Negotiation Agent

Team Members: Alex Taylor (aktaylo4), Galav Sharma (gsharma3), Jonathan Kurian (jgkurian)

1. Relevant Literature

We intend to use two papers to focus on, *Baseline Strategies for the ANAC Automated Negotiation League* by Kotone Ninagaw, Yasser Mohammad, and Amy Greenwall, and *The 13th International Automated Negotiating Agent Competition Challenges and Results* by Reyhan Aydogan, Tim Baarslag, Katsuhide Fujita, Holger H. Hoos, Catholjin M. Jonker, Yasser Mohammad, and Bram M. Renting. *Baseline Strategies for the ANAC Automated Negotiation League* references the approach to creating a negotiation strategy for a finalist agent for the Automated Negotiation League, and the performance of it. It addresses issues that are inherent to automated negotiation, and heuristics their agent used to remedy these issues. Using this paper as a reference, we can design our own heuristic for an agent to follow.

The 13th International Automated Negotiating Agent Competition Challenges and Results discusses the 2022 Automated Negotiating Agents Competition's set up and results, including the utility score and measures for each category for each agent. This included the method of testing, the mechanics for each competition, and the overall results of the competition. The results included the overall results, as well as the winner for each individual category tested, such as individual utility, opponent utility, social welfare, etc. As we're using the ANL's structure to create and test our agent, we can utilize the information here to create our scenarios, as well as evaluate the performance of our agent versus others.

Neither paper includes datasets or code. However, we are using Bram Renting, a previous winner of the ANL 2023 competition's, template repository for the Agent Negotiation League, as a template to create our agent as well as use example agents to test the effectiveness of our agent.

2. Problem Statement

At a high level we aim to address the problem of negotiation through the standards of an already existing competition, the automated negotiation league AKA the ANL. Specifically in our case we are looking at through the lens of someone who would have been competing in the 2023 competition and using the resources provided for them. The GitHub (https://github.com/brenting/ANL-2023-example-agent) for this competition provides a testbed for running an agent through a sample problem or through a tournament against other agents, as well as providing a suite of previous agents to compare to. These previous agents include those who competed in the 2022 version of the event as well as some random agents, stupid agents and

other baseline agents to compare against. We chose this because it allows us to have a metric to compare our agent to since this competition has already established credible rules and regulations that we can become a part of.

Before the negotiation, the agent receives a set of rules, an opponent, and a scenario for setup. This includes a domain and a preference profile for each agent, which is used to judge the utility of different results for an agent. This also includes information for which agent starts and sends out the opening offer. After this turns are taken where the agent can either accept the presented offer, make a counter offer, or end the negotiation (without reaching an agreement). This continues until a deal is made or a preset deadline. A score is then granted by the utility received at the end of negotiation, with 0 being the worst and 1 being the best. A tricky aspect of this problem is that the preference profile of the other agent is unknown to our agent, so we have to negotiate without that knowledge. There are two ways to evaluate the criteria, first is through the agent's individual utility and gain while the second is based on social welfare calculated through the sum of both agent's involved in a negotiation. To simplify, the first score is more competitive while the second score is more cooperative. We will be building our agent to maximize the first category.

The parts that make up this problem are first to develop a strategy to create a bid. Second, to determine whether to accept or reject a bid. Lastly, opponent modeling to make some prediction of the opponent's strategy. An additional stretch goal used in the ANL is a learning method between agent negotiation sessions, however we are focusing on the first three criteria in this project to keep our expectations managed.

With these conditions, and by using the Automated Negotiation League, we aim to see if providing additional information such as environmental information, such as time, as well as utilizing opponent information like previous bids and how much we perceive, will improve gains from negotiation.

3. Importance

This problem simulates real life situations where one must make a decision without complete information of the other parties. For this problem, we will create an agent with enhanced decision making and negotiation capabilities, allowing it to negotiate with other agents without complete information and make decisions about whether to accept the bid, reject the bid, or negotiate. Agents with negotiation and decision making abilities can have much broader applications like healthcare, business, finance, and law. We also can use this to understand how human computing can be

4. Addressing the Problem

To address this problem, we will take the following steps:

- We will start by researching and better understanding the ANL structure, scenario, rules, and evaluation provided by the Agent Negotiation League (ANL) of ANAC. We will look at the resources available, including the provided testbed.
- We will then build our agent with a utility function that allows it to generate a bid to ensure maximum gain, analyze incoming bids and offers and make decisions on whether to accept or reject bids. We will also include models to allow the agent to negotiate with the opponent and analyze bid patterns to understand opponent behavior.
- We will then run simulations of multiple scenarios against random agents, stupid agents, and other baseline agents included in the ANL testbed. We will evaluate performance during tests, and based on these results, we will tune the models and strategies to improve its performance.
- We will then evaluate the agent performance by creating a tournament structure to get its win rate against different types of agents included in the ANL testbed. We will evaluate the agent's performance by calculating the agent's individual utility and overall social welfare. The evaluation approach is written in further detail below.

To generate multiple styles of agents, we each decided to create an agent and compare how different ideas would create different results. Each agent was named directly after its creator so the first agent we will discuss is the AlexAgent.

AlexAgent

The intention behind this agent was to make it focus on having a stronger individual focus, especially towards the start of negotiations. The default opponent-model was used and no changes were made to that aspect, as we didn't have time to reach that stretch goal. Instead, the bulk of the change was made to the accepting offer and scoring bid logic. To start with, let's first discuss the logic behind the accepting offer function. Shown in the small code snippet below, the agent uses the progress to determine how far into the negotiation session the agents are, and then determines based on that how it should react. A progress of 0 would represent a negotiation session that's just begun and a progress of 1 would represent a negotiation session at its very end. Following that logic, a progress level of .25, .5 and .75 would be the first quarter point, halfway point and third quarter point of the negotiation session.

```
#Exponential changing min utility threshold
min_utility_threshold = 0.9 - (0.3 * (progress ** 2))
return self.profile.getUtility(bid) >= min_utility_threshold
```

There is an exponential change in how the utility of an accepted bid changes. You can see in the table below how the change is clearly not linear. Starting at the change from 0 to 0.25, there is barely a decrease in the accepted utility. From 0.25 to 0.5 the change is a bit bigger and then 0.5 to 0.75, it is even bigger. The largest change comes from .75 to 1 where the accepted utility drops almost half of the total decrease in only a quarter of the time.

Progress	**Min Utility Threshold**
0.0	0.9
0.25	0.881
0.5	0.825
0.75	0.731
1.0	0.6

The reasoning behind this change is to employ an early hardball method, to maximize our own gain. Early on we want only the best offers and won't settle. Any opponent will have to provide us higher offers in hopes we accept. As progress goes on we do want to come to a conclusion so we decide to be a bit more lenient but still will only go to an offer with .6 as the lowest score.

The other change below basically sets a floor so that for scoring we will weight our own utility at least 60% of the total score. As time goes on we try to take into account the opponent's utility more in hopes they accept but we don't want to compromise our own needs.

```
self_weight = max(alpha * time_pressure, 0.6)
opponent_weight = 1.0 - self_weight
```

GalavAgent

The intention behind this agent was to maximize its own utility while also trying to maximize social welfare, ensuring the agent seeks mutually beneficial offers while also maximizing its own utility. There were no changes to the opponent model and the bulk of the changes were made to the acceptance criteria and how the bid is scored. Let’s look at how the bid scoring is implemented. As shown in the code snippet below, the agent calculates scores using the agent’s own utility, the total social welfare, and a trade-off parameter between self-interest and altruistic behavior. The score calculates an initial bid score based on the agent’s

utility, alpha, and time pressure. It then computes the social welfare, which is the sum of the agent's utility and the opponent's utility, and adjusts it by scaling it by the alpha value used.

```
our_utility = float(self.profile.getUtility(bid))
time_pressure = 1.0 - progress ** (1 / eps)
score = alpha * our_utility * time_pressure

if self.opponent_model is not None:
    opponent_utility = float(self.opponent_model.get_predicted_utility(bid))
    social_welfare = our_utility + opponent_utility
    score += (1-alpha) * social_welfare * time_pressure
```

This was tested with multiple values, mainly 0, 0.25, 0.5, 0.75, and 1. The agent was then run against the ANL 2022 DreamTeam109Agent and Agent007 and the agent utility and social welfare was documented in the table below. Based on results below and from tournaments with other agents, I decided that an alpha value of 0.75 provided the best agent utility while maintaining a high social welfare.

Alpha Value	**DreamTeam109Agent**	**Agent007**
0	Own Utility: 0.47447 Social Welfare: 1.41979	Own Utility: 0.52820 Social Welfare: 1.46511
0.25	Own Utility: 0.45616 Social Welfare: 1.41430	Own Utility: 0.81257 Social Welfare: 1.39154
0.5	Own Utility: 0.81275 Social Welfare: 1.39154	Own Utility: 0.74071 Social Welfare: 1.34073
0.75	Own Utility: 0.83699 Social Welfare: 1.33090	Own Utility: 0.94725 Social Welfare: 1.32404
1	Own Utility: 0.89353 Social Welfare: 1.27872	Own Utility: 0.97575 Social Welfare: 1.28797

The other changes made were in the acceptance criteria. My agent adjusts its utility threshold based on the agent's predicted utility. The accepts an offer at a lower utility if the opponent's utility is higher than a threshold (I am using 0.75), as it is more likely to satisfy the opponent's requirements and maintain a high social welfare.

```
if self.opponent_model.get_predicted_utility(bid) > 0.75:
    return self.profile.getUtility(bid) > 0.8
```

Jgkurian_Agent

The intention with this agent was to use more learning capabilities in order to use things such as estimated time left, as well as how similar previous bids are, to strengthen the negotiation tactics. This agent saves the opponent's bid, the utility of each bid, as well as the time when offered. We aim to track the recent bids, in order to adjust our scoring procedure for each bid.

Our acceptance procedure is based on current progress, having a much steeper incline, with a minimum utility of 0.5 past 98%, in order to catch agents that would aim for completion, only lessening our conditions later than others began to drop. We also utilized current progress to modify the number of attempts, to try less sampling for a potential bid to save time and make more offers, to reach a higher utility score.

To find a new bid, we experimented with using our previous bids, that, when scored, had a high utility score and were weighted highly. Afterwards we used random sampling to find the highest scoring bid, if there was a better one than the previous few bids. This was to ensure we kept on track if our previous few bids were on the right track.

For scoring, we initially calculated a base score using our initial alpha at 0.95, and used time pressure to dynamically adjust thresholds as time passes. This agent aims to utilize the opponent's predicted utility, reducing the weight for it so we aim for mutually acceptable negotiations, while maintaining our own utility. One thing our agent aims to do is check for recent average utility, using the latest 3 opponent bids. If their utility is high, we aim to increase our own cooperativeness.

The aim of this agent was to use more complex learning and calculation techniques in order to add more utility and thinking compared to the original template agent. However, importantly, this reduced the number of offers made. In the table below, it shows the current Jgkurian Agent below's utility score, welfare, and importantly, count. With a low count of 7.875 per round, compared to the average offers of 1447.5625 from the DreamTeam109Agent we were testing against, a large issue was the inefficiency added by trying to enhance the opponent modeling with trend analysis of the opponent. This added a lot of thinking and calculations to the agent, which, despite aiming to use later time to pressure the opponent, could be easily exploitable due to taking too much time to calculate. Without an efficient way of tracking and storing opponent history, trying to calculate and adjust out acceptance threshold made it more difficult to track.

By removing the additional accept_condition code that tracked opponent modeling, and streamlining additional calculations, the agent was able to perform calculations far fast, raising o 298.0 offers on average, and thus increasing its own utility and nash scores. Scores against agents such as DreamTeam and TemplateAgent increased, going from an average of 0.353, to an average of 0.87 against TemplateAgent specifically. However, this leads to more failed agreements on average, as well as a lower overall social welfare. In order to improve this agent further, we would need to create more efficient ways of saving and loading information, as well as adjusting where calculations are made in order to be able to process offers faster.

	avg_utility	avg_nash_product	avg_social_welfare	avg_num_offers	count	agreement	failed	ERROR
JgkurianAgent 1.0	0.429595641	0.353932045	1.258055987	7.875	56	53	3	0
JgkurianAgent 2.0	0.672634159	0.409185633	1.218777081	298.0	48	41	7	0

5. Alternate Approaches

One alternative approach to creating an agent would be to utilize a rule-based approach to create a negotiation agent. This would involve creating a rigid set of rules to determine whether an agent accepts, rejects, or counterbids, with the counter bid based on its own set of rules. This approach would be easier to implement, and would well perform in static predictable environments. However, in negotiations this would not be beneficial, due to the variety of different negotiation strategies other agents can use, that would lead to erroneous decisions made. It also would need to be manually adjusted when learning new information.

A different alternative approach would be to use machine learning, or reinforcement learning, to train the agent over time. Using multiple trials for the agent to experiment and learn, with different parameters rewarding the agent, we can have the agent learn optimal strategies for more complex environments that would work for negotiations. However, this requires a lot of time to train, as well as not having a clear algorithm or decision making process. The current utility-based model is faster to make, and is more transparent when it comes to decision making, making it a better model in this case.

Those are the two ends of the spectrum for agent decision-making where one side is the rigid agent and one side is the highly flexible agent. But it may be more informative to talk about the actual implementation of the alternatives, in this case, what are some of the agents that exist in this competition. Some of the agents we compete against use more simple methods for the sake of comparison such as a random bidder/accepter agent. There's also a 'stupid agent' that just accepts the first offer given. There's a basic LinearAgent that concedes linearly with time. The

template agent that is started with only accepts an offer in the last 5% of progress time if the offer is better than a certain threshold. Some agents with novelty or gimmick ideas that we beat were part of the ANL2022 competition or the CSE3210 package of agents like the ProcrastinAgent while others in these competitions used formulas and learning methods that were trickier to overcome like the DreamTeam109Agent.

There's also some agents who provided better ideas and we weren't able to overcome them in certain 1v1s or tournaments. Some themes we saw include that two hardballing agents might not reach a conclusion. Another is that a more complicated agent is slower and there are less total offers made, which can affect reaching a decision in some scenarios. Some ideas that we would believe would encompass the ideal agent include:

- Learning opponent strategy and giving offers accordingly
- Changing the choose bid function to search the possible bid space more efficiently
- A time management aspect to change offers based on progress into negotiation
- Agent decision on whether to prioritize self gain vs social welfare
- Basing new offers based on opposition's recent offers

6. Evaluation

By creating a tournament structure, we can evaluate the agent's performance via "win-rate" against other agents, by evaluating who had the higher individual utility gain in each matchup. For the tournament, we evaluate the overall individual utility gain for our agent and the overall social welfare in matchups with our agent, calculated by evaluating the individual utility gain for both our agent and the opponent by adding them together. For agent evaluation, we use individual utility and social welfare to evaluate and tune our approach, and consider the tradeoffs to improving social welfare versus just individual utility.

Our goal was to evaluate our agents and ensure that the agents met our individual goals and expectations. For this, we ran a tournament with 15 agents, including our agents, simple ANAC provided agents, CSE3210 agents, and some ANL2022 agents. Each agent participated in 56 negotiations and was evaluated on the average utility of the agent, average social welfare, average number of offers, total number of negotiations, agreements, fails, and errors. The results are shown in the table below.

	avg_utility	avg_nash_product	avg_social_welfare	avg_num_offers	count	agreement	failed	ERROR
GalavAgent	0.82019624	0.43465615	1.33012164	413.10714	56	52	4	0
DreamTeam109Agent	0.787671247	0.399806996	1.276805651	1501.839286	56	51	5	0
AlexAgent	0.747150786	0.444487146	1.293532046	661.1071429	56	50	6	0
HardlinerAgent	0.725665757	0.285941445	1.013768164	4050.107143	56	41	15	0

Agent2	0.7045003	0.412857464	1.306362325	1527.196429	56	52	4	0
Agent11	0.703476959	0.426522849	1.288633198	1780.25	56	51	5	0
Agent3	0.682363506	0.433895457	1.305207522	2114.517857	56	51	5	0
BoulwareAgent	0.679294742	0.401008575	1.304027937	2783.767857	56	53	3	0
Agent007	0.629546178	0.401879385	1.257205167	873.4107143	56	51	5	0
LinearAgent	0.611485908	0.443625598	1.38555107	603.1785714	56	55	1	0
ConcederAgent	0.593977904	0.430959982	1.377179654	293.1607143	56	56	0	0
RandomAgent	0.524985763	0.368770859	1.159148995	1887.160714	56	49	7	0
Agent7	0.437931649	0.291726569	1.042744311	1212.5	56	47	9	0
JgkurianAgent	0.429595641	0.353932045	1.258055987	7.875	56	53	3	0
StupidAgent	0.305375631	0.287372026	1.168092744	2.357142857	56	52	4	0

Our results show that our agents performed well and met our goals for the project. Our agents performed well and achieved our individual goals. Our agents performed very well with average utilities of 0.82, 0.75, and 0.43, average social welfare of 1.33, 1.29, and 1.26, and an agreement rate of about 92.8%, 89.3%, and 94.6% for our agents.

For analyzing and evaluating our agents, we used a variety of agents such as the Hardliner agent (maximizes self-utility and does not concede), Boulware agent (prioritizes agreements and is reluctant to concede), Random agent (offers random bids till a bid with sufficient utility is offered), and other ANL agents such as Agent007 and DreamTeam109Agent. This allowed us to evaluate our agent's ability to adapt to different strategies and maximize its own utility. The results show high average utilities for our agents, indicating that our agents can successfully adapt to different negotiation strategies while maximizing its own utility. The high average social welfare indicates that our agents contributed to outcomes that were mutually beneficial and came to agreements with maximum utility, while also maintaining fairness. The high agreement rate shows the robustness of our agents in ensuring successful negotiations. These results suggest that our implementation for our agents was effective, allowing the agent to reach agreements with maximum utility and efficiency in diverse negotiation environments.

7. Summary and Results

Based on our results, it showed that going against cooperative agents that were able to adapt and negotiate, our agents performed well by incorporating time pressure as well as the opponent's predicted utility to reach a mutually beneficial agreement. However, there does

appear to be some gaps in utility to cover. For example, the Hardliner agent has a high average utility score, with a simple negotiation tactic of never compromising, despite such a tactic causing multiple failed negotiations. We found that against other agents from different ANL submissions, if they had better ways of storing information or learning metrics, they could make better bids in a timely manner, which other agents struggled with.

Taking that into account, an ideal agent would have an optimal way of storing and utilizing opponent data between negotiations, as well as additional metrics and adjustments to strategies used. Our strategies of taking into account time and adjusting our own perceived utility requirements performed well, but could be improved. Integrating learning will allow for future negotiation agents to perform well against agents that have unusual or unorthodox negotiation metrics, and importantly act closer to human behavior. Other aspects to focus on is adjusting and testing how score heuristics are calculated, with different strategies like utilizing social welfare in a straightforward manner performing well compared to different agents

In practice, our results imply that automated negotiation must take into account different environmental conditions, such as time and the other agent, for optimal outputs. To create successful agents, we needed to address different problems that exist in automated negotiation that represent different factors in real-life negotiation, such as identifying the opponent's strategy, understanding the time remaining and the opponent's tendencies outside of a decided strategy. By refining and creating different heuristic strategies, we can identify successful solutions applicable both in automated learning, as well as in real life, allowing for a better understanding of human and computer behavior.

References

1. Kotone Ninagawa, Yasser Mohammad, and Amy Greenwald. 2021. Baseline Strategies for the ANAC Automated Negotiation League. In Appears at the 3rd Games, Agents, and Incentives Workshop (GAIW 2021). Held as part of the Workshops at the 20th International Conference on Autonomous Agents and Multiagent Systems., London, UK, May 2021, IFAAMAS, 9 pages.
2. Aydogan R., Baarslag T., Fujita K., Hoos H. H., JONKER C. M., MOHAMMAD Y., Renting B. M. The 13th International Automated Negotiating Agent Competition Challenges and Results // Studies in Computational Intelligence. 2023. pp. 87-101.