Automated Configuration and Usage of Strategy Portfolios for Mixed-Motive Bargaining

Bram M. Renting
Leiden University
Delft University of Technology
Leiden & Delft, The Netherlands
B.M.Renting@liacs.leidenuniv.nl

Holger H. Hoos
RWTH Aachen
Leiden University
University of British Colombia
hh@liacs.nl

Catholijn M. Jonker
Delft University of Technology
Leiden University
Delft & Leiden, The Netherlands
C.M.Jonker@tudelft.nl

ABSTRACT

Bargaining can be used to resolve mixed-motive games in multi-agent systems. Although there is an abundance of negotiation strategies implemented in automated negotiating agents, most agents are based on single fixed strategies, while it is acknowledged that there is no single best-performing strategy for all negotiation settings.

In this paper, we focus on bargaining settings where opponents are repeatedly encountered, but the bargaining problems change. We introduce a novel method that automatically creates and deploys a portfolio of complementary negotiation strategies using a training set and optimise pay-off in never-before-seen bargaining settings through per-setting strategy selection. Our method relies on the following contributions. We introduce a feature representation that captures characteristics for both the opponent and the bargaining problem. We model the behaviour of an opponent during a negotiation based on its actions, which is indicative of its negotiation strategy, in order to be more effective in future encounters.

Our combination of feature-based methods generalises to new negotiation settings, as in practice, over time, it selects effective counter strategies in future encounters. Our approach is tested in an Automated Negotiating Agents Competition (ANAC)-like tournament, and we show that we are capable of winning such a tournament with a 5.6% increase in pay-off compared to the runner-up agent.

KEYWORDS

Bargaining;Algorithm Configuration;Algorithm Selection;Mixed-Motive Games

1 INTRODUCTION

Bargaining or negotiation is a prominent method to decentrally solve mixed-motive problems through reaching a mutual agreement. Problems from this area occur prominently in many real-world applications (e.g., transportation of goods using warehouse robotics, coordination of autonomous vehicles, calendar scheduling).

Since the 1980s, there has been research aimed at designing computer negotiators that can replace or assist humans in negotiation. Following early contributions by Jelassi and Foroughi [12], Klein and Lu [14], Robinson [23], Rosenschein [24], Smith [27], Sycara [29], Sycara-Cyranski [30], this research area has evolved considerably, and at the time of this writing, there are regular negotiation competitions (e.g., Automated Negotiating Agents Competition (ANAC) [3]) and standardised test-beds (e.g., General Environment for Negotiation with Intelligent multi-purpose Usage Simulation (GENIUS) [15]) that support the development of algorithmic negotiation strategies. There are now more than one hundred negotiation strategies freely available that can be used as opponents to test against — which is important, since we know that the success of a negotiator also depends on the strategy of the opponent [4].

The improvement of negotiation strategies over time is promising; however, we observe that the strategies almost always remain monolithic, i.e. single strategy with fixed behaviour for every setting. It has been observed that no single strategy is optimal for all negotiation settings [11, 15]. Therefore, a good way to further improve pay-off appears to select from a portfolio of strategies, based on the negotiation setting. This introduces the problem of algorithm selection [22] into bargaining. An early attempt on applying algorithm selection in automated negotiation was made by Ilany and Gal [10, 11], but they only selected a strategy based on the bargaining problem, without considering the opponent, which we know to be an important factor [4]. Furthermore, they relied on a portfolio of existing strategies to select from, which potentially limits robustness.

Our contributions in this paper are as follows: (i) we apply automated algorithm configuration techniques to not only create a single negotiation strategy, but a portfolio of complementary negotiation strategies; and (ii) we introduce a procedure to learn and exploit opponent and problem characteristics during a simulated ANAC tournament. The first contribution uses the approach by Renting et al. [21] to automatically configure negotiation strategies, which we extend by implementing Hydra [31] for portfolio construction and AutoFolio [16] to create a portfolio selector. Empirical results on a variety of bargaining settings show that our method beats the runner-up agent by a (comfortable) margin of 5.6%.

2 RELATED WORK

Thanks to ANAC, new negotiation strategies are developed every year and collected in the GENIUS test-bed [15], to support future research; they are categorised and empirically evaluated [2, 4] to provide a basis for new strategies.

Most negotiation strategies contain policy parameters that influence the behaviour of the agent. To optimise the performance of the agent, the parameters need tuning. So far, tuning is mostly done manually while testing on the available opponents in the GENIUS test-bed. Although manual configuration is conceptually easy, it is also tedious and often leads to unsatisfactory results. Following earlier attempts at automatically configuring strategies using genetic algorithms [7, 8, 18], or reinforcement learning [6, 26], a recent successful approach used a model-based algorithm procedure SMAC [9] to automatically configure a negotiation strategy [21].

As there is no single best strategy for all negotiation problems [11, 15], we should be able to improve pay-off by exploiting differences in problem instances by selecting different strategies per negotiation setting. We see this as a variation of the algorithm selection problem [22]. While algorithm selection methods have been successfully applied to SAT-solving [32], pattern recognition [28] and other problems, only few attempts have been made to apply them in the area of automated negotiation. Ilany and Gal [10, 11] used a set of past ANAC strategies and predicted which strategy would perform best on a given bargaining problem; they then entered that strategy into the negotiation session. Although they managed to improve the pay-off of the agent in this manner, they were unable to win ANAC. Kawata and Fujita [13] used a portfolio of 7 strategies that previously competed in ANAC. They applied a multi-armed bandit approach to find the best performing strategy for every combination of an opponent and problem, while repeating precisely the same bargaining setting 100 times. Unfortunately, this strategy does not generalise to new negotiation settings and problems.

3 PRELIMINARIES

Agent systems that are built to negotiate contain a software-based negotiation strategy. This negotiation strategy must function according to the rules (or protocol) that is set for a negotiation setting. The protocol used in this work is the Stacked Alternating Offers Protocol [1], an extension of the Alternating Offers Protocol [19, 25]. A deadline of 60 seconds is used, normalised to $t \in [0, 1]$, after which a negotiation is aborted without agreement. We refer to a bargaining problem as $p \in P$, which we will negotiate between our own agent and an opponent $o \in O$. The combination of a bargaining problem and an opponent is a bargaining setting defined as $s \in S = O \times P$. Protocols, problems and opponents are all available through the GENIUS test-bed [15] (GPL v3), which we use for benchmarking.

3.1 Bargaining problem

We negotiate over multi-issue (or multi-objective) problems that are defined according to a common standard in automated negotiation [2, 17, 20]. Here, an issue $i \in I$ is an objective in the problem for which an agreement must be found. The set of possible solutions for an issue is denoted by V_i, and the Cartesian product of all the solutions of issues in a problem forms the total outcome space, such that $\prod_{i \in I} V_i = \Omega$. An outcome is denoted by $\omega \in \Omega$.

Preferences over the outcome space Ω are expressed through a utility function $u(\omega)$, such that $u : \Omega \to [0, 1]$, where a value of 1 represents the best possible outcome. We refer to our own utility function as $u(\omega)$ and to that of the opponent as $u_o(\omega)$. Negotiations

Table 1: Configuration space of DA(θ) as set by Renting et al. [21]

Description	Symbol	Domain	Purpose
Scale factor	α	$[1, 1.1]$	Accepting
Utility gap	β	$(0, 0.2]$	Accepting
Accepting time	t_{acc}	$[0.9, 1]$	Accepting
Lower boundary	γ	$\{MAX^W, AVG^W\}$	Accepting
Trade-off factor	δ	$[0, 1]$	Bidding
Conceding factor	e	$(0, 2)$	Bidding
Conceding goal	n	$\{1, 2, 3, 4, 5\}$	Bidding
Population size	N_p	$[50, 400]$	Searching
Tournament size	N_t	$[1, 10]$	Searching
Evolutions	E	$[1, 5]$	Searching
Crossover rate	R_c	$[0.1, 0.5]$	Searching
Mutation rate	R_m	$[0, 0.2]$	Searching
Elitism rate	R_e	$[0, 0.2]$	Searching

are performed under incomplete information, so the utility of the opponent is predicted, which we denote as $\hat{u}_o(\omega)$.

3.2 Dynamic agent

Renting et al. [21] built a flexible agent and automatically configured it using SMAC (described later in this section). They demonstrated that this agent, DA(θ), was able to win an ANAC-like tournament by a significant margin. We implemented the same DA(θ) with configuration $\theta \in \Theta$. The full configuration space Θ of DA(θ) can be found in Table 1. There are three types of parameters that influence the behaviour of DA(θ): four accepting parameters that influence when the agent accepts an offer, three bidding parameters that determine the utility to demand, and six parameters that influence searching in the solution spaces for suitable solutions.

3.3 Automated Configuration

Automated algorithm configuration procedures evaluate configurations of a given algorithm, observe their performance, and use this information to find better-performing configurations for a given set or distribution of problem instances. We attempt to optimise the utility $r(\theta, s) \in [0, 1]$ obtained by playing strategy θ in a negotiation setting s. As we work with a set of settings S, we define the optimisation metric as the average utility:

$$R(\theta, S) = \frac{1}{|S|} \cdot \sum_{s \in S} r(\theta, s), \tag{1}$$

To optimise average utility, we use the freely available general-purpose algorithm configurator SMAC [9] to automatically configure DA(θ), which was shown to be successful by Renting et al. [21]. A pseudocode version of SMAC can be found in Algorithm 1, where the notation is modified for this work. Here, SMAC is used to optimise on single settings ($s \in S$) in a training set to significantly reduce computational expense. SMAC trains a model that maps configurations and setting characteristics to an expected performance. This allows for searching for promising configurations on a model before actually testing the performance. To do so, it

requires bargaining setting features that capture information on setting complexity. We describe these features in Section 3.4.

Algorithm 1 SMAC [9]

Input	Θ	Configuration space
	S	Negotiation settings
	r	Performance metric
	t_{opt}	Optimisation time budget
Variables	H_i	Runhistory of pool i
	H_{full}	Full runhistory of parallel pools, where $H_{full} = [H_1, \ldots, H_m]$
	$\mathcal{M}$	Random forest regression model
	θ_{new}	List of promising configurations
Output	θ_{inc}	Optimised parameter configuration

1: $[H_i, \theta_{inc}] \leftarrow$ Initialise(Θ, S)
2: **loop until** GetTime$() > t_{opt}$
3:    $H_{full} \leftarrow$ ReadParallelRunhistories$()$
4:    $\mathcal{M} \leftarrow$ FitModel(H_{full})
5:    $\theta_{new} \leftarrow$ SelectConfigurations$(\mathcal{M}, \theta_{inc}, \Theta)$
6:    $[H_i, \theta_{inc}] \leftarrow$ Intensify$(\theta_{new}, \theta_{inc}, H_i, S, r)$
7: **return** θ_{inc}

Algorithm 2 Intensify$(\theta_{new}, \theta_{inc}, H, S, r)$ [9]

Input	θ_{new}	List of promising configurations
	θ_{inc}	Incumbent configuration (current best)
	H	Runhistory
	S	Negotiation settings
	r	Performance metric
	t_{int}	Time budget for intensify procedure
Variables	θ_{new}	Challenging configuration
Output	H	Runhistory
	θ_{inc}	Incumbent configuration (current best)

1: **for** $i := 1, \ldots, |\theta_{new}|$ **do**
2:   $S' \leftarrow \{s' \in S : \text{Count}(\theta_{inc} \text{ on } s') \leq \text{Count}(\theta_{inc} \text{ on } s''), \forall s'' \in S\}$
3:   $s \leftarrow$ Random(S')
4:   $H \leftarrow$ ExecuteNegotiation$(H, DA(\theta_{inc}), s)$
5:   $\theta_{new} \leftarrow \theta_{new}[i]$
6:   $N \leftarrow 1$
7:   **loop**
8:     $S_{missing} \leftarrow \{s \in S : \theta_{inc} \text{ is evaluated on } s \text{ but } \theta_{new} \text{ is not}\}$
9:     $S_{torun} \leftarrow$ random subset of $S_{missing}$ of size Min$(N, |S_{missing}|)$
10:     **for** $s \in S_{torun}$ **do** $H \leftarrow$ ExecuteNegotiation$(H, DA(\theta_{new}), s)$
11:     $S_{missing} \leftarrow S_{missing}/S_{torun}$
12:     $S_{common} \leftarrow \{s \in S : \theta_{new} \text{ and } \theta_{inc} \text{ are evaluated on } s\}$
13:     **if** $\sum_{s \in S_{common}} r(\theta_{new}, s) < \sum_{s \in S_{common}} r(\theta_{inc}, s)$ **then break**
14:     **else if** $S_{missing} = \emptyset$ **then** $\theta_{inc} \leftarrow \theta_{new}$; **break**
15:     **else** $N \leftarrow 2 * N$
16:   **if** (GetTime$() > t_{int}) \wedge i \geq 2$ **then break**
17: **return** $[H, \theta_{inc}]$

3.4 Bargaining setting features

SMAC needs a feature description of the bargaining settings that captures key properties of those settings. We also need these features for strategy selection (Section 5), which is a multi-class classification problem with features as input and a prediction of the best-performing strategy as output.

Renting et al. [21] created a set of features to describe a bargaining setting, which was partly based on previous work by Ilany and Gal [11] and Baarslag et al. [5]. We adopt this set of features

Description	Definition
Number of issues	$|I|$
Average number of values per issue	$\frac{1}{|I|} \cdot \sum_{i \in I} |V_i|$
Number of possible outcomes	$|\Omega|$
Standard deviation of issue weights	$\sqrt{\frac{1}{|I|} \cdot \sum_{i \in I} \left(w_i - \frac{1}{|I|}\right)^2}$
Average utility of all possible outcomes	$\frac{1}{|\Omega|} \cdot \sum_{\omega \in \Omega} u(\omega) = u(\bar{\omega})$
Standard deviation utility of all possible outcomes	$\sqrt{\frac{1}{|\Omega|} \cdot \sum_{\omega \in \Omega} (u(\omega) - u(\bar{\omega}))^2}$

Table 2: Bargaining problem features (X_p) [11]

Description	Definition
The time it takes to reach an agreement	t
Concession rate of opponent	$\begin{cases} 1 & \text{if } \hat{u}_o(x_o^-) \leq \hat{u}_o(\omega^+), \\ \frac{1 - \hat{u}_o(x_o^-)}{1 - \hat{u}_o(\omega^+)} & \text{otherwise.} \end{cases}$
Average offer rate of opponent	$\begin{cases} 1 & \text{if } \hat{u}_o(\bar{x}) \leq \hat{u}_o(\omega^+), \\ \frac{1 - \hat{u}_o(\bar{x})}{1 - \hat{u}_o(\omega^+)} & \text{otherwise.} \end{cases}$
Default strategy performance	$\begin{cases} 0 & \text{if } u(\omega_a) \leq u(\omega^-), \\ \frac{u(\omega_a) - u(\omega^-)}{1 - u(\omega^-)} & \text{otherwise.} \end{cases}$

Table 3: Opponent features (X_o) [21]. x_o^- is the lowest offer by the opponent in their utility. ω^+/ω^- is our best/worst possible outcome. $\bar{x}$ is the (fictional) average offer by the opponent in their utility. ω_a is the agreement.

consisting of problem features (X_p) and opponent features (X_o) (Equation 2). An overview of the bargaining setting features we use is given in Table 2 & Table 3. Opponent behaviour depends partly on the problem and is not always deterministic. We therefore calculate both the mean and covariance of the opponent features over multiple bargaining settings as opponent features for a total of eight opponent features.

$$\mathcal{F} : S \rightarrow (X_p \times X_o) \tag{2}$$

3.5 Problem definition

Strategy portfolio creation. We have an agent with a dynamic strategy DA(θ) based on configuration space Θ. Can we create a portfolio of configurations $\theta \subset \Theta$ using a training set of negotiation settings S consisting of configurations that outperform each other on specific subsets of a test set of negotiation settings $S'_{test} \subset S_{test}$ that have never been encountered before?

Algorithm selection. We have an agent with a dynamic strategy DA(θ), and a portfolio of configurations $\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$, where θ_1 is the single best-performing configuration (Equation 4). Can

we apply an algorithm selection method $\theta_s = AS(\boldsymbol{\theta}, s)$ that selects a configuration θ_s from $\boldsymbol{\theta}$ based on negotiation setting s, such that $R(AS(\boldsymbol{\theta}, s), S_{test}) > R(\theta_1, S_{test})$. The real goal here is to let $R(AS(\boldsymbol{\theta}, s), S_{test})$ approach the performance of the oracle selector (Equation 3) $R(OR(\boldsymbol{\theta}, s), S_{test})$ as closely as possible.

$$OR(\boldsymbol{\theta}, s) \in \arg\max_{\theta \in \boldsymbol{\theta}} r(\theta, s) \tag{3}$$

$$\theta_1 \in \arg\max_{\theta \in \boldsymbol{\theta}} R(\theta, S) \tag{4}$$

4 PORTFOLIO CREATION

As a basis for algorithm selection, we need a portfolio of negotiation strategies to select from. A simple approach is to build a portfolio of negotiation strategies that already exist within the GENIUS environment, which is the approach used by Ilany and Gal [11]. However, for several reasons, we consider this a less-than-ideal approach:

(1) It relies on strategies that already exist, thus limiting our choices for a portfolio to strategies that have been previously implemented and are available to be re-used.
(2) The strategies might not be optimised or optimised for a different objective, resulting in a low-performance portfolio.
(3) There might be dominated strategies in the portfolio, which are outperformed in all cases by some other strategy in the portfolio, needlessly complicating the selection problem.
(4) The portfolio might not be robust. There can be negotiation settings for which all the negotiation strategies fail to achieve decent performance, causing "weak spots" in our portfolio.

4.1 Portfolio creation

We aim to expand upon the work of Renting et al. [21], by not only automatically configuring a single negotiation strategy, but by building a portfolio of complementary strategies to better exploit differences between negotiation settings. The portfolio of strategies $\boldsymbol{\theta}$ we create is thus a portfolio of configurations for our DA(θ). In our method we will therefore enforce that every strategy must add value to the portfolio:

$$\forall \theta \in \boldsymbol{\theta}, \; \exists s \in S, \; \forall \theta' \in (\boldsymbol{\theta} \setminus \theta) \; : \; r(\theta, s) > r(\theta', s) \tag{5}$$

The portfolio can be viewed as a set of strategies that each specialise on a region within the bargaining setting space. Similarities in this space are found by mapping the space to the feature space. One could obtain such a portfolio by automatically configuring strategies on sets of negotiation settings that are separated in feature space by dividing the feature space either manually or using clustering techniques. However, both methods rely on human input without clear insight into the effects. The quality of the sets is disputable, as they are created based on similarities in the given feature space without regard for the performance gains thus achieved. Therefore, instead we chose to automate the portfolio creation method by using Hydra [31], removing the requirement of human input in feature space separation.

4.2 Hydra

Hydra automatically generates a portfolio given only a parameterised strategy (Section 3.2) and a set of negotiation settings with features (Section 3.4) while using an algorithm configurator and an algorithm selector (Section 5). We provide a pseudo-code description of Hydra in Algorithm 3, modified for this work.

Algorithm 3 Hydra [31]

Input	Θ	Configuration space
	S	Training set of negotiation settings
	r	Performance metric
Variables	θ_k	Configuration
	$\boldsymbol{\theta}$	Portfolio of configurations
	r_k	Modified performance metric
Output	$\boldsymbol{\theta}$	Portfolio of configurations
	AS	Algorithm selector

1: $\boldsymbol{\theta} \leftarrow \emptyset; r_k \leftarrow r$
2: **for** $k = 1; k = k + 1$ **do**
3: $\quad \theta_k \leftarrow SMAC(\Theta, S, r_k)$ ▷ Algorithm 1
4: $\quad TestPerformance(S, \theta_k)$
5: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} \cup \{\theta_k\}$
6: $\quad AS \leftarrow FitAlgorithmSelector(\boldsymbol{\theta}, S)$
7: $\quad r_k \leftarrow GetModifiedPerformanceMetric(r, AS)$
8: $\quad$ **if** θ_k is not contributing to $\boldsymbol{\theta}$ on S **then**
9: $\quad\quad$ End for loop
10: **return** $AS, \boldsymbol{\theta}$

The main idea of Hydra is to perform multiple configurator runs on an identical set of training settings, while only modifying the performance metric. Due to the modifications to the metric, the configurator produces different strategies. In Algorithm 3, the modified performance metric is computed by "$GetModifiedPerformanceMetric$" and formally defined as:

$$r_k(\theta, s) = \max\{r(\theta, s), \; r(AS(\boldsymbol{\theta}, s), s)\}. \tag{6}$$

The modified performance is the better of the performance of the strategy that is assessed and the performance of the strategy that is selected by the algorithm selector. By optimising the increase of performance as compared to the current portfolio, the configurator aims to find a configuration that adds the most value to the portfolio. In the first configurator run, the default performance metric is used. The resulting configuration θ_1 is therefore a locally optimal configuration over the full set of training settings, also known as the *single best strategy* in the portfolio.

5 STRATEGY SELECTION

The next step in our approach is strategy selection. We now have a portfolio of strategies $\boldsymbol{\theta}$, but still need to decide which of these strategies best fits our current problem and opponent. We therefore desire a mapping from the feature space X to a one-hot distribution over the possible strategies. This is an algorithm selection problem [22] and is illustrated in Figure 1, modified for our work. Essentially, it is a classification problem, for which we can train a classifier on examples generated from our training set. Subsequently, we hope the learned function will generalise to new bargaining problems and unknown opponents in the test set, allowing us to select the most suitable strategy from our portfolio.

Ilany and Gal [11] also considered this algorithm selection problem and analysed the performance of multiple classifiers that map
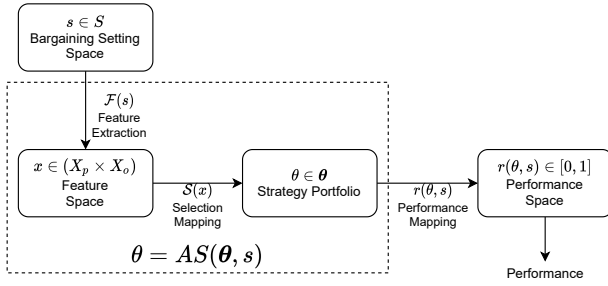
Figure 1: Algorithm selection schematics [22]

feature vectors to algorithms. The process of selecting a classifier and configuring the accompanying parameters can again be seen as an algorithm configuration problem. In line with the rest of this paper, we chose to automate the configuration of an algorithm selector by using AutoFolio [16], leveraging the power of a broad range of algorithm selection methods and removing human bias.

5.1 AutoFolio

The algorithm selection system AutoFolio is used to construct the algorithm selector. It has a range of regression and classification methods to choose from and uses SMAC to determine both the selection method to use and the settings of its hyperparameters. The data AutoFolio requires as input is the performance $r(\theta, s)$ of every strategy ($\theta \in \boldsymbol{\theta}$) on every setting ($s \in S$) in the training set and a set of features. Its goal is to select the best-performing strategy for every negotiation setting.

5.2 Cross validation.

AutoFolio uses 10-fold cross validation during optimisation to avoid overfitting, by dividing the bargaining settings in the training set in 10 subsets and leaving one subset out for performance testing. However, due to the nature of a bargaining setting being a combination of an opponent and a bargaining problem, this leads to overfitting of the algorithm selector. The training set of bargaining settings is the Cartesian product of the training set of opponents and problems $S = O \times S$, so both components are included multiple times in the training set.

To address this issue, we modified AutoFolio to split the cross validation folds based on the set of opponents and problems that build the bargaining settings. The set of opponents and the set of problems are each split into 4 subsets, such that we obtain a total of $4 \cdot 4 = 16$ folds. When selecting a fold ($|S_{fold}| = \frac{1}{16} \cdot |S|$), we must eliminate the part of the remaining training set ($|S_{elim}| = \frac{6}{16} \cdot |S|$) that overlaps with the fold based on opponents and problems and use the remaining setting ($|S_{fit}| = \frac{9}{16} \cdot |S|$) to fit the algorithm selector. This cross validation approach reduces the workable size of the training set, but it does prevent training on test opponents/problems.

5.3 Performance baselines

The oracle selector (Equation 3) always makes the perfect choice for every negotiation setting and is an upper bound on the performance of a selector using the given portfolio. It is obtained by simply trying every strategy on every setting and selecting the best strategy. The

single best strategy is the strategy in the portfolio that obtains the highest performance on the full set of negotiation settings (Equation 4). We refer to this strategy as θ_1, as it is the first strategy in the portfolio produced by Hydra. The performance of the single best strategy is considered to be the baseline.

6 EMPIRICAL EVALUATION

We will first describe the method that was used to obtain the results of this work before we show the results.

6.1 Method

The first configurator run with the default performance metric results in the single best strategy θ_1 on the training set of negotiation settings. We iterated through Hydra until $k = 4$. At that point, the Hydra loop was terminated, as the last strategy that was added did not contribute to the portfolio based on the training set, which will be shown in Section 6. This also allows us to analyse the performance of portfolios of sizes 1, 2 and 3, due to the incremental approach of Hydra. The configurations thus obtained were tested 10 times on every negotiation setting in the training set, to capture performance variation due to randomness in the negotiation strategies. Finally, the portfolio and the performance data was used along with the setting features to configure an algorithm selector using AutoFolio.

6.1.1 Input. An overview of the opponents that are used in this work can be found in Table 4. The test set of opponents O_{test} consists of the bug-free ANAC 2017 agents. More recent ANAC agents are not compatible with this work, due to different challenges, such as partially defined preferences and a change of benchmarking platform since 2020. In line with the competition, we allow ourselves access to the agents of previous ANAC editions (before 2017) that we use as a training set S for the Hydra procedure (Algorithm 3). Two additional agents are added to the test set in order to compare our work to the work of Ilany and Gal [11], which adopted a similar portfolio selection method. 36 agents from the ANAC are used, split up in 20 training agents and 16 test agents.

The set of problems is provided in Table 5. A total of 42 problems is used of which both sides can be played by our agent resulting in 84 playable problems. The set of bargaining problems is selected based on diversity using the features as described in Section 3.4 and their discount factor and reservation utility are removed. The set is split up into 56 training problems and 28 test problems.

The training set is of size $|S| = |O| \cdot |P| = 20 \times 56 = 1120$ and the test set is of size $|S_{test}| = |O_{test}| \cdot |P_{test}| = 16 \times 28 = 448$.

The bargaining problem features were calculated in advance, as described in Section 3.4. The opponent features can only be gathered by performing negotiations against the opponents. We gathered these features in advance for the first configurator run, by negotiating 10 times on every setting with a manually set strategy. After the first configurator run, opponent features are extracted based on negotiations with strategies that are already in the portfolio. Note that during training, we use the actual opponent's utility function (u_o) to calculate the features in Section 3.4 to reduce estimation noise.

Training set		Test set	
Agent	**ANAC**	**Agent**	**ANAC**
ParsCat	2016	SimpleAgent	2017
YXAgent	2016	Rubick	2017
Terra	2016	PonPokoAgent	2017
MyAgent	2016	ParsCat2	2017
GrandmaAgent	2016	ShahAgent	2017
Farma	2016	Mosa	2017
Caduceus	2016	Mamenchis	2017
Atlas3201	2016	MadAgent	2017
AgentHP2_main	2016	Imitator	2017
RandomDance	2015	GeneKing	2017
PokerFace	2015	Farma17	2017
PhoenixParty	2015	CaduceusDC16	2017
ParsAgent	2015	AgentKN	2017
kawaii	2015	AgentF	2017
Atlas3	2015	MetaAgent2013	2013
AgentX	2015	MetaAgent	2012
AgentH	2015		
AgentBuyogMain	2015		
Gangster	2014		
DoNA	2014		

Table 4: Overview of opponent set used in this work. The last column indicates in which year the opponent participated in ANAC.

6.1.2 Hardware & budget.
We followed Renting et al. [21] in terms of computational budget, in order to be able to compare results. Each run of SMAC was given a 1200-hour budget, divided over 300 parallel runs. Every run was performed on a single Intel® Xeon® CPU core with 2 threads and 12 GB of RAM. 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 same computing cluster, assigned it 4 GB of RAM, and provided it with a budget of 0.5 hours.

Output. The final algorithm selector was saved as a binary file at the final step of Hydra, along with the parameter settings of every strategy configuration (Table 6). We use both when faced with a new negotiation setting for which we want to select a configuration.

6.2 Results

We now present the results using a test set of bargaining settings S_{test}. More specifically, we investigated two aspects:

(1) the quality of the portfolio;
(2) the performance of the algorithm selector.

6.2.1 Quality of the portfolio.
We assessed the quality of the portfolio by measuring the performance (Equation 1) of every configuration in the portfolio on the training and testing sets of negotiation settings. The results can be found in Table 7. We included ratios that indicate how often a strategy is part of the set of best strategies per setting ("Sum" in Table 7). As a final quality check, the performance of the oracle selector (Equation 3) is evaluated for varying sizes of the portfolio. We present the results in Table 8.

Table 7 shows the results per strategy in the portfolio in the form of individual performance over a set of settings $R(\theta, S)$. It

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

Table 5: Overview of bargaining problem set used in this work.

is evident that θ_1 is the single best strategy over the full training set S. Furthermore, as every strategy is at least once the single best on individual settings (single best ratio > 0), we can conclude that every strategy contributes to the portfolio, thus satisfying our requirement from Section 3.5.

Finally, Table 8 shows us that, at every iteration of Hydra, the oracle performance of the portfolio increases on both S and S_{test}. The improvement decreases on S as the number of iterations increases, indicating that Hydra fills the largest "weaknesses" in the portfolio first.

6.2.2 Performance of the algorithm selector.
Table 8 shows that there is potential in the portfolio to improve utility of DA(θ) by $\frac{0.840-0.742}{0.742} \cdot 100\% \approx 13.0\%$ on the test set, if we use the oracle selector rather than θ_1. We now replace the oracle selector with the actual selector and test its performance in two ways.

Performance against known opponents. 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

Table 6: Final configurations in the portfolio. These are the final parameter settings that make up the different bargaining strategies in the portfolio.

θ	Accepting				Bidding			Searching					
	α	β	t_{acc}	γ	n_{fit}	δ	e	N_{pop}	N_{tour}	E	R_c	R_m	R_e
θ_1	1.038	0.03201	0.942	AVG^W	3	0.927	0.00199	262	6	4	0.290	0.140	0.085
θ_2	1.001	0.00166	0.935	AVG^W	3	0.998	0.06232	94	2	5	0.168	0.002	0.108
θ_3	1.007	0.01970	0.912	AVG^W	4	0.917	0.01093	305	10	1	0.107	0.063	0.184
θ_4	1.056	0.00003	0.900	MAX^W	5	0.997	0.02090	139	10	4	0.463	0.176	0.101

Table 7: Individual configuration performance on S and S_{test}. The left two columns show the average utility of every individual strategy in the portfolio on the training and test set of negotiation settings. The next four columns show the fraction of the amount settings in the test set for which a single strategy belongs to a set of best-performing strategies.

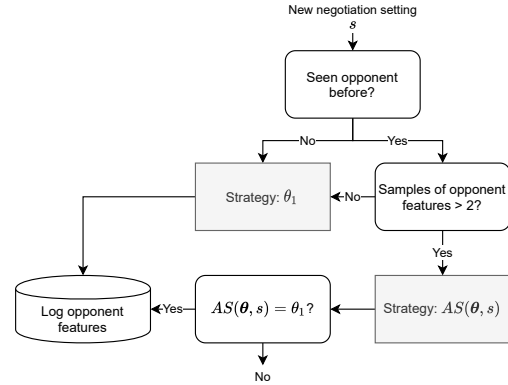| θ	$R(\theta, \cdot)$		Best performing on S_{test} by ratio				
	S	S_{test}	Single best	In top 2	In top 3	In top 4	Sum
θ_1	0.815	0.742	0.281	0.100	0.016	0.123	0.520
θ_2	0.788	0.734	0.167	0.022	0.020	0.123	0.333
θ_3	0.789	0.754	0.154	0.065	0.031	0.123	0.373
θ_4	0.773	0.721	0.118	0.058	0.033	0.123	0.333

Table 8: Algorithm selector performance compared to oracle performance. The left two columns show the upper limit in average utility for various sizes of the portfolio on the training and test set of negotiation settings. The right two columns show the average utility obtained by applying the trained algorithm selector on every setting in both sets.

θ	$R(OR(\theta, s), \cdot)$		$R(AS(\theta, s), \cdot)$	
	S	S_{test}	S	S_{test}
$\{\theta_1\}$	0.815	0.742	0.815	0.742
$\{\theta_1, \theta_2\}$	0.870	0.824	0.865	0.785
$\{\theta_1, \theta_2, \theta_3\}$	0.875	0.832	0.869	0.776
$\{\theta_1, \theta_2, \theta_3, \theta_4\}$	0.879	0.840	0.868	0.784

setting S_{test}. The opponent features are gathered by running 10 negotiation sessions with configuration θ_1 on the test set.

We trained and tested multiple algorithm selectors on different portfolio sizes by extending the portfolio, starting with the single best strategy θ_1. We report the performance in Table 8. For the oracle selector, the performance of DA(θ) increases with the size of the portfolio. However, the performance increase plateaus on S_{test} after adding the fourth strategy to the portfolio. Based on the results on the training set, we conclude that the fourth strategy in the portfolio is redundant and needlessly complicates the strategy selection procedure; we therefore omitted it in the final evaluation step reported in the following.

Performance with unknown opponents. Opponent features, in contrast to the problem features, must be learned from previous

Figure 2: Realistic strategy selection of DA($AS(\theta, s)$)

encounters. Up to this point, we assumed the opponents to always be known in advance, which is not realistic. We now simulate a realistic negotiation tournament where this problem occurs. The agents in S_{test} can also learn from their opponents, but we cannot guarantee fair learning chances due to parallelisation. To address this issue, we negotiate once against all of them and then clean up and restart our agent, giving every opponent a head start, favouring a handicap over any advantage for our agent.

The question arises of what strategy to select at first encounters with opponents when no opponent features are available. If strategy selection is not possible, we select the single best strategy θ_1. Opponent features are influenced by the strategy that is selected by DA(θ), 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. The coefficient of variation of an opponent feature (Section 3.4) needs at least two samples to be meaningful, so we set a second condition to select strategy θ_1 for the first two encounters with an opponent to "sample" the opponent. We illustrate this behaviour in Figure 2.

To obtain the results, we iterate randomly through the test settings S_{test} and use DA($AS(\theta, s)$) with $\theta = \{\theta_1, \theta_2, \theta_3\}$ to negotiate, following the procedure as described in Figure 2. Additionally, we let every opponent in the test set negotiate with every other opponent in the test set on every test problem and combine the results with the results of the DA(θ). This procedure is repeated 10 times to reduce the influence of variance for a total of 38 080 negotiations. The results averaged per agent show that we are capable of winning

Table 9: ANAC tournament results using DA($AS(\theta, s)$) where all scores are averaged over all bargaining settings. The goal of ANAC is to obtain the highest utility. We show the top 5 agents and all the outliers for every performance measure. Here, social welfare is the summation of utility and opponent utility, Pareto distance is the smallest distance to a Pareto efficient bargaining outcome, Nash distance is the distance to the Nash bargaining solution of the problem, and agreement ratio represents the fraction of settings that resulted in an agreement. (bold = best, underline = worst)

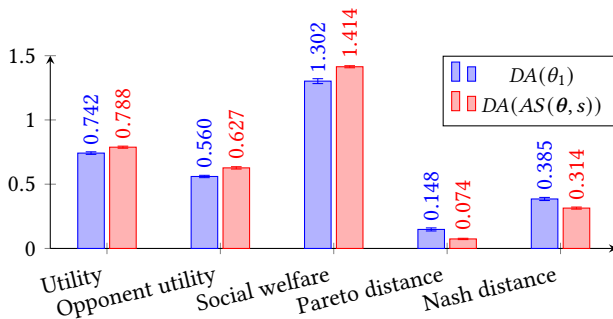| Agent	Utility	Opponent utility	Social welfare	Pareto distance	Nash distance
Imitator	<u>0.446</u>	**0.901**	1.347	0.091	<u>0.428</u>
GeneKing	0.612	0.783	1.396	0.065	0.378
Mamenchis	0.636	0.863	**1.498**	**0.016**	0.272
ParsCat2	0.642	0.773	1.414	0.090	0.273
MadAgent	0.669	0.536	<u>1.204</u>	<u>0.232</u>	0.383
Farma17	0.676	0.690	1.366	0.115	0.311
CaduceusDC16	0.688	0.599	1.287	0.181	0.327
AgentKN	0.690	0.757	1.447	0.065	**0.252**
SimpleAgent	0.699	<u>0.531</u>	1.230	0.204	0.398
Mosa	0.702	0.781	1.483	0.026	0.271
Rubick	0.716	0.715	1.431	0.070	0.282
PonPokoAgent	0.730	0.589	1.320	0.158	0.307
AgentF	0.738	0.679	1.417	0.076	0.301
ShahAgent	0.741	0.554	1.296	0.172	0.342
MetaAgent2013	0.746	0.659	1.405	0.092	0.284
MetaAgent	0.752	0.634	1.386	0.106	0.296
DA($AS(\theta, s)$)	**0.788**	0.627	1.414	0.074	0.314

Figure 3: Comparison of two DA(θ) strategies in an ANAC tournament setting. Here, DA(θ_1) is comparable to the agent configured by Renting et al. [21] and DA($AS(\theta, s)$) represents this work. See Table 9 for an explanation of the measures.

an ANAC-like bilateral tournament with our DA(θ) using the strategy selector, see Table 9. We beat the runner-up agent (MetaAgent) by $\frac{0.788-0.752}{0.752} \cdot 100\% \approx 5.6\%$ (significant at $\alpha = 0.05$ according to a one-tailed t-test p-value of $p = 0.0022$).

Finally, we compare the performances including error bars of DA(θ) with θ_1 and with a portfolio of strategies in a realistic ANAC tournament setup, see Figure 3. Notice that our utility improved with $\frac{0.788-0.742}{0.742} \cdot 100\% \approx 6.2\%$ by using a portfolio instead of a single fixed strategy and that the portfolio approach also improves all other performance measures.

7 CONCLUSIONS AND FUTURE WORK

In previous work [21], automatic algorithm configuration was used to obtain a single best strategy. Here, we have introduced a method to configure and use a portfolio of strategies for negotiation agents, adding a combination of Hydra, AutoFolio, and a procedure to learn opponent behaviour. Our approach is fully automated and represents a significant step beyond the use of single best strategies in automated negotiation. In principle, it can be applied to any negotiation agent with a flexible, parameterised strategy.

We created a portfolio of 4 strategies θ and tested the performance of every strategy on a broad set of negotiation settings. In Table 7, we showed that every configured strategy contributes to the portfolio by specialising on separate sets of negotiation settings. By adding algorithm selection to the Dynamic Agent to exploit differences between settings in a realistic tournament, we increased the performance of Dynamic Agent by 6.2% compared to the single best strategy and won the tournament by a margin of 5.6%. We note that the single best strategy is comparable to the agent configured by Renting et al. [21], indicating that a portfolio-based agent provides another significant boost to negotiation pay-off.

Limitations lie in the required mutual agreement on the norms of how to conduct a negotiation. In this work, a predefined protocol is used that is supported by all used agents. Agents that do not support this protocol cannot participate in the negotiation. Another important limitation is that this method has no safeguards to detect whether the strategy portfolio is still performing well and that we are not being exploited. Finally, due to the train-then-test principle of our method, we still rely on a training set that is reasonably representative of the actual application. Ethical concerns arise in the design of bargaining agents for use in real-life applications. Persons that have more resources to design quality bargaining agents can gain even more resources in the process, leading to more inequality. There are risks of exploitation, unfair play, and deception, due to a lack of explainability and a high level of complexity for laypersons.

In future work, we intend to study the influence of the strategies employed by the Dynamic Agent on the opponent characteristics that we learn during negotiation to improve opponent learning. Secondly, strategy selection could be improved for first encounters with opponents, where currently the single best strategy is selected without regard of the setting characteristics. We intend to investigate strategy selection for bargaining settings through neural networks to relax the reliance on manually designed setting features. Finally, it would be interesting to explore the use of reinforcement learning for training negotiation strategies instead of the algorithm configuration approach that we leveraged here.

REFERENCES

[1] Reyhan Aydoğan, David Festen, Koen V. Hindriks, and Catholijn M. Jonker. 2017. Alternating Offers Protocols for Multilateral Negotiation. In *Modern Approaches to Agent-based Complex Automated Negotiation*, Katsuhide Fujita, Quan Bai, Takayuki Ito, Minjie Zhang, Fenghui Ren, Reyhan Aydoğan, and Rafik Hadfi (Eds.). Springer International Publishing, Cham, 153–167. https://doi.org/10.1007/978-3-319-51563-2_10

[2] T Baarslag. 2014. *What to bid and when to stop.* Ph.D. Dissertation. Delft University of Technology. https://doi.org/10.4233/uuid:3df6e234-a7c1-4dbe-9eb9-baadabc04bca

[3] Tim Baarslag, Reyhan Aydoğan, Koen V. Hindriks, Katsuhide Fujita, Takayuki Ito, and Catholijn M. Jonker. 2015. The Automated Negotiating Agents Competition, 2010–2015. *AI Magazine* 36, 4 (2015), 2010–2014. https://doi.org/10.1609/aimag.v36i4.2609

[4] Tim Baarslag, Katsuhide Fujita, Enrico H. Gerding, Koen Hindriks, Takayuki Ito, Nicholas R. Jennings, Catholijn Jonker, Sarit Kraus, Raz Lin, Valentin Robu, and Colin R. Williams. 2013. Evaluating practical negotiating agents: Results and analysis of the 2011 international competition. *Artificial Intelligence* 198 (2013), 73–103. https://doi.org/10.1016/j.artint.2012.09.004

[5] Tim Baarslag, Koen Hindriks, and Catholijn Jonker. 2011. 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)* 7047 LNAI (2011), 143–158. https://doi.org/10.1007/978-3-642-25044-6_13

[6] Jasper Bakker, Aron Hammond, Daan Bloembergen, and Tim Baarslag. 2019. RLBOA: A Modular Reinforcement Learning Framework for Autonomous Negotiating Agents. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '19)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 260–268. https://www.ifaamas.org/Proceedings/aamas2019/pdfs/p260.pdf

[7] Garett Dworman, Steven O. Kimbrough, and James D. Laing. 1996. Bargaining by artificial agents in two coalition games: a study in genetic programming for electronic commerce. In *Proceedings of the 1st annual conference on genetic programming*. MIT Press, Cambridge, MA, USA, 54–62. https://dl.acm.org/doi/abs/10.5555/1595536.1595544

[8] Torsten Eymann. 2001. Co-Evolution of Bargaining Strategies in a Decentralized Multi-Agent System. In *symposium on negotiation methods for autonomous cooperative systems*. 126–134.

[9] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization (Lecture Notes in Computer Science)*, Carlos A. Coello Coello (Ed.). Springer, Berlin, Heidelberg, 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

[10] Litan Ilany and Ya'akov Gal. 2014. The Simple-Meta Agent. In *Novel insights in agent-based complex automated negotiation*, Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Takayuki Ito, Minjie Zhang, Quan Bai, and Katsuhide Fujita (Eds.). Vol. 535. Springer, 197–200. https://doi.org/10.1007/978-4-431-54758-7

[11] Litan Ilany and Ya'akov Gal. 2016. Algorithm selection in bilateral negotiation. *Autonomous Agents and Multi-Agent Systems* 30, 4 (2016), 697–723. https://doi.org/10.1007/s10458-015-9302-8

[12] M Tawfik Jelassi and Abbas Foroughi. 1989. Negotiation support systems: An overview of design issues and existing software. *Decision Support Systems* 5, 2 (1989), 167–181.

[13] Ryohei Kawata and Katsuhide Fujita. 2019. Meta-Strategy for Multi-Time Negotiation: A Multi-Armed Bandit Approach. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '19)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2048–2050.

[14] Mark Klein and Stephen C. Y. Lu. 1989. Conflict resolution in cooperative design. *Artificial Intelligence in Engineering* 4, 4 (Oct. 1989), 168–180. https://doi.org/10.1016/0954-1810(89)90013-7

[15] Raz Lin, Sarit Kraus, Tim Baarslag, Dmytro Tykhonov, Koen Hindriks, and Catholijn M. Jonker. 2014. Genius: An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence* 30, 1 (2014), 48–70. https://doi.org/10.1111/j.1467-8640.2012.00463.x

[16] Marius Lindauer, Frank Hutter, Holger H. Hoos, and Torsten Schaub. 2017. AutoFolio: An automatically configured algorithm selector. *IJCAI International Joint Conference on Artificial Intelligence* 53 (2017), 5025–5029. https://doi.org/10.1613/jair.4726

[17] Ivan Marsa-Maestre, Mark Klein, Catholijn M. Jonker, and Reyhan Aydoğan. 2014. From problems to protocols: Towards a negotiation handbook. *Decision Support Systems* 60, 1 (2014), 39–54. https://doi.org/10.1016/j.dss.2013.05.019

[18] Noyda Matos, Carles Sierra, and Nick R. Jennings. 1998. Determining successful negotiation strategies: An evolutionary approach. *Proceedings - International Conference on Multi Agent Systems, ICMAS 1998* (1998), 182–189. https://doi.org/10.1109/ICMAS.1998.699048

[19] Martin J. Osborne and Ariel Rubinstein. 1994. *A Course in Game Theory.* (1 ed.). Vol. 1. MIT press. https://doi.org/10.2307/2554642

[20] Howard Raiffa. 1985. *The art and science of negotiation.* Harvard University Press. 384 pages.

[21] Bram M. Renting, Holger H. Hoos, and Catholijn M. Jonker. 2020. Automated Configuration of Negotiation Strategies. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems.* International Foundation for Autonomous Agents and Multiagent Systems, Auckland, 1116–1124. http://arxiv.org/abs/2004.00094

[22] John R. Rice. 1976. The Algorithm Selection Problem. *Advances in Computers* 15, C (1976), 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

[23] W.N. Robinson. 1990. Negotiation behavior during requirement specification. *[1990] Proceedings. 12th International Conference on Software Engineering* (1990), 268–276. https://doi.org/10.1109/ICSE.1990.63633

[24] J. S. Rosenschein. 1986. *Rational interaction: cooperation among intelligent agents.* Ph.D. Dissertation. Stanford University, Stanford, CA, USA. http://www.osti.gov/energycitations/product.biblio.jsp?osti_id=5310977

[25] Ariel Rubinstein. 1982. Perfect Equilibrium in a Bargaining Model. *Econometrica* 50, 1 (1982), 97. https://doi.org/10.2307/1912531

[26] Ayan Sengupta, Yasser Mohammad, and Shinji Nakadai. 2021. An Autonomous Negotiating Agent Framework with Reinforcement Learning based Strategies and Adaptive Strategy Switching Mechanism. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (Richland, SC, 2021-05-03) *(AAMAS '21)*. International Foundation for Autonomous Agents and Multiagent Systems, 1163–1172. https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p1163.pdf

[27] Reid G. Smith. 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Trans. Comput.* C-29, 12 (1980), 1104–1113. https://doi.org/10.1109/TC.1980.1675516

[28] Kate A. Smith-Miles. 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. *Comput. Surveys* 41, 1 (2009), 1–25. https://doi.org/10.1145/1456650.1456656

[29] Katia Sycara. 1988. Resolving Goal Conflicts via Negotiation. *The Seventh National Conference on Artificial Intelligence* (1988), 245–249. http://www.aaai.org/Papers/AAAI/1988/AAAI88-044.pdf

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

[31] Lin Xu, Holger Hoos, and Kevin Leyton-Brown. 2010. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In *Twenty-Fourth AAAI Conference on Artificial Intelligence* (2010-07-03). https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1929

[32] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. 2008. SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research* 32 (July 2008), 565–606. https://doi.org/10.1613/jair.2490