CSE3210 Project 3: Negotiation
Automated Agent Implementation Report

Group 15
Junwon Yoon 5024722
Justin Jo 5047463
Kwangjin Lee 5464609
Hongwoo Cho 5028647

1. Introduction

This report outlines an exercise in collaborative AI focused on negotiation, as part of CSE3210 coursework. This report focuses on creating a negotiating agent using the GeniusWeb framework.

2. Agents

The team's negotiating agent, *'Group15Agent'*, was developed in file '*group15_agent.py'* under the *'agents/group15_agent/'* folder, using the *'TemplateAgent'* as the basic skeleton. The group focused on developing the agent's ability to find the next bid based on a heuristic, and assessing the received bid to decide whether to accept the bid or not.

2.1. Organization of Source Code

The source code of the developed agent and the required files for it are developed under *'agents/group15_agent/'* folder. The final version of the agent is implemented in *'group15_agent.py'* file, and the previous versions of the agents can be found in the files '*group15_agent_v1.py'* and '*group15_agent_v2.py'*.

Under *'agents/group15_agent/utils'* folder is the *'group15_opponent_model.py'*, which is the opponent model that the agent uses to predict the opponent's score on a bid.

The agent was tested against different agents by running differently configured files, namely *"run.py"*, *"run_tournament.py"*, and *"run_agent15_versus_all.py"* under the root directory.

2.2. Agent Structure

The agent's overarching structure can be classified into four key sections: Initialization, Action Handling, Capabilities and Description, and Action Implementation.

In the Initialization section, the agent starts by recognizing several key parameters such as its domain, profile, last offered bid, last received bid, and opponent model. It sets the *concession_threshold*, which dictates how much the agent is willing to compromise on its positions. It also maintains a list of previously offered bids to ensure the same bid is not made again.

In Action Handling section, the method *notifyChange()* manages all communications from the negotiation server. It classifies incoming messages and responds accordingly: it sets up the agent's settings for the negotiation session on the *'Settings'* message, processes the counterpart's offer on the *'ActionDone'* message, decides upon an action to be taken on *'YourTurn'* message, and finally, properly closes the agent on the *'Finished'* message.

In the Capabilities and Description section, the agent (using *getCapabilities()* & *getDescription()* methods) responds to the GeniusWeb negotiation platform about its functional capabilities and provides a brief description about itself.

Lastly, in the Action Implementation section, the agent uses multiple helper methods to generate the next action (using *my_turn()* method). The functions in this section like *find_bid(), select_bids(), score_bid()* and opponent*_score_bid()* drive the crucial decision-making process of the agent.

find_bid() generates the next bid the agent will propose. If it is the very first bid, then it randomly generates 500 bids and chooses the one with the highest heuristic score. For subsequent bids, *select_bids()* is used to generate a subset of bids with a similar utility to the previous bid, applying a larger amount of concession each time if no suitable bids are found. Then the best bid is found by following the trade-off strategy. A more detailed description about this procedure could be found in section 2.3.2.

select_bids() is used to select all bids that have utilities that are larger than the criterion utility and have never been offered by the agent yet. Criterion utility is

calculated by subtracting the concession from the utility of the last offered bid.

score_bid() calculates the heuristic score for a bid, considering several parameters such as the agent's utility and the negotiation's time progress. The score is calculated by *score = time_pressure * our_utility*. The constant *time_pressure* dictates the time-dependent concession strategy that was implemented in the agent: this gets larger as the negotiation progresses in order to give higher scores for the same bid later in the negotiation. This is to make the agent gradually reach towards an agreement. More detailed description on the implementation of this feature is elaborated in section 2.3.1.

opponent_score_bid() is similar to *score_bid(),* however it predicts the rating of a bid from the perspective of the opponent. This function uses an opponent model to predict the utility of a bid for the opponent, and combines this with the same *time-pressure* factor used in *score_bid()*.

These functions collectively empower the agent to evaluate, generate and offer effective bids throughout the negotiation process, thereby enhancing its decision-making capabilities.

2.3. Bidding Strategy

In order to optimize the agent, the group has decided to test different bidding strategies: the team has considered 'Random Walker', 'Time dependent concession strategy', 'Trade off', and 'Tit for Tat'.

Random walker strategy was not chosen because although the random walker strategy prioritizes reaching agreements quickly, it may result in suboptimal outcomes for the agent.

Tit for Tat strategy was also not implemented because in the given negotiation settings, the values for each issue are discrete values (strings) not numeric values, and therefore it was difficult to mirror the opponent's action.

2.3.1. Time-Dependent Concession

Time dependent concession strategy adapts each agent's behavior depending on the time progress. This strategy is needed in order to make sure that the agent reaches an agreement at the end, even though the agreement might not be as good as the bids in the beginning. The closer the agent reaches the deadline, it should prefer to concede and meet an agreement with the opponent.

This strategy was implemented in two methods - *'score_bid'* and *'opponent_score_bid'* methods. The two methods take constant epsilon (*'eps'*) as its input and use it to make changes in the time pressure. The epsilon represents the inverse of beta value that determines which tactic the agent will take for its time-dependent concession. If the epsilon value is 1, it means that the rate of utility change over time will stay constant over time. If the epsilon value is high, it means that the agent will concede very easily after some time has passed. In contrast, if the epsilon value is low, it means that the agent will barely yield and will only concede when the deadline is very close.

Initially, the group set a relatively large epsilon value of 2 and iteratively decreased it until the optimal value of 0.1 was identified. This epsilon value struck the best balance between the probability of reaching an agreement and maximizing average utility. Therefore, the group has chosen to settle on this value as it represents the point where the trade-off between these two factors is most favorable.

2.3.2. Trade-Off

With a trade-off strategy, the agent starts considering not only its utility but also its opponent's utility.

Trade-off strategy was chosen to be the group's agent's main strategy for finding bids. *'opponent_score_bid'* method predicts how much the opposing agent favors a bid. The agent selects a list of bids that have the same (or similar) utility values as the previously offered bid by using *'select_bids'* method. Among these bids, the agent offers the bid that is predicted to be the most favorable by the opponent.

An important change that was made during the testing and development stage was the implementation of the *'select_bids'* method and the negotiation parameter. By gradually using larger concession values until the bids are found, the agent could explore more bids to offer, and this resulted in the increase of the agent's ability to reach an agreement. The choice of the concession value that will be added for each step was also crucial; this value was set to 0.05 initially and was optimized later (see section 4).

2.4. Acceptance Strategy

For the acceptance strategies, the team has considered 'AC_{const}', 'AC_{next}', and 'AC_{time}'. The team did not choose one and apply only that strategy, but rather combined them together to build a superior agent that gives better results.

The performance of the agent under different acceptance strategies was analyzed using a self-created scoring model to demonstrate the performance of the agent in different scenarios through charts and tables. The results for each test were given by running *'run_agent15_versus_all.py'* file, which runs the group's agent against all other agents that are given by the project. These results were processed and averaged, and were analyzed thoroughly by the team members.

2.4.1. AC_const

The AC_{const} strategy entails accepting bids with utility greater than a predetermined constant value. During the testing phase, the team initially set this constant value to

0.9. However, it was observed that bids below this threshold were consistently rejected, leading to fewer negotiations reaching an agreement. Consequently, this constant value was adjusted to 0.8, resulting in a higher number of agreements and an acceptable average utility of 0.8.

The AC_{const} was not chosen as the final acceptance strategy because if the opponent offers a bid with higher utility than this constant value, the agent accepts it, which overlooks opportunities for higher utility bids later in the negotiation process. This ultimately limits the agent's ability to fully explore the negotiation space and maximize utility. Furthermore, only having this condition made the agent very "greedy" with having a lower chance to reach an agreement at the end.

2.4.2. Accept_next

An accept condition of AC_{next} is an acceptance strategy where the agent accepts a bid when the opponent's bid is better than the upcoming bid. The team also tested the agent with AC_{next} strategy. However, it was observed that because the agent offered bids that favored itself more, this condition was rarely met and most agreements were made by the opponent accepting the agent's offer. Therefore, AC_{next} was not chosen as the final acceptance strategy as it did not function as a good acceptance strategy just by itself.

2.4.3. Accept_time

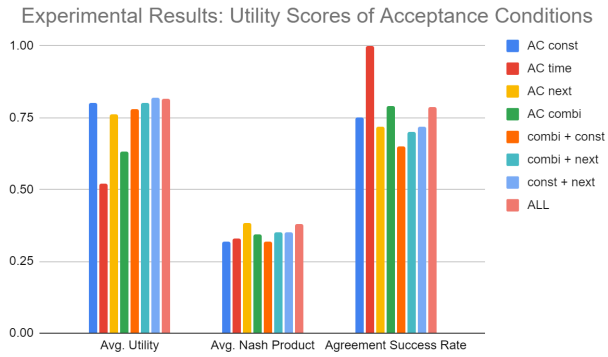
An accept condition of AC_{time}(T) is an acceptance strategy where the agent accepts bids when the progress has passed a certain time T. During the testing phase, T was initially set to 0.95. Testing with only AC_{time} strategy gave a successful negotiation rate of 100%. However, as this strategy accepts all bids eventually, the final average utility was very low. Therefore, this acceptance strategy was not chosen as the final strategy.

2.4.4. Accept_combinated

To tackle this problem, the group combined AC_{time} and AC_{const} to give AC_{combi}(α, T). After trying different combinations of α and T, it was found that AC_{combi}(0.6, 0.95) displays a most balanced result, yielding satisfactory values for both average utility value and agreement rate. However, as this strategy alone discarded the possibility of reaching a better agreement before T = 0.95, more combinations of different acceptance conditions were explored.

2.4.4. Chosen Acceptance Condition

After testing with the multiple accepted strategies 10 times, the average values of utility, Nash product and agreement success rate were plotted in the following bar chart.



This bar chart shows the average results for the following acceptance strategies: AC_{const}, AC_{time}, AC_{next}, AC_{combi}, $AC_{combi} + AC_{const}$, $AC_{combi} + AC_{next}$, $AC_{const} + AC_{next}$ and ALL. "+" means OR in this case: as an example, combi + const is an acceptance strategy where the agent accepts bids with utility greater than 0.8 OR the time is greater than 0.95 with utility higher than 0.6. Finally, ALL is an acceptance strategy which is a combination of AC_{const}, AC_{next} and AC_{time}.

As the average utility, average Nash product, and average agreement success rate are fundamental in determining the quality of the acceptance strategy, the acceptance strategy with the highest sum of these values was chosen as the final acceptance condition, in which the result was ALL acceptance combination.

2.5. Parameter Optimization

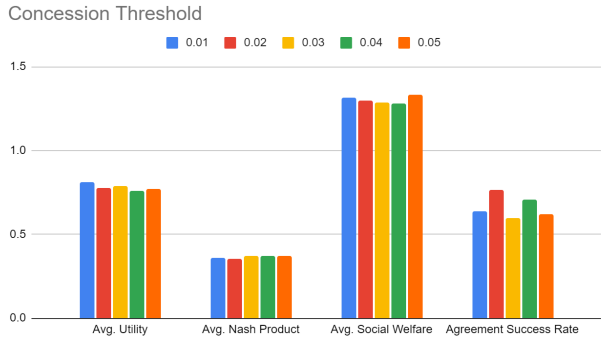
In order to optimize hyperparameters discussed in the description, a performance comparison was conducted with different combinations of values. The test was conducted by using the same procedure as used for deciding the acceptance strategy (section 2.4). The results were analyzed and used to optimize parameters and identify strong and weak points of the agent.

While the number of negotiation instances varied across tests, no significant differences were found, leading to the exclusion of this factor from the comparison criteria.

2.5.1. Concession Threshold

Initially set at the default value of 0.05 as per the implementation step, the concession threshold was experimented with by decrementing it by 0.01 increments. Further reductions below 0.01 were deemed insignificant as the agent was unsuccessful to reach an agreement in most of its negotiations. Analysis revealed that utility, Nash Product, and social welfare exhibited similar trends.

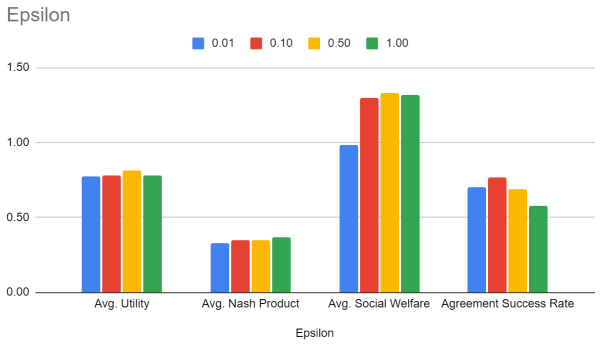
Notably, when the concession threshold was set to 0.02, a substantial increase in the agreement rate to 77% was observed. Consequently, the concession threshold was set at 0.02 for subsequent tests.



	0.01	0.02	0.03	0.04	0.05
Avg. Utility	0.81	0.78	0.79	0.76	0.77
Avg. Nash Product	0.36	0.35	0.37	0.37	0.37
Avg. Social Welfare	1.32	1.30	1.29	1.28	1.33
Agreement Success	0.64	0.77	0.60	0.71	0.62
Avg no. of negotiati	127.94	183.71	170.50	192.05	169.68

2.5.2. Epsilon

As previously discussed, epsilon value impacts time pressure. Leveraging the default value of 0.1, higher and lower values were tested. Similar to the concession threshold, no significant changes in utility, Nash product, and social welfare for each input were observed. However, the default value of 0.1 yielded overall better results, showcasing a 77% agreement rate. Given that increasing epsilon led to a decrease in the agreement success rate, it was concluded that further increments were unnecessary. Likewise, the same rationale applied to lower values. As a result, Epsilon is set at 0.1.



Epsilon	0.01	0.10	0.50	1.00
Avg. Utility	0.78	0.78	0.81	0.78
Avg. Nash Produ	0.33	0.35	0.35	0.37
Avg. Social Welf	0.99	1.30	1.33	1.32
Agreement Succ	0.70	0.77	0.69	0.58
Avg no. of negot	171.21	183.71	113.75	126.62

3. Performance

The team ran the agent against all the provided agents by running the *'run_tournament.py'* file. Below is the result of the session.

	avg_utility	avg_nash_	avg_social	avg_num_of	% of agree	ERROR
Agent2	0.747	0.431	1.359	2601.99	97%	0
Agent26	0.746	0.414	1.295	2205.35	93%	1
Agent24	0.717	0.401	1.267	1893.89	92%	0
Agent27	0.714	0.423	1.309	405.97	94%	0
HardlinerAgent	0.711	0.275	0.988	4815.49	72%	0
Group15Agent	0.703	0.385	1.280	167.07	82%	0
Agent32	0.702	0.397	1.237	562.60	89%	0
Agent52	0.698	0.438	1.334	3357.03	94%	0
Agent11	0.687	0.437	1.280	2102.07	90%	0
BoulwareAgent	0.680	0.407	1.306	3512.80	94%	0
AVG	0.616	0.387	1.235	1592.47	89%	0.125

Out of 32 agents that participated in the tournament, the team's agent had the 6th highest average utility with 82% chance of reaching an agreement.

In reviewing the results of multiple negotiation sessions with various negotiation agents, a number of strengths and weaknesses have become apparent in regards to the team's own agent, Group15Agent. A detailed analysis of these findings yields valuable insights into our agent's performance and provides significant information for future development and improvements.

3.1. Strengths

The agent's average utility stands at 0.703, securing a steady 6th rank position out of all the agents considered. Considering the average of 0.601, the team perceived the agent's performance as very promising. This suggests that Group15Agent is notably successful in reaching agreements that are correspondingly beneficial for itself.

Furthermore, the group found that Group15Agent demands a significantly smaller number of offers than the average to arrive at a consensus. This suggests that the agent's ability to find bids that can be favorable to its opponent is highly effective in the negotiations that reached an agreement, leading to highly efficient negotiation processes.

3.2. Weaknesses

The weak point of the agent is when targeting a model that continuously demands a fixed value, achieving an agreement becomes challenging. Referencing the trace chart in Appendix A, the *'HardlinerAgent'* stands out as a prime example. This agent consistently presents identical values for the same issue across rounds, except two

instances where it provides two values for Issue E. Consequently, the utility of a model presenting varying values diminishes progressively, leading to a failure to reach an agreement.

This could be seen as the agent's tendency to not concede to agents that are too self-interested, but also as a weakness in opponent modeling because the agent misses the opportunity to assess bids that are similar to the opponent's bid that also fits the agent itself's interest.

4. Further Improvements

To tackle the weak point of the agent, the agent's opponent model could be developed specifically to discern recurring patterns of consistent value propositions, as the current opponent model does record values but just uses these to estimate issue weights of the opponent.. By comparing these patterns with the opponent's behavior, strategies can be developed to the optimal value for mutual benefit.

Taking this further, the agent could record the negotiation sessions with other opponents and use the data to learn about the opponent after the negotiation is finished. This improvement in opponent modeling could benefit the agent for later negotiations with the opponent, as the agent could use the past data to offer a bid that is more appealing for the opponent. For the data recorded after a negotiation session that did not reach an agreement, the agent could find what the opponent's interest was, and can aim to reach an agreement in later sessions by offering bids that can relate to the opponent's predicted interest.

These changes could improve the agent's weaknesses, which is expected to result in more agreements reaching an agreement. Also, because the agent will have more understanding of other agents, this could positively affect the average Nash product and social welfare score, benefiting both the agent itself and the opponent.

5 Individual Contribution

Junwon Yoon has contributed in the project by performing the following tasks:

- Implementation of the Time-Dependent Concession strategy
- Optimization of different parameters (e.g. epsilon, concession threshold)
- Collection and quantification of performance test results

Justin Jo has contributed in the project by performing the following tasks:

- Implementation of the trade-off and tit-for-tat negotiation strategy
- Implementation and preparation of the performance analysis of the agent against other agents
- Collection and quantification of performance test results

Kwangjin Lee has contributed in the project by performing the following tasks:

- Implementation of the tit-for-tat strategy
- Performance testing and optimization

Hongwoo Cho has contributed in the project by performing the following tasks:

- Implementation of the tit-for-tat strategy
- Performance testing and optimization

All unspecified tasks are done in coordination.