Negotiation Agent Group 37 Report

Thanakorn Panyapiang
tp2n19@soton.ac.uk

Vasin Srisupavanich
vs2n19@soton.ac.uk

ABSTRACT

In this work, we design and develop a negotiation agent using the GENIUS platform [6], a platform for developing automated multi-issues negotiation agents. The negotiation strategy of our agent is created using the combination of many state-of-the-art approaches from the literature including opponent modelling using frequency heuristic and Gaussian process regression, user model estimation using linear programming, and concession strategy that is adaptive based on opponent's behavior. The agent is developed as part of the class competition in *Intelligent Agent* (COMP6203) module at the University of Southampton in the academic year 2019. Excluding the problem of our agent's scalability issue on the Energy domain, our agent was able to achieve the third highest average utility while maintaining the fifth lowest average distance to Nash point of agreement in the class competition.

KEYWORDS

Automated bilateral negotiation, Multi-agent systems, Opponent modelling, User preference uncertainty

1 INTRODUCTION

This paper describes our agent's strategies and the motivation behind each strategy. We designed our agent to maximise its own utility and concede as little as possible, while taking into consideration the opponent's preference when making an offer. We used frequency heuristic based on [9, 12] to model the opponent utility function, and model the opponent concession by using Gaussian process regression [11], which, in turn, is used to adjust our agent's concession rate. The estimation of our user model is created using linear programming approach based on [8]. We also implemented a hybrid offering strategy where we try to prevent the opponent from modelling our agent's strategy. The rest of the paper discuss in detail the negotiation techniques we apply, and their performance compares to other techniques.

2 ARCHITECTURE

Our agent consists of six main components which are responsible for different aspects of the negotiation strategy. These components are designed as separated modules on top of the GENIUS framework, so they can be tested and experimented separately without running the simulation. This approach also helps the team to collaborate more efficiently and reduce conflicts during code integration as the logic of each part is clearly separated. The design of our agent's components and how they are integrated into the agent are illustrated in Figure 1.

PreferenceModel is responsible for estimating the utility space of the agent. It estimates the utility space from *User* and *UserModel*. Once the UtilitySpace is created, it will be used by ***OfferGenerator*** to create an offer space, a list of pre-generated offers which the agent will operate on during the negotiation.

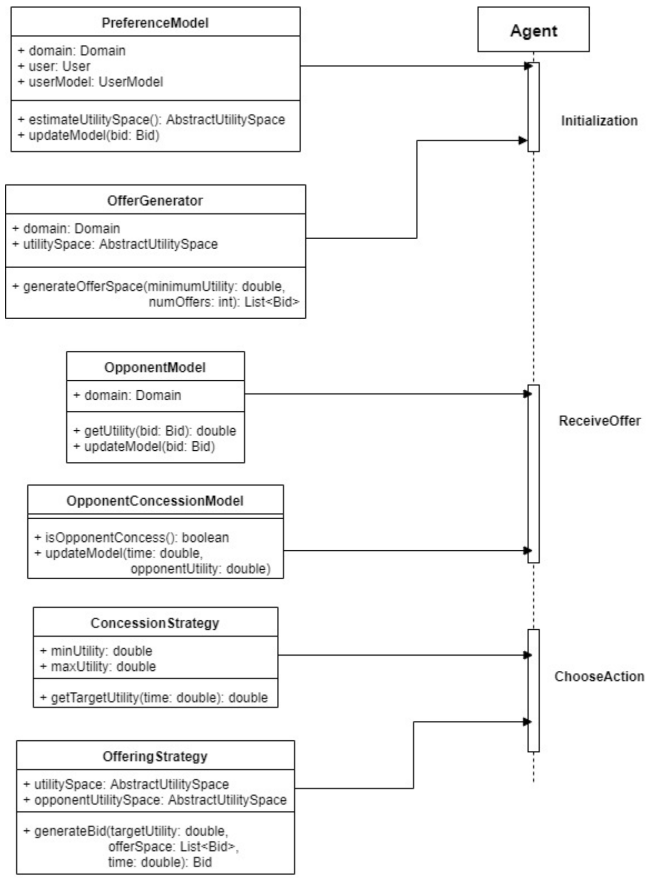

PreferenceModel
+ domain: Domain
+ user: User
+ userModel: UserModel
+ estimateUtilitySpace(): AbstractUtilitySpace
+ updateModel(bid: Bid)
Agent
Initialization
OfferGenerator
+ domain: Domain
+ utilitySpace: AbstractUtilitySpace
+ generateOfferSpace(minimumUtility: double, numOffers: int): List<Bid>
OpponentModel
+ domain: Domain
+ getUtility(bid: Bid): double
+ updateModel(bid: Bid)
ReceiveOffer
OpponentConcessionModel
+ isOpponentConcess(): boolean
+ updateModel(time: double, opponentUtility: double)
ConcessionStrategy
+ minUtility: double
+ maxUtility: double
+ getTargetUtility(time: double): double
ChooseAction
OfferingStrategy
+ utilitySpace: AbstractUtilitySpace
+ opponentUtilitySpace: AbstractUtilitySpace
+ generateBid(targetUtility: double, offerSpace: List<Bid>, time: double): Bid

Figure 1: Agent Architecture

When receiving an offer from the opponent, the agent passes the received offer to the ***OpponentModel*** in order to update the opponent preference. Moreover, it is also used by the ***OpponentConcessionModel*** to predict their concession strategy. In choosing an appropriate response, the agent makes a decision using ***ConcessionStrategy*** and ***OfferingStrategy***. *ConcessionStrategy* determines the target utility of the agent at a particular time of the negotiation. The agent will accept any offer not lower than the target utility, otherwise, the agent will ask *OfferingStrategy* to find the best counter-offer from the pre-generated offer space.

3 USER MODEL ESTIMATION

Our agent deals with preference uncertainty by using the Linear Programming[8] to estimate user's preference model. We transform *BidRank* derived from the *UserModel* into a linear optimization problem where each option is considered as a variable in a linear equation system. Then solve it using the *Simplex Method*[10] with the goal to minimize the sum of all slack variables.

To construct the objective function and all constraints, we generate the *option-constraints matrix* which has *(m + n)* columns where the first *m* columns represent the coefficients of all options in the domain, the last *n* columns represents coefficients of all slack variables(*n* is the bid order size), and each row represents one constraint. The structure of the constraint matrix can be defined as follow:

$$\left[\begin{array}{cccccccccc} v_{1,1} & v_{1,2} & .. & v_{2,1} & v_{2,2} & .. & v_{k,m} & z_1 & .. & z_{n-1} \\ \hline 0 & 1 & .. & 0 & 0 & .. & 0 & 1 & .. & 0 \\ 1 & 0 & .. & 0 & 1 & .. & 0 & 0 & .. & 0 \\ 0 & 1 & .. & 0 & 0 & .. & 0 & 0 & .. & 1 \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \end{array}\right]$$

where:

k = Number of options in the domain
n = Number of bids in the *BidRank*
$v_{i,j}$ = Coefficient of option j^{th} for issue i^{th}
z_i = Coefficient of the i^{th} slack variable

Once the constraint matrix is built, we use **SCPSolver**, the Java Linear Programming API, to minimize the objective function subject to all the constraints. Solving this problem gives estimated utility value for all options in the domain.

3.1 Weight Estimation

The approach for estimating weights is similar to the utility values estimation except the unknown variables are the issue weights. We construct the *weight-constraints matrix* where the first *k* columns represents coefficients of the weights, the last *n* columns represent coefficients of all slack variables. Note that the coefficient of the issue weight is computed by subtracting the utility of the options(which are estimated in the previous step) which belong to the same issue from two different offers in the *BidRank*. The *weight-constraints matrix* has the structure as follow:

$$\left[\begin{array}{ccccccc} w_1 & w_2 & .. & w_k & z_1 & .. & z_{n-1} \\ \hline u_{1,1} - u_{1,2} & u_{2,1} - u_{2,2} & .. & .. & 1 & .. & 0 \\ u_{1,2} - u_{1,3} & u_{2,2} - u_{2,1} & .. & .. & 0 & .. & 0 \\ u_{1,3} - u_{1,3} & u_{2,2} - u_{2,4} & .. & .. & 0 & .. & 1 \\ .. & .. & .. & .. & .. & .. & .. \end{array}\right]$$

where:

k = Number of issues in the domain
n = Number of bids in the *BidRank*
w_i = The weight of i^{th} issue
$u_{i,j}$ = The utility of value j^{th} of i^{th} issue

3.2 Evaluation

To evaluate the performance, we experiment our method using the **Party** domain provided by GENIUS. In the experiment, we randomly generated a *BidRank* with several numbers of offers and let the agent estimates the utility space. Next, we find the average utility error of all possible offers in the domain, then compare the result with GENIUS's default estimator. The result of the experiment is shown in Table 1.

The table shows that the average utility error of the linear programming approach is lower than the average utility of the GENIUS default estimator especially when the number of offers in *BidRank* is relatively small.

Num Offers	GENIUS Estimator	LinearProgramming
100	0.21	0.12
250	0.19	0.11
500	0.22	0.09
1000	0.15	0.10
1500	0.18	0.11
2000	0.13	0.11

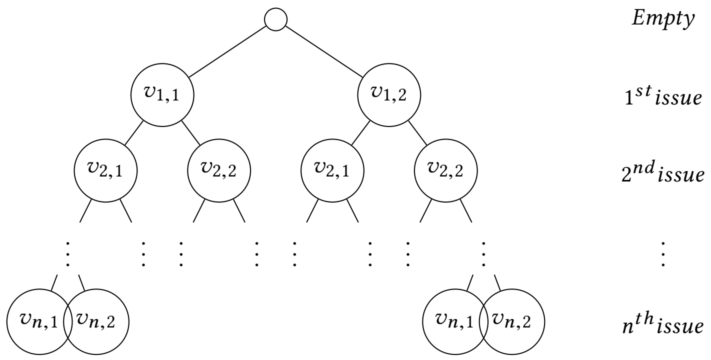
Table 1: Average Utility Error between GENIUS DefaultEstimator and LinearProgramming

4 OFFER SPACE GENERATION

Apart from user model estimation, another important task that our agent does during the initialization is generating an offer space. Generating an offer space in advance gives a huge benefit in terms of computational time especially in a large domain because the agent can operate under a significantly smaller offers list while still obtain a desirable utility.

Although the GENIUS platform has a built-in method for generating an offer, it is very inefficient especially in a huge domain as it uses a random approach. Therefore, we decide to develop our own offer-generating technique.

The principle of our approach is the whole offer space can be modeled as a search tree with depth *n* where *n* is the numbers of issues in the domain. The root node is an empty offer, and each node in the tree represents one variable assignment. All nodes in the first level represents all possible assignments for the most important(highest weight) issue, all nodes in the second level represents all possible assignments for the second-most important issue as illustrated in the figure below. A path from the root node to any leave node is equivalent to a complete offer.



The offer space which our agent operates on during the negotiation is generated by using *depth-first search* to explore the whole offer space and choose only offers that have desired utility.

To evaluate the performance of our approach, we run the experiment to compare the latency in generating an offer space between the GENIUS built-in method and our approach on the **Energy** domain which has a very large number of possible bids(400,000). The result is shown in Figure 2.

From the result, it can be observed that the DFS approach that we develop significantly outperforms the random-generating approach in terms of latency especially when the number of offers are large.

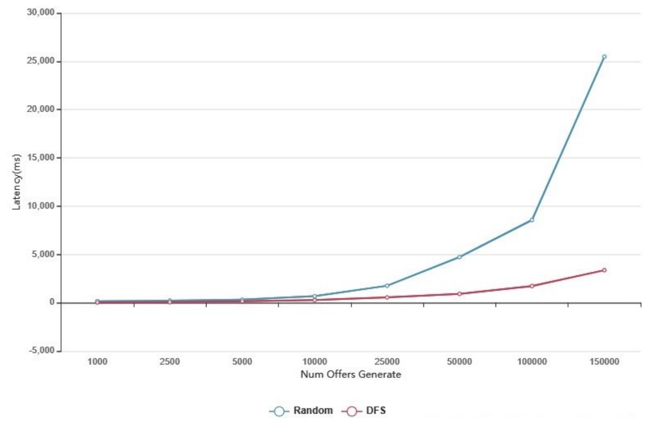


Figure 2: Offer Generating Latency

5 OPPONENT MODELLING

The key challenge in the automated negotiation tournament is that there is no knowledge about the opponent prior to the negotiation. We can only try to estimate the opponent model during the negotiation. According to the work by Tim Baarslag et al. [2], having a good opponent model can greatly improve the quality of the negotiation outcome, where we can avoid non-agreements, and find win-win situation for both agents. In this project, our agent try to learn both *what the opponent want* and *what the opponent do* by learning the opponent's utility preference model and the opponent's concession function.

5.1 Opponent Preference Modelling

We chose frequency based approach to do the opponent preference modelling due to its accuracy, simplicity, and speed. According to [1, 3], frequency model outperforms more complex model like Bayesian learning, due to the fact that frequency model making less assumptions. The main idea behind the frequency based approach is that the opponent's preferred option will appear more frequently in the negotiation trace, and the issue, where the opponent changes the option often, could be relatively less important to the opponent.

5.1.1 Value Function Estimation. Our agent's value estimation function is implemented according to the work by Okan Tunalı et al. This is the state-of-the-art frequency based technique which introduced Laplace smoothing and exponential filter [9]. Laplace smoothing gives the importance to the options that are yet to appear in the negotiation trace, and exponential filter can be used to prevent the unbalance distribution of option values when the opponent keep sending the same offer. The value estimation function is defined as follow:

$$V_i(j) = \frac{(1 + C_{i_j})^\gamma}{(1 + M_i)^\gamma}$$

where $V_i(j)$ is the estimated value for option j in issue i. C_{i_j} represents the frequency of option j in issue i, M_i represents the highest count of option in issue i, and γ represent the exponential filter which has the range between 0 and 1. The addition of 1 in both numerator and denominator is the Laplace smoothing.

5.1.2 Issue Weight Estimation. The weight of each issue is estimated based on [12]. This method uses Gini-Impurity scores to determine the weight, and is defined as follow:

$$w_i = \sum_{o \in O_i} \frac{C_o^2}{T^2}$$

where w_i is the estimated weight of issue i, C_o represents the frequency of option o, and T represents the total offers in the negotiation trace.

5.2 Opponent Concession Modelling

Opponent concession model can be estimated by analysing the opponent's bidding history. By learning the opponent's behavior, we can exploit this information to maximise our agent's utility [2]. Our approach use **Gaussian process regression** based on [11] to predict the opponent's behavior. This technique is chosen because it gives us the prediction along with uncertainty. To detect whether the opponent is conceding, only the prediction results with high certainty are used. We use the library from [7], which exists in the GENIUS framework, to do the regression. The output from the regression is a Gaussian distribution which is defined as follow:

$$f(u; u_t, \sigma_t) = \frac{1}{\sqrt{2\pi\sigma_t^2}} e^{-\frac{u - u_t}{2\sigma_t^2}}$$

where u_t is the predicted mean (our agent's predicted utility) at time t, and σ_t is the predicted variance or the uncertainty of the prediction at time t.

Our opponent concession modelling approach works as follow.

(1) The negotiation period is separated into 10 time slots.
(2) The opponent offers is recorded in term of our agent's utility. The assumption is that if the opponent is conceding, our agent's utility would increase.
(3) Only the maximum utility in each time slot is used. This helps reduce the noise in the regression process.
(4) After every time slot, the training model is updated, and the prediction is made for all the time period.
(5) In every round after the third time slot, we calculate the opponent's concession by checking the predictions for all the time period, where only the predictions, which the 95% confidence interval of the variance is less than 0.2, are used.
(6) If our agent's predicted utility keep increasing over our specified time period, the opponent's concession is detected.

Algorithm 1 shows the pseudo-code which summarize our opponent concession modelling approach where t_c represents the current round of the negotiation and t_{max} represents the deadline. O_{opp} represents the opponent's latest offer and V_o is our agent's utility value from opponent's latest offer. μ, σ are the mean and variance vectors, outputted from the regression.

6 CONCESSION STRATEGY

Our concession strategy is based on time-dependent tactic and is adaptive based on opponent's behavior. The target utility outputted from our concession function is used for both our agent's acceptance and offering strategy. We designed our agent to concede very little at the beginning, and only concede more when the deadline

Algorithm 1: Pseudo-code for opponent concession modelling

```
InitializeRegressionModel();
$\mu$, $\sigma$ ← EmptyVector;
$MaxUtilityInTimeSlot$ ← 0;
$IsOpponentConcede$ ← False;
while $t_c < t_{max}$ do
    $O_{opp}$ ← ReceiveOffer();
    $V_o$ ← GetAgentUtility($O_{opp}$);
    if $V_o$ > $MaxUtilityInTimeSlot$ then
        $MaxUtilityInTimeSlot$ ← $V_o$;
    end
    if IsNewTimeSlot() then
        UpdateRegressionModel();
        $\mu$, $\sigma$ ← PerformRegression();
        $MaxUtilityInTimeSlot$ ← 0;
    end
    $IsOpponentConcede$ ← DetectConcession($\mu$, $\sigma$);
    if $IsOpponentConcede$ then
        AdjustAgentConcession(HARD_HEAD);
    else
        AdjustAgentConcession(NORMAL);
    end
end
```

is approaching. We believed that by conceding later than the opponent, we have a higher chance of receiving higher utility. Since there is no time discount in the competition, there is no benefit in reaching the agreement early, so our agent can explore the outcome space and have a more accurate model of the opponent. The concession function is based on [4, 5], and is defined as follow:

$$F(t) = (C_i + ((1 - C_i) * t^{1/\beta}))$$

$$U_{target}(t) = U_{min} + (1 - F(t)) * (U_{max} - U_{min})$$

where t is the normalized time period ranging from 0 and 1. U_{min} and U_{max} represent the minimum and maximum acceptable utility of our agent. C_i is the initial concession value and β is the concession rate. The initial concession value allows more offers to be generated in the initial phase of negotiation, and was set to be 0.05. There are two modes of concession rate: HARD_HEAD (0.05) and NORMAL (0.1). During the negotiation, the concession rate is changed according to the opponent's behaviour, as explained in the previous section (last section of Algorithm 1).

7 OFFERING STRATEGY

Our agent use a hybrid offering process which contains some randomness while taking into account the opponent's preferences. The random element was added in the first half of the negotiation to prevent the opponent from modeling our concession strategy, and gain an advantage on our agent. During the negotiation, our agent considers only the subset of the offer space, which is pre-generated during the initialisation phase (section 4). When selecting the offer, our agent will find the bids that are higher than the target utility, and choose the one which is best for the opponent according to our opponent model. By considering opponent's preference, we would have a higher chance of agreement, and an outcome that is nearer to the Pareto efficient frontier and Nash bargaining solution, which would give us a higher score in the competition.

Algorithm 2 shows the pseudo-code which summarize the function for choosing the offer for the opponent. The function takes the following input: *PreparedBids*, which is the generated subset of all bids, *TargetUtility*, which is the value calculated from our agent's concession function, and *BestBidFromOpponent*, which is the highest utility offer our agent receive from the opponent.

Algorithm 2: Pseudo-code for offering strategy

```
input  :PreparedBids, TargetUtility, BestBidFromOpponent
ChosenBid ← NULL;
$TargetBids$ ← FilterBidsHigherThanTarget(PreparedBids,
  TargetUtility);
$BestBidForOpponent$ ← GetOpponentBestBid($TargetBids$);
if t <= 0.5 then
    RandomBid ← GetRandomBid();
    ChosenBid ← GetRandomBidBetween(RandomBid,
      BestBidForOpponent);
else
    ChosenBid ← BestBidForOpponent;
end
if ChosenBid != NULL then
    Return ChosenBid;
else
    Return BestBidFromOpponent;
end
```

8 PERFORMANCE ANALYSIS

Regarding the competition result, our agent was able to achieve 0.87491 average utility, which is the third highest among all the competing agents. With regards to distance to Nash solution, our agent achieved 0.15963, which is the fifth lowest distance in the competition. Out of all the completed negotiations (5733 negotiations), only 20 negotiations resulted in non-agreements. However, due to the NULL pointer exceptions with our linear programming approach, our agent was unable to complete the negotiations in the Energy domain, which resulted in a low expected score of 0.5799.

Next, we analyse how our agent perform against previous finalists of ANAC Competition by running a tournament in a GENIUS framework. We wanted to test our negotiation strategy without considering the preference uncertainty, so the tournament was run with these settings: 5 agents (3 previous finalists and 1 standard Agent), 5 rounds with 30 seconds deadline, using default party domain. The result from the tournament is shown in Table 2. Our agent was able to achieve an average utility of 0.787, while the average distance to Nash agreement is 0.217. The result shows that our agent performs relatively well, considering the opponents are the previous finalists of ANAC competition.

Agent	Ave. Utility	Ave. Distance to Nash
Agent37	0.787	0.217
Atlas3	0.905	0.247
IAMhaggler2012	0.720	0.160
CUHKAgent	0.943	0.225
BoulwareAgent	0.681	0.240

Table 2: Tournament Result

9 CONCLUSION AND FUTURE WORK

The results from the competition and our analysis show that our approach in maximising the utility while maintaining low distance to Nash agreement is successful. However, there are several areas which our agent can be further improved. First, the agent estimates user's preference profile using only offers in a *BidRank* at the beginning of the negotiation. The limitation of this is that, in the negotiation with high uncertainty, the estimated utility space could be inaccurate. To overcome this issue, the agent should be able to update the user's model during the negotiation by asking the user to rank an unknown offer proposed by an opponent as it will improve the accuracy of the user model over time. Second, we could also improve our negotiation strategy by applying the result from Gaussian process regression to predict the best utility our agent can receive at the end of the negotiation, which can potentially improve the quality of the outcome.

10 INDIVIDUAL CONTRIBUTION

Our group has only 2 members, since Tanat Tanyapongpanich (tt2n19@soton.ac.uk) has withdrawn from the program. In this project, Vasin Srisupavanich (vs2n19@soton.ac.uk) and Thanakorn Panyapiang (tp2n19@soton.ac.uk) have contributed equally to both the development of agent and the report. The tasks were divided as follow.
Vasin: Opponent Modelling, Concession Strategy, and Offering Strategy
Thanakorn: Setting up Agent's architecture, User Model Estimation, and Offer Space Generation

REFERENCES

[1] Tim Baarslag, Mark Hendrikx, Koen Hindriks, and Catholijn Jonker. 2012. Measuring the Performance of Online Opponent Models in Automated Bilateral Negotiation, Vol. 7691. 1–14. https://doi.org/10.1007/978-3-642-35101-3_1

[2] Tim Baarslag, Mark Hendrikx, Koen Hindriks, and Catholijn Jonker. 2016. Learning about the opponent in automated bilateral negotiation: a comprehensive survey of opponent modeling techniques. *Autonomous Agents and Multi-Agent Systems* (09 2016). https://doi.org/10.1007/s10458-015-9309-1

[3] Tim Baarslag, Koen Hindriks, Mark Hendrikx, and Catholijn Jonker. 2013. Predicting the Performance of Opponent Models in Automated Negotiation. *Proceedings - 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2013* 2. https://doi.org/10.1109/WI-IAT.2013.91

[4] Peyman Faratin, Carles Sierra, and Nick Jennings. 1998. Negotiation Decision Functions for Autonomous Agents. *Robotics and Autonomous Systems* 24 (09 1998), 159–182. https://doi.org/10.1016/S0921-8890(98)00029-3

[5] S. Fatima, Michael Wooldridge, and Nicholas Jennings. 2001. Optimal Negotiation Strategies for Agents with Incomplete Information. *Lect. Notes Comput. Sci. Vol.* 2333, 377–392. https://doi.org/10.1007/3-540-45448-9_28

[6] Raz Lin, Sarit Kraus, Tim Baarslag, Dmytro Tykhonov, Koen Hindriks, and Catholijn M. Jonker. 2014. Genius: An Integrated Environment for Supporting the Design of Generic Automated Negotiators. *Computational Intelligence* 30, 1 (2014), 48–70. https://doi.org/10.1111/j.1467-8640.2012.00463.x

[7] Ruben Stranders. 2013. Gaussian Processes for Java. https://github.com/rhomeister/gp4j.

[8] Dimitrios Tsimpoukis, Tim Baarslag, Michael Kaisers, and Nikolaos Paterakis. 2019. *Automated Negotiations Under User Preference Uncertainty: A Linear Programming Approach.* 115–129. https://doi.org/10.1007/978-3-030-17294-7_9

[9] Okan Tunalı, Reyhan Aydoğan, and Victor Sanchez-Anguix. 2017. Rethinking Frequency Opponent Modeling in Automated Negotiation. In *PRIMA 2017: Principles and Practice of Multi-Agent Systems*, Bo An, Ana Bazzan, João Leite, Serena Villata, and Leendert van der Torre (Eds.). Springer International Publishing, Cham, 263–279.

[10] Wikipedia. 2019. *Simplex algorithm.* https://en.wikipedia.org/wiki/Simplex_algorithm

[11] Colin Williams, Valentin Robu, Enrico Gerding, and Nicholas Jennings. 2011. Using Gaussian Processes to Optimise Concession in Complex Negotiations against Unknown Opponents. *IJCAI International Joint Conference on Artificial Intelligence* (01 2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-080

[12] Osman Yucel, Jon Hoffman, and Sandip Sen. 2017. *Jonny Black: A Mediating Approach to Multilateral Negotiations.* Springer International Publishing, Cham, 231–238. https://doi.org/10.1007/978-3-319-51563-2_18