Learning in automated negotiation

Renting, B.M.

I

Learning to Negotiate

3

AUTOMATED CONFIGURATION OF NEGOTIATION STRATEGIES

This chapter has been published at AAMAS 2020 [124].

3.1 INTRODUCTION

In this chapter, we use algorithm configuration techniques to configure negotiating agents on large and diverse sets of negotiation scenarios and opponent types. We recreate a negotiation agent from literature [92] that is configured manually, combine it with contemporary opponent learning techniques and create a configuration space of its strategic behaviour. To automatically configure this conceptually rich and highly parametric design, we use Sequential Model-based optimization for general Algorithm Configuration (SMAC), a general-purpose automated algorithm configuration procedure that has been used previously to optimize the performance of cutting-edge solvers for Boolean Satisfiability (SAT), Mixed Integer Programming (MIP) and other NP-hard problems. We note that here, we apply automated algorithm configuration for the first time to a multi-agent problem.

Earlier attempts to solve the automated configuration problem in automated negotiation mostly used basic approaches, such as random and grid search. More advanced methods, in the form of genetic algorithms, have also been attempted. Matos et al. [104] encoded a mix of baseline tactics as a chromosome and deployed a genetic algorithm to find the best mix. They assumed perfect knowledge of the opponent's preferences, and their strategy is only tested against itself in a single negotiation scenario. Eymann [46] encoded a more complex strategy as a chromosome with six parameters, again only testing its performance against itself and using the same scenario. Dworman et al. [43] implement the genetic algorithm in a coalition game with three 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 in a coalition game with varying coalition values. Lau et al. [92] use a genetic algorithm to explore the outcome space during a negotiation session but do not use it to change the strategy.

This work aims to automatically configure a negotiation algorithm with no fixed or pre-defined strategy. This agent can be configured to perform well on a user-defined set of training problem instances, with little restrictions on the size of the instances or instance sets. To demonstrate its performance, we configure the agent in an attempt to win an Automated Negotiating Agents Competition (ANAC)-like bilateral tournament [10].

We show that we can win such a tournament with a comfortable margin of 5.1% in increased negotiation payoff compared to the number two. These margins are not observed in a tournament without our negotiation agent, where the winning strategy obtains a marginal improvement in negotiation payoff of 0.012%.

3.2 PROBLEM DESCRIPTION

In this section, we discuss the problem that we attempt to solve. We first describe the parameterised agent that we will configure (Section 3.2.1) and then give a formal problem definition using this parameterised agent (Section 3.2.2). For background on automated negotiation and algorithm configuration, we refer the reader to Section 2.1 and Section 2.2, respectively.

3.2.1 DYNAMIC AGENT

We first create a Dynamic Agent with a flexible strategy equivalent to a configuration space. We implement a few popular components and add their design choices to the configuration space, increasing the chances that it contains a successful strategy. We refer to this configuration space (or strategy space) with Θ. We name the constructed agent Dynamic Agent $DA(\theta)$, with strategy $\theta \in \Theta$.

The dynamic agent is constructed on the basis of the BOA-architecture [8]. We use this structure to give a brief overview of the workings of the dynamic agent and its configuration space.

BIDDING STRATEGY

The implemented bidding strategy applies a fitness value to the outcome space Ω and selects the outcome with the highest fitness as the offer, which is an approach used by Lau et al. [92]. This fitness function $f(\boldsymbol{\omega}, t)$ balances between our utility, the opponent's utility and the remaining time towards the deadline. Such a tactic is also known as a time-dependent tactic, and it generally concedes to the opponent as time passes.

The fitness function in Equation 3.1 has three parameters:

- A trade-off factor δ that balances between the importance of our own utility and the importance of reaching an agreement.
- A factor to control an agent's eagerness to concede e relative to time, where the behaviour is Boulware if $0 < e < 1$, linear conceder if $e = 1$, and conceder if $e > 1$.
- A categorical parameter n that sets the outcome where the fitness function concedes towards over time (Equation 3.2). Here, x^{last} is the last offer made by the opponent, and x^+ is the best offer the opponent made in terms of our utility.

$$\begin{aligned} f(\boldsymbol{\omega}, t) &= F(t) * u(\boldsymbol{\omega}) + (1 - F(t)) * f_n(\boldsymbol{\omega}) \\ F(t) &= \delta * (1 - t^{\frac{1}{e}}) \end{aligned} \tag{3.1}$$

$$\begin{aligned} f_1(\boldsymbol{\omega}) &= 1 - |\hat{u}_o(\boldsymbol{\omega}) - \hat{u}_o(x^{last})| \\ f_2(\boldsymbol{\omega}) &= \min(1 + \hat{u}_o(\boldsymbol{\omega}) - \hat{u}_o(x^{last}), 1) \\ f_3(\boldsymbol{\omega}) &= 1 - |\hat{u}_o(\boldsymbol{\omega}) - \hat{u}_o(x^+)| \\ f_4(\boldsymbol{\omega}) &= \min(1 + \hat{u}_o(\boldsymbol{\omega}) - \hat{u}_o(x^+), 1) \\ f_5(\boldsymbol{\omega}) &= \hat{u}_o(\boldsymbol{\omega}) \end{aligned} \tag{3.2}$$

Outcome space exploration The outcome space is potentially large. To reduce computational time and to ensure a fast response time for our agent, we apply a genetic algorithm to explore the outcome space in search of the best outcome. Standard procedures such as elitism, mutation and uniform crossover [109] are

Table 3.1: Configuration space in bidding strategy

Description	Symbol	Domain
Trade-off factor	δ	$[0,1]$
Conceding factor	e	$(0,2]$
Conceding goal	n	$\{1,2,3,4,5\}$
Population size	N_p	$[50,400]$
Tournament size	N_t	$[1,10]$
Evolutions	E	$[1,5]$
Crossover rate	R_c	$[0.1,0.5]$
Mutation rate	R_m	$[0,0.2]$
Elitism rate	R_e	$[0,0.2]$

Table 3.2: Configuration space in acceptance strategy

Description	Symbol	Domain
Scale factor	α	$[1,1.1]$
Utility gap	β	$(0,0.2]$
Accepting time	t_{acc}	$[0.9,1]$
Lower boundary utility	γ	$\{MAX^W, AVG^W\}$

applied, and the parameters of the genetic algorithm are added to the configuration space.

Configuration space The configuration space of the bidding strategy is summarized in Table 3.1.

OPPONENT MODEL

The Smith Frequency model [59] is used to estimate the opponent utility function $\hat{u}_o(\boldsymbol{\omega})$. According to an analysis by Baarslag et al. [11], the performance of this opponent modelling method is already quite close to that of the perfect model. No parameters are added to the configuration space of the Dynamic Agent.

ACCEPTANCE STRATEGY

The acceptance strategy decides when to accept an offer from the opponent. Baarslag et al. [14] performed an isolated and empirical research on popular acceptance conditions. They combined acceptance conditions and showed that a combined approach outperforms its parts. Baarslag et al. defined four parameters and performed a grid search in search of the best strategy. We adopt the combined approach and add its parameters (Table 3.2) to the configuration space of the Dynamic Agent. For more details on the combined acceptance condition, see Baarslag et al. [14].

3.2.2 PROBLEM DEFINITION

The negotiation agents in the General Environment for Negotiation with Intelligent multi-purpose Usage Simulation (GENIUS) environment [1] [99] are mostly based

[1] https://ii.tudelft.nl/genius/

on manually configured strategies by competitors in ANAC. These agents almost always contain parameters that are set by trial and error, despite the abundance of automated algorithm configuration techniques (e.g. Genetic Algorithm [71]). Manual configuration is a difficult and tedious job due to the dimensionality of both the configuration and the negotiation instance space.

A few attempts were made to automate this process as discussed in Section 3.1, but only on very specific negotiation settings with few configuration parameters. The main reason for this is that many automated configuration algorithms require to evaluate a challenging configuration on the full training set. To illustrate, evaluating the performance of a single configuration on the full training set that we use in this paper would take approximately 18.5 hours, regardless of the hardware due to the real-time deadline. These methods of algorithm configuration are, therefore, impractical.

Automated strategy configuration We have an agent called Dynamic Agent $DA(\theta)$, with strategy θ. We want to configure this agent such that it performs generally well using automated 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 $\mathcal{I}_{train}$ of size $|\mathcal{I}_{train}| = 20$ and scenarios S_{train} of size $|S_{train}| = 56$ that we use for training, making the total amount of training instances $|\mathcal{P}_{train}| = |\mathcal{I}_{train}| * |S_{train}| = 1120$. 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 literature θ_l and a setting that is hand tuned based on intuition, modern literature and manual tuning θ_m that we consider baselines. Can we automatically configure a strategy $\theta_{opt} \in \Theta$ that outperforms the baselines and wins an ANAC-like bilateral tournament on a never before seen test set of negotiation instances $\mathcal{P}_{test}$?

3.3 AUTOMATED CONFIGURATION

The goal of our work is to create an agent that can be configured to obtain a negotiation strategy that performs well in a given setting. This requires us to define what it means for a strategy to perform well. An obvious performance metric $m(\theta, p)$ is the utility obtained using strategy θ in negotiation instance p. As we are interested in optimizing performance on the full set of training instances rather than for a single instance, we define the performance of a configuration on an instance set as the average utility:

$$M(\theta, \mathcal{P}) = \frac{1}{|\mathcal{P}|} \cdot \sum_{p \in \mathcal{P}} m(\theta, p), \tag{3.3}$$

where:

m : utility obtained by using strategy θ in negotiation instance p
M : average utility of configuration θ on instance set $\mathcal{P}$
$\theta \in \Theta$: parameter configuration
p : single negotiation instance consisting of opponent agent $i \in \mathcal{I}$ and scenario $s \in S$, where $p = \langle a, s \rangle \in \mathcal{P}$
$\mathcal{P}$: set of negotiation instances

As stated in Section 3.2.2, automated configuration methods that require evaluation on the full training set of instances, thus requiring Equation 3.3 to be calculated, are impractical for our application. A second component that influences the amount of required evaluations is the mechanism that selects configurations for evaluation. This is not a straightforward problem, as the configuration space is large, and simple approaches, such as random search and grid search, suffer from the curse of dimensionality.

3.3.1 SMAC

To solve the problem defined in Section 3.2.2, we bring SMAC, a prominent, general-purpose algorithm configuration procedure [75], into the research area of automated negotiation. We note that SMAC is well suited for tackling the configuration problem arising in the context of our study:

1. It can handle different types of parameters, including real- and integer-valued as well as categorical parameters.
2. It can configure on subsets of the training instance set, reducing the computational expense.
3. It has a mechanism to terminate poorly performing configurations early, saving computation time. If it detects that a configuration is performing very poorly on a small set of instances (e.g., a very eager conceder), it stops evaluating and drops the configuration.
4. It models the relationship between parameter settings, negotiation instance features and performance, which tends to significantly reduce the effort of finding good configurations.
5. It permits parallelisation of the configuration process by means of multiple independent runs, which leads to significant reductions in wall-clock time.

SMAC keeps a run history (Equation 3.4), consisting of a configuration θ_i with its associated utility m_i on a negotiation instance that is modelled by a feature set $\mathcal{F}(p)$. A random forest regression model is fitted to this run history, mapping the configuration space and negotiation instance space to a performance estimate $\hat{m}$ (Equation 3.5). This model is then used to predict promising configurations, which are subsequently raced against the best configuration found so far, until an overall time budget is exhausted. We refer the reader to Hutter et al. [75] for further details on SMAC.

$$R = \{(\langle \theta_1, \mathcal{F}(p) \rangle, o_1), \ldots, (\langle \theta_n, \mathcal{F}(p) \rangle, o_n)\} \tag{3.4}$$

Table 3.3: Scenario features

Feature type	Description	Equation	Notes
Domain	Number of issues	$\|B\|$	
Domain	Average number of values per issue	$\frac{1}{\|B\|}\sum_{b\in B}\|\Omega_b\|$	
Domain	Number of possible outcomes	$\|\Omega\|$	
Preference	Standard deviation of issue weights	$\sqrt{\frac{1}{\|B\|}\sum_{b\in B}(w(b)-\frac{1}{\|B\|})^2}$	
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	$\sqrt{\frac{1}{\|\Omega\|}\sum_{\omega\in\Omega}(u(\omega)-u(\bar{\omega}))^2}$	

$$\mathcal{M} : (\Theta \times \mathcal{P}) \rightarrow \hat{m} \tag{3.5}$$

In order for SMAC to be successful in predicting promising configurations, it requires an accurate feature description of the negotiation instances that captures differences in complexity between these instances.

Automated algorithm configuration Suppose we have a set of opponent agents $\mathcal{I}$ and a set of negotiation scenarios S, such that combining a single agent $i \in \mathcal{I}$ and a single scenario $s \in S$ creates a new negotiation setting or instance $p \in \mathcal{P}$. Can we derive a set of features for both the opponent and the scenario that characterize the complexity of the negotiation instance?

We approach this question empirically by analyzing if a candidate feature set helps the automated algorithm configuration method find better configurations within the same computational budget.

3.4 INSTANCE FEATURES

The negotiation instances consist of an opponent and a scenario. We will extract features for both components separately and then combine them as a feature set of an instance (Equation 3.6). This feature description is used by the configuration method to predict promising strategies for our Dynamic Agent $DA(\theta)$.

$$\mathcal{F} : \mathcal{P} \rightarrow (X_{sc} \times X_{opp}) \tag{3.6}$$

3.4.1 SCENARIO FEATURES

A negotiation scenario consists of a shared domain and individual preference profiles. Ilany and Gal [76] specified a list of features to model a scenario that they used for strategy selection in bilateral negotiation. Although the usage differs in their paper, the goal to model the scenario is the same, so we will follow Ilany et al.. The features are fully independent of the opponent's behaviour. An overview of the scenario features is provided in Table 3.3.

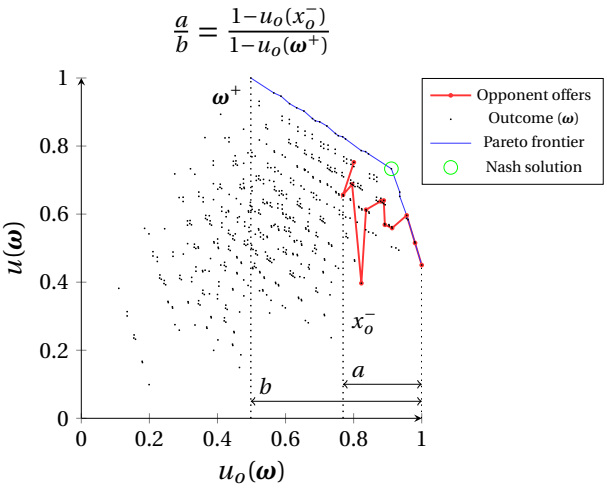

Figure 3.1: Visualisation of Concession Rate (CR)

3.4.2 OPPONENT FEATURES

This section describes the opponent features in detail. For each opponent, we store both the *mean* and the *Coefficient of Variance (CoV)* of all features.

NORMALIZED TIME

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

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. [15]. The CR is a normalized 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.

To calculate the CR, Baarslag et al. [15] used two constants. The minimum utility an opponent has demanded during the negotiation session $u_o(x_o^-)$ and the Full Yield Utility (FYU), which is the utility that the opponent receives at our maximum outcome $u_o(\omega^+)$.

We present a formal description of the CR in Equation 3.7 and a visualisation in Figure 3.1.

$$CR(x_o^-) = \begin{cases} 1 & \text{if } u_o(x_o^-) \leq u_o(\omega^+), \\ \frac{1-u_o(x_o^-)}{1-u_o(\omega^+)} & \text{otherwise.} \end{cases} \tag{3.7}$$

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

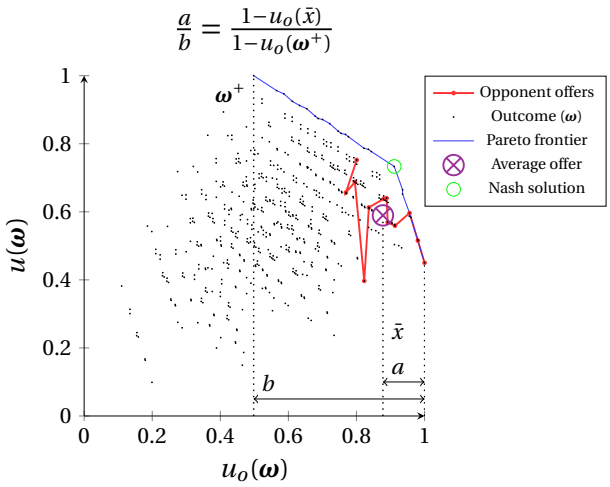

Figure 3.2: Visualisation of Average Rate (AR)

opponent demanded ($u_o(\bar{x})$). The AR is a normalized 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. We present a definition of the AR in Equation 3.8 and a visualisation in Figure 3.2.

$$AR(\bar{x}) = \begin{cases} 1 & \text{if } u_o(\bar{x}) \leq u_o(\boldsymbol{\omega}^+), \\ \frac{1-u_o(\bar{x})}{1-u_o(\boldsymbol{\omega}^+)} & \text{otherwise.} \end{cases} \tag{3.8}$$

The AR is another indication of competitiveness of the opponent based on average utility demanded instead of minimum demanded utility as the CR is.

DEFAULT CONFIGURATION PERFORMANCE

According to Hutter et al. [75], the performance of any default configuration on a problem works 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 normalized 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 using it to obtain an agreement $\boldsymbol{\omega}_{agree}$ with the opponent. We then normalize the obtained utility and use it as the Default Configuration Performance (DCP) feature. We present the formal definition of this feature in Equation 3.9 and a visualisation in Figure 3.3.

$$DCP(\boldsymbol{\omega}_{agree}) = \begin{cases} 0 & \text{if } u(\boldsymbol{\omega}_{agree}) \leq u(\boldsymbol{\omega}^-), \\ \frac{u(\boldsymbol{\omega}_{agree})-u(\boldsymbol{\omega}^-)}{1-u(\boldsymbol{\omega}^-)} & \text{otherwise.} \end{cases} \tag{3.9}$$

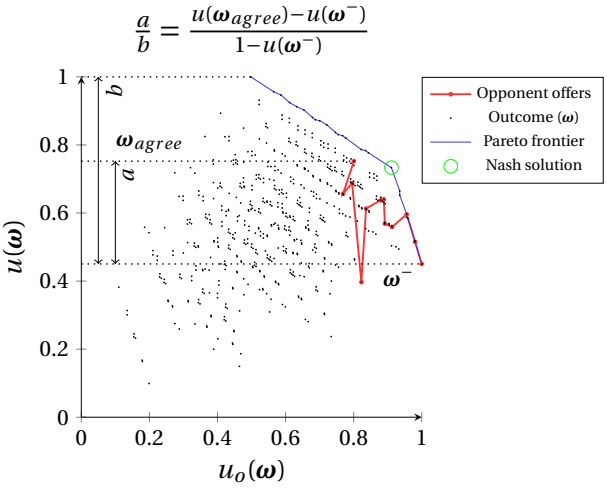

Figure 3.3: Visualisation of Default Configuration Performance (DCP)

Table 3.4: Opponent utility function usage

	Training	**Testing**
$DA(\theta)$	$\hat{u}_o(\omega)$	$\hat{u}_o(\omega)$
SMAC	$u_o(\omega)$	N/A

3.4.3 OPPONENT UTILITY FUNCTION

As can be seen in Figure 3.1, Figure 3.2, and Figure 3.3, the actual opponent utility function $u_o(\omega)$ is used to calculate the opponent features. SMAC is only used to configure the Dynamic Agent on the training set. As the opponent features are only used by SMAC, we can safely use the opponent's utility function to construct those features (Equation 3.7, Equation 3.8 and Equation 3.9) without giving the Dynamic Agent an unfair advantage during testing. The Dynamic Agent always uses the predicted opponent utility $\hat{u}_o(\omega)$ obtained through the model (Table 3.2.1), as is conventional in the ANAC.

We provide an overview of when the predicted opponent utility function and when the actual opponent utility function is used in Table 3.4.

3.5 EMPIRICAL EVALUATION

We must set baseline configurations to compare to the result of the optimisation. The basis of our Dynamic Agent is derived from a paper by Lau et al. [92]. Though some functionality is added, it is possible to set our agent's strategy to resemble that of the original agent. We refer to this configuration from the literature as θ_l, its parameters can be found in Table 3.5.

Another baseline strategy is added, which is configured manually, as the literature configuration is outdated. A combination of intuition, past research, and

Table 3.5: Baseline configurations parameters

	Accepting				Fitness function			Space exploration					
θ	α	β	t_{acc}	γ	n	δ	e	N_p	N_t	E	R_c	R_m	R_e
θ_l	1	0	1	MAX^W	1	0.5	0.5	200	3	3	0.6	0.05	0.1
θ_m	1	0	0.98	MAX^W	4	0.95	0.05	300	5	4	0.6	0.05	0.05

manual search is used for this manual configuration, which we consider the default method for current ANAC competitors. We present the manually configured parameters θ_m in Table 3.5 and an explanation below:

- *Accepting*: The acceptance condition parameters of θ_l set a pure AC_{next} strategy with parameters $\alpha = 1$, $\beta = 0$. Baarslag et al. [14] performed 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. [14].
- *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.

3.5.1 METHOD

SMAC is run in *embarrassingly parallel* mode on a computing cluster by starting a separate SMAC process on chunks of allocated hardware. SMAC selects a negotiation instance and a configuration to evaluate that instance and calls the negotiation environment GENIUS through a wrapper function.

Input The training instances were created by selecting a diverse set of opponents and scenarios from the GENIUS environment. The scenarios have non-linear utility functions and vary in competitiveness and outcome space size (between 9 and 400 000). The scenario features were calculated in advance as described in Section 3.4.1, and the configuration space is defined in Section 3.2.1.

The opponent features, as defined in Section 3.4.2, can only be gathered by performing negotiations against the opponents. We gather these features in advance by negotiating 10 times in every instance with the manual strategy θ_m.

Hardware & configuration budget We perform 300 independent parallel runs of SMAC for 4 hours of wall-clock time each, on a computing cluster running Simple

Table 3.6: Configurations overview

	Accepting				Fitness function			Space exploration					
θ	α	β	t_{acc}	γ	n	δ	e	N_p	N_t	E	R_c	R_m	R_e
θ_l	1	0	1	MAX^W	1	0.5	0.5	200	3	3	0.6	0.05	0.1
θ_m	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

Linux Utility for Resource Management (SLURM). To ensure consistent results, all runs were performed on Intel® Xeon® CPU, allocating 1 CPU core, with 2 processing threads and 12 GB RAM to each run of SMAC.

Output Every parallel SMAC process outputs its best configuration θ_{inc} after the time budget is exhausted. As there are 300 parallel processes, a decision must be made on which of the 300 configurations to use. To do so, the SMAC random forest regression model conform Equation 3.5 is rebuilt and used to predict the performance of every θ_{inc}. The configuration with the best-predicted performance is selected as the best configuration θ_{opt}.

3.5.2 RESULTS

The configuration process, as described, is run three times without instance features and three times with instance features, under identical conditions. There is now a total of 8 strategies: 2 baselines $[\theta_l, \theta_m]$, 3 optimized 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 3.6.

The obtained configurations are now analyzed with an emphasis on the following three topics:

1. The influence of the instance features on the convergence of the configuration process.

2. The performance of the obtained configurations on a never-before-seen set of instances.

3. The performance of the best configuration in an ANAC-like bilateral tournament.

INFLUENCE OF INSTANCE FEATURES

To study the influence of the instance features on the configuration process, we compare the strategies obtained by configuring with features and configuring without features. Only the training set of instances is used for the performance comparison, as we are purely interested in the convergence towards a higher utility.

Table 3.7: Performance of configurations on $\mathcal{P} = \mathcal{P}_{train}$

θ	$M(\theta, \mathcal{P})$	$\frac{M(\theta,\mathcal{P})-M(\theta_m,\mathcal{P})}{M(\theta_m,\mathcal{P})}$	**Description**
θ_l	0.533	-0.307	Literature
θ_m	0.769	0	Manually configured
θ_1	0.785	0.020	Configured without features
θ_2	0.770	0.000	Configured without features
θ_3	0.792	0.029	Configured without features
θ_4	0.800	0.040	Configured with features
θ_5	0.816	0.060	Configured with features
θ_6	0.803	0.044	Configured with features

Table 3.8: Performance of configurations on $\mathcal{P} = \mathcal{P}_{test}$

θ	$M(\theta, \mathcal{P})$	$\frac{M(\theta,\mathcal{P})-M(\theta_m,\mathcal{P})}{M(\theta_m,\mathcal{P})}$	**Description**
θ_l	0.563	-0.261	Literature
θ_m	0.763	0	Manually configured
θ_1	0.779	0.021	Configured without features
θ_2	0.760	-0.004	Configured without features
θ_3	0.774	0.015	Configured without features
θ_4	0.792	0.038	Configured with features
θ_5	0.795	0.042	Configured with features
θ_6	0.789	0.034	Configured with features

Every configuration is run 10 times on the set of training instances $\mathcal{P}_{train}$, and the average obtained utility is calculated by Equation 3.3. The results are presented in Table 3.7, including an improvement ratio over θ_m.

SMAC is capable of improving the performance of the Dynamic Agent above our capabilities of manual configuration. We observe that configuration without instance features potentially leads to marginal improvements on the training set. Finally, we observe that the usage of instance features leads to less variation in final configuration parameters (Table 3.6) and to a significant improvement of obtained utility.

PERFORMANCE ON TEST SET

Testing the configurations on a never-before-seen set of opponent agents and scenarios is needed to rule out potential overfitting. We selected a diverse set of scenarios and opponents for testing, such that $|\mathcal{P}_{test}| = |A_{test}| * |S_{test}| = 16 * 28 = 448$.

Every configuration is once again run 10 times on the set of training instances $\mathcal{P}_{test}$ and the average obtained utility is calculated by Equation 3.3. The results are presented in Table 3.8, including an improvement ratio over θ_m.

It is now clear that strategy configuration without instance features is undesirable as it potentially leads to a worse-performing strategy. Configuration with instance feature, on the other hand, still leads to a significant performance increase on a never-before-seen set of negotiation instances.

ANAC TOURNAMENT PERFORMANCE OF BEST CONFIGURATION

The strategy configuration method is successful in finding improved configurations, but the results are only compared against the other configurations of our Dynamic Agent. No comparison is yet made with agents built by ANAC competitors. We now compare the performance of the best configuration that we found to the ANAC agents in the test set of opponents.

We select θ_5 as the best strategy based on performance on the training set and enter the Dynamic Agent in an ANAC-like a bilateral tournament with a 60-second deadline. The Dynamic Agent is combined with the test set of opponents and scenarios. Every combination of 2 agents negotiated 10 times on every scenario, for a total amount of 38080 negotiation sessions. The averaged results are presented in Table 3.9. We elaborate on the performance measures found in the table:

- **Utility:** The utility of the agreement.
- **Opp. utility:** The opponent's utility of the agreement.
- **Social welfare:** The sum of utilities of the agreement.
- **Pareto distance:** Euclidean distance of the agreement to the nearest outcome on the Pareto frontier in terms of utility.
- **Nash distance:** Euclidean distance of the agreement to the Nash solution in terms of utility (Equation 2.3).
- **Agreement ratio:** The ratio of negotiation sessions that result in an agreement.

Using the Dynamic Agent with θ_5 results in a successful negotiation agent that is capable of winning a ANAC-like bilateral tournament by outperforming all other agents (two-tailed t-test: $p < 0.001$). It 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, while also outperforming it on every other performance measure.

Since the presence of our agent in the tournament also influences the performance of other agents, we also ran the full tournament without our Dynamic Agent as a sanity check. The top 5 performers of this tournament are presented in Table 3.10, along with their margins over the respective next lower-ranking agent in terms of utility.

3.6 CONCLUSION

The two main contributions of this chapter are (1) the success of automated configuration of negotiation strategies using a general-purpose configuration procedure (here: SMAC), and (2) an investigation of the importance of the features of negotiation settings.

Table 3.9: Bilateral ANAC tournament results using $DA(\theta_5)$ (bold = best, underline = worst)

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 3.10: Bilateral ANAC tournament without $DA(\theta_5)$

Agent	**Utility**	**Margin**
Mosa	0.715	3.01%
ShahAgent	0.736	2.43%
RandomDance	0.754	0.65%
AgentF	0.759	0.01%
SimpleAgent	0.759	

3.6.1 CONFIGURATION

Two baseline strategies were selected for our comparison. The first configuration, θ_l, is based on publications from which we derived the agent [92, 14]. The second configuration, θ_m, is configured based on intuition, recent literature and manual search, which we considered the default approach for current ANAC competitors. In Section 3.5, we automatically configured our dynamic Agent using SMAC.

The configuration based on earlier work θ_l [92] performed poorly compared to the manually configured configuration θ_m, and achieved 26.1% lower utility on our test set. The best automatically configured strategy θ_5 outperformed both baseline configurations and achieved a 4.2% increase in utility compared to θ_m. From this, we conclude that the automated configuration method is successful in outperforming manual configuration.

Our experiments show that the automated configuration method can produce a strategy that can win an ANAC-like bilateral tournament by a margin of 5.1% (Table 3.9). This is particularly striking when considering that without our agent, the winner of the same tournament beats the next-based agent only by a margin of 0.01%.

3.6.2 FEATURES

We consider a set of features that characterizes the negotiation scenario as well as the opponent. Our empirical results indicate that when using the negotiation instance features, SMAC is able to find good configurations faster.

Overall, using SMAC in combination with instance features leads to less variation in the parameter settings between the final configurations obtained in multiple independent runs (Table 3.6, Table 3.7), as well as significant and consistent performance improvement. Furthermore, our results show that automated configuration without features does not always outperform manual configuration. Therefore, we conclude that the instance features presented in this chapter are a necessary ingredient for the successful automated configuration of negotiation strategies.

3.6.3 NEXT STEPS

For this initial step towards automated configuration of negotiation agents, the negotiation scenarios were simplified by removing the reservation utility and the discount factor. Now that we have demonstrated that our general approach can be successful, additional validation should be performed in more complex and different negotiation environments.

Over the years, it became clear that there is no single best negotiation strategy for all negotiation settings [99]. In this chapter, we have presented a method to automatically configure an effective strategy for a specific set of negotiation settings. However, if this set becomes too diverse, we inherently end up in a situation where the automatically configured best strategy may not perform too well. In the next chapter, we exploit the strategy space of the dynamic agent by extracting multiple complementary strategies for specific settings, along with an online selection mechanism that determines the strategy to be used in a specific instance.