# Automated Configuration and Usage of Strategy Portfolios for Bargaining

**Bram M. Renting**
Leiden University
Delft University of Technology
B.M.Renting@liacs.leidenuniv.nl

**Holger H. Hoos**
Leiden University
University of British Columbia
hh@liacs.nl

**Catholijn M. Jonker**
Delft University of Technology
Leiden University
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 widely 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.

## 1 Introduction

Bargaining or negotiation is a prominent method to decentrally solve mixed-motive problems through reaching 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 Smith [27], Sycara-Cyranski [30], Rosenschein [24], Sycara [29], Jelassi and Foroughi [12], Klein and Lu [14], Robinson [23], 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) [5]) and standardised test-beds (e.g., GENIUS [15]) that support the development of algorithmic negotiation strategies. There are now more than one hundred negotiation strategies freely available that

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

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: 1) we apply automated algorithm configuration techniques to not only create a single negotiation strategy, but a portfolio of complementary negotiation strategies; and 2) 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 [32] 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 [15] test-bed, to support future research; they are categorised and empirically evaluated [4, 2] 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 [18, 8, 7], or reinforcement learning [6, 26], a recent successful approach used a model-based algorithm procedure (Sequential Model-based optimization for general Algorithm Configuration (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]. Note that algorithm selection has been successfully applied to, e.g., SAT-solving [31] and pattern recognition [28]. However, in the field of automated negotation, only a few attempts were made to use algorithm selection methods. 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. 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 [25, 19]. 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 ($s \in S = O \times P$). Protocols, problems and opponents are all available through the GENIUS [15] test-bed (GPL v3), which we use to benchmark our agents.

### 3.1 Bargaining problem

We negotiate over multi-issue (or multi-objective) problems that are defined according to a common standard in automated negotiation [20, 17, 2]. 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 ($\prod_{i \in I} V_i = \Omega$). An outcome is denoted by $\omega \in \Omega$.

Preferences over the outcome space $\Omega$ are expressed through a utility function $u(\omega)$, such that $u : \Omega \to [0, 1]$, where a score 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 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 DA($\theta$) was able to win an ANAC-like tournament by a significant margin. We implemented the same DA($\theta$) with configuration $\theta \in \Theta$. The full configuration space $\Theta$ of DA($\theta$) can be found in Appendix B. There are three types of parameters that influence the behaviour of DA($\theta$): 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 obtained utility $r(\theta, s) \in [0, 1]$ by playing strategy $\theta$ in a negotiation setting $s$. As we work with a set of settings $S$, we define the optimisation metric as the average utility

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

**SMAC.** We use the freely available general-purpose algorithm configurator SMAC [9] to automatically configure DA($\theta$), following the successful implementation of Renting et al. [21]. A pseudocode version of SMAC can be found in Appendix D, 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 attempts to model differences between negotiation settings through features that capture information on setting complexity. We describe these features in Section 3.4.

### 3.4 Negotiation setting features

To perform algorithm selection, we need some features that describe 1) the characteristics of the negotiation problem that we currently face, and 2) the characteristics of the current opponent. Then, given these features, algorithm selection essentially becomes an classification problem, where we map the current features to a selected negotiation algorithm from our portfolio. We also use these features to guide the model-based optimisation procedure of SMAC.

Renting et al. [21] created a set of features to describe a negotiation setting, which was partly based on previous work by Ilany and Gal [11] and Baarslag et al. [3]. We adopt this set of features consisting of bargaining problem features ($X_p$) and opponent features ($X_o$). An overview of the bargaining setting features we use is given in Appendix A. 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 negotiation settings as opponent features for a total of 8 opponent features.

### 3.5 Problem definition

**Strategy portfolio creation.** We have an agent with a dynamic strategy DA($\theta$) based on configuration space $\Theta$. Can we create a portfolio of configurations $\boldsymbol{\theta} \subset \Theta$ 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($\theta$), and a portfolio of configurations $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_n\}$, where $\theta_1$ is the single best-performing configuration (Equation 5). Can we apply an algorithm selection method $\theta_s = AS(\boldsymbol{\theta}, s)$ that selects a configuration $\theta_s$ 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 4) $R(OR(\boldsymbol{\theta}, s), S_{test})$ as closely as possible.

# 4 Portfolio of bargaining strategies

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 instances for which all the negotiation strategies fail to achieve a 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($\theta$). In our method we will therefore enforce that every strategy must add value to the portfolio:

$$\forall \theta \in \boldsymbol{\theta},\ \exists s \in S,\ \forall \theta' \in (\boldsymbol{\theta} \setminus \theta)\ :\ r(\theta, s) > r(\theta', s) \tag{2}$$

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 [32], 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 Appendix E, modified for this work.

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 Appendix E, the modified performance metric is computed by "*GetModifiedPerformanceMetric*" and formally defined as

$$r_k(\theta, s) = \max\left(r(\theta, s),\ r(AS(\boldsymbol{\theta}, s), s)\right). \tag{3}$$

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 using 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 $\theta_1$ 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 important 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 essentially a classification problem, which we can train on examples generated from our training set. Subsequently, we hope the learned function can 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 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 constructs 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 setting 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 Performance measure

We measure the algorithm selector's performance as a normalised value between a baseline and the oracle selector (Equation 4) on the test set of negotiation settings. The oracle selector always makes the perfect choice for every negotiation setting and is an upper bound on the performance of an 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 5). We refer to this strategy as $\theta_1$, 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.

$$OR(\boldsymbol{\theta}, s) \in \arg\max_{\theta \in \boldsymbol{\theta}} r(\theta, s) \tag{4}$$

$$\theta_1 \in \arg\max_{\theta \in \boldsymbol{\theta}} R(\theta, S) \tag{5}$$

# 6 Empirical evaluation

## 6.1 Method

The first configurator run with the default performance metric results in the single best strategy $\theta_1$ on the training set of negotiation settings. We aim to complement the portfolio with an additional three strategies, so we iterate through HYDRA until $k = 4$. This also allows us to analyse the performance of portfolios of size 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 reduce stochastic influence. Finally, the portfolio and the performance data was used along with the setting features to configure an algorithm selector using AutoFolio.

Table 1: 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.

| | $R(\theta, \cdot)$ | | **Best performing on $S_{test}$ by ratio** | | | | |
|---|---|---|---|---|---|---|---|
| $\theta$ | $S$ | $S_{test}$ | Single best | In top 2 | In top 3 | In top 4 | Sum |
| $\theta_1$ | 0.815 | 0.742 | 0.281 | 0.100 | 0.016 | 0.123 | 0.520 |
| $\theta_2$ | 0.788 | 0.734 | 0.167 | 0.022 | 0.020 | 0.123 | 0.333 |
| $\theta_3$ | 0.789 | 0.754 | 0.154 | 0.065 | 0.031 | 0.123 | 0.373 |
| $\theta_4$ | 0.773 | 0.721 | 0.118 | 0.058 | 0.033 | 0.123 | 0.333 |

#### 6.1.1 Input.

Specifics on the training and test set can be found in Appendix C. The bargaining problem features were calculated in advance, as described in Appendix A. 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 opponents utility function ($u_o$) to calculate the features in Appendix A to reduce estimation noise.

#### 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 GBs of RAM. We ran AutoFolio on a single dual core processor on the same computing cluster, assigned it 4 gigabytes of RAM, and provided it with a budget of 0.5 hours.

### 6.2 Results

#### 6.2.1 Quality of the portfolio.

We tested the quality of the portfolio by testing 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 1. We included ratios that indicate how often a strategy is part of the set of best strategies per setting ("Sum" in Table 1). As a final quality check, the performance of the oracle selector (Equation 4) is evaluated for varying sizes of the portfolio. We present the results in Table 2.

Table 1 shows the results per strategy in the portfolio in the form of an individual performance over a set of settings $R(\theta, S)$. It is evident that $\theta_1$ 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 the first problem statement in Section 3.5.

Finally, Table 2 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 amount of iterations increase, indicating that HYDRA fills the largest "weaknesses" in the portfolio first.

#### 6.2.2 Performance of the algorithm selector.

Table 2 shows that there is potential in the portfolio to improve utility of DA($\theta$) 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 $\theta_1$. We now replace the oracle selector with the actual selector and test its performance in two ways.

**Performance with 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 setting $S_{test}$. The opponent features are gathered by running 10 negotiation sessions with configuration $\theta_1$ on the test set.

Table 2: 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.

| | $R(OR, \cdot)$ | | $R(AS, \cdot)$ | |
|---|---|---|---|---|
| $\boldsymbol{\theta}$ | $S$ | $S_{test}$ | $S$ | $S_{test}$ |
| $\{\theta_1\}$ | 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 |

We trained and tested multiple algorithm selectors on different portfolio sizes by extending the portfolio, starting with the single best strategy $\theta_1$. We report the performance in Table 2. For the oracle selector $OR$ the performance of DA($\theta$) increases with the size of the portfolio. However, the performance of the algorithm selector $AS$ plateaus on $S$ 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 encounters. Up to this point, we assumed the opponents to always be known in advance, which is not realistic. We 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 solve this, we negotiate once against all of them and then "wiping our memory", giving every opponent a head start.

The question arises 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 $\theta_1$. Opponent features are influenced by the strategy that is selected by DA($\theta$), so we simplify the feature extraction process and only gather features when strategy $\theta_1$ is selected. This aligns with the decision to select $\theta_1$ at first opponent encounters. The Coefficient of Variance (CoV) of an opponent feature (Section 3.4) needs at least two samples to be meaningful, so we set a second condition to select strategy $\theta_1$ for the first two encounters with an opponent to "sample" the opponent.

To obtain the results, we iterate randomly through the test settings $S_{test}$ and use DA($AS(\boldsymbol{\theta}, s)$) with $\boldsymbol{\theta} = \{\theta_1, \theta_2, \theta_3\}$ to negotiate, following the procedure as described. 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($\theta$). This procedure is repeated 10 times to reduce variance for a total of 38 080 negotiations. The results averaged per agent show that we are capable of winning an ANAC-like tournament with our DA($\theta$) using the strategy selector, see Table 3. We beat the runner-up agent (MetaAgent) by $\frac{0.788-0.752}{0.752} \cdot 100\% \approx 5.6\%$ (one-tailed t-test $p < 0.0022$).

Finally, we compare the performances of DA($\theta$) with $\theta_1$ and with a portfolio of strategies in a realistic ANAC tournament setup, see Figure 1. 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], automated 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. It requires only a negotiation agent with a flexible, parameterised strategy.

We created a portfolio of 4 strategies $\boldsymbol{\theta}$ and tested the performance of every strategy on a broad set of negotiation settings. In Table 1, 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

Table 3: ANAC tournament results using DA($AS(\boldsymbol{\theta}, s)$) 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** | **Agreement ratio** |
|---|---|---|---|---|---|---|
| Imitator | <u>0.446</u> | **0.901** | 1.347 | 0.091 | <u>0.428</u> | 0.953 |
| GeneKing | 0.612 | 0.783 | 1.396 | 0.065 | 0.378 | **0.994** |
| Mamenchis | 0.636 | 0.863 | **1.498** | **0.016** | 0.272 | 0.993 |
| MadAgent | 0.669 | 0.536 | <u>1.204</u> | <u>0.232</u> | 0.383 | <u>0.768</u> |
| AgentKN | 0.690 | 0.757 | 1.447 | 0.065 | **0.252** | 0.934 |
| SimpleAgent | 0.699 | <u>0.531</u> | 1.230 | 0.204 | 0.398 | 0.805 |
| AgentF | 0.738 | 0.679 | 1.417 | 0.076 | 0.301 | 0.941 |
| ShahAgent | 0.741 | 0.554 | 1.296 | 0.172 | 0.342 | 0.829 |
| MetaAgent2013 | 0.746 | 0.659 | 1.405 | 0.092 | 0.284 | 0.917 |
| MetaAgent | 0.752 | 0.634 | 1.386 | 0.106 | 0.296 | 0.894 |
| DA($AS(\boldsymbol{\theta}, s)$) | **0.788** | 0.627 | 1.414 | 0.074 | 0.314 | 0.923 |

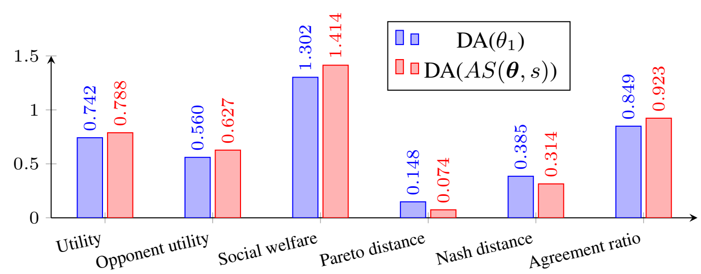


Figure 1: Comparison of two DA($\theta$) strategies in an ANAC tournament setting. Here, DA($\theta_1$) is comparable to the agent configured by Renting et al. [21] and DA($AS(\boldsymbol{\theta}, s)$) represents this work. See Table 3 for an explanation of the performance measures.

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 that check whether the currently trained agent is still performing well, and not being exploited. Finally, due to the train-then-test principle of our method, we still rely on a training set that is a decent representation of the actual application. Ethical concerns in the design of bargaining agents lie in the application of these agents in real life. Persons that have more resources to design quality bargaining strategies can gain even more resources in the process, leading to more inequality. There are risk of exploitation, unfair play, and deception due to a lack of explainability and a high level of complexity for the layman.

In future work, we intend to study the influence of Dynamic Agent's strategies 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 want to investigate strategy selection for bargaining settings through neural networks to relax the reliance on manually designed 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.

## Acknowledgement

The authors would like to thank Thomas Moerland for his help in proof-reading this paper.

This research was (partly) funded by the Hybrid Intelligence Center, a 10-year programme funded the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research, grant number 024.004.022 and by EU H2020 ICT48 project"Humane AI Net" under contract # 952026.

This research was also partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

## References

[1] Reyhan Aydoğan, David Festen, Koen V. Hindriks, and Catholijn M. Jonker. Alternating offers protocols for multilateral negotiation. In *Studies in Computational Intelligence*, volume 674, pages 153–167. Springer, 2017. ISBN 978-3-319-51563-2. doi: 10.1007/978-3-319-51563-2_10. URL `https://link.springer.com/chapter/10.1007%2F978-3-319-51563-2_10`.

[2] T Baarslag. *What to bid and when to stop*. PhD thesis, Delft University of Technology, 2014. URL `http://dx.doi.org/10.4233/uuid:3df6e234-a7c1-4dbe-9eb9-baadabc04bca`.

[3] Tim Baarslag, Koen Hindriks, and Catholijn 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)*, 7047 LNAI:143–158, 2011. ISSN 03029743. doi: 10.1007/978-3-642-25044-6_13.

[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. Evaluating practical negotiating agents: Results and analysis of the 2011 international competition. *Artificial Intelligence*, 198:73–103, 2013. ISSN 00043702. doi: 10.1016/j.artint.2012.09.004. URL `http://dx.doi.org/10.1016/j.artint.2012.09.004`.

[5] Tim Baarslag, Reyhan Aydoğan, Koen V. Hindriks, Katsuhide Fujita, Takayuki Ito, and Catholijn M. Jonker. The Automated Negotiating Agents Competition, 2010–2015. *AI Magazine*, 36(4):2010–2014, 2015. ISSN 0738-4602. doi: 10.1609/aimag.v36i4.2609. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/115-118`.

[6] Jasper Bakker, Aron Hammond, Daan Bloembergen, and Tim Baarslag. 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, pages 260–268. International Foundation for Autonomous Agents and Multiagent Systems, 2019. ISBN 978-1-4503-6309-9. URL `https://www.ifaamas.org/Proceedings/aamas2019/pdfs/p260.pdf`.

[7] Garett Dworman, Steven O. Kimbrough, and James 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*, page 54–62, 1996. URL `http://portal.acm.org/citation.cfm?id=1595536.1595544`.

[8] T Eymann. Co-evolution of bargaining strategies in a decentralized multi-agent system. *AAAI Fall 2001 Symposium on Negotiation Methods for Autonomous Cooperative Systems*, pages 126–134, 2001. URL `http://www.aaai.org/Papers/Symposia/Fall/2001/FS-01-03/FS01-03-016.pdf`.

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

[10] Litan Ilany and Ya'akov Gal. The Simple-Meta Agent. In Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Takayuki Ito, Minjie Zhang, Quan Bai, and Katsuhide Fujita, editors, *Novel insights in agent-based complex automated negotiation*, volume 535, pages 197–200. Springer, 2014. ISBN 978-4-431-54757-0. doi: 10.1007/978-4-431-54758-7.

[11] Litan Ilany and Ya'akov Gal. Algorithm selection in bilateral negotiation. *Autonomous Agents and Multi-Agent Systems*, 30(4):697–723, 2016. ISSN 15737454. doi: 10.1007/s10458-015-9302-8.

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

[13] Ryohei Kawata and Katsuhide Fujita. 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, pages 2048–2050, Richland, SC, May 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-6309-9.

[14] Mark Klein and Stephen C. Y. Lu. Conflict resolution in cooperative design. *Artificial Intelligence in Engineering*, 4(4):168–180, October 1989. ISSN 0954-1810. doi: 10.1016/0954-1810(89)90013-7. URL `https://www.sciencedirect.com/science/article/pii/0954181089900137`.

[15] Raz Lin, Sarit Kraus, Tim Baarslag, Dmytro Tykhonov, Koen Hindriks, and Catholijn M. Jonker. Genius: An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence*, 30(1):48–70, 2014. ISSN 08247935. doi: 10.1111/j.1467-8640.2012.00463.x. URL `http://doi.wiley.com/10.1111/j.1467-8640.2012.00463.x`.

[16] Marius Lindauer, Frank Hutter, Holger H. Hoos, and Torsten Schaub. AutoFolio: An automatically configured algorithm selector. *IJCAI International Joint Conference on Artificial Intelligence*, 53:5025–5029, 2017. ISSN 10450823. doi: 10.1613/jair.4726.

[17] Ivan Marsa-Maestre, Mark Klein, Catholijn M. Jonker, and Reyhan Aydoğan. From problems to protocols: Towards a negotiation handbook. *Decision Support Systems*, 60(1):39–54, 2014. ISSN 01679236. doi: 10.1016/j.dss.2013.05.019.

[18] Noyda Matos, Carles Sierra, and Nick R. Jennings. Determining successful negotiation strategies: An evolutionary approach. *Proceedings - International Conference on Multi Agent Systems, ICMAS 1998*, pages 182–189, 1998. ISSN 0254-1319. doi: 10.1109/ICMAS.1998.699048.

[19] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.*, volume 1. MIT press, 1 edition, 1994. ISBN 0262650401. doi: 10.2307/2554642. URL `http://www.jstor.org/stable/10.2307/2554642?origin=crossref`.

[20] Howard Raiffa. *The art and science of negotiation*. Harvard University Press, 1985. ISBN 9780674048133.

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

[22] John R. Rice. The Algorithm Selection Problem. *Advances in Computers*, 15(C):65–118, 1976. ISSN 00652458. doi: 10.1016/S0065-2458(08)60520-3.

[23] W.N. Robinson. Negotiation behavior during requirement specification. *[1990] Proceedings. 12th International Conference on Software Engineering*, pages 268–276, 1990. ISSN 02705257. doi: 10.1109/ICSE.1990.63633. URL `http://ieeexplore.ieee.org/document/63633/`.

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

[25] Ariel Rubinstein. Perfect Equilibrium in a Bargaining Model. *Econometrica*, 50(1):97, 1982. ISSN 00129682. doi: 10.2307/1912531. URL `http://www.jstor.org/stable/1912531`.

[26] Ayan Sengupta, Yasser Mohammad, and Shinji Nakadai. 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*, AAMAS '21, pages 1163–1172. International Foundation for Autonomous Agents and Multiagent Systems, 2021. ISBN 978-1-4503-8307-3. URL `https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p1163.pdf`.

[27] Reid G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980. ISSN 00189340. doi: 10.1109/TC.1980.1675516. URL `http://www.csupomona.edu/~ftang/courses/CS599-DI/notes/papers/contractnetprotocol.pdf`.

[28] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):1–25, 2009. ISSN 03600300. doi: 10.1145/1456650.1456656. URL `http://portal.acm.org/citation.cfm?doid=1456650.1456656`.

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

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

[31] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, July 2008. ISSN 1076-9757. doi: 10.1613/jair.2490. URL `https://www.jair.org/index.php/jair/article/view/10556`.

[32] Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1929`.

## A Feature representations of bargaining setting

Table 4: Bargaining problem features ($X_p$) [11]. The utility functions of the problems that are used in the paper are linear additive. An issue weight is a linear weight that is associated with an issue. The scores of every issue are multiplied by this weight and then summed to obtain the final utility. The sum of the issue weights is 1.

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

Table 5: Opponent features ($X_o$) [21]. $x_o^-$ is the lowest offer by the opponent in their predicted utility. $\omega^+$/$\omega^-$ is our best/worst possible outcome. $\bar{x}$ is the (fictional) average offer by the opponent in their predicted utility. $\omega_{agree}$ is the agreement.

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

## B Configuration space and configured portfolio of the Dynamic Agent

Table 6: Configuration space of DA($\theta$) as set by Renting et al. [21]

| **Description** | **Symbol** | **Domain** | **Purpose** |
|---|---|---|---|
| Scale factor | $\alpha$ | $[1, 1.1]$ | Accepting |
| Utility gap | $\beta$ | $(0, 0.2]$ | Accepting |
| Accepting time | $t_{acc}$ | $[0.9, 1]$ | Accepting |
| Lower boundary | $\gamma$ | $\{MAX^W, AVG^W\}$ | Accepting |
| Trade-off factor | $\delta$ | $[0, 1]$ | 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 |

Table 7: Final configurations in the portfolio. These are the final parameter settings that make up the different bargaining strategies in the portfolio.

| | **Accepting** | | | | **Bidding** | | | **Searching** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\theta$ | $\alpha$ | $\beta$ | $t_{acc}$ | $\gamma$ | $n_{fit}$ | $\delta$ | $e$ | $N_{pop}$ | $N_{tour}$ | $E$ | $R_c$ | $R_m$ | $R_e$ |
| $\theta_1$ | 1.038 | 0.03201 | 0.942 | $AVG^W$ | 3 | 0.927 | 0.00199 | 262 | 6 | 4 | 0.290 | 0.140 | 0.085 |
| $\theta_2$ | 1.001 | 0.00166 | 0.935 | $AVG^W$ | 3 | 0.998 | 0.06232 | 94 | 2 | 5 | 0.168 | 0.002 | 0.108 |
| $\theta_3$ | 1.007 | 0.01970 | 0.912 | $AVG^W$ | 4 | 0.917 | 0.01093 | 305 | 10 | 1 | 0.107 | 0.063 | 0.184 |
| $\theta_4$ | 1.056 | 0.00003 | 0.900 | $MAX^W$ | 5 | 0.997 | 0.02090 | 139 | 10 | 4 | 0.463 | 0.176 | 0.101 |

## C Training and testing set of opponents and problems

This appendix provides an overview of the training and testing set of both opponents and bargaining problems that is used throughout this paper. A single training setting requires an agent as opponent and problem from the train set, the same is true for a test setting.

The set of agents is provided in Table 8. We used a total of 36 agents from the ANAC. The set of ANAC agents is split up in 20 training agents and 16 test agents. The set of problems is provided in Table 9. 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 Appendix A and their discount factor and reservation utility are removed. The set is split up in 56 training problems and 28 test problems.

The total amount of training settings:

$$|S| = |O| * |P| = 20 * 56 = 1120 \tag{6}$$

The total amount of test settings:

$$|S_{test}| = |O_{test}| * |P_{test}| = 16 * 28 = 448 \tag{7}$$

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

Table 8: Overview of agent set used in this work. The last column indicates in which year the agent participated in ANAC.

| 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 9: Overview of bargaining problem set used in this work

## D SMAC

Algorithm 1 forms the main body of SMAC [9]. The sub-procedure *Intensify* is described in Algorithm 2. We used the SMAC3 implementation of SMAC, which is released under a BSD 3-Clause License (`https://github.com/automl/SMAC3`).

**Algorithm 1** Parallel Sequential Model-Based Optimisation [9] (SMBO)

| | | |
|---|---|---|
| **Input** | $\Theta$ | Configuration space |
| | $S$ | Negotiation settings |
| | $O$ | 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 |
| | $\boldsymbol{\theta}_{new}$ | List of promising configurations |
| **Output** | $\theta_{inc}$ | Optimised parameter configuration |

$[R_i, \theta_{inc}] \leftarrow Initialise(\Theta, S)$
**loop until** $GetTime() > t_{opt}$
    $R_{full} \leftarrow ReadParallelRunhistories()$
    $\mathcal{M} \leftarrow FitModel(R_{full})$
    $\boldsymbol{\theta}_{new} \leftarrow SelectConfigurations(\mathcal{M}, \theta_{inc}, \Theta)$
    $[R_i, \theta_{inc}] \leftarrow Intensify(\boldsymbol{\theta}_{new}, \theta_{inc}, R_i, S, O)$
**return** $\theta_{inc}$

**Algorithm 2** $Intensify(\boldsymbol{\theta}_{new}, \theta_{inc}, R, S, O)$ [9]

| | | |
|---|---|---|
| **Input** | $\boldsymbol{\theta}_{new}$ | List of promising configurations |
| | $\theta_{inc}$ | Incumbent configuration (current best) |
| | $R$ | Runhistory |
| | $S$ | Negotiation settings |
| | $O$ | Performance metric |
| | $t_{int}$ | Time budget for intensify procedure |
| **Variables** | $\theta_{new}$ | Challenging configuration |
| **Output** | $R$ | Runhistory |
| | $\theta_{inc}$ | Incumbent configuration (current best) |

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

## E HYDRA

**Algorithm 3** HYDRA [32]

| | | |
|---|---|---|
| **Input** | $\Theta$ | Configuration space |
| | $S$ | Training set of negotiation settings |
| | $o$ | Performance metric |
| **Variables** | $\theta_k$ | Configuration |
| | $\boldsymbol{\theta}$ | Portfolio of configurations |
| | $r_k$ | Modified performance metric |
| **Output** | $\boldsymbol{\theta}$ | Portfolio of configurations |
| | $AS$ | Algorithm selector |

$\boldsymbol{\theta} \leftarrow \emptyset$; $r_k \leftarrow o$
**for** $k = 1$; Until portfolio size is reached; $k = k + 1$ **do**
    $\theta_k \leftarrow SMBO(\Theta, S, r_k)$ ▷ Appendix D
    $TestPerformance(S, \theta_k)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} \cup \{\theta_k\}$
    $AS \leftarrow FitAlgorithmSelector(\boldsymbol{\theta}, S)$
    $r_k \leftarrow GetModifiedPerformanceMetric(o, AS)$
**return** $AS$, $\boldsymbol{\theta}$