BidBot: The Adaptive Negotiation Agent

Azra Oymaağaç[1], Hüseyin Acemli[1] and Emirhan Tandoğan[1*]

[1]Engineering Faculty, Ozyegin University

Abstract

This project centers on the design and enhancement of an automated negotiation agent within the NegMAS framework, specifically tailored to address the challenges outlined in the Automated Negotiation League (ANL) 2024. Our agent is designed to effectively handle bilateral negotiations with the Bidding, Opponent model, and Acceptance strategy (BOA) framework, without access to opponents' reservation values. Although automated negotiating agents employ a wide tane of negotiation strategies, the majority of these agents rely on just one specific strategy. However, it is well known that no single strategy is the best for every negotiation situation. Our strategic approach integrate dynamic adaptations based on opponent actions and evolving negotiation scenarios. This report describes our negotiation approach in depth, explains how it complies with the needs of the competition, and explores how it might offer accurate responses in the context of automated negotiations.

1. Introduction

The applications of automated negotiation systems in multi-agent settings has grown dramatically in tandem with developments in computational negotiation models and artificial intelligence. In this project, we describe the creation and deployment of an automated negotiation agent specifically designed for the Automated Negotiation League (ANL) 2024. Based on the NegMAS framework this project attempts to tackle the special problem that the league has presented: creating an agent that can negotiate bilaterally without having access to the opponent's reservation value, which is a crucial element of strategic negotiation preparation. Our work utilizes the components of the Bidding, Opponent model, and Acceptance (BOA) strategy framework to build an agent that learns from its opponents' evolving strategies and adjusts dynamically to maximize its own utility outcomes. This report unveils our comprehensive negotiation strategy that integrates innovative approaches to performance measurement, bidding, opponent modeling, and acceptance strategies. This introduction serves as a prelude to the intricate details of our negotiation strategy with the subsequent sections, we share more detail of all these components of our agent with the code explanations and quantify the performance of our agent.

2. High Level Description of Our Awesome Agent

Our agent BidBot which designed for specifically for the ANAC 2024 Automated Negotiation League is a sophisticated negotiation system built to manage the complexities of bilateral negotiation dialogues. The important component of our agent is strategic flexibility which is emphasized in the architecture's construction, enables it to adjust to changing negotiation dynamics and opponent behaviors.

System Architecture

Our agent employs a modular design consists of separate but interlinked components named the Bidding Strategy, Acceptance Strategy and Opponent Model. Each part is made to carry out specific tasks that when combined, aid in the agent's decision-making process during negotiating. It employs a sophisticated opponent modeling system which constantly updates its understanding of the opponent's strategies and reservation values. Because of this, our agent can make offers that are sensitive to the circumstances of the negotiation and properly calibrated. Using information from the Opponent Model, which is updated constantly with information on the opponent's tactics and reserve values, the Bidding Strategy modifies offers in real-time. In order to maximize utility, the Acceptance Strategy uses complex algorithms to assess if an offer is acceptable. The innovative Opponent Model plays an essential part in directing the agent's strategic replies by offering dynamic estimations.

Performance Measures

We use three carefully chosen performance indicators to assess our agent's effectiveness, which when combined provide a complete picture of its negotiating skills. Firstly, average Utility is the primary indicator, showing how consistently our representative closes favorable agreements in a variety of negotiation situ-

*Corresponding author: azra.oymaagac@ozu.edu.tr
Received: October 20, 2023, **Published:** December 14, 2023

ations [e.g., [3]]. Its ease of application and capacity to provide valuable insights into our agent's overall performance make it a favored choice. Average utility is calculated based on the following formula, in this case it only takes into our own agent's utility.

$$U_{\text{avg}} = \frac{1}{n} \sum_{k=1}^{n} U_k$$

Moving beyond the simplicity of averages, we use a more sophisticated indicator of negotiation effectiveness called the Distance to Pareto Frontier which will help us to understand how fair and optimal the negotiated point. A deal achieves Pareto optimality, when it cannot be improved for one party without simultaneously worsening it for another, thus representing the optimal scenario for both agents [2]. As this metric considers both agents' utility, it can be used to measure mutual benefit. Minimizing the distance to the Pareto-optimal frontier improves fairness and the probability of acceptance.

- U_a, U_b: utility of agents in agreement
- P_a, P_b: utility of agents in closest pareto frontier point

$$D = \sqrt{(U_a - P_a)^2 + (U_b - P_b)^2}$$

Finally, we consider the competitive context of the negotiations through our innovative Negative Opponent Utility metric. The motivation behind the creation of this metric is that our agent competes with other competitors' agents. Therefore, opponent utility should be considered as a negative, adverse component of our metric.

- U_a: Own agent utility
- U_b: Opponent agent utility
- γ: Number of agents in competition except our own agent
- N: Number of negotiations

$$U_{\text{neg}} = \frac{1}{N} \sum \left(\frac{U_a^{\gamma}}{U_b} \right)$$

3. Agent Components and Strategy

Bidding Strategy

Our bidding approach is a hybrid model that takes inspiration from the best practices in the field of negotiation agents. The bidding strategy aims to calculate the opponent's concession level for each offer. According to concession level, bidding strategy behaves like Boulware if opponent is enough concessive. Conversely, if the opponent does not seem close to conceding, we employ a Nice Tit-for-Tat strategy.According to concession level, bidding strategy behaves like Boulware if opponent is enough concessive. Recall for Tit-for-Tat and Boulware strategies:

$U_A(t, B \to A)$: utility of A in offer from B to A at time t

$$\Delta U_A(t) : U_A(t, B \to A) - U_A(t-2, B \to A)$$

$$\Delta U_B(t) : U_B(t, B \to A) - U_B(t-2, B \to A)$$

$$k : \text{hyperparameter}$$

$$\text{opp\_behaviour} = \begin{cases} \text{not Conceder,} & \text{if } \frac{k \cdot \Delta U_A(t) - \Delta U_B(t)}{k+1} < 0 \\ \text{Conceder,} & \text{if } \frac{k \cdot \Delta U_A(t) - \Delta U_B(t)}{k+1} \geq 0 \end{cases}$$

"Opponent behavior" indicates the behavior of opponent for certain time, doesn't indicate agent type. We will decide the counteroffer strategy by using the table 1. Recall that the Conceder agent concedes fast during the negotiation while the Boulware agent hardly concedes until the deadline. Tit-For-Tat agent can switch its strategy between these tactics stochastically [4].

Predicted Opponent model for unit time	**Applied model for unit time**
Conceder	Boulware(P1)
Not Conceder	Tit-For-Tat

Table 1: *Strategies based on opponent model prediction.*

Here is the pseudocode of bidding strategy:

Algorithm 1 Bidding Strategy

```
function BiddingStrategy(recent_opp_offer, past_opp_offer, k)
    opp_behavior ← k × (ufun(recent_opp_offer) − ufun(past_opp_offer)) − (opp_ufun(recent_opp_offer) − opp_ufun(past_opp_offer)) / (k+1)
    if opp_behavior > 0 then
        create_boulware_strategy(p1)        ▷ p1: curvature of function
    else
        create_tit_for_tat()
    end if
end function
```

BidBot Nice Tit-for-Tat Strategy: The BidBot Nice-Tit-For-Tat strategy adapts the traditional Nice Tit-for-Tat approach to consider the behaviors of the opponent. Unlike conventional Nice Tit-for-Tat strategy also taking the opponent's first bids into account, BidBot focuses on the opponent's last 5 offers instead. This adjustment is made to account for potential changes in the opponent's behavior over time, providing more accurate insights into their concession patterns. Besides, since BidBot is designed as a win-focused agent, instead of nash point utility calculation in the original formula, BidBot takes its previous and new target utility as the parameter. To determine the appropriate concession level for the next bid, BidBot calculates several key metrics:

Opponent Concession Analysis

- BidBot evaluates the opponent's concession behavior by computing the utility difference of the offers with the maximum and minimum utility by the opponent. This concession metric reflects the opponent's willingness to compromise during the negotiation process.

Concession Factor Calculation

- BidBot computes a concession factor based on the opponent's recent concession behavior and BidBot's current target utility. This factor ensures that BidBot adjusts its concession level dynamically in response to the opponent's actions.

Adjusting Bid Strategy

- BidBot determines its next bid using a uniquely developed bidding strategy, which takes into account the current negotiation time and BidBot's updated target utility. This strategy is designed to optimize BidBot's bid placement for maximizing utility while considering the dynamic nature of the negotiation process.

By leveraging these adaptive strategies, BidBot aims to maximize its utility and maintain a competitive edge in negotiations against diverse opponent behaviors.

Algorithm 2 Tit-For-Tat Bidding Strategy

```
function CreateBiddingTitForTat(state, t)
    opponent_last_bid ← state.current_offer if state.current_offer is not None else None
    my_utility_of_opponent_last_bid ← self.previous_ufun
    maximum_offered_utility_by_opponent ← max(self.opponent_ufun_in_opponent_offers)
    minimum_offered_utility_by_opponent ← min(self.opponent_ufun_in_opponent_offers)
    min_utility_of_opponent_last_bids ← min(self.ufun_in_opponent_offers[-5:])
    opponent_concession ← maximum_offered_utility_by_opponent - minimum_offered_utility_by_opponent
    opponent_concede_factor ← min(1, opponent_concession / (self.my_current_target_utility - min_utility_of_opponent_last_bids + 1e-12))
    my_concession ← opponent_concede_factor × (1 - self.my_current_target_utility)
    my_current_target_utility ← max(0.0, (1.0 - my_concession))
    my_current_target_utility ← min(my_current_target_utility, 1.0)
    move ← self.interval_bidding_strategy(t, my_current_target_utility)
    return move[0]
end function
```

Interval Bidding: After we have calculated the target utility for our agent. Also, we can control the opponent's utility because we can access the opponent utility function. This situation helps to create a technique which creates offers having different opponent utility by staying our agent utility almost same. In this technique, we set an interval centering target utility, and store all offers in the interval and order according to opponent utility by increasing order. Lastly, we need to choose an offer from sorted offers. We create a formula returning the index linearly increases according to time.

$$\text{index} = \text{nint}(\text{time} \times (\#\text{ of interval array}))$$

We use the index in order to return the desired offer. Also, in our design we set interval as 0.05 because we target approximately 100 offers for interval array.

$$2 \times \text{interval} = \frac{100}{\text{outcome spaces}} = \frac{100}{1000} = 0.1$$

$$\text{interval} = 0.05$$

Here is the pseudocode of interval bidding strategy:

Algorithm 3 Interval Bidding Function

```
function IntervalBidding(time, target_utility, interval, min_val, max_val)
    possible_outcomes ← list()                                   ▷ create a list
    for outcome in rational_outcomes do
        if target_utility - our utility of outcome < interval then   ▷ if outcome is within interval, then add to possible_outcomes
            possible_outcomes.add(outcome)
        end if
    end for
    sort(possible_outcomes, by=opponent_utility)                 ▷ sort list by opponent utility by increasing
    index ← round((max_val − min_val) ∗ time + min_val) ∗ (len(possible_outcomes) − 1))   ▷ decide index depending on time
    return possible_outcomes[index]                              ▷ return the offer
end function
```

Here is an example environment with interval(figure 1):

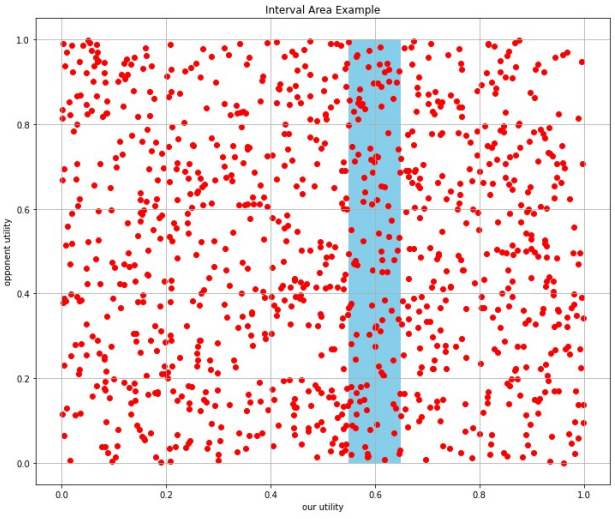

Figure 1: *Example environemnt with interval*

Opponent Model

One of the main critical point of our agent is our opponent model. The opponent model of our agent

utilizes a dual-estimation technique to precisely determine the adversary's concession strategy and reservation value in real-time. First, with every offer made throughout the negotiation, it improves the estimation of the opponent's reservation value. The model dynamically updates the reservation value to reflect the most accurate estimate by incorporating insights from the opponent's past behavior into each negotiating move through the use of a proprietary formula. Opponent's estimated reservation value is recalculated after each offering according to following formula:

- R: Reservation value
- U_B: Utility of B in B offer

$$R = \min(U_B, R)$$

Second, during the course of the negotiation, the model determines the degree of concession made by the other party. Our agent can determine whether the opponent is choosing a more yielding attitude or a hard-liner stance by using a specially designed algorithm.

Let $U_A(t, B \to A)$ be the utility of A in offer from B to A at time t.

$$\Delta U_A(t) : U_A(t, B \to A) - U_A(t-2, B \to A)$$

$$\Delta U_B(t) : U_B(t, B \to A) - U_B(t-2, B \to A)$$

Let k be the hyperparameter.
The type is defined as:

$$\text{Type} = \begin{cases} \text{Boulware}, & \text{if } \frac{k \cdot \Delta U_A(t) - \Delta U_B(t)}{k+1} < 0 \\ \text{Conceder}, & \text{if } \frac{k \cdot \Delta U_A(t) - \Delta U_B(t)}{k+1} \geq 0 \end{cases}$$

Reservation Value Prediction: In this competition, the opponent model is almost transparent because we can access opponent utility function, but we do not know opponent's reservation value. Therefore, opponent model must be aimed to detect reservation value as true as possible. We have developed a technique to predict reservation value. We store the minimum utility between opponent utilities for opponent offer for each step. After, our technique is based on the fact that usual agent must start high utility for itself, and it decreases over time. Therefore, we add a time dependent coefficient for minimum reservation value to increase accuracy of predicted reservation value. We are inspired by the gaussian function while creating our function, so we have named it as gaussian function in our context.

$$\text{res\_coef} = \frac{e^{(k \cdot \text{time})} - 1}{e^k - 1}$$

$$\text{res\_coef} \in [0, 1]$$

$$\text{predicted\_res\_value} = \text{res\_coef} \cdot \text{min\_res\_value}$$

We plot coefficient–time graph for different k values. (Figure 2)

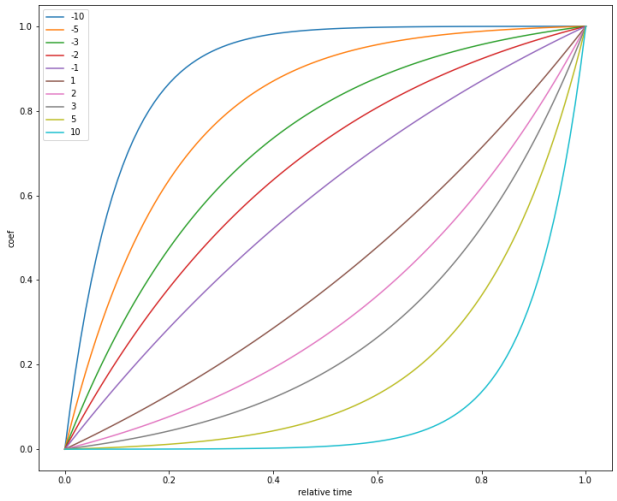

Figure 2: *Coefficient-time graph for different k values.*

According to experimental and heuristic information, when k >0 it is better predictor against strict agents, when k<0 it is better predictor against loose agents (like Conceder). As a result, we have decided to use k = 5 for our agent because we need to focus on strict agent. We can already deal with loose agents.

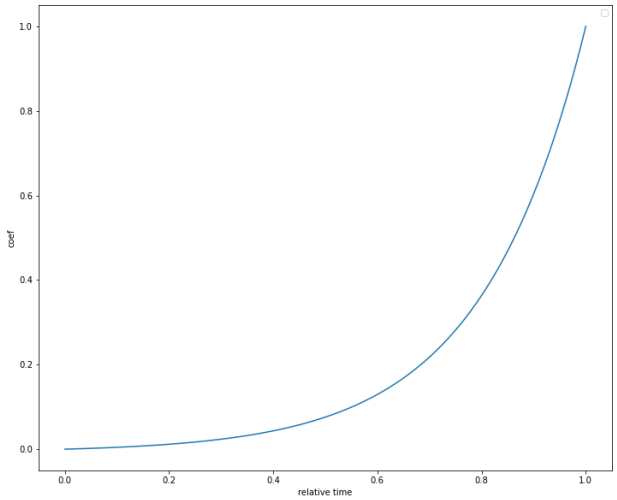

Figure 3: *Coefficient-time graph for k=5 values.*

Also, here is the pseudocode for partner reservation value:

Algorithm 4 Update Partner Reservation Value Algorithm

```
function UpdatePartnerReservationValue(opponent_utility)
    opponent_utility_history.add(opponent_utility)
    min_reservation_value ← min(opponent_utility_history)
    res_coef ← GaussianFunction(time, k)
    partner_reservation_value ← res_coef × min_reservation_value
    for all outcome in rational_outcomes do
        if opponent utility of outcome < partner_reservation_value then
            rational_outcomes.remove(outcome)
        end if
    end for
end function
function GaussianFunction(time, k)
    return (e^(k×time) − 1) / (e^k − 1)
end function
```

Acceptance Strategy

At the essential of our agent's decision-making process is the Acceptance Strategy. We use the AC_next strategy which is a highly preferred, simple, and effective acceptance strategy. AC_next performs well with a strong bidding strategy and is selected due to the dynamic nature of our opponent modelling and bidding strategy. If we create a good bidding strategy, AC_next also will be good because AC_next accept the opponent offer if opponent offer is better than our agent's future.

$$AC_{\text{next}} = U_A(t, B \to A) \geq U_A(t+1, A \to B)$$

Here is the pseudocode of acceptance strategy:

Algorithm 5 Acceptance Strategy

```
function AcceptanceStrategy(future_offer, opponent_offer)
    return ufun(future_offer) < ufun(opponent_offer)   ▷ check if our utility of opponent offer is higher than our utility of future offer
end function
```

Quantifying the Agent's Performance

A number of simulations and critical analysis went into the comprehensive empirical evaluation of our agent's negotiation prowess. We carefully recorded the agent's performance with 2450 negotiations spread over 10 scenarios between 7 competitors in order to identify the agent's strengths and shortcomings, which would guide future iterations of strategy improvement.

Initially, it was thought to take k parameter in self.bidding_strategy_formula as equal to 0, which means only considering opponent agent's current and previous utility when calculating the bidding strategy. Results showed in Table 2.

Then, it is thought that the time dependent k value might be logical, and the tests were rerun, and results are as shown in Table 3.

Agent Name	Score
BidBot	0.536955
Boulware	0.472314
RVFitter	0.422178
Linear	0.397354
NashSeeker	0.378827
Conceder	0.302175
MiCRO	0.267771

Table 2: *Results in tournament between default agents with 2450 negotiations, 10 scenarios and 5 repetitions when k = 0 where k in self.bidding_strategy_formula k * (ufun - prev_ufun) - (opponent_ufun - prev_opponent_ufun) / (k + 1)*

Agent Name	Score
BidBot	0.491486
Boulware	0.417739
RVFitter	0.356125
Linear	0.320371
NashSeeker	0.312097
MiCRO	0.242405
Conceder	0.224233

Table 3: *Results in tournament between default agents with 2450 negotiations, 10 scenarios and 5 repetitions when k = 1-state.relative_time where k in self.bidding_strategy_formula k * (ufun - prev_ufun) - (opponent_ufun - prev_opponent_ufun) / (k + 1)*

Taking the average self-scores taken in several repeated tests into account, it is observed that better results appear when k = 0 thus k is taken as 0 for the rest of the parameter decision process.

Agent Name	Score
BidBot	0.606405
Boulware	0.564752
RVFitter	0.501827
Linear	0.439011
NashSeeker	0.432746
MiCRO	0.422539
Conceder	0.315184

Table 4: *Results in tournament between default agents with 2450 negotiations, 10 scenarios and 5 repetitions when P1 = 2+opponent_behavior_decision where P1 in create_time_dependant_bidding(state, P1) when calculating target_utility target_utility = min(((1 - t) * 2 * P0 + 2 * (1 - t) * t * P1 + t * t * 2 * P2), 1)*

After that, when calling create_time_dependent_bidding function which leading target_utility calculation, the value of parameter P1 was investigated. And the best results taken when P1 = 2 + opponent_concession_behaviour value as shown in Table 4. P1 takes values between 2 and 3 in this case.

Lastly, the current model including reservation

Agent Name	Score
BidBot	0.461024
Boulware	0.397214
RVFitter	0.335579
Linear	0.304782
NashSeeker	0.300081
MiCRO	0.214231
Conceder	0.201193

Table 5: *Results in tournament between default agents with 2450 negotiations, 10 scenarios and 5 repetitions when opponent agent's reservation value is not estimated.*

value estimation with chosen parameters shown in Table 4, is compared with the model excluding opponent agent's reservation value estimation shown in Table 5. And it is obvious that reservation value estimation improves our agents' utility, so our agent is determined as the agent whose results shown in Table 4.

Table 6: *Results in tournament (Part 1)*

Agent Name	Score	Nash Optimality	Kalai Optimality	Max Welfare Optimality
BidBot	0.606405	0.821354	0.831740	0.907691
Boulware	0.564752	0.861969	0.865165	0.924275
RVFitter	0.501827	0.830389	0.833498	0.910037
Linear	0.439011	0.854445	0.855975	0.945891
NashSeeker	0.432746	0.895643	0.862144	0.903133
MiCRO	0.422539	0.719354	0.747475	0.753214
Conceder	0.315184	0.817812	0.816592	0.933264

Table 7: *Results in tournament (Part 2)*

Agent Name	Pareto Optimality	Negative Opponent Utility Score
BidBot	0.968390	0.690754
Boulware	0.977454	0.446760
RVFitter	0.973656	0.216368
Linear	0.993634	0.094948
NashSeeker	0.950255	0.086886
MiCRO	0.872394	0.074974
Conceder	0.989686	0.012125

The results presented in Table 6 and Table 7 suggest that BidBot outperforms default agents in ANAC 2024 across two key metrics: average utility ("Score") and negative opponent utility. This performance underscores BidBot's status as a win-focused agent, leveraging a unique blend of time-based and behavior-based strategies coupled with efficient reservation value and concession level calculations derived from opponent modeling and dynamic bidding strategies.

However, due to its win-focused nature, BidBot tends to perform averagely compared to other agents in terms of Nash, Kalai, Max Welfare, and Pareto Optimality metrics. Future enhancements aim to explore additional strategies to foster mutual benefit in scenarios requiring cooperation.

Future Perspectives

In order to make a negotiation in real life, our agent must have emotions and feel and analyze the opponent's emotion because human is not perfect creature and it acts with emotions, which is not usually most sense move. Therefore, if we will create a agent based on completely abstract and strict math formula, it does not give good results. Instead, we will create our agent emotion and understand the opponent emotion. Also, we communicate with our opponent emotionally. We need to add emotional intelligence to our negotiation agent's capabilities in order for it to function well in practical situations, going beyond strict mathematical calculations. The next stage is to create our agent with emotional awareness so that it can recognize and react to emotional stimuli. This will make it easier for the agent to comprehend and perhaps even anticipate the emotional undertones of negotiations, allowing for the development of more flexible and successful strategies.

Further enhancements will involve developing mechanisms for the agent to express emotions strategically. This capability will allow the agent to engage more authentically with human negotiators, potentially swaying the course of discussions by mirroring emotions or demonstrating empathy. One way to make encounters feel less transactional and more real is to demonstrate enthusiasm in reaction to a positive offer or to be patient throughout lengthy talks.

In the end, our agent will analyze information and interact with humans more like a human by fusing emotional intelligence with its current analytical skills. This will increase its usefulness in a variety of negotiation situations, from high-stakes business deals to routine interpersonal negotiations. Our agent will get much closer to the complex reality of human bargaining dynamics with the incorporation of these additional emotional skills.

Conclusion

Our agent is an example of creative thinking and flexibility in the field of automated negotiations, as it was created for the ANAC 2024 Automated Negotiation League. It is distinguished by its sensitive opponent modeling and flexible bidding strategy, which together guarantee customized answers to shifting negotiation dynamics. Our agent represents a strategic combination of assertiveness and adaptability, leveraging a sophisticated mix of strategies driven by performance metrics and ongoing recalibration.

All things considered, our time at ANL 2024 has been incredibly fulfilling. It has given us insight-

ful knowledge about automated negotiation and equipped us to make more contributions to the multi-agent system's development.

References

[1] T. Baarslag, *Exploring the strategy space of negotiating agents*, Springer Theses, 2016. https://doi.org/10.1007/978-3-319-28243-5.

[2] R. Bahgat, S. AbdelRahman, and G. Farag, "Order Statistics Bayesian–Mining Agent Modeling for Automated Negotiation," *An International Journal of Computing and Informatics (Informatica)*, vol. 33, issue 1, pp. 123-137, 2011.

[3] S. Chen, & G. Weiss, "An Efficient and Adaptive Approach to Negotiation in Complex Environments," In L. De Raedt et al. (Eds.), *ECAI 2012* (pp. 228-233). IOS Press, 2012. doi:10.3233/978-1-61499-098-7-228.

[4] R. Aydoğan, O. Keskin and U. Çakan, "Would You Imagine Yourself Negotiating With a Robot, Jennifer? Why Not?," in *IEEE Transactions on Human-Machine Systems*, vol. 52, no. 1, pp. 41-51, Feb. 2022, doi: 10.1109/THMS.2021.3121664.

[5] Tim Baarslag, Koen Hindriks, Catholijn Jonker, "Effective acceptance conditions in real-time automated negotiation," *Decision Support Systems*, Volume 60, 2014, ISSN 0167-9236, https://doi.org/10.1016/j.dss.2013.05.021.