configuration space to an estimated performance $\hat{o}$ (Equation 6.2). This model is then used to predict promising configurations $\vec{\Theta}_{new}$, which are send to the *Intensify* procedure. We want to refer the reader to [22] for further details on the *Initialise*, *FitModel*, and *SelectConfigurations* procedures, as they are less relevant for the topic of this thesis.

$$\mathcal{M} : \Theta \rightarrow \hat{o} \tag{6.2}$$

$$R = \{(\theta_1, o_1), \ldots, (\theta_n, o_n)\} \tag{6.3}$$

Intensify procedure

The *Intensify* procedure, see Algorithm 4 for a simplified form, provides valuable insight into the optimisation process. The intensify procedure starts with a single random negotiation instance on which it evaluates θ_{inc}. It then races new configurations from $\vec{\Theta}_{new}$ against the incumbent configuration, starting with a comparison on a single negotiation instance. If the challenging configuration loses, it is dropped immediately. If not, then it compares the two configurations on an incremented set of instances. Finally, the challenging configuration replaces the incumbent configuration if it outperformed the incumbent on all instances that the incumbent was previously run on. The performance metric used to compare the performance of configurations is identical to

Algorithm 3 Parallel Sequential Model-Based Optimisation [22] (SMBO)

Input	Θ	Configuration space
	Π	Negotiation instances
	O	Performance metric
	t_{opt}	Optimisation time budget
Variables	R_i	Runhistory of pool i
	R_{full}	Full runhistory of parallel pools, where $R_{full} = [R_1, \ldots, R_m]$
	$\mathcal{M}$	Random forest regression model
	$\vec{\Theta}_{new}$	List of promising configurations
Output	θ_{inc}	Optimised parameter configuration

1: $[R_i, \theta_{inc}] \leftarrow Initialise(\Theta, \Pi)$
2: **loop until** $GetTime() > t_{opt}$
3: $\quad R_{full} \leftarrow ReadParallelRunhistories()$
4: $\quad \mathcal{M} \leftarrow FitModel(R_{full})$
5: $\quad \vec{\Theta}_{new} \leftarrow SelectConfigurations(\mathcal{M}, \theta_{inc}, \Theta)$
6: $\quad [R_i, \theta_{inc}] \leftarrow Intensify(\vec{\Theta}_{new}, \theta_{inc}, R_i, \Pi, O)$
7: **return** θ_{inc}

Equation 6.1, but is calculated using a subset of instances to approximate the actual performance (Equation 6.4).

$$O(\theta, \Pi) \approx O(\theta, \Pi') = \frac{1}{|\Pi'|} \sum_{\pi \in \Pi'} o(\theta, \pi) \tag{6.4}$$

where:

Π' : Subset of instance, where $\Pi' \subseteq \Pi$

6.2.2 Adding features

The random forest regression model tries to capture relations between parameter configurations and performances by training on the runhistory as illustrated in Equation 6.3. This runhistory is gathered by running a configuration θ_i on an instance $\pi \in \Pi$ and observing its obtained utility o_i. However, the instances vary in opponents and scenarios, meaning that a configuration θ_i might obtain a different utility on another instance. The current regression model does not consider differences between instances and assumes a direct relation between parameter configuration and performance. With sufficient training data, the model captures configuration trends that perform well in general, rendering it capable of finding promising configurations.

Hutter et al. [22] introduced the use of features to capture the differences between instances and implemented them into SMAC. Given a vector of features $\mathcal{F}(\pi) = x \in X$ describing a training instance $\pi \in \Pi$, a model is fitted using the joint vector of configuration and features $[\theta_i, \mathcal{F}(\pi)]$ with its observed utility o_i as output. We make a distinction between opponent features and scenario features, as the two feature spaces are supposedly independent. Therefore, the total feature space X is the Cartesian product between the opponent feature space and the scenario feature space

Algorithm 4 $Intensify(\vec{\Theta}_{new}, \theta_{inc}, R, \Pi, O)$ [22]

Input	$\vec{\Theta}_{new}$	List of promising configurations
	θ_{inc}	Incumbent configuration (current best)
	R	Runhistory
	Π	Negotiation instances
	O	Performance metric
	t_{int}	Time budget for intensify procedure
Variables	θ_{new}	Challenging configuration
Output	R	Runhistory
	θ_{inc}	Incumbent configuration (current best)

for $i := 1, \ldots, |\vec{\Theta}_{new}|$ **do**
  $\Pi' \leftarrow \{\pi' \in \Pi : Count(\theta_{inc} \text{ on } \pi') \leq Count(\theta_{inc} \text{ on } \pi''), \forall \pi'' \in \Pi\}$
  $\pi \leftarrow Random(\Pi')$
  $R \leftarrow ExecuteNegotiation(R,\ DA(\theta_{inc}),\ \pi)$
  $\theta_{new} \leftarrow \vec{\Theta}_{new}[i]$
  $N \leftarrow 1$
  loop
    $\Pi_{missing} \leftarrow \{\pi \in \Pi : Exists(\theta_{inc} \text{ on } \pi) \wedge \neg Exists(\theta_{new} \text{ on } \pi)\}$
    $\Pi_{torun} \leftarrow$ random subset of $\Pi_{missing}$ of size $Min(N,\ |\Pi_{missing}|)$
    for $\pi \in \Pi_{torun}$ **do** $R \leftarrow ExecuteNegotiation(R,\ DA(\theta_{new}),\ \pi)$
    $\Pi_{missing} \leftarrow \Pi_{missing} / \Pi_{torun}$
    $\Pi_{common} \leftarrow \{\pi \in \Pi : Exists(\theta_{new} \text{ on } \pi) \wedge Exists(\theta_{inc} \text{ on } \pi)\}$
    if $O(\theta_{new},\ \Pi_{common}) < O(\theta_{inc},\ \Pi_{common})$ **then break**
    else if $\Pi_{missing} = \emptyset$ **then** $\theta_{inc} \leftarrow \theta_{new}$; **break**
    else $N \leftarrow 2 * N$
  if $(GetTime() > t_{int}) \wedge i \geq 2$ **then break**
return $[R, \theta_{inc}]$

$X = (X_{opp} \times X_{sc})$. The regression model that maps the combined configuration space and feature space to a predicted utility is described by Equation 6.5. The model now considers differences of instances when predicting performance of a configuration over the full set of instances. The expanded version of the runhistory is illustrated in Equation 6.6.

$$\mathcal{M} : (\Theta \times (X_{sc} \times X_{opp})) \rightarrow \hat{o} \tag{6.5}$$

$$R = \{(\langle \theta_1, \mathcal{F}(\pi) \rangle, o_1), \ldots, (\langle \theta_n, \mathcal{F}(\pi) \rangle, o_n)\} \tag{6.6}$$

We already defined features of a negotiation instance in Chapter 5. If these features manage to capture the differences between instances accurately, then we should observe an improved optimisation result compared to optimisation without features.

6.3 Baselines

We must set baseline configurations to analyse the result of the optimisation. The basis of our Dynamic Agent is derived from a paper by Lau et al. [29]. Though we added some functionalities, it is possible to set our agents strategy configuration to resemble that of the original agent. We refer to this configuration from the literature as θ_{lit}, its parameters can be found in Table 6.2.

The paper was published in times when there was not a general testbed for automated negotiations agent like GENIUS. As a result this agent was only tested against itself on a single domain with varying preference profiles. We decided to add a baseline configuration by hand, as the literature configuration is outdated. To achieve this configuration, we use a combination of intuition, past research, and manual search, which we consider the standard procedure for all agents that are tuned by hand. We present the hand tuned configuration parameters θ_{hand} in Table 6.2 and an explanation below:

- *Accepting*: The acceptance condition parameters of the literature configuration set a pure AC_{next} strategy with parameters $\alpha = 1,\ \beta = 0$. Baarslag et al. [45] performed an empirical research on a variety of acceptance conditions and showed that there are better alternatives. We set the accepting parameters of our configuration to the best performing condition as found by Baarslag et al. [45].
- *Fitness function*: Preliminary testing showed that the literature configuration concedes much faster than the average ANAC agent, resulting in a poor performing strategy. We set a more competitive parameter configuration for the fitness function by manual search, to match the competitiveness of the ANAC agents.
- *Space exploration*: The domain used in the paper has a relatively small set of outcomes. We increased the population size, added an extra evolution to the genetic algorithm and made some minor adjustments to cope with larger outcome spaces.

	Accepting				Fitness function			Space exploration					
θ	α	β	t_{acc}	γ	n_{fit}	δ	e	N_{pop}	N_{tour}	E	R_c	R_m	R_e
θ_{lit}	1	0	1	MAX^W	1	0.5	0.5	200	3	3	0.6	0.05	0.1
θ_{hand}	1	0	0.98	MAX^W	4	0.95	0.05	300	5	4	0.6	0.05	0.05

Table 6.2: Baseline configurations parameters

6.4 Method

Figure 6.1 provides an overview of the structure of the optimisation process.

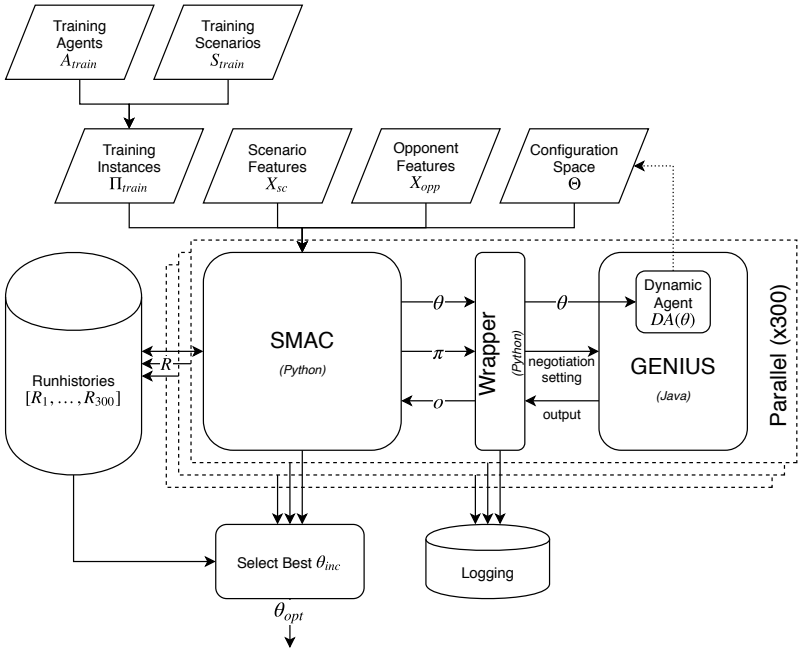


Figure 6.1: Overview of SMAC structure

Runtime specifics

The interaction between SMAC, the wrapper function, and GENIUS happen via the command line interface. SMAC is run in *embarrassingly parallel* mode on a computing cluster by starting a separate SMAC thread within every computation pool. SMAC selects a negotiation instance and a configuration to evaluate on that instance and calls a wrapper function that handles the communication with GENIUS. This wrapper function sends the configuration to the Dynamic Agent, creates the negotiation setting, calls GENIUS to execute the negotiation, post-processes

the results, and finally returns the obtained utility to SMAC.

Input

The training instances are provided in Appendix A, the scenario features are calculated in advance according to Section 5.1, and the configuration space is defined is Table 6.1.

The opponent features, as defined in Section 5.2, can only be gathered by performing negotiations against the opponent, which is not done in this thesis up until this point. To gather these features, we use our baseline hand configuration θ_{hand}, evaluate it 10 times on every training instance, and construct the features.

Hardware & budget

We run the optimisation in parallel spread over 300 computation pools on a computing cluster running Simple Linux Utility for Resource Management [49] (SLURM). As some opponent agents are very RAM inefficient, we had to assign 12 gigabytes of RAM to every pool. Each parallel optimisation pool receives a budget of 4 hours (t_{opt} in Algorithm 3), resulting in 1200 hours of total optimisation time or 2400 CPU hours (2 CPU's per pool). An overview of the hardware and budget is presented in Table 6.3.

Brand	Intel® Xeon® E5-2683 v4
Frequency	2.10 [GHz]
Cache L3	40 [Mb]
Computation pools	300
Cores per pool	1
CPU's per pool	2
RAM per pool	12 [Gb]
Budget per pool	4 [hours]

Table 6.3: Hardware & budget SMAC

Output

Every parallel optimisation problem outputs its own incumbent configuration θ_{inc} at the end of the optimisation process. As we ran 300 parallel processes, we had to decide which of the 300 incumbent configurations to use. To do so, we created a post-processing script that rebuilds the random forest regression model conform Equation 6.5 and used it to predict the performance of every incumbent over the full training set of instances. The incumbent with the best predicted performance was selected as final output.

6.5 Results

The results of the optimisation process and method described in this chapter, are presented with an emphasis on the following two topics:

1. The influence of the instance features on the optimisation process.
2. The performance of the optimised configurations.

6.5.1 Influence of instance features

To study the influence of the instance features on the optimisation process, we compare strategies obtained by optimisation with features and by optimisation without features. We also compare the strategies with the baseline. Only the training set of instances is used for the performance comparison, as we are purely interested in convergence of the optimisation. We have run the optimisation process as described in this chapter three times without instance features and three times with instance features, all with the same optimisation budget. A total of 8 strategies is obtained: 2 baselines $[\theta_{lit}, \theta_{hand}]$, 3 optimised without features $[\theta_1, \theta_2, \theta_3]$, and 3 optimised with features $[\theta_4, \theta_5, \theta_6]$. An overview of the final configurations is presented in Table 6.4.

	Accepting				Fitness function			Space exploration					
θ	α	β	t_{acc}	γ	n_{fit}	δ	e	N_{pop}	N_{tour}	E	R_c	R_m	R_e
θ_{lit}	1	0	1	MAX^W	1	0.5	0.5	200	3	3	0.6	0.05	0.1
θ_{hand}	1	0	0.98	MAX^W	4	0.98	0.05	300	5	4	0.4	0.05	0.05
θ_1	1.001	0.048	0.901	AVG^W	3	0.879	0.00183	345	10	4	0.437	0.003	0.176
θ_2	1.041	0.001	0.904	AVG^W	4	0.913	0.00130	384	5	4	0.431	0.126	0.198
θ_3	1.009	0.026	0.910	MAX^W	1	0.977	0.00113	361	2	5	0.279	0.181	0.072
θ_4	1.032	0.022	0.931	AVG^W	3	0.914	0.00429	311	8	3	0.251	0.082	0.132
θ_5	1.015	0.017	0.925	AVG^W	5	0.961	0.00105	337	5	3	0.192	0.090	0.138
θ_6	1.027	0.022	0.943	AVG^W	3	0.985	0.00227	283	7	4	0.294	0.057	0.156

Table 6.4: Optimised configurations parameters

We run each of these configurations 10 times on the set of training instances and calculate the average obtained utility by Equation 6.1. The results are presented in Table 6.5, including an improvement ratio over θ_{hand}.

6.5.2 Performance on test set

In the previous section we compared the obtained configurations by using the training set of instances. We showed that automated optimisation results in a better performing configuration than manual search on the training set of instances. However, it is still possible that overfitting plays a role in this process. Testing the optimised configurations on a never before seen set of opponent agents and scenarios is needed to rule this out. The test set of instances is presented in Appendix A.

θ	$O(\theta, \Pi)$	$\frac{O(\theta,\Pi)-O(\theta_{hand},\Pi)}{O(\theta_{hand},\Pi)}$	**Description**
θ_{lit}	0.533	-0.307	Literature
θ_{hand}	0.769	0	Hand tuned
θ_1	0.785	0.020	Optimised without features
θ_2	0.770	0.000	Optimised without features
θ_3	0.792	0.029	Optimised without features
θ_4	0.800	0.040	Optimised with features
θ_5	0.816	0.060	Optimised with features
θ_6	0.803	0.044	Optimised with features

Table 6.5: Optimised configurations performance on Π_{train}

Performance of configurations

We compare the configurations by running them 10 times on the test set of instances Π_{test} and calculate the performance by Equation 6.1. We compare the same configurations as presented in Table 6.4. The results are presented in Table 6.6, including an improvement ratio over θ_{hand}.

In both Table 6.5 and Table 6.6 we observe that SMAC is capable of improving the performance of the Dynamic Agent on the training set of negotiation instances. The usage of instance features leads to less variation in final configuration parameters (Table 6.4), which also reflects in the performance of these configurations.

θ	$O(\theta, \Pi_{test})$	$\frac{O(\theta,\Pi_{test})-O(\theta_{hand},\Pi_{test})}{O(\theta_{hand},\Pi_{test})}$	**Description**
θ_{lit}	0.563	-0.261	Literature
θ_{hand}	0.763	0	Hand tuned
θ_1	0.779	0.021	Optimised without features
θ_2	0.760	-0.004	Optimised without features
θ_3	0.774	0.015	Optimised without features
θ_4	0.792	0.038	Optimised with features
θ_5	0.795	0.042	Optimised with features
θ_6	0.789	0.034	Optimised with features

Table 6.6: Optimised configurations performance on Π_{test}

ANAC tournament performance of best configuration

The results from the previous section indicate that the configurator is successful in finding improved configurations, but the results are only compared against the other configurations of our Dynamic Agent. This does not say much about the performance compared to other negotiation agents. In this section, we show the performance of the best configuration that we found compared to the ANAC agents in the test set of opponents.

We select θ_5 as our best configurations based on performance on the training set and enter our

Dynamic Agent in an ANAC-comparable bilateral tournament with a 60 second deadline. We take the agents (16) and the scenarios (14) from the test set as defined in Appendix A and add our Dynamic Agent to the pool of agents. Next, we let every combination of 2 agents negotiate 10 times on both sides of every scenario, for a total amount of 38080 negotiation sessions (Equation 6.7). To speed up the process, we use a computing cluster to run the negotiations and assign that same hardware to a negotiation session as we assigned to a single SMAC run (Table 6.3). We present the results in Table 6.7.

Using the Dynamic Agent with θ_5 results in a successful negotiation agent that is capable of winning a ANAC-like bilateral tournament. We managed to obtain a $\frac{0.795-0.756}{0.756} * 100\% \approx 5.1\%$ higher utility than SimpleAgent, the number two in the ranking. We also beat the number two on every other performance measure and were able to achieve a fairly low average distance to the Pareto frontier. We refer the reader to Chapter 8 for a more elaborate discussion of the results.

$$\binom{17}{2} * 14 * 2 * 10 = 136 * 14 * 2 * 10 = 38080 \tag{6.7}$$

Agent	**Utility**	**Opp. utility**	**Social welfare**	**Pareto distance**	**Nash distance**	**Agreement ratio**
RandomCounterOfferParty	0.440	**0.957**	1.398	0.045	0.415	1.000
HardlinerParty	0.496	0.240	0.735	0.507	0.754	0.496
AgentH	0.518	0.801	1.319	0.118	0.408	0.904
ConcederParty	0.577	0.848	1.425	0.047	0.358	0.964
LinearConcederParty	0.600	0.831	1.431	0.046	0.350	0.964
PhoenixParty	0.625	0.501	1.125	0.263	0.468	0.748
GeneKing	0.637	0.760	1.396	0.061	0.383	0.993
Mamenchis	0.651	0.725	1.377	0.087	0.360	0.927
BoulwareParty	0.662	0.786	1.448	0.043	0.319	0.968
Caduceus	0.677	0.486	1.163	0.241	0.453	0.784
Mosa	0.699	0.640	1.339	0.113	0.385	0.902
ParsCat2	0.716	0.671	1.386	0.108	**0.286**	0.904
RandomDance	0.737	0.716	**1.453**	**0.024**	0.344	0.998
ShahAgent	0.744	0.512	1.256	0.188	0.389	0.821
AgentF	0.751	0.605	1.356	0.100	0.367	0.918
SimpleAgent	0.756	0.437	1.194	0.212	0.470	0.801
$DA(\theta_5)$	**0.795**	0.566	1.361	0.087	0.407	0.922

Table 6.7: Bilateral ANAC tournament results using $DA(\theta_5)$ (bold = best, underline = worst)

Chapter 7

Algorithm selection for negotiation

In Chapter 4, we developed a Dynamic Agent $DA(\theta)$ with a parameterized strategy that depends on a parameter configuration θ. We created a successful method to automatically configure this agent (Chapter 6), such that it outperforms both the comparable configuration from the literature θ_{lit} and the configuration that we tuned by hand θ_{hand}. However, up until this point, we ignored the fact that there is no single strategy that is optimal for all negotiation settings [11]. Although the configuration results in a well performing strategy, it is still a fixed strategy.

In this chapter, we add a layer of algorithm selection [19] to our Dynamic Agent to exploit the differences between negotiation instances. We attempt to improve the average obtained utility by switching between strategies depending on the negotiation instance at hand. Instead of a single best strategy, we aim to form a range of strategies that might include less efficient strategies that specialise on specific instances.

Figure 7.1 provides a schematic overview of the algorithm selection problem at hand.

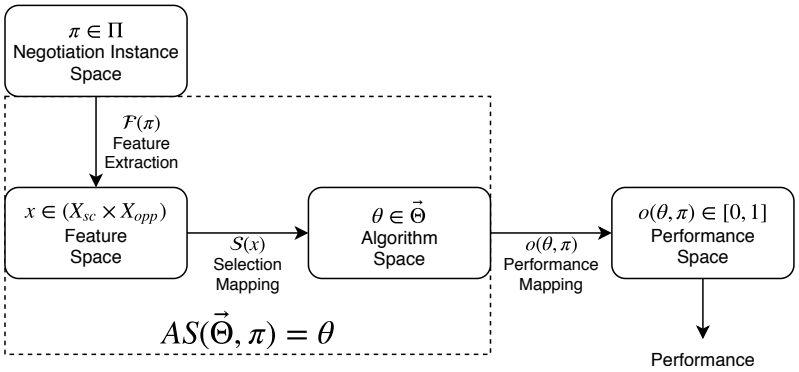


Figure 7.1: Algorithm selection schematics [19]

The subscript "*train*" (e.g. Π_{train}) is dropped throughout this chapter to improve readability.

7.1 Algorithm selector

Suppose we already have a portfolio of strategies $\vec{\Theta} = \{\theta_1, \ldots, \theta_n\}$ for the Dynamic Agent $DA(\theta)$, a feature space X that is the combined feature space of scenario features X_{sc} and opponent features X_{opp}, a training set Π and test set Π_{test} of negotiation instances. We must train an algorithm selector using the training set of negotiation instances to improve the performance of the Dynamic Agent on the test set. We define the algorithm selector $AS(\vec{\Theta}, \pi)$ as a mapping from the negotiation instance space to the algorithm space (Figure 7.1 & Equation 7.1).

$$AS : \Pi \times \vec{\Theta} \rightarrow \vec{\Theta} \tag{7.1}$$

Ilany et al. [18] also faced this algorithm selection problem and analysed the performance of multiple classifiers that map the feature space to the algorithm space. The process of selecting a classifier and configuring the accompanying parameters is again an algorithm configuration problem like we faced in Chapter 6. We chose to automate the configuration of an algorithm selector by using AutoFolio [50], leveraging the full power of a broad range of algorithm selection methods and removing human bias.

7.1.1 AutoFolio

The algorithm selection tool AutoFolio constructs the algorithm selector. The benefit of this tool is that we are not bothered with the decision which algorithm selection approach to use. The tool has a range of regression and classification methods to choose from and uses SMAC to determine both the selection method to use and the setting of its hyperparameters. The data AutoFolio requires as input is the performance $o(\theta, \pi)$ of every strategy on every instance and a feature vector x for every instance in the training set.

AutoFolio's goal is set to maximise utility for the instances in the training set. For each negotiation instance, it obtains the feature set and assigns a strategy from the portfolio based on the feature data. It then reads the performance of the selected strategy on that negotiation instance for optimisation purposes. As we collected performance data of every strategy on every negotiation instance in advance, no negotiation sessions had to be run during the optimisation process. As $|\Pi| = 1120$ and $X \in \mathbb{R}^{14}$, we can state that fitting a selector is computationally inexpensive relative to the process of gathering performance data on the negotiation instances in this thesis.

Cross validation

AutoFolio uses 10-fold cross validation during optimisation to avoid overfitting by dividing the negotiation instances in the training set in 10 subsets and leaving one subset out for performance testing. However, due to the nature of our negotiation instance being a combined problem of an

opponent and a scenario, this still leads to overfitting of the algorithm selector. We demonstrate this behaviour by example.

The training set of negotiation instances is the Cartesian product of the training set of opponents and scenarios $\Pi = A \times S$. We plot the training set of negotiation instances represented by square markers in Figure 7.2 and indicated a random subset of instances that represents a single fold Π_{fold}. We refer to the remaining negotiation instances as Π_{rest}. By fitting the algorithm selector to Π_{rest}, the unique combinations of opponent and scenario in Π_{fold} are not seen before at performance testing. However, we must be thoughtful that it is highly likely that we did see both the opponent and scenario before in Π_{rest}, but combined with a different scenario/opponent. As a result, the algorithm selector will overfit. A straightforward solution is to remove all the instances in Π_{rest} that contain either the opponent or the scenario, but this heavily reduces or even eliminates the entire set Π_{rest}.

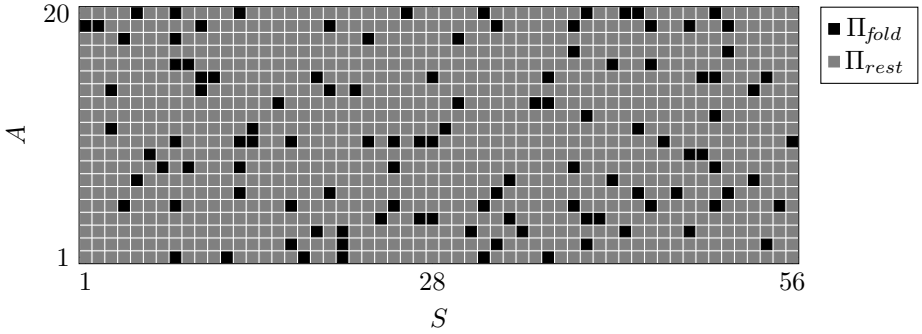


Figure 7.2: Default AutoFolio cross validation visualisation

As a solution, we modified AutoFolio to split the cross validation folds based on the set of opponents and scenarios that build the negotiation instances. Both the set of opponents and the set of scenarios are split into 4 subsets, such that $A_{fold} \subset A$, $|A_{fold}| = 5$ and $S_{fold} \subset S$, $|S_{fold}| = 14$. We now define a single cross validation fold as $\Pi_{fold} = A_{fold} \times S_{fold}$, so that the total amount of folds is $4 * 4 = 16$.

We eliminate the set of negotiation instances Π_{elim} that contain either an opponent or a scenario that is present in the fold Π_{fold} (but not both as those instances are in Π_{fold}). The remaining set of negotiations is again indicated by Π_{rest}, such that $\Pi = \Pi_{fold} \cup \Pi_{elim} \cup \Pi_{rest}$. We present a visualisation of the three separate instance sets for a single fold in Figure 7.3.

The presented approach of cross validation reduces the workable size of the instance set, but it does prevent overfitting. The decision to split both the opponent set and scenario set into 4 subsets is explained by the wish to maintain the ratio between training and test set size within the cross validation procedure to the original AutoFolio ratio. AutoFolio originally uses 10-fold cross validation, giving a train-test ratio of $\frac{1}{9}$. The presented approach results in a ratio of:

$$\frac{|A_{fold}| * |S_{fold}|}{|A \setminus A_{fold}| * |S \setminus S_{fold}|} = \frac{5 * 14}{15 * 42} = \frac{70}{630} = \frac{1}{9} \tag{7.2}$$

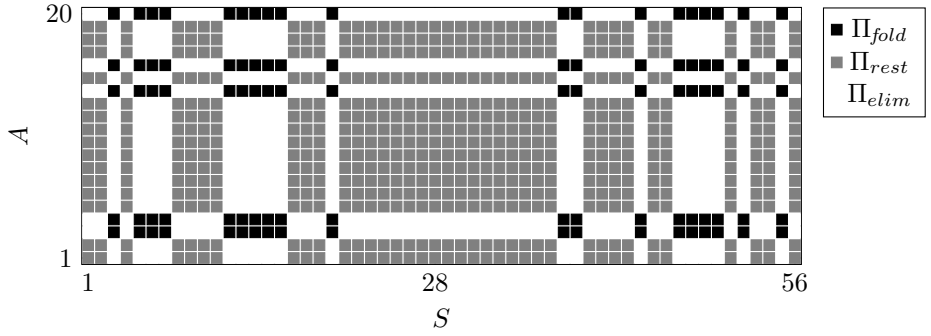


Figure 7.3: Modified cross validation visualisation

7.1.2 Performance measure

We measure the algorithm selector's performance as a normalised value between a baseline and the oracle selector (Equation 7.3) on the test set of negotiation instances. The oracle selector always makes the perfect choice for every negotiation instance and is an upper bound of selection performance of the considered portfolio.

We define the *single best strategy* as the fixed strategy in the portfolio that obtains the highest performance on the set of negotiation instances (Equation 7.4). We refer to this strategy as θ_1 as it is the first strategy in the portfolio due to the working of HYDRA (see next section). The performance of the single best strategy is considered to be the baseline. We define the performance measure in Equation 7.5, where $p(AS, \Pi) = 0$ means no improvement over the single best strategy and $p(AS, \Pi) = 1$ means perfect performance (Oracle).

$$OR(\vec{\Theta}, \pi) = \arg\max_{\theta \in \vec{\Theta}} o(\theta, \pi) \tag{7.3}$$

$$\theta_1 = \arg\max_{\theta \in \vec{\Theta}} O(\theta, \Pi) \tag{7.4}$$

$$p(AS, \Pi) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \left(\frac{o(AS(\vec{\Theta}, \pi), \pi) - o(\theta_1, \pi)}{o(OR(\vec{\Theta}, \pi), \pi) - o(\theta_1, \pi)} \right) \tag{7.5}$$

7.2 Algorithm portfolio

In this section we discuss the details of the problem of creating an algorithm portfolio that we defined in Section 3.2. Algorithm selection is the study of selecting the best performing algorithm for a certain problem at hand. In this thesis, the problems are the negotiation instances and the algorithms are negotiation strategies. The algorithm selector is discussed in the previous section, however, making a selection requires a portfolio of negotiation strategies to select from.

A simple solution is to build a portfolio of negotiation agents that already exist within the GENIUS environment, which is the approach used by Ilany et al. [18]. However, we consider this a less ideal

approach for the following reasons:

1. It requires a number of agents that already exist, which is the case for the negotiation settings used in this thesis, but might not be the case if the settings are changed (e.g. using a different protocol).
2. The agents might all be optimised with a different performance measure than we use for algorithm selection (optimised for Nash solution vs. optimised for utility), resulting in a low performing portfolio to start with.
3. There might be agents in the portfolio that are always outperformed by another agent. They only increase the size of the portfolio without contributing, needlessly complicating the algorithm selection problem.
4. The portfolio might not be robust, i.e. there can be a negotiation instance for which all the negotiation agents in the portfolio fail to achieve a decent performance, thus introducing "weak spots" in our agent.

In Chapter 6, we developed a method to automatically set a well performing negotiation strategy to our Dynamic Agent. We used the method to create a single best strategy based on the performance on the training set of negotiation instance. However, due to the flexible nature of the Dynamic Agent, there is more than a single strategy to find within its hyperparameter space. Theoretically, the hyperparameter space of the Dynamic Agent can be increased until every possible strategy is within its domain, but we will continue with the Dynamic Agent as defined in Chapter 4.

We aim to use the automated configuration method and the Dynamic Agent to create multiple strategies for the portfolio. This solves the issue of relying on a pool of already existing negotiation agents as it only requires a single Dynamic Agent. It also allows us to specialise the portfolio, since we can influence both the training set of instances as well as the performance measure used for configuration. This can be useful if we are certain about boundaries in the negotiation instance space that the agent will face (e.g. only larger domains, no discount, etc.), or if we aim to reach a fair share of utility instead of being greedy.

The portfolio of strategies we create is thus a portfolio of configurations for our Dynamic Agent $\vec{\Theta}$. We set the condition that every strategy must add value to the portfolio:

$$\forall \theta \in \vec{\Theta}\ \exists \pi \in \Pi\ \forall \theta' \in (\vec{\Theta} \setminus \{\theta\}) : o(\theta, \pi) > o(\theta', \pi) \tag{7.6}$$

Which means that for every strategy θ there must exist an instance π, such that using θ on that instance provides better results than any of the other configurations in $\vec{\Theta}$.

The portfolio can be viewed as a set of strategies that each specialise on a sub-space of the negotiation instance space. Similarities in the negotiation instance space are found by mapping the space to the feature space. A simple solution to achieve a portfolio is to divide the feature space of the negotiation instances into multiple sectors by hand and run the configurator on training instances who's features lie within that sector. We than obtain a portfolio of configurations that each specialise on a subspace of the feature space. We risk, however, cutting the feature space in such a way that we split negotiation instances that are similar.

A more sophisticated approach would be to use clustering techniques to cluster the training instances based on their features and run the configurator on every cluster separately. We thus would

obtain a portfolio of strategies that each specialise on a cluster of negotiation instances. However, both approaches lean on human input (how many sectors/clusters, which clustering technique, where to cut the feature space, etc.) and dictate the configurator on which sub-spaces it can optimise. The quality of the sectors/clusters is disputable, as they are created based on similarities in the feature space without regards for the performance gain it may bring. Instead we chose to automate the portfolio creation method by using HYDRA [21], removing the requirement of human input in problem space separation.

7.2.1 HYDRA

HYDRA automatically generates a portfolio given only a parameterised algorithm (Chapter 4) and a set of negotiation instances with features (Chapter 5) while using an algorithm configurator (Chapter 6) and an algorithm selector (Section 7.1). We provide a pseudocode description of HYDRA in Algorithm 5, modified for this thesis. The input is already introduced throughout this thesis, as well as the "*RunAlgorithmConfigurator*" procedure (Algorithm 3) and the algorithm selector (Section 7.1).

Algorithm 5 HYDRA [21]

Input	Θ	Configuration space
	Π	Training set of negotiation instances
	O	Performance metric
Variables	θ_k	Configuration
	$\vec{\Theta}$	Portfolio of configurations
	O_k	Modified performance metric
	AS	Algorithm selector
Output	$\vec{\Theta}$	Portfolio of configurations

$k \leftarrow 1$
$\theta_1 \leftarrow RunAlgorithmConfigurator(\Theta, \Pi, O)$
$TestPerformance(\Pi, \theta_1)$
$\vec{\Theta} \leftarrow \{\theta_1\}$
$AS \leftarrow FitAlgorithmSelector(\vec{\Theta}, \Pi)$
loop until *condition*
    $k \leftarrow k + 1$
    $O_k \leftarrow GetModifiedPerformanceMetric(O, AS)$
    $\theta_k \leftarrow RunAlgorithmConfigurator(\Theta, \Pi, O_k)$
    $TestPerformance(\Pi, \theta_k)$
    $\vec{\Theta} \leftarrow \vec{\Theta} \cup \{\theta_k\}$
    $AS \leftarrow FitAlgorithmSelector(\vec{\Theta}, \Pi)$
return AS, $\vec{\Theta}$

The main idea of HYDRA is to run multiple identical configurator sessions on an identical training set of instances, while only modifying the performance metric. Due to the modifications to the metric, the configurator converges towards different strategies. In Algorithm 5, the modified performance metric is obtained through "*GetModifiedPerformanceMetric*". We provide a formal

definition of the modified performance measure in Equation 7.7.

$$O_k(\theta, \Pi) = \sum_{\pi \in \Pi} \max\Big(o(\theta, \pi),\ o(AS(\vec{\Theta}, \pi), \pi)\Big) \tag{7.7}$$

The modified performance is the better of the strategy that is assessed and the performance of the strategy that is selected by the algorithm selector. By optimising using the increase of performance as compared to the current portfolio, the configurator converges to the configuration that adds the most value to the portfolio. In other words, the automated configuration will "fill the weaker spots" of the current portfolio in the negotiation instance space.

At the first configurator run, the default performance metric is used. The resulting configuration θ_1 is, therefore, the configuration that works best in general over the full set of training instances, also known as the *single best strategy*. We already ran the configuration process under these conditions three times in Chapter 6, so we select θ_5 from Table 6.4 and rename it to θ_1 in this current chapter.

7.3 Method

Figure 7.4 provides an overview of the total HYDRA procedure, which is explained in this section.

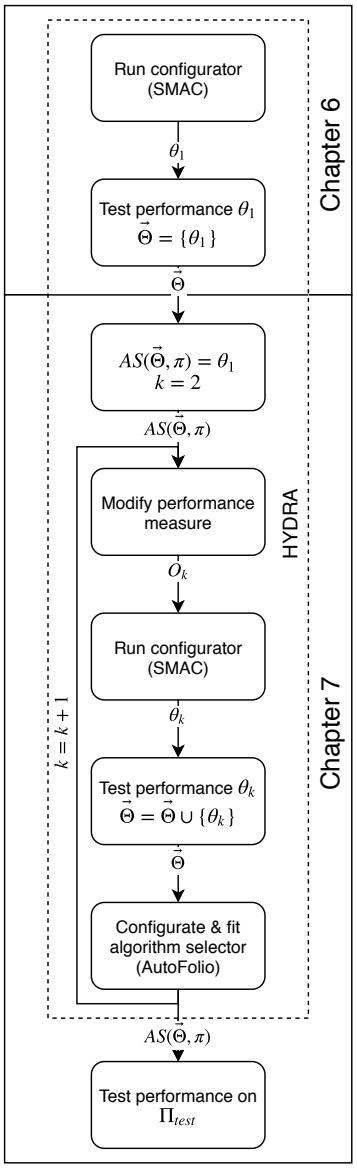


Figure 7.4: Overview HYDRA

Runtime specifics

The first configurator run with the default performance metric results in the single best strategy θ_1 on the training set of negotiation instances. This step is the main topic of Chapter 6, where we obtain three of these configurations using instance features. We pick the best performing configuration θ_5 from Table 6.4 based on observed performance on the training set of negotiation instances as our single best strategy θ_1 and use it to initialise the portfolio and the algorithm selector.

We aim to complement the portfolio with an additional three strategies, so we iterate through HYDRA until $|\vec{\Theta}| = 4$ keeping a small portfolio for initial testing. It also allows us to analyse the performance of portfolios of size 1, 2 and 3 due to the incremental approach of HYDRA. The working of SMAC is elaborated upon in Chapter 6. The obtained configuration is tested 10 times on every negotiation instance in the training set and the obtained

utility is stored on disc. Finally the portfolio and the performance data is used along with the instance features, all in csv-file format, to configure an algorithm selector using AutoFolio. The configured algorithm selector is fitted to the training set and saved as binary file to disk.

Input

The training instances are provided in Appendix A, the scenario features are calculated in advance according to Section 5.1, and the configuration space is defined is Table 6.1.

The opponent features, as defined in Section 5.2, can already be extracted from the negotiation results of Chapter 6. Since θ_1 is the single best strategy, we consider θ_1 to be the default configuration for the DAP feature. We will not update the opponent features during the HYDRA procedure.

Hardware & budget

We ran SMAC under identical conditions and as we did in Chapter 6, so we refer the reader to Table 6.3 for details on hardware and budget.

Running AutoFolio for our problem is not computationally expensive, so we chose to not run it in parallel for convenience. We used a single dual core processor on the computing cluster, assigned it 4 gigabytes of RAM, and provided it with a budget of 0.5 hours. An overview of the hardware and budgets is provided in Table 7.1.

Brand	Intel® Xeon® E5-2683 v4
Frequency	2.10 [GHz]
Cache L3	40 [Mb]
Cores	1
CPU's	2
RAM	4 [Gb]
Budget	0.5 [hours]

Table 7.1: Hardware & budget AutoFolio

Output

The fitted algorithm selector is saved as a binary file at the final step of HYDRA and forms the final result of the process. If we are faced with a new negotiation instance for which we want to select a configuration, we obtain the instance features, load the binary file, use the selector to select a configuration, and start the negotiation session with that configuration.

7.4 Results

We discussed the process and method of building a portfolio and selecting configurations in this chapter. We now present the results of this process. More specifically, there are two topics we address:

1. The quality of the portfolio.
2. The performance of the algorithm selector.

7.4.1 Quality of the portfolio

By applying HYDRA, we obtained an algorithm selector with a portfolio of four configurations, which correspond to four strategies, which are presented in Table 7.2. We tested the quality of the portfolio by running every configuration on the train and test set of negotiation instances. The results can be found in Table 7.3.

	Accepting				**Fitness function**			**Space exploration**					
θ	α	β	t_{acc}	γ	n_{fit}	δ	e	N_{pop}	N_{tour}	E	R_c	R_m	R_e
θ_1	1.015	0.017	0.925	AVG^W	5	0.961	0.00105	337	5	3	0.192	0.090	0.138
θ_2	1.073	0.049	0.916	AVG^W	4	0.951	0.01920	362	3	1	0.460	0.198	0.020
θ_3	1.015	0.001	0.939	MAX^W	4	0.998	0.00133	367	8	2	0.146	0.001	0.009
θ_4	1.018	0.001	0.993	AVG^W	3	0.914	0.05114	365	10	1	0.488	0.170	0.195

Table 7.2: Portfolio configurations parameters

		Best performing on Π by ratio				
θ	$O(\theta, \Pi)$	Best	Single best	1 of 2 best	1 of 3 best	1 of 4 best
θ_1	0.816	0.579	0.213	0.152	0.121	0.094
θ_2	0.772	0.233	0.077	0.025	0.038	0.094
θ_3	0.808	0.598	0.156	0.201	0.147	0.094
θ_4	0.785	0.388	0.093	0.062	0.139	0.094

		Best performing on Π_{test} by ratio				
θ	$O(\theta, \Pi_{test})$	Best	Single best	1 of 2 best	1 of 3 best	1 of 4 best
θ_1	0.793	0.576	0.188	0.145	0.087	0.156
θ_2	0.767	0.308	0.071	0.027	0.049	0.156
θ_3	0.782	0.621	0.150	0.183	0.132	0.156
θ_4	0.780	0.438	0.094	0.060	0.127	0.156

Table 7.3: Individual configuration performance on Π and Π_{test}

For every configuration in the portfolio, the performance on a set of instances is calculated according to Equation 6.1. We also include ratios to demonstrate how many times on the instances a configuration is the best, the single best, one of two best, one of three best, or one of four best in the portfolio.

As a final quality check, the performance of the oracle selector (Equation 7.3) is calculated for varying sizes and combinations of the portfolio. This selector always makes the best choice for every instance and is an upper limit of performance of the Dynamic Agent. We present the results in Table 7.4.

$\vec{\Theta}$	$O(OR(\vec{\Theta}, \pi), \Pi)$	$O(OR(\vec{\Theta}, \pi), \Pi_{test})$
$\{\theta_1\}$	0.816	0.793
$\{\theta_1, \theta_2\}$	0.868	0.857
$\{\theta_1, \theta_2, \theta_3\}$	0.876	0.860
$\{\theta_1, \theta_2, \theta_3, \theta_4\}$	0.882	0.870
$\{\theta_1, \theta_3\}$	0.846	0.819
$\{\theta_1, \theta_4\}$	0.866	0.855

Table 7.4: Oracle selector performance

Table 7.2 shows the selected parameter values of all strategies in the portfolio, separated by agent component (Chapter 4). The configuration space in which the values of these parameters lie can be found in Table 6.1. The differences between the strategies are clearly visible.

Table 7.3 shows the results per strategy in the portfolio in the form of an individual performance over a set of instances $O(\theta, \Pi)$. It is clearly visible that θ_1 is the single best strategy over the full set Π. Furthermore, as every strategy is at least once the single best on individual instances (single best ratio > 0), we can conclude that every strategy contributes to the portfolio.

Finally, Table 7.4 shows us that, at every iteration of HYDRA, potential performance of the portfolio is increased on both Π and Π_{test}. This improvement decreases on Π as the amount of iterations increase, indicating that HYDRA finds the largest "weaknesses" in the portfolio first. We also show the performance of alternative portfolios by combining arbitrary iterations of HYDRA.

Further discussion on the results can be found in Chapter 8.

7.4.2 Performance of the algorithm selector

We now present the performance results of the algorithm selector. The results of Section 7.4.1 show that there is potential in the portfolio to improve utility of the Dynamic Agent by $0.870 - 0.793 = 0.077$ on the test set of instances if we use the Oracle selector. In this section, we replace the Oracle selector with the actual selector and test its performance in two parts:

- In the first part we assume that opponent features are already fully known to the Dynamic Agent, as if we already faced the opponents before. Doing so allows us to observe the absolute performance of the algorithm selector by eliminating the lack of opponent features of first encounters.

- In the second part we once again test our agent in an ANAC tournament setting, comparing our agents performance against other agents from the test set. We simulate a realistic setting including first encounters of opponents at which there is a lack of opponent features.

Absolute performance of algorithm selector

In this section, we test the absolute performance of the algorithm selector by assuming perfect knowledge of opponent features of the opponents in the test set of negotiation instance Π_{test}. The opponent features are gathered by running 10 negotiation sessions with configuration θ_1 on the test set of instances.

We train and test multiple algorithm selectors on different portfolio sizes by extending the portfolio as we did in Table 7.4, starting with the single best strategy θ_1. We report both the performance of the agent using Equation 6.1, as well as the normalised performance of the algorithm selector according to Equation 7.5 per portfolio $\vec{\Theta}$. The results are presented in Table 7.5.

On both Π and Π_{test} the performance of the Dynamic Agent increases as the size of the portfolio increases. The normalised performance $p(AS, \Pi)$ also increases for Π, but fluctuates for Π_{test}. The "N/A" values are explained by Equation 7.5, which calculates performance of the portfolio relative to the single best strategy θ_1. Since $\{\theta_1\}$ is a portfolio with only the single best strategy, the value would make no sense. Further discussion on the absolute performance can be found in Chapter 8.

	Π		Π_{test}	
$\vec{\Theta}$	$O(AS(\vec{\Theta}, \pi), \Pi)$	$p(AS, \Pi)$	$O(AS(\vec{\Theta}, \pi), \Pi_{test})$	$p(AS, \Pi_{test})$
$\{\theta_1\}$	0.816	N/A	0.793	N/A
$\{\theta_1, \theta_2\}$	0.853	0.725	0.819	0.416
$\{\theta_1, \theta_2, \theta_3\}$	0.861	0.760	0.810	0.256
$\{\theta_1, \theta_2, \theta_3, \theta_4\}$	0.872	0.856	0.825	0.419

Table 7.5: Absolute algorithm selector performance

ANAC tournament performance of algorithm selector

The final results we present in this thesis combines the effort of all previous chapters. We test the performance of the Dynamic Agent with a portfolio of strategies in a bilateral ANAC tournament setup where opponent learning between negotiation sessions is allowed. The tournament is run using the test set of agents, the test set of scenarios and a deadline of 60 seconds to resemble the original deadline of 180 seconds.

First encounters Up until now, we assumed opponent features to be fully known, which is not realistic. Opponent features cannot be calculated in advance, in contrast to the scenario features, but must be learned from previous encounters. In this section, we simulate a realistic negotiation tournament where this problem occurs. The question rises what strategy to select at first encounters with opponents, when no opponent features are available. The first strategy in the

portfolio is the single best strategy due to the working of HYDRA, which is confirmed in Table 7.3. The logical answer is, therefore, to select the single best strategy θ_1 if no opponent features are available and algorithm selection is impossible. We illustrate this behaviour in Figure 7.5.

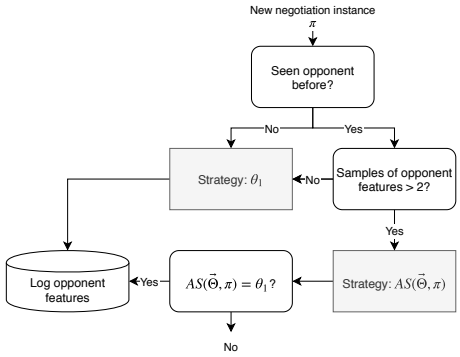


Figure 7.5: Realistic strategy selection of Dynamic Agent

Opponent features All opponent features in this chapter are pre-calculated by negotiation with the single best strategy θ_1, which is also required to calculate the DAP feature. Opponent features are influenced by the strategy that is selected by the Dynamic Agent, so we simplify the feature extraction process and only gather features when strategy θ_1 is selected. This aligns with the decision to select θ_1 at first opponent encounters.

Both the CoV and the mean of every opponent feature are stored for future use (Section 5.2). The CoV of a feature needs at least two samples to be meaningful, so we set a second condition that forces the Dynamic Agent to select strategy θ_1 for the first two encounters with an opponent (Figure 7.5).

Results To obtain the results, we take the Dynamic Agent with $\vec{\Theta} = \{\theta_1, \theta_2, \theta_3, \theta_4\}$ and the training set of negotiation instances Π_{test}. We iterate randomly through the test instances and use the Dynamic Agent to negotiate following the procedure as illustrated in Figure 7.5. Additionally, we let every opponent in the test set negotiate with every other opponent in the test set on every test scenario and combine the results with the results of the Dynamic Agent. This procedure is repeated 10 times to reduce influence of variance for a total of 38080 negotiations. We present the results averaged per agent in Table 7.6.

Finally, we compare the performances of three Dynamic Agents in a realistic ANAC tournament setup in Figure 7.6:

1. Dynamic Agent with the hand tuned strategy θ_{hand} that was used throughout Chapter 6
2. Dynamic Agent with the single best strategy θ_1 from Table 7.2
3. Dynamic Agent with the strategy selector $AS(\vec{\Theta}, \pi)$ with $\vec{\Theta} = \{\theta_1, \theta_2, \theta_3, \theta_4\}$

Agent	Utility	Opp. utility	Social welfare	Pareto distance	Nash distance	Agreement ratio
RandomCounterOfferParty	0.440	**0.956**	1.397	0.046	0.415	**1.000**
HardlinerParty	0.504	0.244	0.748	0.498	0.747	0.504
AgentH	0.509	0.802	1.312	0.122	0.411	0.900
ConcederParty	0.577	0.848	1.425	0.046	0.357	0.964
LinearConcederParty	0.600	0.831	1.431	0.046	0.350	0.964
PhoenixParty	0.627	0.504	1.131	0.259	0.464	0.751
GeneKing	0.637	0.760	1.397	0.061	0.383	0.993
Mamenchis	0.652	0.725	1.377	0.087	0.360	0.927
BoulwareParty	0.662	0.786	**1.448**	0.043	0.318	0.968
Caduceus	0.680	0.491	1.171	0.236	0.447	0.790
Mosa	0.699	0.640	1.339	0.113	0.385	0.902
ParsCat2	0.710	0.673	1.383	0.111	**0.287**	0.901
RandomDance	0.737	0.716	1.453	**0.024**	0.344	0.998
ShahAgent	0.744	0.517	1.260	0.186	0.384	0.823
AgentF	0.751	0.605	1.356	0.100	0.365	0.918
SimpleAgent	0.760	0.443	1.203	0.207	0.463	0.806
$DA(AS(\vec{\Theta}, \pi))$	**0.819**	0.570	1.390	0.069	0.381	0.939

Table 7.6: Bilateral ANAC tournament results using $DA(AS(\vec{\Theta}, \pi))$ (bold = best, underline = worst)

In Table 7.6 we see that we are capable of winning an ANAC-like bilateral tournament with our Dynamic Agent using the strategy selector. We outperform the number two (SimpleAgent) by $\frac{0.819-0.760}{0.760} * 100\% \approx 7.8\%$, which is a significant increase. We also beat the number two on all the other performance measures. Further discussion on the result can be found in Chapter 8.

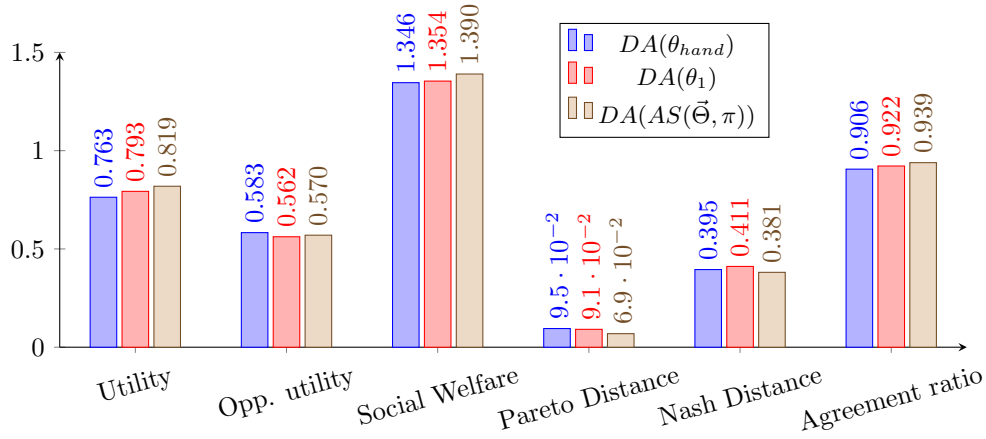


Figure 7.6: Comparison of three Dynamic Agents strategies in an ANAC tournament setting

Chapter 8

Discussion

We created a negotiation agent with a flexible strategy that is capable of winning an ANAC-like bilateral tournament when properly configured. We showed that configuring this agent can be done automatically to achieve such a winning strategy, even with a computational expensive training set. Finally, we showed that we can find multiple strategies within the strategy space of the Dynamic Agent that complement each other and can apply strategy selection to improve agent performance.

In this chapter we will discuss the contributions of this thesis and interpret the most important results split into three topics in line with the thesis: The Dynamic Agent, Strategy configuration and Strategy selection. We finalise the chapter by discussing future work.

8.1 Dynamic Agent

The agents that are designed for the Automated Negotiating Agents Competition [12] (ANAC) contain parameters that are set by its creators. Obtaining a well performing strategy depends on the settings of the parameters, which requires human effort to optimise. We faced the same problem with the Dynamic Agent that we developed in Chapter 4, but we automated this configuration process. We only define the domain of a parameter (so-called hyperparameter) that is likely to contain a successful strategy, instead of finding the exact setting for a successful strategy.

The configuration process used (SMAC) allows specification of categorical parameters and conditional parameters, providing large flexibility to the Dynamic Agent. Additional functionalities with their own parameters can be made optionally by setting a boolean parameter with conditionals. This allows us to add functionalities to the agent while being unsure if it is beneficial for agent performance. The configuration process will decide whether to use the module or not.

8.2 Strategy configuration

Developing a successful strategy for the ANAC is not straightforward, we can justify this statement by looking at the differences in utility between the winning and losing agents in the ANAC. We attempted to set our Dynamic Agent by hand θ_{hand} to create a decent strategy and were quite successful in achieving a high utility, but we must keep in mind that the opponents were designed for different negotiation settings and that we had access to more modern literature. Despite already being a successful strategy, we managed to improve the utility of the Dynamic Agent by $\frac{0.795-0.763}{0.763} * 100\% \approx 4.2\%$ by using automated configuration methods. This is a considerable increase in utility and suggests that manual search configuration for such parameterised agents is less optimal.

Suppose we would use a configuration process that requires testing of strategies on the full set of training instances, then a time budget of 18.7 hours (Appendix A) would be required per challenging strategy. The total configuration budget that is assigned in this thesis is 1200 hours (Table 6.3), which allows approximately 64 strategies to be tested on the full training set in a single configuration run. Far to few considering the configuration space of the Dynamic Agent and the random nature of many automated configuration methods. By using SMAC, we optimised on subsets of the training set, enabling early detection of low performing configurations. Even with the relatively low budget, we managed to obtain consistent performance gains and similar configurations on three independent configuration runs. We, therefore, deem SMAC to be an effective strategy configurator for automated negotiation agents.

The training set is a good representation of the test set of instances, as the improvement ratios compared to the hand tuned configuration show similarities for both sets. The best configuration on the training set θ_5 is also the best configuration on the test set and sets an improvement of 4.2% over the hand tuned configuration θ_{hand}. Surprisingly, this configuration also outperformed all other agents in the ANAC tournament simulation and beat the second place winner on all 5 measures. Also surprising is the position of the ANAC 2016 winner Caduceus, which is unable to achieve a decent agreement ratio. Caduceus applied a very competitive strategy, which might have back fired in the composition of test agents used in this agent.

8.2.1 Strategies

The baseline strategy that resembled the configuration of the original agent θ_{lit} performed very poor, as we expected. Compared to the strategy that is optimised by hand θ_{hand}, it results in a 31% decrease of average obtained utility.

The three best strategies $[\theta_4, \theta_5, \theta_6]$ show some interesting settings. The parameter t_{acc} of all three strategies is set to a value that was not included in the range that Baarslag et al. [45] defined in their grid search for the best possible acceptance condition. The parameter setting $\gamma = MAX^W$ was found to be superior by Baarslag, yet none of the strategies preferred that setting. We also left both α and β flexible within AC_{combi}, instead of setting them to 1 and 0 respectively as was done by Baarslag, which seems to be less optimal.

All three strategies use a fitness function where the opponent part ($f_o(\omega, t)$ in Equation 4.2) does not depend on the last offered outcome x^{last} as proposed by Lau et al. [29]. Both the trade-off

factor δ and the concede eagerness e are also set much more selfish than the original proposed values by Lau. Finally, the concede eagerness factor that we set for our hand tuned strategy is still more then 10 times higher than the values obtained via configuration.

8.3 Strategy selection

We repeated the statement that there is no single best strategy optimal for all negotiation settings [11] multiple times. In Chapter 6, we obtained a single best strategy that was already performing well by winning an ANAC-like tournament with a margin. Despite being a successful strategy, we found an additional 3 strategies within the strategy space of the Dynamic Agent using HYDRA that boost performance by nearly 10% in case of perfect selection. This increase is substantial and justifies the statement.

The quality of the portfolio is good, but the performance of the strategy selector on the test set is not in line with the performance on the training set, so we conclude that the selector is overfitted. A possible reason can be the small size of the workable training set (Figure 7.3) during configuration with AutoFolio. Nevertheless, 41.9% of the potential performance increase (0.077) on the test set is reached, but we must keep in mind that these results are achieved by assuming fully known opponent features.

Applying the algorithm selector with the full portfolio (Table 7.2) in a realistic ANAC-like tournament without pre-known opponent features yields an improvement in performance of $0.819 - 0.793 = 0.026$ compared to the single best strategy. As the single best strategy was already the winning strategy in an anac setup using the test set of opponents and scenarios, this achievement furthers our dominance in performance.

It is interesting that the performance of the Dynamic Agent, in the realistic setting with first encounters, approaches the performance of the case where opponent features are known in advance. The final performance of the Dynamic Agent with strategy selection is 7.8% higher than the performance of the second placed SimpleAgent in an ANAC-like tournament setup. This is a significant margin that was not achieved with the single best strategy. We conclude that strategy selection can greatly improve agent performance if repeated encountering of opponents is applicable.

Finally, we compared the performance of the Dynamic Agent with the baseline strategy θ_{hand}, the single best strategy θ_1 and the strategy selector $AS(\tilde{\Theta}, \pi)$ in Figure 7.6. We note that the single best strategy is more competitive than the baseline as it increases our utility and decreases opponent utility, but still manages to increase the agreement ratio. What is also notable, is that the utility of both the Dynamic Agent and the opponent increase by using the algorithm selector instead of the single best strategy. Being able to find outcomes with a higher utility for the opponent increases our chances of them accepting our offer, which in turn increases our utility.

8.3.1 Portfolio

The usage of HYDRA automates the creation of a portfolio for the Dynamic Agent, without requiring human input in making divisions in the problem space for specialised strategies. As HYDRA always uses the full set of negotiation instances for portfolio construction, the risk of

weaknesses of the portfolio in sub-spaces of the problem space is reduced. We see similar potential performance increase of the portfolio on the training set and on the test set as the portfolio increases. Apparently, HYDRA is capable of finding general weaknesses of the incumbent portfolio and does not overfit by adding strategies that counter single negotiation instances.

The resulting four created strategies (Table 7.2) are time-dependent Boulware with low values for e making them competitive strategies. Between the four, θ_4 is the most cooperative strategy in the portfolio.

Due to the working of HYDRA the first strategy in the portfolio is the single best strategy, which is confirmed in Table 7.3 for both the test and the training set of instances. Having a single best strategy in the portfolio is useful for strategy selection when new opponents are encountered and no selection is possible due to a lack of opponent features. Additional strategies are found iteratively based on performance gain compared to the current portfolio. This behaviour is visible in Table 7.4, as the largest performance increase of the Oracle selector on the training set of instances is achieved by adding the second strategy θ_2.

Interestingly, the second strategy in the portfolio θ_2 achieves the lowest performance on both the test and train set of instances by itself (Table 7.3), but adds the most potential performance increase to the portfolio as a second strategy (Equation 7.3) besides the single best θ_1. Notable are the low ratios of best performing strategy for θ_2. Apparently, the single best strategy θ_1 has a weak spot in the problem space where it fails to achieve a decent performance. This weak spot is filled by θ_2.

The condition that every strategy must contribute to the portfolio by outperforming all other strategies on at least one negotiation instance is satisfied (Table 7.3). There is a single best strategy for 53.8% of the instances in the training set and for 50.3% in the test set. The best performing strategy ratios are similar for the test set and training set of instances.

8.4 Overall contribution

The overall contribution of this thesis is to reduce human effort in the design of automated negotiation agents and to provide a general approach to add strategy selection to automated negotiation agents.

Human effort shifts by replacing manual search methods with automated search methods for strategy configuration, while drastically increasing the size of the training set. It allows us to focus more on defining the boundaries of the problem space, e.g. the same Dynamic Agent can be configured for negotiation scenarios with a discount by adding those scenarios to the training set. In other words, we can find a single best strategy for the Dynamic Agent for every possible negotiation problem space by altering the training set accordingly.

We demonstrated that algorithm selection for automated negotiations is beneficial for agent performance, which is also found in literature [11], [18]. Still, the amount of agents that apply algorithm selection in the ANAC is surprisingly low. We believe that this is due to the difficulties it brings in the form of creating multiple strategies that complement each other and creating a successful method of strategy selection. In this thesis, we demonstrated a method to automate both, reducing human effort by limiting it to designing a single parameterised strategy and gathering a

representative training set of negotiation instances.

8.5 Future work

In this section we propose future work to expand upon this thesis. The future work is focussed on weaknesses that we spotted in our work during the experiments. We also propose steps towards removing the limitations we defined in the beginning of this work for simplification. Now that the initial is taken we can look towards expansion of the problem space.

Expanding problem space complexity

In the beginning of the thesis we set some boundaries in the negotiation problem space to lower the step towards our goal. Now that the initial step was successful, the problem space should be expanded to check whether our approach remains useful. For example, by adding discount factors and reservation utilities (Section 3.1.1) to the scenarios.

Extension to multilateral negotiation

We simplified in this thesis by limiting the work to bilateral negotiation settings. The current training and testing set of opponents and scenarios can also be used to create a training set of multilateral negotiation instances, but this increases the size of the set of instance set significantly, making automated configuration more difficult. It might be possible to configure the Dynamic Agent exclusively on bilateral settings and apply the resulting strategies in multilateral negotiations settings. There are three separate parts of the Dynamic Agent with parameters to configure as displayed in Table 6.1. We would like to investigate the performance of applying strategy selection in multilateral negotiation based on selecting a separate strategy for every opponent as if it would be a bilateral setup. As an initial suggestion, we propose the following steps to select and combine bilateral strategies:

1. Construct two algorithm selectors, one to select outcome space exploration configuration based on scenario features and one to select bidding strategy and accepting condition configuration based on opponent and scenario features.
2. Select a configuration for outcome space exploration based on the scenario
3. Select a configuration for bidding strategy and acceptance condition per opponent based on the opponent and scenario
4. Apply the acceptance conditions only to offers from the corresponding opponent
5. Use the bidding strategy to obtain a separate fitness value per opponent for an outcome and combine them via min operator

The handling of multiple opponents does add complexity to the problem. Failing to select a decent strategy for one of the opponents can result in failed negotiation for not only yourself, but also for

all the other parties. Another potential complexity lies in the opponent modelling, as opponents not only react to our offers but also to other opponent offers.

Opponent features

The opponent features are influenced by the strategy of the Dynamic Agent. To solve this, we restricted the gathering of opponent features to negotiation sessions where the Dynamic Agent uses the single best strategy, which reduces opportunities to learn opponent characteristics. More research is needed to study the influence of a strategy on the opponent features.

Strategy selection on first encounters

In a first encounter with an opponent we solve the incapability of strategy selection by selecting the single best strategy. Although it is a decent approach, it is a missed optimisation opportunity since we do know characteristics of the scenario. This is especially true if scenarios become more complex due to discount factors or reservation utilities. Future work should be focused on an approach of strategy selection at first encounters with opponents.

Partner selection

The automated configuration cost is linked directly to utility. Since we simplified in this thesis by assuming a reservation value of 0 for every scenario, the configurator ends up with a strategy that will desperately seek an agreement. After all, a small utility is more than 0 utility. We believe that opponents should not be punished of being very competitive by walking away from a small utility, refusing the opponent a good deal. Instead, opponents should be punished by our willingness to negotiate with them in the future, which is possible if it is possible to choose an opponent for a scenario (e.g. multiple sellers of the same product).

Implement multi-objective outcome space exploration

The Dynamic Agent uses a single-objective genetic algorithm to explore the outcome space. As we must optimise not only our own utility, but also the opponents utility, we added a heuristic to obtain a single objective value. However, there are genetic algorithms that are specifically build for multi-objective optimisation by focusing on finding Pareto solutions [51]. This seems ideal for outcome space exploration.

Chapter 9

Conclusion & Reflection

9.1 Conclusion

We now look back to the original problem statements as defined in Section 3.2 and answer the research questions in retrospect. We structure this section based on the separate problem definitions.

Dynamic Agent

We created a Dynamic Agent in Chapter 4 with a strategy that is based on a parameter configuration within a manually designed configuration space. We made the following statement as a requirement for a Dynamic Agent to be successful:

> We deem the Dynamic Agent successful if there is a configuration $\theta \in \Theta$ such that our agent $DA(\theta)$ outperforms the test set of opponent agents A_{test} on the test set of scenarios S_{test}. We define outperforming as achieving a higher average utility in a bilateral ANAC-like tournament setup.

In Chapter 6, the Dynamic Agent was automatically configured on the training set to create a single best strategy for the agent. We tested the configuration in an ANAC-like tournament setting as described above and showed that the resulting strategy was able to outperforming all other agents with a 5% performance increase compared to the number 2. We conclude that the Dynamic Agent is successful as its strategy space contains successful strategies.

Features

We constructed a set of features that describe a negotiation setting in Chapter 5, split up between opponent features and scenario features. These features are used to indicate differences between

negotiation settings for a potential faster convergence of the configuration procedure (SMAC) and as a problem description for selecting a good strategy.

To check the quality of the features, we wanted to test if the following two statements are true:

> 1. The feature set helps the automated algorithm configuration method in converging to a better configuration while maintaining the computational budget.
> 2. The feature set contains sufficient information, such that algorithm selection improves the utility of the Dynamic Agent.

Using SMAC in combination with instance features leads to less variation in parameter values between the final configurations of separate optimisation runs (Table 6.4, Table 6.5). We also see that the performance improvement is more significant and consistent for these configurations. One of the configurations that is optimised without features θ_2 leads to a worse performance on the test set of negotiation instances compared to the baseline. We conclude that the features have a notable positive impact on the optimisation process.

In Table 7.5 we showed that strategy selection based on the features increases the utility of the Dynamic Agent. We conclude that the features provide sufficient information for successful strategy selection.

Opponent preference estimation

Accurately modelling the preference profile of an opponent is important, as it allows us to observe the opponents behaviour and to find Pareto efficient outcomes. We measured the performance of a preference estimation model by calculating the Pearson correlation of bids and the difference in Pareto frontier surface. Both measures correlate highly with the negotiation performance, as is showed in opponent model comparison literature. We took an existing state-of-the-art preference estimation model (SFM) that is know to perform well and defined the following research question:

> Can we improve the Smith Frequency Model [46] (SFM) based on the Pearson correlation of bids and the difference in Pareto frontier surface measures?

We tested the SFM method of preference estimation and made observations about its shortcomings. We proposed two modifications in estimated utility calculation and applied it in our negotiation agent. In Table 4.5 we showed the results of the accuracy measures after 11200 negotiation sessions. Both modifications improved the accuracy of the opponent model for both measures. The combined improvement is 6.55% for the Pearson correlation measure and 3.91% for the Pareto surface measure. We conclude that the modifications are beneficial for preference estimation.

Configuration

The Dynamic Agent we created does not have a fixed strategy, but a configuration space that maps to a strategy. We set two baseline strategies by defining two parameter configurations. The

first configuration θ_{lit} is based on the original paper that we derived the agent from. The second configuration θ_{hand} is set by hand based on intuition, modern literature and manual search, which we considered the default approach if no automated configuration method is applied. We defined the following research question:

> Can we automatically tune a parameter configuration θ_{opt} that outperforms the baselines on average on a never before seen test set of negotiation instances?

In Chapter 6 we automatically configured the Dynamic Agent and obtained an optimised configuration using SMAC. We tested the performance of both baseline configurations and the optimised configuration on the test set of negotiation instances and presented the results in Table 6.6.

The configuration based on the original paper θ_{lit}, performed poorly compared to the hand tuned configuration θ_{hand} by achieving a 26.1% lower utility. The optimised configuration θ_{opt} outperformed both baseline configuration, obtaining a 4.2% increase in utility compared to the hand tuned configuration. We conclude that the automated configuration method is successful in finding strategies that outperform manual configuration.

Portfolio creation

We created the Dynamic Agent $DA(\theta)$ with a configuration space Θ and found a single best strategy within that configuration space in Chapter 6. We also stated that no single strategy is optimal for all negotiation instances, so we neded multiple strategies that complement each other. To avoid the need of constructing a portfolio of strategies manually, we automated this process by using HYDRA to find multiple successful strategies within the configuration space of the Dynamic Agent and used them as portfolio $\vec{\Theta}$. As a benefit, this approach required us to only design a single negotiation agent with a flexible parameterised strategy. We set the following requirement to verify the quality of the portfolio:

> Assuming that we are capable of selecting the best strategy for every negotiation instance (oracle selector $\theta_\pi = OR(\vec{\Theta}, \pi)$), $DA(OR(\vec{\Theta}, \pi))$ must outperform $DA(\theta_i)$ on average over the testing set of negotiation instances Π_{test}, for all $\theta_i \in \vec{\Theta}$.

We created a portfolio of 4 strategies and tested the performance of every strategy on the testing set of negotiation instances. In Table 7.4 and Table 7.3 we showed that, at perfect selection, the Dynamic Agent achieves a utility of 0.870 compared to a utility of 0.793 for the single best strategy. We conclude that the requirement is satisfied.

Selection

In the final part of the thesis we used the portfolio of strategies $\vec{\Theta}$ and added a layer of algorithm selection to the Dynamic Agent that selects a strategy based on the negotiation instance to exploit differences between instances. We defined the following research question:

Can we apply an algorithm selection method $AS(\vec{\Theta}, \pi) = \theta_\pi$ that selects a configuration θ_π from $\vec{\Theta}$ based on the negotiation instance π, such that $DA(AS(\vec{\Theta}, \pi))$ outperforms the single best strategy $DA(\theta_1)$ in an ANAC-like bilateral tournament?

To avoid having to choose the method of algorithm selection, we implemented AutoFolio that treats the construction of an algorithm selection pipeline as an algorithm configuration problem. We trained and fitted an algorithm selector on the training set of negotiation instances and implemented this selector in the Dynamic Agent, increasing the performance of the Dynamic Agent by $\frac{0.819-0.793}{0.793} * 100\% \approx 3.4\%$ compared to the Dynamic Agent with the single best strategy.

9.2 Reflection

In this section we reflect on the research of this thesis and point towards limitations that impact the value of the results presented.

Dynamic Agent

We aimed to create a flexible negotiation agent with a broad strategy space, set by a parameter configuration. However, we limited the strategy to a time-dependent tactic, which is the more popular tactic in the ANAC, undermining our goal. Other common tactics, like the behavioural dependent tactic, are not within the configuration space of the Dynamic Agent (Chapter 4) for reasons of simplifying our initial step towards automated configuration on a broad problem space.

The most famous behavioural tactic, tit-for-tat, performs less when optimising for utility, as it behaves as a Matcher instead of an Inverter [48]. Optimising for utility means exploiting conceding opponents, which is opposite behaviour of the tit-for-tat tactic. Since the Dynamic Agent already has a component of opponent behaviour adaptation (strategy selection), we chose to use time-dependent tactics in the Dynamic Agent.

Scenarios

We simplified the scenarios by setting the discount factor to 0. A discount factor of 0 makes competitive strategies more effective, as the time it takes to reach agreement is not penalised. It is likely that the single best strategy found in Chapter 6 would be more cooperative if the training set of scenarios contained scenarios with a large discount factor. The current single best strategy is probably ineffective in discounted scenarios, regardless of the opponent. This makes the decision to select the single best strategy if no opponent features are present disputable.

The scenarios do not contain inter-dependent issues, i.e. issues that are only considered when specific values are set to other issues. This allows us the usage of the linear additive utility function (Equation 3.2). Although these scenarios are most commonly used, we must note that

the opponent models (Section 4.3) and outcome space exploration method (Section 4.1.2) used by our Dynamic Agent are not suited to handle scenarios with inter-dependent issues.

Deadline

We set a time based deadline for the negotiation settings, following the ANAC competition. The idea behind this is that automated negotiation is commonly used in applications where agreements are needed within a real-time deadline. Examples are bidding on advertisement spots online while loading a web page, selling green energy to the grid, or regulating traffic at intersections. A round based deadline is an alternative, but that allows agents to use unlimited real-time, which is impractical in these applications. By setting a real-time deadline, agent designers are forced to focus on computational efficiency.

Real-time deadlines cause inconsistencies between tested performance on different hardware. More powerful hardware allows for more negotiation rounds, which in turn gives agents more time to explore/learn. In an attempt to resolve this issue, we limited ourselves to computation nodes with identical architecture, but we are not able to eliminate dynamic effects within those nodes. For example, slower memory due to memory clogging, CPU's going in saving mode due to lower computational node demand, and input output delays. These are all affecting the negotiation sessions, and thus influencing the performance.

Opponents

We focused on bilateral negotiation settings in this thesis. As we showed in Chapter 6 we are capable of performing very well against the opponents from the ANAC competition. However, these agents were made for multilateral negotiation settings with a deadline of 180 seconds, which is comparable but not identical. The Dynamic Agent in this thesis can also be used in multilateral negotiations with adaptations (Section 8.5), but its performance might differ.

Opponent model

We use the opponent model to extract opponent features for future use. The model is updated every time the opponent makes an offer. However, in this work we treated an accept as an offer made by the opponent and updated the model accordingly. There is something to say for this approach, as the opponent is willing to accept that outcome, but it is likely not identical to the offer it would have made otherwise. In a bilateral negotiation settings, there is only 1 accept at max, making this discussion less important. However, in multilateral negotiation, it is possible that there are many more accepts by opponents (Algorithm 1), making this an important discussion.

Performance metric

The performance metric in Equation 6.1 perfectly aligns with the goal of creating a well performing agent in general. However, it does not align with the goal to win the ANAC. In ANAC, it is not only important to obtain a high utility, but also to prevent opponents from achieving an even

higher utility. A very competitive agent will achieve a high performance in a competition with many conceders. In such a case, it is better to walk away from a negotiation with a competitive agent when you are not capable of winning the negotiation, resulting in 0 utility for both. The current performance metric results in a strategy that "desperately" tries to obtain an agreement, as little utility is always more than 0 utility.

Abbreviations

ANAC Automated Negotiating Agents Competition [12]. 4, 5, 8, 13, 16, 18–22, 30, 49, 53, 54, 65–71, 74, 77, 78, 86

AR Average Rate. 40, 41, 43, 93, 94

ASP Answer Set Programming. 8

BATNA Best Alternative To a Negotiation Agreement [34]. 11

CoV Coefficient of Variance. 39, 43, 66

CR Concession Rate. 39–41, 43, 90–92

CSP Constraint Satisfaction Problem. 8

DAP Default Algorithm Performance. 42, 43, 62, 66

FYU Full Yield Utility. 39, 40

GENIUS General Environment for Negotiation with Intelligent multi-purpose Usage Simulation [11]. 4, 9, 11, 13, 18, 20, 24, 49, 50, 58, 87, 88

MIP Mixed Integer Programming. 5, 7

SAOP Stacked Alternating Offers Protocol [37]. 13, 14

SAT Boolean Satisfiability. 5, 8

SFM Smith Frequency Model [46]. 30–37, 75

SLURM Simple Linux Utility for Resource Management [49]. 51

SMAC Sequential Model-based optimization for general Algorithm Configuration [22]. 45–47, 50, 51, 53, 54, 56, 61, 62, 68, 69, 75, 76

SMBO Sequential Model-Based Optimisation [22]. 46, 47

Nomenclature

AutoFolio an automatically configured algorithm selector [50]. 56, 57, 62, 70, 77

HYDRA a method for automatically designing algorithms to complement a portfolio [21]. 5, 58, 60–64, 66, 70, 71, 76

Bibliography

[1] H. Raiffa, *The art and science of negotiation*. Harvard University Press, 1982.

[2] J. Z. Rubin, *Negotiation behavior*, 13. Academic Press, 1983, vol. 17, pp. 911–912, ISBN: 9780125662505. DOI: 10.1016/0277-9536(83)90290-3.

[3] R. G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1104–1113, 1980, ISSN: 00189340. DOI: 10.1109/TC.1980.1675516.

[4] K. Sycara, "Resolving Goal Conflicts via Negotiation," *The Seventh National Conference on Artificial Intelligence*, pp. 245–249, 1988.

[5] K. Sycara-Cyranski, "Arguments Of Persuasion In Labour Mediation," *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 1, pp. 294–296, 1985.

[6] W. Robinson, "Negotiation behavior during requirement specification," *[1990] Proceedings. 12th International Conference on Software Engineering*, pp. 268–276, 1990, ISSN: 02705257. DOI: 10.1109/ICSE.1990.63633.

[7] J. S. Rosenschein, "Rational interaction: cooperation among intelligent agents," PhD thesis, Stanford, CA, USA, 1986, p. 145.

[8] M. Klein and S. C. Lu, "Conflict resolution in cooperative design," *Artificial Intelligence in Engineering*, vol. 4, no. 4, pp. 168–180, 1989, ISSN: 09541810. DOI: 10.1016/0954-1810(89)90013-7.

[9] T. Bosse and C. M. Jonker, "Human vs. computer behaviour in multi-issue negotiation," *Proceedings - First International Workshop on Rational, Robust, and Secure Negotiation Mechanisms in Multi-Agent Systems, RRS 2005*, vol. 2005, pp. 11–24, 2005, ISSN: 18387640. DOI: 10.1109/RRS.2005.8.

[10] T. Baarslag, K. Fujita, E. H. Gerding, K. Hindriks, T. Ito, N. R. Jennings, C. Jonker, S. Kraus, R. Lin, V. Robu, and C. R. Williams, "Evaluating practical negotiating agents: Results and analysis of the 2011 international competition," *Artificial Intelligence*, vol. 198, pp. 73–103, 2013, ISSN: 00043702. DOI: 10.1016/j.artint.2012.09.004.

[11] R. Lin, S. Kraus, T. Baarslag, D. Tykhonov, K. Hindriks, and C. M. Jonker, "Genius: An integrated environment for supporting the design of generic automated negotiators," *Computational Intelligence*, vol. 30, no. 1, pp. 48–70, 2014, ISSN: 08247935. DOI: 10.1111/j.1467-8640.2012.00463.x.

[12] T. Baarslag, K. Hindriks, C. Jonker, S. Kraus, and R. Lin, "The first automated negotiating agents competition (ANAC 2010)," *Studies in Computational Intelligence*, vol. 383, no. Anac, pp. 113–135, 2012, ISSN: 1860949X. DOI: 10.1007/978-3-642-24696-8_7.

[13] T. Baarslag, R. Aydoğan, K. V. Hindriks, K. Fujita, T. Ito, and C. M. Jonker, "The Automated Negotiating Agents Competition, 2010–2015," *AI Magazine*, vol. 36, no. 4, pp. 2010–2014, 2015, ISSN: 0738-4602. DOI: 10.1609/aimag.v36i4.2609.

[14] N. Matos, C. Sierra, and N. R. Jennings, "Determining successful negotiation strategies: An evolutionary approach," *Proceedings - International Conference on Multi Agent Systems, ICMAS 1998*, pp. 182–189, 1998, ISSN: 0254-1319. DOI: 10.1109/ICMAS.1998.699048.

[15] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992, p. 232, ISBN: 9780262082136.

[16] T. Eymann, "Co-evolution of bargaining strategies in a decentralized multi-agent system," *AAAI Fall 2001 Symposium on Negotiation Methods for Autonomous Cooperative Systems*, pp. 126–134, 2001.

[17] G. Dworman, S. O. Kimbrough, and J. D. Laing, "Bargaining by artificial agents in two coalition games: A study in genetic programming for electronic commerce," *Proceedings of the First Annual Conference on Genetic Programming*, pp. 54–62, 1996.

[18] L. Ilany and Y. Gal, "Algorithm selection in bilateral negotiation," *Autonomous Agents and Multi-Agent Systems*, vol. 30, no. 4, pp. 697–723, 2016, ISSN: 15737454. DOI: 10.1007/s10458-015-9302-8.

[19] J. R. Rice, "The Algorithm Selection Problem," *Advances in Computers*, vol. 15, no. C, pp. 65–118, 1976, ISSN: 00652458. DOI: 10.1016/S0065-2458(08)60520-3.

[20] L. Ilany and Y. Gal, "The Simple-Meta Agent," in *Novel insights in agent-based complex automated negotiation*, I. Marsa-Maestre, M. A. Lopez-Carmona, T. Ito, M. Zhang, Q. Bai, and K. Fujita, Eds., vol. 535, Springer, 2014, pp. 197–200, ISBN: 978-4-431-54757-0. DOI: 10.1007/978-4-431-54758-7.

[21] L. Xu, H. Hoos, and K. Leyton-Brown, "Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection.," *Aaai 2010*, vol. 10, pp. 210–216, 2010.

[22] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6683 LNCS, pp. 507–523, 2011, ISSN: 03029743. DOI: 10.1007/978-3-642-25566-3_40.

[23] ——, "Automated configuration of mixed integer programming solvers," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6140 LNCS, pp. 186–202, 2010, ISSN: 03029743. DOI: 10.1007/978-3-642-13520-0_23.

[24] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-Race and Iterated F-Race: An Overview," in *Experimental Methods for the Analysis of Optimization Algorithms*, T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, Eds., Springer Berlin Heidelberg, 2010, pp. 311–336, ISBN: 978-3-642-02538-9. DOI: 10.1007/978-3-642-02538-9_13.

[25] P. Balaprakash, M. Birattari, and T. St, "Improvement Strategies for the F-Race Algorithm :" *Strategies*, pp. 108–122, 2007.

[26] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo, "Model-based search for combinatorial optimization: A critical survey," *Annals of Operations Research*, vol. 131, no. 1-4, pp. 373–395, 2004, ISSN: 02545330. DOI: 10.1023/B:ANOR.0000039526.52305.af.

[27] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, 2009, ISSN: 10769757. DOI: 10.1613/jair.2861.

[28] C. Ansótegui, M. Sellmann, and K. Tierney, "A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms," in *Principles and Practice of Constraint Programming - CP 2009*, I. P. Gent, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 142–157, ISBN: 978-3-642-04244-7.

[29] R. Y. Lau, M. Tang, O. Wong, S. W. Milliner, and Y. P. P. Chen, "An evolutionary learning approach for adaptive negotiation agents," *International Journal of Intelligent Systems*, vol. 21, no. 1, pp. 41–72, 2006, ISSN: 08848173. DOI: 10.1002/int.20120.

[30] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based Algorithm Selection for {SAT}," *J. Artif. Intell. Res.*, vol. 32, pp. 565–606, 2008, ISSN: 1076-9757. DOI: 10.1613/jair.2490.

[31] H. Hoos, M. Lindauer, and T. Schaub, "Claspfolio 2: Advances in algorithm selection for answer set programming," *Theory and Practice of Logic Programming*, vol. 14, no. 4-5, pp. 569–585, 2014, ISSN: 14753081. DOI: 10.1017/S1471068414000210.

[32] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan, "Using case-based reasoning in an algorithm portfolio for constraint solving," *Irish Conference on Artificial Intelligence and Cognitive Science*, no. 05, pp. 210–216, 2008.

[33] T. Baarslag, *What to bid and when to stop*, september. 2014, ISBN: 9789461863058. DOI: 10.4233/uuid:3df6e234-a7c1-4dbe-9eb9-baadabc04bca.

[34] R. Fisher and W. L. Ury, *Getting to Yes: Negotiating Agreement Without Giving In*. Penguin Group, 1981, p. 200, ISBN: 9780395317570.

[35] J. F. Nash, "The Bargaining Problem," *Econometrica*, vol. 18, no. 2, p. 155, 1950, ISSN: 00129682. DOI: 10.2307/1907266.

[36] E. Kalai and M. Smorodinsky, "Other Solutions to Nash's Bargaining Problem," *Econometrica*, vol. 43, no. 3, p. 513, 1975, ISSN: 00129682. DOI: 10.2307/1914280.

[37] R. Aydoğan, D. Festen, K. V. Hindriks, and C. M. Jonker, "Alternating offers protocols for multilateral negotiation," in *Studies in Computational Intelligence*, vol. 674, Springer, 2017, pp. 153–167, ISBN: 978-3-319-51563-2. DOI: 10.1007/978-3-319-51563-2_10.

[38] T. Baarslag, A. Dirkzwager, K. V. Hindriks, and C. M. Jonker, "The significance of bidding, accepting and opponent modeling in automated negotiation," *Frontiers in Artificial Intelligence and Applications*, vol. 263, pp. 27–32, 2014, ISSN: 09226389. DOI: 10.3233/978-1-61499-419-0-27.

[39] P. Faratin, C. Sierra, and N. R. Jennings, "Negotiation decision functions for autonomous agents," *Robotics and Autonomous Systems*, vol. 24, no. 3-4, pp. 159–182, 1998.

[40] R. J. Lewicky, D. M. Saunders, and B. Barry, *Essentials of Negotiation*. McGraw-Hill/Irwin Boston, MA, 2011, p. 289.

[41] D. K. Gode and S. Sunder, "Allocative Efficiency of Markets with Zero-Intelligence Traders: Market as a Partial Substitute for Individual Rationality," *Journal of Political Economy*, vol. 101, no. 1, pp. 119–137, 1993, ISSN: 0022-3808. DOI: 10.1086/261868.

[42] T. Baarslag, K. Hindriks, and C. Jonker, "Acceptance conditions in automated negotiation," in *Complex Automated Negotiations: Theories, Models, and Software Competitions*, Springer, 2013, pp. 95–111.

[43] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys*, vol. 41, no. 1, pp. 1–25, 2008, ISSN: 03600300. DOI: 10.1145/1456650.1456656.

[44] T. Baarslag, M. Hendrikx, K. Hindriks, and C. Jonker, "Predicting the performance of opponent models in automated negotiation," in *Proceedings - 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2013*, vol. 2, IEEE, Nov. 2013, pp. 59–66, ISBN: 9781479929023. DOI: 10.1109/WI-IAT.2013.91.

[45] T. Baarslag, K. Hindriks, and C. Jonker, "Effective acceptance conditions in real-time automated negotiation," *Decision Support Systems*, vol. 60, no. 1, pp. 68–77, 2014, ISSN: 01679236. DOI: 10.1016/j.dss.2013.05.021.

[46] N. Van Galen Last, "Agent Smith: Opponent model estimation in bilateral multi-issue negotiation," *Studies in Computational Intelligence*, vol. 383, pp. 167–174, 2012, ISSN: 1860949X. DOI: 10.1007/978-3-642-24696-8_12.

[47] T. Baarslag, M. Hendrikx, K. Hindriks, and C. Jonker, "Measuring the performance of online opponent models in automated bilateral negotiation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, M. Thielscher and D. Zhang, Eds., ser. Lecture Notes in Computer Science, vol. 7691 LNAI, Springer Berlin Heidelberg, 2012, pp. 1–14, ISBN: 9783642351006. DOI: 10.1007/978-3-642-35101-3_1.

[48] T. Baarslag, K. Hindriks, and C. Jonker, "Towards a quantitative concession-based classification method of negotiation strategies," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7047 LNAI, pp. 143–158, 2011, ISSN: 03029743. DOI: 10.1007/978-3-642-25044-6_13.

[49] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," pp. 44–60, 2006. DOI: 10.1007/10968987_3.

[50] M. Lindauer, F. Hutter, H. H. Hoos, and T. Schaub, "AutoFolio: An automatically configured algorithm selector," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 53, pp. 5025–5029, 2017, ISSN: 10450823. DOI: 10.1613/jair.4726.

[51] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002, ISSN: 1089778X. DOI: 10.1109/4235.996017.

Appendix A

Training and testing set

This appendix provides an overview of the training and testing set of both agents and scenarios that is used throughout this thesis. A single training instance requires an agent as opponent and scenario from the train set, the same is true for a test instance.

The set of agents is provided in Table A.1. We used a total of 31 agents from the ANAC, which were all finalists in either the nash category or the individual utility category. The set of ANAC agents is split up in 20 training agents and 11 test agents. We added an additional 5 baseline tactic agents as described in Section 3.1.3 to the test set for analysis. The set of scenarios is provided in Table A.2. A total of 42 scenarios is used of which both sides can be played by our agent resulting in 84 playable scenarios. The set of scenarios is selected based on diversity using the features as described in Section 5.1 and their discount factor and reservation utility are removed. The set is split up in 56 training scenarios and 28 test scenarios.

The total amount of training instances:

$$|\Pi_{train}| = |A_{train}| * |S_{train}| = 20 * 56 = 1120 \tag{A.1}$$

The total amount of test instances:

$$|\Pi_{test}| = |A_{test}| * |S_{test}| = 16 * 28 = 448 \tag{A.2}$$

Train/Test	Agent	ANAC	Finalist Type
train	Rubick	2017	individual/nash
train	PonPokoAgent	2017	individual
train	CaduceusDC16	2017	individual
train	AgentKN	2017	individual/nash
train	ParsCat	2016	individual
train	YXAgent	2016	individual
train	Terra	2016	individual
train	MyAgent	2016	individual
train	GrandmaAgent	2016	individual
train	Farma	2016	individual
train	Atlas32016	2016	individual
train	AgentHP2_main	2016	individual
train	PokerFace	2015	individual
train	ParsAgent	2015	individual
train	kawaii	2015	individual
train	Atlas3	2015	individual/nash
train	AgentX	2015	nash
train	AgentBuyogMain	2015	individual/nash
train	Gangster	2014	nash
train	DoNA	2014	individual
test	SimpleAgent	2017	individual
test	ParsCat2	2017	nash
test	ShahAgent	2017	individual/nash
test	Mosa	2017	nash
test	Mamenchis	2017	individual/nash
test	GeneKing	2017	nash
test	AgentF	2017	individual
test	Caduceus	2016	individual
test	RandomDance	2015	individual/nash
test	PhoenixParty	2015	individual
test	AgentH	2015	nash
test	RandomCounterOfferParty	N/A	N/A
test	BoulwareParty	N/A	N/A
test	ConcederParty	N/A	N/A
test	HardlinerParty	N/A	N/A
test	LinearConcederParty	N/A	N/A

Table A.1: Overview of agent set used in GENIUS

Train/Test	Profile 1	Profile 2	Comment
train	ItexvsCypress_Cypress.xml	ItexvsCypress_Itex.xml	x2 (both sides are played)
train	laptop_buyer_utility.xml	laptop_seller_utility.xml	x2 (both sides are played)
train	Grocery_domain_mary.xml	Grocery_domain_sam.xml	x2 (both sides are played)
train	Amsterdam_party1.xml	Amsterdam_party2.xml	x2 (both sides are played)
train	camera_buyer_utility.xml	camera_seller_utility.xml	x2 (both sides are played)
train	energy_consumer.xml	energy_distributor.xml	x2 (both sides are played)
train	EnergySmall-A-prof1.xml	EnergySmall-A-prof2.xml	x2 (both sides are played)
train	Barter-A-prof1.xml	Barter-A-prof2.xml	x2 (both sides are played)
train	FlightBooking-A-prof1.xml	FlightBooking-A-prof2.xml	x2 (both sides are played)
train	HouseKeeping-A-prof1.xml	HouseKeeping-A-prof2.xml	x2 (both sides are played)
train	MusicCollection-A-prof1.xml	MusicCollection-A-prof2.xml	x2 (both sides are played)
train	Outfit-A-prof1.xml	Outfit-A-prof2.xml	x2 (both sides are played)
train	RentalHouse-A-prof1.xml	RentalHouse-A-prof2.xml	x2 (both sides are played)
train	Supermarket-A-prof1.xml	Supermarket-A-prof2.xml	x2 (both sides are played)
train	Animal_util1.xml	Animal_util2.xml	x2 (both sides are played)
train	DogChoosing_util1.xml	DogChoosing_util2.xml	x2 (both sides are played)
train	Icecream_util1.xml	Icecream_util2.xml	x2 (both sides are played)
train	Lunch_util1.xml	Lunch_util2.xml	x2 (both sides are played)
train	Ultimatum_util1.xml	Ultimatum_util2.xml	x2 (both sides are played)
train	DefensiveCharms_util1.xml	DefensiveCharms_util2.xml	x2 (both sides are played)
train	SmartEnergyGrid_util1.xml	SmartEnergyGrid_util2.xml	x2 (both sides are played)
train	DomainAce_util1.xml	DomainAce_util2.xml	x2 (both sides are played)
train	Smart_Grid_util1.xml	Smart_Grid_util2.xml	x2 (both sides are played)
train	DomainTwF_util1.xml	DomainTwF_util2.xml	x2 (both sides are played)
train	ElectricVehicle_profile1.xml	ElectricVehicle_profile2.xml	x2 (both sides are played)
train	PEnergy_util1.xml	PEnergy_util2.xml	x2 (both sides are played)
train	JapanTrip_util1.xml	JapanTrip_util2.xml	x2 (both sides are played)
train	NewDomain_util1.xml	NewDomain_util2.xml	x2 (both sides are played)
test	England.xml	Zimbabwe.xml	x2 (both sides are played)
test	travel_chox.xml	travel_fanny.xml	x2 (both sides are played)
test	IS_BT_Acquisition_BT_prof.xml	IS_BT_Acquisition_IS_prof.xml	x2 (both sides are played)
test	AirportSiteSelection-A-prof1.xml	AirportSiteSelection-A-prof2.xml	x2 (both sides are played)
test	Barbecue-A-prof1.xml	Barbecue-A-prof2.xml	x2 (both sides are played)
test	EnergySmall-A-prof1.xml	EnergySmall-A-prof2.xml	x2 (both sides are played)
test	FiftyFifty-A-prof1.xml	FiftyFifty-A-prof2.xml	x2 (both sides are played)
test	Coffee_util1.xml	Coffee_util2.xml	x2 (both sides are played)
test	Kitchen-husband.xml	Kitchen-wife.xml	x2 (both sides are played)
test	Wholesaler-prof1.xml	Wholesaler-prof2.xml	x2 (both sides are played)
test	triangularFight_util1.xml	triangularFight_util2.xml	x2 (both sides are played)
test	SmartGridDomain_util1.xml	SmartGridDomain_util2.xml	x2 (both sides are played)
test	WindFarm_util1.xml	WindFarm_util2.xml	x2 (both sides are played)
test	KDomain_util1.xml	KDomain_util2.xml	x2 (both sides are played)

Table A.2: Overview of scenario set used in GENIUS

Appendix B

Genetic algorithm procedures

Algorithm 6 Crossover

Input	Ind_a	Individual, where $Ind = \langle fitness, \omega \rangle$
	Ind_b	Individual, where $Ind = \langle fitness, \omega \rangle$
	R_c	Crossover rate
Variables	v	Value in outcome ω
Output	Ind	Individual, where $Ind = \langle fitness, \omega \rangle$

1: $\omega_a \leftarrow GetOutcome(Ind_a)$
2: $\omega_b \leftarrow GetOutcome(Ind_b)$
3: **for** v **in** ω_b **do**
4: **if** $Random() < R_c$ **then**
5: $\omega_a \leftarrow ReplaceValue(\omega_a, v)$
6: **return** $\langle null, \omega_a \rangle$

Algorithm 7 Mutate

Input	Ind	Individual, where $Ind = \langle fitness, \omega \rangle$
	R_m	Mutation rate
Output	Ind	Individual, where $Ind = \langle fitness, \omega \rangle$

1: $\omega \leftarrow GetOutcome(Ind)$
2: **for** $i := 1, \ldots, Length(\omega)$ **do**
3: **if** $Random() < R_m$ **then**
4: $\omega[i] \leftarrow GetRandomValue(\omega[i])$
5: **return** $\langle null, \omega \rangle$

Appendix C

Predicted concession rate

The Concession Rate (CR), as described by Baarslag et al. [48], is calculated under perfect information, meaning that the opponents utility function $u_o(\omega)$ is available. We do not have the luxury of knowing the opponents preferences, so we must revert to the estimated utility function $\hat{u}_o(\omega)$. As a result, the minimum demanded utility by the opponent is an estimation as well, so the prediction of the CR relies heavily on the quality of the opponent model.

We propose two alternative methods of predicting the CR that rely less on the quality of the opponent preference estimation method. Instead, they rely more on the capability of the opponent to offer Pareto efficient outcomes and on the shape of the outcome utility space. We present all three methods of CR prediction in Equation C.1 and their visualisations in Figure C.1. We also present a brief textual description of the three methods below:

- $\hat{CR}_1$ does not use an estimated opponent utility function. In fact, the only uncertainty is the estimation of the opponents best outcome $\hat{\omega}_o^+$. It approaches the true CR if the Pareto frontier is a straight line, if we are capable of accurately predicting $\hat{\omega}_o^+$, and if the opponent is capable of offering Pareto efficient outcomes.
- $\hat{CR}_2$ uses an estimated opponent utility function, but does not have to predict special outcomes or offers of the opponent. The best offered outcome by the opponent x^+, and our best outcome ω^+ are fully known, since they are measured in our utility. It approaches the true CR if our estimated opponent utility function is accurate and if the opponent is capable of offering Pareto efficient outcomes.
- $\hat{CR}_3$ is the original version of the CR, but without perfect information. It uses an estimated opponent utility function and has to estimate the opponents minimum demanded utility $\hat{x}_o^-$. It approaches the true CR if our estimated opponent utility function is accurate, as well as our prediction of $\hat{x}_o^-$. It does not rely on the capability of the opponent to offer Pareto efficient outcomes.

$$
\begin{aligned}
\hat{CR}_1(x^+) &= \begin{cases} 0 & \text{if } u(x^+) \leq u(\hat{\omega}_o^+), \\ \frac{u(x^+)-u(\hat{\omega}_o^+)}{1-u(\hat{\omega}_o^+)} & \text{otherwise.} \end{cases} \\
\hat{CR}_2(x^+) &= \begin{cases} 1 & \text{if } \hat{u}_o(x^+) \leq \hat{u}_o(\omega^+), \\ \frac{1-\hat{u}_o(x^+)}{1-\hat{u}_o(\omega^+)} & \text{otherwise.} \end{cases} \\
\hat{CR}_3(\hat{x}_o^-) &= \begin{cases} 1 & \text{if } \hat{u}_o(\hat{x}_o^-) \leq \hat{u}_o(\omega^+), \\ \frac{1-\hat{u}_o(\hat{x}_o^-)}{1-\hat{u}_o(\omega^+)} & \text{otherwise.} \end{cases}
\end{aligned}
\tag{C.1}
$$

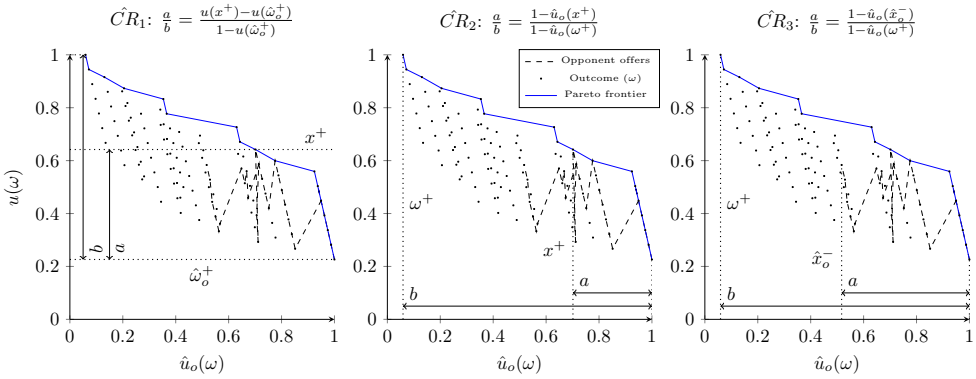


Figure C.1: Visualisation of predicted Concession Rate (CR)

C.1 Results

We want to use the CR as an informational feature of the opponent. We compared the three methods of CR prediction with the true CR on the basis of maintaining informational value. We, thus, use comparison via Pearson correlation coefficient instead of absolute or squared error. We ran a total of 11200 negotiation sessions on the training set as described in Appendix A using the single best strategy for our Dynamic Agent as described in Chapter 7, while measuring the true and predicted CR. The results showed that $\hat{CR}_3$ best approaches the true CR (Figure C.2), so we use this approximation as CR feature.

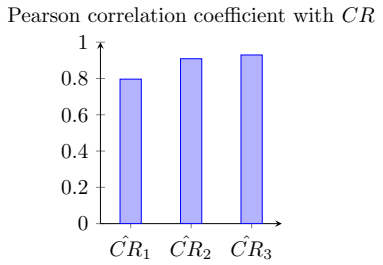


Figure C.2: Comparison predicted and true Concession Rate (CR)

Appendix D

Predicted average rate

The Average Rate (AR) is calculated under perfect information, meaning that the opponents utility function $u_o(\omega)$ is available. We do not have the luxury of knowing the opponents preferences, so we must revert to the estimated utility function $\hat{u}_o(\omega)$.

We present two methods of predicting the AR in Equation D.1 and their visualisations in Figure D.1. We also present a brief textual description of the two methods below:

- $\hat{AR}_1$ does not use an estimated opponent utility function. In fact, the only uncertainty is the estimation of the opponents best outcome $\hat{\omega}_o^+$. It approaches the true AR if the Pareto frontier is a straight line, if we are capable of accurately predicting $\hat{\omega}_o^+$, and if the opponent is capable of offering Pareto efficient outcomes.
- $\hat{AR}_2$ is the original version of the AR, using an estimated opponent utility function. It approaches the true AR if our estimated opponent utility function is accurate.

$$\begin{aligned} \hat{AR}_1(\bar{x}) &= \begin{cases} 0 & \text{if } u(\bar{x}) \leq u(\hat{\omega}_o^+), \\ \frac{u(\bar{x})-u(\hat{\omega}_o^+)}{1-u(\hat{\omega}_o^+)} & \text{otherwise.} \end{cases} \\ \hat{AR}_2(\hat{\bar{x}}) &= \begin{cases} 1 & \text{if } \hat{u}_o(\hat{\bar{x}}) \leq \hat{u}_o(\omega^+), \\ \frac{1-\hat{u}_o(\hat{\bar{x}})}{1-\hat{u}_o(\omega^+)} & \text{otherwise.} \end{cases} \end{aligned} \tag{D.1}$$

D.1 Results

As in Appendix C, we compared the true AR with the predicted versions using the Pearson correlation coefficient. We ran a total of 11200 negotiation sessions on the training set as described in Appendix A using the single best strategy for our Dynamic Agent as described in Chapter 7, while measuring the true and predicted AR. The results showed that $\hat{AR}_2$ best approaches the true AR (Figure D.2), so we use this approximation as AR feature.

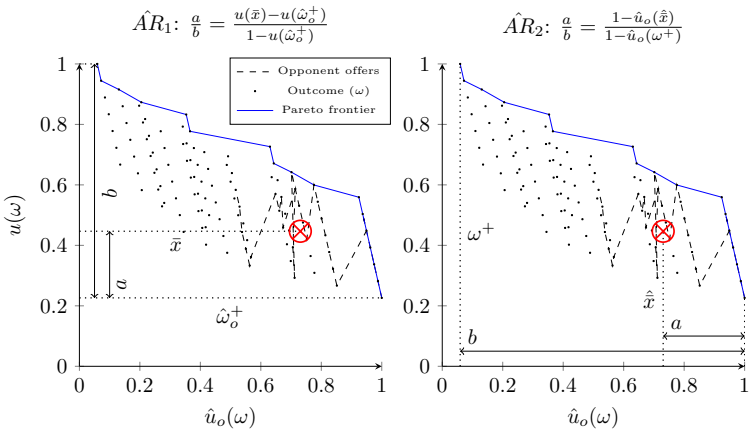


Figure D.1: Visualisation of predicted Average Rate (AR)

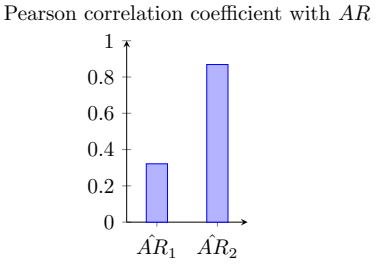


Figure D.2: Comparison predicted and true Average Rate (AR)