Learning in automated negotiation

Renting, B.M.

4

Configuration of Strategy Portfolios

This chapter has been published at the Cooperative AI workshop at NeurIPS 2021 [122] and AAMAS 2022 [123].

4.1 INTRODUCTION

The strategies of negotiation agents almost always remain monolithic, i.e. single strategy with fixed behaviour for every setting, the exceptions we found are, e.g., [76, 137]. It has been observed that no single strategy is optimal for all negotiation instances [76, 99] and that the success of a negotiator also depends on the strategy of the opponent [10]. In Chapter 3, we successfully showed that we can use automated configuration techniques to optimise negotiation strategies. However, the obtained strategy is also fixed in that it does not adapt to various opponent types or negotiation scenarios. A good way to further improve pay-off would be to select from a portfolio of strategies based on the negotiation game. This introduces the problem of algorithm selection [127] into negotiation. An early attempt to apply algorithm selection in automated negotiation was made by Ilany and Gal [77, 76], but they only selected a strategy based on the negotiation scenario, without considering the opponent, which we know to be an essential factor [10]. Furthermore, they relied on a portfolio of existing strategies to select from, which potentially limits robustness.

Our contributions in this chapter 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 scenario characteristics during a simulated Automated Negotiating Agents Competition (ANAC) tournament. Our method uses the approach from Chapter 3 to automatically configure negotiation strategies, which we extend by implementing HYDRA [162] for portfolio construction and AutoFolio [100] to create a portfolio selector. Empirical results on a variety of negotiation instances show that our method beats the runner-up agent by a (comfortable) margin of 5.6%.

4.2 RELATED WORK

Thanks to ANAC, new negotiation strategies are developed every year and collected in the General Environment for Negotiation with Intelligent multi-purpose Usage Simulation (GENIUS) test-bed [99], to support future research; they are categorised and empirically evaluated [10, 8] to provide a basis for new strategies.

As there is no single best strategy for all negotiation instances [76, 99], we should be able to improve pay-off by exploiting differences in instances by selecting different strategies per negotiation instance. We see this as a variation of the algorithm selection problem [127]. While algorithm selection methods have been successfully applied to other problems, only few attempts have been made to apply them in the area of automated negotiation. Ilany and Gal [77, 76] and Güneş et al. [64] used a set of past ANAC strategies and predicted which strategy would perform best on a given negotiation instance; 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 [84] 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 scenario while repeating precisely the same negotiation setting 100 times. Unfortunately,

this strategy does not generalise to unseen negotiation instances. Sengupta et al. [137] trained both a set of strategies and a selection mechanism. Strategy selection on a more fine-grained level by selecting modular components of a negotiation strategy using reinforcement learning has also been attempted [18].

4.3 PRELIMINARIES

This chapter requires background knowledge in automated negotiation (Section 2.1), algorithm configuration (Section 2.2), and algorithm selection (Section 2.3).

We continue the work of the previous chapter and extend it with portfolio-building methods and algorithm selection techniques. The dynamic agent $DA(\theta)$ with parameter configuration space Θ from Section 3.2.1 is used as a basis. We also use the same scenario features and opponent features as described in Section 3.4. A deadline of 60 seconds is used for the Alternating Offers Protocol (AOP) [132] in this chapter, normalised to $t \in [0,1]$, after which negotiation is aborted without agreement. The performance metric $m(\theta, p)$ of a configuration θ on negotiation instance p is the obtained utility, where the performance on a set of negotiation instances is:

$$M(\theta, \mathcal{P}) = \frac{1}{|\mathcal{P}|} \cdot \sum_{p \in \mathcal{P}} m(\theta, p), \tag{4.1}$$

4.3.1 PROBLEM DEFINITION

Note that in this chapter, we consider algorithm selection to be performed on portfolios of parameter configurations of the same algorithm, thus replacing ψ in Section 2.3 with θ.

Strategy portfolio creation. We have an agent with a dynamic strategy $DA(\theta)$ based on configuration space Θ. Can we create a portfolio of configurations $\boldsymbol{\theta} \subset \Theta$ using a training set of negotiation instances $\mathcal{P}$ consisting of configurations that outperform each other on specific subsets of a test set of negotiation instances $\mathcal{P}'_{test} \subset \mathcal{P}_{test}$ 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 θ_1 is the single best-performing configuration (Equation 4.3). Can we apply an algorithm selection method $\theta_p = AS(\boldsymbol{\theta}, p)$ that selects a configuration θ_p from $\boldsymbol{\theta}$ based on negotiation setting p, such that $M(AS(\boldsymbol{\theta}, p), \mathcal{P}_{test}) > M(\theta_1, \mathcal{P}_{test})$. The real goal here is to let $M(AS(\boldsymbol{\theta}, p), \mathcal{P}_{test})$ approach the performance of the oracle selector (Equation 4.2) $M(OR(\boldsymbol{\theta}, p), \mathcal{P}_{test})$ as closely as possible.

$$OR(\boldsymbol{\theta}, p) \in \arg\max_{\theta \in \boldsymbol{\theta}} m(\theta, p) \tag{4.2}$$

$$\theta_1 \in \arg\max_{\theta \in \boldsymbol{\theta}} M(\theta, \mathcal{P}) \tag{4.3}$$

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 [76]. 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.

4.4.1 PORTFOLIO CREATION

We aim to expand upon the work of Chapter 3, by not only automatically configuring a single negotiation strategy, but by building a portfolio of complementary strategies to better exploit differences between negotiation instances. 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:

$$\forall \theta \in \boldsymbol{\theta},\ \exists p \in \mathcal{P},\ \forall \theta' \in (\boldsymbol{\theta} \setminus \theta)\ :\ m(\theta, p) > m(\theta', p) \tag{4.4}$$

The portfolio can be viewed as a set of strategies that each specialise on a region within the negotiation instance 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 instances 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 [162], removing the requirement of human input in feature space separation.

4.4.2 HYDRA

HYDRA automatically generates a portfolio given only a parameterised strategy (Section 3.2.1) and a set of negotiation instances with features (Section 3.4) while

Algorithm 4.1 HYDRA [162]

Input	Θ	Configuration space
	$\mathcal{P}$	Training set of negotiation instances
	m	Performance metric
Variables	θ_k	Configuration
	$\boldsymbol{\theta}$	Portfolio of configurations
	m_k	Modified performance metric
Output	$\boldsymbol{\theta}$	Portfolio of configurations
	AS	Algorithm selector

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

using an algorithm configurator and an algorithm selector (Section 4.5). We provide a pseudo-code description of HYDRA in Algorithm 4.1, modified for this work.

The main idea of HYDRA is to perform multiple configurator runs on an identical set of training instances while only modifying the performance metric. Due to the modifications to the metric, the configurator produces different strategies. In Algorithm 4.1, the modified performance metric is computed by "*GetModifiedPerformanceMetric*" and formally defined as:

$$m_k(\theta, p) = \max\{m(\theta, p),\ m(AS(\boldsymbol{\theta}, p), p)\}. \tag{4.5}$$

The modified performance is the better of the performance of the strategy that is assessed and the performance of the strategy that the algorithm selector selects. 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.

4.5 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 scenario 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 [127] and is illustrated in Figure 4.1, 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 negotiation scenarios and unknown opponents in the test set, allowing us to select the most suitable strategy from our portfolio.

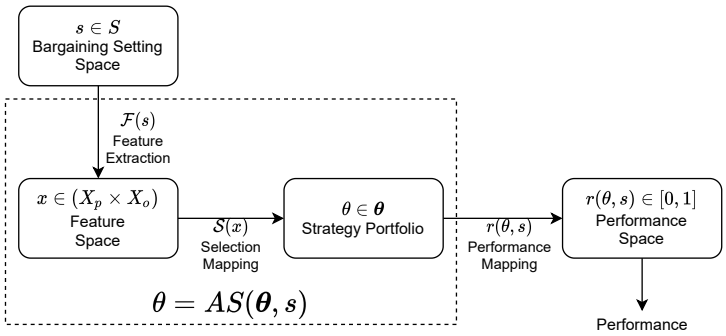

Figure 4.1: Algorithm selection schematics [127], modified for this work.

4

Ilany and Gal [76] 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 [100], leveraging the power of a broad range of algorithm selection methods and removing human bias.

4.5.1 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 Sequential Model-based optimization for general Algorithm Configuration (SMAC) to determine both the selection method to use and the settings of its hyperparameters. The data AutoFolio requires as input is the performance $m(\theta, p)$ of every strategy ($\theta \in \boldsymbol{\theta}$) on every setting ($p \in \mathcal{P}$) in the training set and a set of features. Its goal is to select the best-performing strategy for every negotiation setting.

4.5.2 CROSS VALIDATION.

AutoFolio uses 10-fold cross-validation during optimisation to avoid overfitting by dividing the negotiation instances in the training set into 10 subsets and leaving one subset out for performance testing. However, due to the nature of a negotiation instance being a combination of an opponent and a negotiation scenario, this leads to overfitting of the algorithm selector. The training set of negotiation instances is the Cartesian product of the training set of opponents and scenarios, 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 scenarios that build the negotiation instances. The set of opponents and the set of scenarios are each split into 4 subsets, such that we obtain a total of $4 \cdot 4 = 16$ folds. When selecting a fold ($|\mathcal{P}_{fold}| = \frac{1}{16} \cdot |\mathcal{P}|$), we must eliminate the part of the remaining training set ($|\mathcal{P}_{elim}| = \frac{6}{16} \cdot |\mathcal{P}|$) that overlaps with the fold based on opponents and scenarios and use the remaining

instances ($|\mathcal{P}_{fit}| = \frac{9}{16} \cdot |\mathcal{P}|$) 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/scenarios.

4.5.3 PERFORMANCE BASELINES

The oracle selector (Equation 4.2) 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 4.3). 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.

4.6 EMPIRICAL EVALUATION

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

4.6.1 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 4.6. 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 were used along with the setting features to configure an algorithm selector using AutoFolio.

INPUT

An overview of the opponents that are used in this work can be found in Table 4.1. The test set of opponents $\mathcal{I}_{opp,test}$ 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 $\mathcal{P}$ for the HYDRA procedure (Algorithm 4.1). Two additional agents are added to the test set in order to compare our work to the work of Ilany and Gal [76], which adopted a similar portfolio selection method. 36 agents from the ANAC are used, split up into 20 training agents and 16 test agents.

The set of scenarios is provided in Table 4.2. A total of 42 scenarios is used, of which both sides can be played by our agent, resulting in 84 playable scenarios. The set of negotiation scenarios is selected based on diversity using the features

Table 4.1: Overview of opponent set used in this work. The last column indicates in which year the opponent participated in ANAC.

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		

as described in Section 3.4, and their discount factor and reservation utility are removed. The set is split up into 56 training scenarios and 28 test scenarios. The training set is of size $|\mathcal{P}| = 20 \times 56 = 1120$ and the test set is of size $|\mathcal{P}_{test}| = 16 \times 28 = 448$.

The negotiation scenario 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.

HARDWARE & BUDGET

We followed Renting et al. [124] 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 not to 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.

Table 4.2: Overview of the negotiation scenarios sets used in this work. These scenarios are part of the GENIUS package.

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 4.3: Final configurations in the portfolio. These are the final parameter settings that make up the different negotiation strategies in the portfolio.

	Accepting				**Bidding**			**Searching**					
θ	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 4.4: Individual configuration performance on $\mathcal{P}$ and $\mathcal{P}_{test}$. The two left 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.

	$M(\theta,\cdot)$		**Best performing on $\mathcal{P}_{test}$ by ratio**				
θ	$\mathcal{P}$	$\mathcal{P}_{test}$	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

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 4.3). We use both when faced with a new negotiation setting for which we want to select a configuration.

4.6.2 RESULTS

We now present the results using a test set of negotiation instances $\mathcal{P}_{test}$. More specifically, we investigated two aspects:

1. the quality of the portfolio;

2. the performance of the algorithm selector.

QUALITY OF THE PORTFOLIO

We assessed the quality of the portfolio by measuring the performance (Equation 4.1) of every configuration in the portfolio on the training and testing sets of negotiation settings. The results can be found in Table 4.4. We included ratios that indicate how often a strategy is part of the set of best strategies per setting ("Sum" in Table 4.4). As a final quality check, the performance of the oracle selector (Equation 4.2) is evaluated for varying sizes of the portfolio. We present the results in Table 4.5.

Table 4.4 shows the results per strategy in the portfolio in the form of individual performance over a set of settings $M(\theta,\mathcal{P})$. It is evident that θ_1 is the single best strategy over the full training set $\mathcal{P}$. Furthermore, as every strategy is at least once the single best on individual settings (single best ratio > 0), we can conclude that

Table 4.5: Algorithm selector performance compared to oracle performance. The two columns on the left 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.

	$M(OR(\boldsymbol{\theta}, p), \cdot)$		$M(AS(\boldsymbol{\theta}, p), \cdot)$	
$\boldsymbol{\theta}$	$\mathcal{P}$	$\mathcal{P}_{test}$	$\mathcal{P}$	$\mathcal{P}_{test}$
$\{\theta_1\}$	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

every strategy contributes to the portfolio, thus satisfying our requirement from Section 4.3.1.

Finally, Table 4.5 shows us that, at every iteration of HYDRA, the oracle performance of the portfolio increases on both $\mathcal{P}$ and $\mathcal{P}_{test}$. The improvement decreases on $\mathcal{P}$ as the number of iterations increases, indicating that HYDRA fills the largest "weaknesses" in the portfolio first.

PERFORMANCE OF THE ALGORITHM SELECTOR

Table 4.5 shows that there is potential in the portfolio to improve the 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 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 setting $\mathcal{P}_{test}$. 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 4.5. For the oracle selector, the performance of $DA(\theta)$ increases with the size of the portfolio. However, the performance increase plateaus on $\mathcal{P}_{test}$ 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 scenario 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 now simulate a realistic negotiation tournament where this problem occurs. The agents in $\mathcal{P}_{test}$ 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.

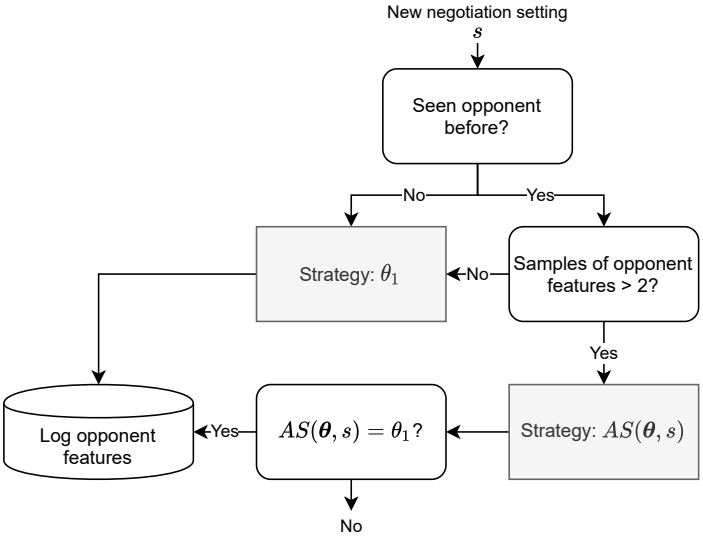

Figure 4.2: Realistic strategy selection of $DA(AS(\boldsymbol{\theta}, p))$

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 $DA(\theta)$, 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 4.2.

To obtain the results, we iterate randomly through the test settings $\mathcal{P}_{test}$ and use $DA(AS(\boldsymbol{\theta}, p))$ with $\boldsymbol{\theta} = \{\theta_1, \theta_2, \theta_3\}$ to negotiate, following the procedure as described in Figure 4.2. Additionally, we let every opponent in the test set negotiate with every other opponent in the test set on every test scenario and combine the results with the results of the $DA(\theta)$. 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 an ANAC-like bilateral tournament with our $DA(\theta)$ using the strategy selector, see Table 4.6. 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 $DA(\theta)$ with θ_1 and with a portfolio of strategies in a realistic ANAC tournament setup, see Figure 4.3. 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.

Table 4.6: ANAC tournament results using $DA(AS(\boldsymbol{\theta}, p))$ where all scores are averaged over all negotiation instances. 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 negotiation outcome, Nash distance is the distance to the Nash bargaining solution [111] of the scenario, 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	0.446	**0.901**	1.347	0.091	0.428
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	1.204	0.232	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	0.531	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
$DA(AS(\boldsymbol{\theta}, p))$	**0.788**	0.627	1.414	0.074	0.314

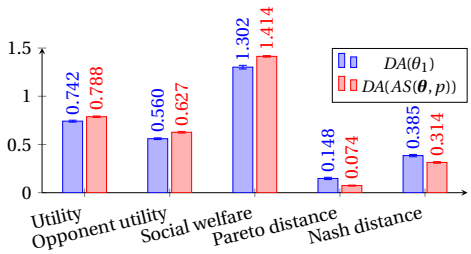

Figure 4.3: Comparison of two $DA(\theta)$ strategies in an ANAC tournament setting. Here, $DA(\theta_1)$ is comparable to the agent configured by Renting et al. [124] and $DA(AS(\boldsymbol{\theta}, p))$ represents this work. See Table 4.6 for an explanation of the measures.

4.7 CONCLUSION

In previous work [124], 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 $\boldsymbol{\theta}$ and tested the performance of every strategy on a broad set of negotiation settings. In Table 4.4, 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. [124], 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 negotiation agents for use in real-life applications. Persons who have more resources to design quality negotiation 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 to the instance characteristics. We intend to investigate strategy selection for negotiation instances through neural networks to relax the reliance on manually designed instance 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.