GreedyAgent: A Semi-Random Frequency-Based Tit-for-Tat Bilateral Negotiation Agent

Sameen Islam
si1u19@soton.ac.uk

Mohammed Mossuily
mtm91994@soton.ac.uk

Sai Pandian
ssp1e17@soton.ac.uk

Pouria Toopchi
pt3g20@soton.ac.uk

ABSTRACT

In this paper, we present GreedyAgent, a multi-algorithm bilateral negotiation agent for the Automated Negotiation Agent Competition (ANAC). GreedyAgent utilises a modified tit-for-tat concession strategy, and a hybrid semi-random and frequency-based bidding strategy. The agent is optimised for balancing maximising personal utility and social welfare (in the form of distance to Nash Bargaining Solution). GreedyAgent employs a linear program solver for estimating its own utility weights in negotiation scenarios which contain preference uncertainty. The agent performs excellently in larger domains, with a higher achieved utility and social welfare than numerous previous ANAC winners, but encounters difficulties in smaller domains.

Word Count: 2487 (excl. abstract, captions, tables, algorithms)

1 INTRODUCTION

The Automated Negotiation Agent Competition (ANAC) [4] is an annual event in which researchers compete with self-designed automated negotiation agents. This paper presents GreedyAgent, an automated agent designed to compete in a replica ANAC competition, with numerous multi-issue domains in a Stacked Alternating Offers Protocol (SAOP). In the competition, agents are measured on the final achieved utility and distance of agreement to the Nash Bargaining Solution (NBS).

As a result, GreedyAgent is optimised to balance achieving the maximum possible utility with finding an agreement close to the NBS. The agent was designed following the BOA (Bidding Strategy, Opponent Modelling and Acceptance Strategy) architecture, with the added function of preference uncertainty modelling. GreedyAgent employs a hybrid frequency-based and semi-random approach for bidding strategy, a modified tit-for-tat approach for negotiation and acceptance strategy, and a linear program solver for preference uncertainty modelling.

This paper presents the employed strategy for each of these areas and explores the effectiveness of each strategy and the overall agent in different scenarios.

2 PREFERENCE UNCERTAINTY MODELLING

One particular challenge in this negotiation scenario is the existence of preference uncertainty. This means that the agent does not have complete knowledge of its own additive utility function, instead only having a partial ranking of outcomes. The problem is presented to compute an approximation of the issue weights used in the utility function, using the provided partial bid ranking. The agent can also submit an outcome for ranking, at the cost of "user bother." User bother is subtracted from the final utility of the agent.

Heuristic and linear programming methods are the most popular solutions to this problem. The heuristic method investigated in this paper is the frequency-based method that is part of the GENIUS framework [10].

Another method involves solving a Linear Program (LP) as detailed in [12]. With this method, we have the same starting information, i.e. a partial user ranking $O \subseteq \Omega$ consisting of a total of d outcomes [11], where Ω is the set of all outcomes. We then construct the set $\mathcal{D}$ of all pairwise comparisons of the bids in the ranking.

$$\mathcal{D} = \{(o^{(j)}, o^{(j+1)}) \mid o^{(j)} \in O \text{ and } 0 < j \leq d-1\} \tag{1}$$

where o is an individual bid in the ranking. For each pairwise comparison, we expect that the subtraction of the evaluation of a lower-ranked bid from a higher-ranked bid would yield a positive result. This is expressed as:

$$\Delta u_{o,o'} = \sum_{i=1}^{m} w_i \cdot v_i(o_i) - w_i \cdot v_i(o_i') \geq 0 \tag{2}$$

Another piece of information known to the agent is that the utility function u is linearly additive, meaning the value of each issue i is calculable using an evaluation function v_i:

$$u(o) = \sum_{i=1}^{m} w_i \cdot v_i(o_i) \text{ where } \sum_{i=1}^{m} w_i = 1 \tag{3}$$

where w_i corresponds to the weight of issue i. The evaluator function is known by the agent.

Using these we can construct a linear program as follows.

$$\begin{aligned} \text{Minimize:} \quad & F = \sum_{(o,o') \in \mathcal{D}} z_{o,o'} \\ \text{Subject to:} \quad & z_{o,o'} + \Delta u_{o,o'} \geq 0 \\ & z_{o,o'} \geq 0 \\ & w_i \cdot v_i(x_j^{(i)}) \geq 0 \\ & \sum_{i=1}^{m} w_i = 1 \end{aligned} \tag{4}$$

The LP in Equation 4 can be solved using any computational LP solver, though GreedyAgent used the Gurobi optimiser [6] due to its relatively high speed comparing to other solvers [1].

Figure 1 presents the discrepancy between calculated weights and true weights using the heuristic method and LP solver method for different sized domains. We observe that the LP solver method estimates issue weights more accurately for all tested domains. The difference between the LP and Heuristic methods is more noticeable

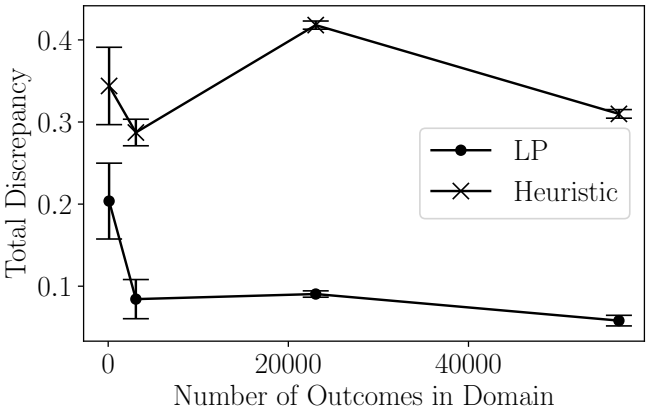

Figure 1: Discrepancy between true issue weights and estimated issue weights in different sized domains, estimated using LP method and Heuristic method. LP method appears to be a better estimator in a diverse range of domain sizes.

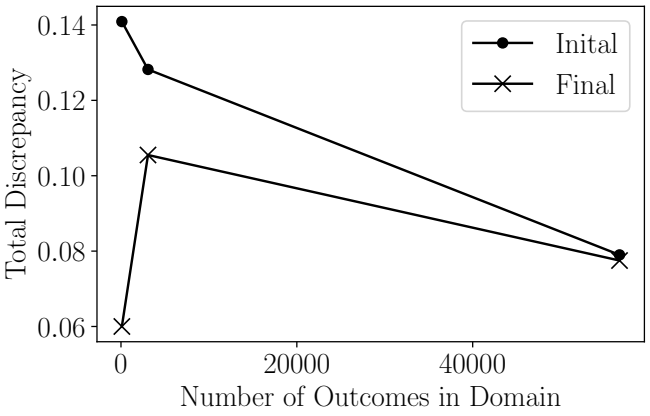

Figure 2: Discrepancy between true and estimated issue weights in different sized domains, estimated using LP method calculated at start of negotiation and at round 25. The estimated weights are closer to true weights when calculated at a later round with a larger ranked bid list.

in larger domains. Due to the greater accuracy of the LP method in a wide range of domains, GreedyAgent uses this method as its sole preference uncertainty model.

One additional method of improving the accuracy of the LP solver is to recompute the weights upon receiving new bids. The new bids are added to the bid ranking (at the cost of user bother) and the weights are recalculated. This means the agent has continually updating weights. Figure 2 presents the accuracy of calculated weights at the start of a negotiation and after 25 rounds. We see that for all domains, the latter calculation is more accurate. Thus, GreedyAgent will recalculate the weights every round until the 25^{th} round. It will not recalculate further to not continually increase user bother.

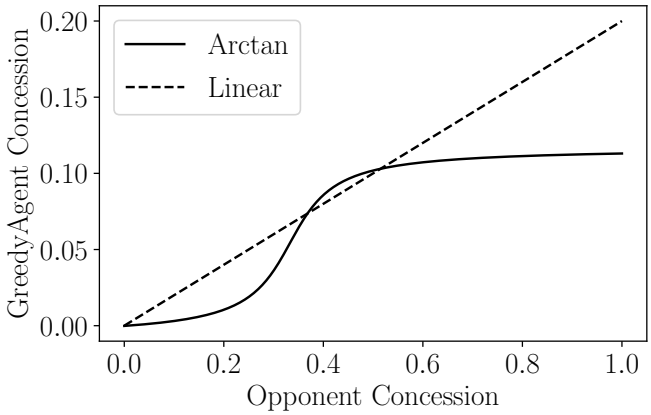

Figure 3: Comparison of two different tit for tat concession strategy curves. We observe that the arc-tan curve allows for a different response depending on the amount opponent concession.

3 CONCESSION & ACCEPTANCE STRATEGY

Since GreedyAgent is designed to balance fairness and personal utility, it employs a "tit-for-tat" concession strategy [3] in which the opponent concession is used to determine the agent concession. The opponent concession is evaluated in terms of the agent's own utility function. GreedyAgent uses an "arc-tangent tit-for-tat" concession, in which the opponent concession is transformed by an arc-tangent function, to ensure the agent does not concede too much. This concession is given by:

$$C_g = \frac{\arctan(\alpha \cdot C_o - 5)}{8\pi} + 0.05 \tag{5}$$

where C_g is GreedyAgent's concession and C_o is the opponent concession. α represents a parameter that controls the slope of the curve. GreedyAgent uses $\alpha = 15$.

Figure 3 compares the described strategy to a common approach, a linear tit-for-tat concession. The linear concession is conceding steadily 5 times slower than the opponent, while the arc-tangent concession allows for adaptive rates of concession depending on the speed at which the opponent is conceding. Table 1 presents a comparison of the two strategies in different domains competing against a Boulware Agent. We observe that arc-tangent concession outperforms linear concession in a wide range of domains. Thus, GreedyAgent uses it as the primary method of concession. The agent will accept any offer that is above its target utility.

4 OPPONENT MODELLING

GreedyAgent is a multi-algorithm frequency-based agent and as such the opponent modelling is handled independently by the different bidding algorithms. Due to the intelligent nature of the competition, we can assume that opponent agents will make sensible offers based around their own utility functions, and not random offers. Thus, GreedyAgent uses past opponent offers to construct GreedyAgent's bids, such as employing a frequency table similar to the Johnny Black agent [14]. Therefore, GreedyAgent does not

Table 1: Comparison of Arctan, Linear concession functions ADN: Average Distance to NBS, AUU: Average User Utility

Domain	Arctan		Linear	
	ADN	AUU	ADN	AUU
Wholesaler-domain (Size: 56700)	0.1852	0.6710	0.2032	0.6060
Party Domain (Size: 3072)	0.1233	0.7331	0.1619	0.6826
Smart Grid Domain (Size: 100)	0.1651	0.8229	0.1735	0.7200

require complicated estimation of the opponent utility function weights, which is the approach taken by other agents.

5 BIDDING STRATEGY

The bidding strategy of GreedyAgent builds upon the approach pioneered by ParsAgent [8], using multiple algorithms each optimised for different-sized domains. The flowchart for the agent is presented in Figure 4. GreedyAgent has four algorithms, with the next algorithm used when one fails.

5.1 Algorithm 1

The first algorithm provides the opening bid to the negotiation. Usually, agents provide their best bid as an opening bid to the negotiation. However, GreedyAgent randomises its least-preferred issue value in the best bid as the first move. This is in an attempt to prevent the opponent from modelling GreedyAgent's utility function. The pseudocode is shown in Algorithm 1.

Algorithm 1: GreedyAgent First Mover Bidding Strategy

```
Output: Random Bid Offer
Input: All issues, best bid, Agent Utility at time t
maxRandomBid ← set to the max utility bid
for i to listOfIssues do
    maxRandomBid ← update issue i of
      maxRandomBid with random value
    if maxRandomBidUtility ≥ currentUtility then
        return maxRandomBid
    end
    else
        maxRandomBid ← set to the max utility bid
    end
end
```

5.2 Algorithm 2

The second algorithm aims to create a bid using the opponent's most recent offer. GreedyAgent modifies at random one issue in this offer, with its own more preferred issues more likely to be chosen. The new bid evaluated against current target utility. If the bid is unsuitable, the issue is reset and another issue chosen at random. The advantage of this method is that it avoids having to model the opponent utility function to generate a suitable bid, since the modified bid will likely still be highly desired by the opponent. This algorithm is well-suited to small domains, where the number of values in an issue is small. The pseudocode is given in Algorithm 2.

Algorithm 2: GreedyAgent Second Bidding Strategy

```
Output: Random Bid Offer
Input: All issues, opponent offer, Agent Utility at time t
ourMostWantedIssue ← select issue based on probability
for numberOfIssues do
    newBid ← insert ourMostWantedIssue into bid
    for j to numberOfIssues do
        if j != ourMostWantedIssue then
            newBid ← insert random value for issue j
        end
    end
    if newBidUtility ≥ currentUtility then
        return newBid
    end
end
```

5.3 Algorithm 3

The third algorithm uses a frequency table of opponent offers to create a bid. GreedyAgent will select the most occurring value in the opponent's bids. A new bid is then created with this value for the given issue. The empty issues will be updated with random values. The new bid is then checked against GreedyAgent's target utility. Since this bid is generated using the opponent's most desired issue variable, it will likely be desirable to the opponent. This algorithm is well suited to large domains where the frequency table is likely to be populated with more values. The pseudocode is shown in Algorithm 3.

Algorithm 3: GreedyAgent Third Bidding Strategy

```
Output: Random Bid Offer
Input: All issues, Agent Utility at time t, numberOfRuns
opponentPreferredIssue ← opponent preferred issue
  obtained from frequency table
for numberOfRuns do
    RandomBid ← insert opponentPreferredIssue into bid
    RandomBid ← fill remaining issues with random values
    if RandomBidUtility ≥ currentUtility then
        return randomBid
    end
    else
        RandomBid ← is set to null
    end
end
```

5.4 Backup Algorithm

The fourth algorithm is executed if the other algorithms fail to create a bid. The backup algorithm modifies half of the issues in the best bid with a random value. This modified bid is checked to see if the utility is greater than or equal to GreedyAgent's target utility. The pseudocode shown in Algorithm 4 demonstrates how the random bid is generated.

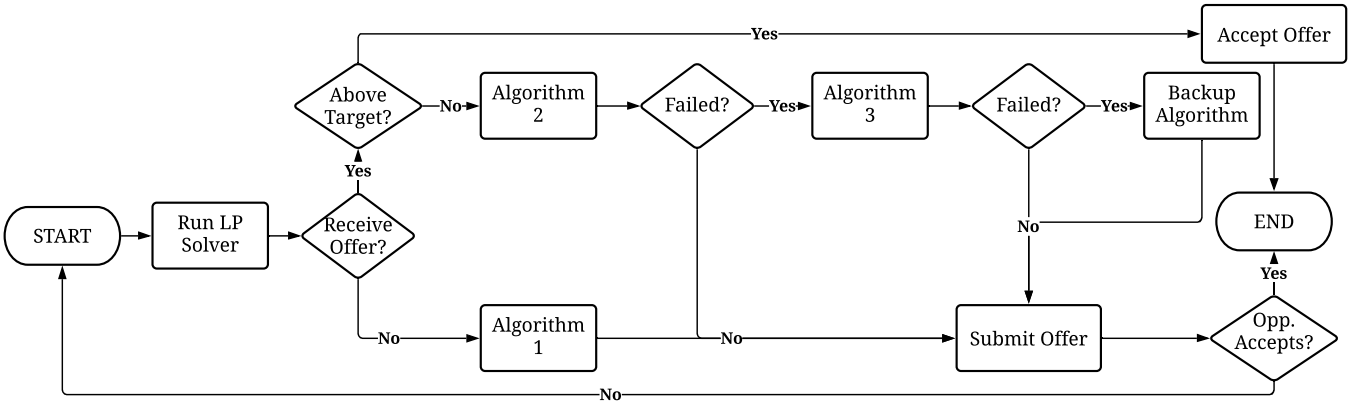

Figure 4: Flowchart representing Agent architecture. The fundamental concept is the use of three different algorithms (as well as a backup algorithm), each of which is more effective in different domains. This ensures the agent has high performance in a diverse set of domains.

Algorithm 4: GreedyAgent Backup Bidding Strategy

Output: Random Bid Offer
Input: All issues, Agent Utility at time t, *numberOfRuns*
randomListOfIssues ← randomly selected half of issues
for *numberOfRuns* **do**
  randomBid ← empty bid declared
  for *i* **to** *randomListOfIssues* **do**
    randomBid ← set issue *i* with random value
  end
  if $randomBidUtility \geq currentUtility$ **then**
    return *randomBid*
  end
end

Table 2: Agent performances against Boulware Agent. ADN: Average Distance to NBS, AUU: Average User Utility

Domain	Party	ADN	AUU
Wholesaler-domain (Size: 56700)	GreedyAgent	0.1852	0.6710
	ParsAgent	0.3810	0.7024
	IAMHaggler2012	0.2618	0.6567
PartyDomain (Size: 3072)	GreedyAgent	0.1323	0.7733
	ParsAgent	0.3119	0.8544
	IAMHaggler2012	0.1542	0.7801
SmartGridDomain (Size: 100)	GreedyAgent	0.2871	0.6983
	ParsAgent	0.1517	0.7259
	IAMHaggler2012	0.1990	0.6756

6 AGENT TESTING & PERFORMANCE

We measure agent performance with two metrics: The average distance to Nash Bargaining Solution (ADN), and the average user utility (AUU) with GreedyAgent aiming to achieve a balance between ADN and AUU.

GreedyAgent's performance was tested against Boulware Agent in many different domains. Table 2 presents different agents' performance in three example domains, representing three different sizes. We observe that in large domains, GreedyAgent achieves ADN that is significantly lower than two other smart agents. However, the AUU is slightly worse than ParsAgent [8]. This is likely due to ParsAgent and IAMHaggler2012 [13] being optimised for maximising personal utility, and not social welfare.

If we consider GreedyAgent's performance in medium-sized domains, we observe a similar trend. GreedyAgent outperforms the other agents in its ADN, but is beaten in AUU though by a small margin. Thus by adjusting the balance between prioritising ADN and AUU, this could be mitigated.

The high performance in large and medium-sized domains is mainly attributed to Algorithm 3. We expect that in larger domains, Algorithm 2 is more likely to fail since with a greater number of values associated with each issue, it is more difficult to obtain an offer greater than GreedyAgent's target utility by only randomising one issue value. Thus, bid generation in large domains is dominated by Algorithm 3. This algorithm is also more effective in long negotiations, occurring more in larger domains, where the frequency table of opponent offers is populated with a greater number of opponent bids.

When we observe GreedyAgent's performance in small domains, we see that the agent is soundly beaten in ADN. One reason for this may be that Algorithm 2, the more commonly expected algorithm in smaller domains, is not performing adequately. In earlier negotiation rounds, the opponent agent will propose bids that are very close to the maximum of their utility function. Depending on the preference profiles of both agents, it may not be possible for Algorithm 2 to modify the opponent offer to create a suitable bid in these earlier rounds. If Algorithm 3 also fails, as it is likely to do in earlier rounds when the frequency table is sparse, then the Backup Algorithm will be used. This algorithm has no opponent-modelling features, and will thus not submit a desirable bid. In later rounds,

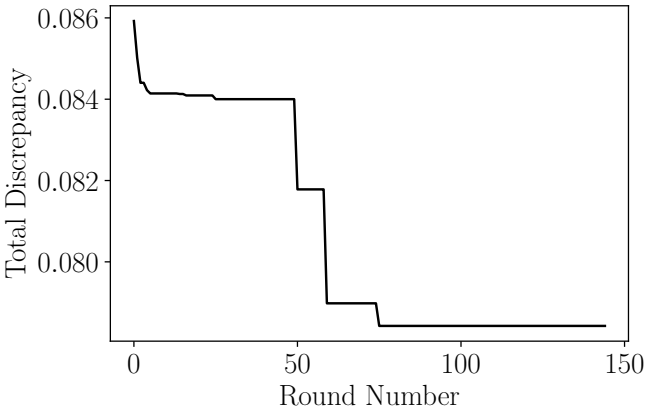

Figure 5: Discrepancy between true and estimated issue weights, calculated every round until agreement. We observe that the calculated issue weights are more inaccurate at start of negotiation.

Table 3: GreedyAgent performance against other agents ADN: Average Distance to NBS, AUU: Average User Utility

Agent	ADN	AUU
GreedyAgent	0.1712	0.8452
IAMHaggler2012	0.1825	0.8268
NiceTitForTat	0.0847	0.7941
ParsAgent	0.1231	0.7036

once the opponent has sufficiently conceded, we expect that Algorithm 2 is more effective, but by this point, GreedyAgent has also conceded too much, often past the NBS.

Another reason for poor performance in smaller domains could be the use of LP solver for weight estimation. In Figure 2, we observe that the initial estimate for weights is more inaccurate in smaller domains when compared to larger domains. Figure 5 presents the discrepancy between true and estimated weights every round. We observe that early in the negotiation, the estimated weights are more inaccurate, mainly due to the small ranked bid list. Thus, GreedyAgent has poorly estimated issue weights at the beginning of the negotiation and is not able to propose suitable offers. In later rounds, it has more accurate estimations of weights, but has conceded too much by this point.

It is vital to measure GreedyAgent's performance against other smart agents as we can observe how its anti-modelling strategies work against agents actively attempting to model its behaviour. Table 3 presents this performance in the PartyDomain.

When GreedyAgent negotiates against IAmHaggler2012, we observe that it achieves the highest AUU, compared to all other opponents. However, the ADN is also the highest. IAmHaggler2012 employs a Bayesian learning technique for modelling the opponent utility function to decide on a concession. However, as GreedyAgent is a semi-random agent, it is inherently difficult to model its utility function based on proposed offers. Thus, IAmHaggler2012 has a slow concession rate, leading to a slower concession for GreedyAgent, resulting in an agreement with a high utility, but far from the NBS. Nonetheless, this is good evidence for the effectiveness of GreedyAgent's non-modellable nature.

When negotiating against NiceTitForTat [3], we observe that the ADN is very small, while the AUU remained relatively high. NiceTitForTat is an agent that is well-optimised for maximising social welfare, conceding proportionally towards the NBS. As a result, the agreement is likely to be found around the NBS. As NiceTitForTat also creates offers around an estimated Pareto frontier, the AUU for GreedyAgent is high, as the accepted offer is close to Pareto optimal.

When competing against ParsAgent, we find that while the ADN is quite small, the AUU is also significantly lower than compared to other opponents. This is likely due to the "stubborn" nature of ParsAgent, which employs a time-dependent Boulware concession strategy with a lower limit. This means that ParsAgent will not accept any offer that it evaluates to have a utility lower than 0.7. Thus, GreedyAgent would have to concede significantly more to achieve an agreement.

7 EVALUATION & FUTURE WORK

One important observation was that GreedyAgent often concedes past the NBS, resulting in a large ADN and a low AUU. One way of mitigating this issue is to implement a similar strategy to the NiceTitForTat agent, in which the agent concedes up to the NBS. One method of implementing this would be with smart Pareto front estimation, from which we could discern the NBS. Methods for this are explored in [5] and [9]. By conceding only up to NBS and making offers specifically around this point, GreedyAgent could perform similarly to how it performed against NiceTitForTat against all other opponents.

We observed that in small domains, GreedyAgent performs poorly due to Algorithm 2 not generating suitable offers in earlier negotiation rounds. One reason for this is that the methodology by which opponent offer is modified to make a suitable offer is not effective if the opponent has not yet conceded a sufficient amount. Thus, an approach in which the opponent's utility function is directly modelled may be necessary. One approach for this would be the use of Bayesian learning [7]. However, this is a similar approach to IAmHaggler2012, and we observed that it performs poorly against semi-random opponents.

Finally, GreedyAgent had difficulty estimating the issue weights at the start of a negotiation in small domains. This could be improved by utilising a heuristic method in small domains early in the negotiation, and only using the LP solver method later in the negotiation. This is a similar approach to WinkyAgent, which took part in ANAC 2019 [2]. A better heuristic would have to be found, compared to the one tested in Section 2.

8 CONCLUSION

This paper presented GreedyAgent, a semi-random frequency-based negotiation agent which employed a modified tit-for-tat concession strategy. The agent is optimised to balance maximising utility and social welfare. This investigation found that GreedyAgent has high performance in large domains, beating numerous past ANAC

winners. However, the agent encountered performance issues in smaller domains, which could be improved upon by utilising a more sophisticated opponent-modelling method or improved weight estimation method.

REFERENCES

[1] Rimmi Anand, Divya Aggarwal, and Vijay Kumar. 2017. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems* 20, 4 (7 2017), 623–635. https://doi.org/10.1080/09720510.2017.1395182

[2] Reyhan Aydoğan, Tim Baarslag, Katsuhide Fujita, Johnathan Mell, Jonathan Gratch, Dave de Jonge, Yasser Mohammad, Shinji Nakadai, Satoshi Morinaga, Hirotaka Osawa, Claus Aranha, and Catholijn M. Jonker. 2020. Challenges and Main Results of the Automated Negotiating Agents Competition (ANAC) 2019. Springer, Cham, 366–381. https://doi.org/10.1007/978-3-030-66412-1{_}23

[3] Tim Baarslag, Koen Hindriks, and Catholijn Jonker. 2013. A tit for tat negotiation strategy for real-time bilateral negotiations. *Studies in Computational Intelligence* 435 (2013), 229–233. https://doi.org/10.1007/978-3-642-30737-9{_}18

[4] Tim Baarslag, Koen Hindriks, Catholijn Jonker, Sarit Kraus, and Raz Lin. 2012. The first automated negotiating agents competition (ANAC 2010). *Studies in Computational Intelligence* 383 (2012), 113–135. https://doi.org/10.1007/978-3-642-24696-8{_}7

[5] Shubhangi Deshpande, Layne T Watson, and Robert A Canfield. 2013. Pareto Front Approximation Using A Hybrid Approach. *Procedia Computer Science* 18 (2013), 521–530. https://doi.org/10.1016/j.procs.2013.05.216

[6] Gurobi Optimization LLC. 2020. Gurobi Optimizer Reference Manual. https://www.gurobi.com/

[7] Koen Hindriks and Dmytro Tykhonov. 2008. *Opponent Modelling in Automated Multi-Issue Negotiation Using Bayesian Learning*. Technical Report. https://doi.org/10.5555/1402383.1402433

[8] Zahra Khosravimehr and Faria Nassiri-Mofakham. 2017. Pars agent: Hybrid time-dependent, random and frequency-based bidding and acceptance strategies in multilateral negotiations. *Studies in Computational Intelligence* 674 (2017), 175–183. https://doi.org/10.1007/978-3-319-51563-2{_}12

[9] Max W.Y. Lam and Ho Fung Leung. 2017. Phoenix: A threshold function based negotiation strategy using gaussian process regression and distance-based pareto frontier approximation. *Studies in Computational Intelligence* 674 (2017), 201–212. https://doi.org/10.1007/978-3-319-51563-2{_}15

[10] 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 (2 2014), 48–70. https://doi.org/10.1111/j.1467-8640.2012.00463.x

[11] V. Srinivasan and Allan D. Shocker. 1973. Estimating the weights for multiple attributes in a composite criterion using pairwise judgments. *Psychometrika* 38, 4 (12 1973), 473–493. https://doi.org/10.1007/BF02291490

[12] Dimitrios Tsimpoukis, Tim Baarslag, Michael Kaisers, and Nikolaos G. Paterakis. 2019. Automated Negotiations Under User Preference Uncertainty: A Linear Programming Approach. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 11327 LNAI. Springer Verlag, 115–129. https://doi.org/10.1007/978-3-030-17294-7{_}9

[13] Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. 2012. IAMhaggler: A negotiation agent for complex environments. *Studies in Computational Intelligence* 383 (2012), 151–158. https://doi.org/10.1007/978-3-642-24696-8{_}10

[14] Osman Yucel, Jon Hoffman, and Sandip Sen. 2017. Jonny black: A mediating approach to multilateral negotiations. *Studies in Computational Intelligence* 674 (2017), 231–238. https://doi.org/10.1007/978-3-319-51563-2{_}18