CS-451 Report
Automated Negotiation Agent
Group-1

Mustafa Alper Sayan
Ertugrul Ozvardar

1. Introduction

a. Description

This project is about developing an automated negotiation agent using the GeniusWeb environment. In this assignment, negotiations are also called as negotiation sessions. Those negotiations are going to be run on the SHAOP protocol that is an extension of SAOP protocol. In this protocol, the first party initializes the negotiation with a bid which is observed from the other agents. Then, three actions can be performed in a turn taking clockwise fashion such as accepting a bid, making a counter bid, and ending the negotiation without reaching an agreement. Additionally, agents can implement information requests to obtain more information about the utility space and develop their own partial profile. This can be done by using ElicitComparison and Compare actions provided by the SHAOP protocol. GeniusWeb interface consists of profiles and parties to compare our agent against. It handles the communication between agents and reports the outcome as a graph. GeniusWeb also allows to monitor which bids are exchanged between agents. The most significant part of this assignment is that each agent has incomplete information about their opponent.

2. Agent Structure

a. Our agent consist of 5 classes:

1. **FrequencyOpponentModel:** This method is used to estimate the opponents' preference profile. It takes domains as an input and each time our agent gets a bid we update it to get closer utility values to the actual preference profile of the opponent. Methods of this class are as follows

    1. **getUtility():** Takes bid as an input, if there are no bids to compare it outputs 1, otherwise it asks for its helper method getFraction() to count value occurrences and sums them up, then divides this number to total number.
    2. **getFraction():**Takes issue and value as inputs, if the total number of the bids are 0, it gives an output 1, otherwise it gets the frequency of the asked issue value and divides it to the total number of bids.
    3. **Update():** Takes bid as an input, copies old bid frequencies and updates the frequency model according to the entered bids' issue values. Then, returns the updated frequency model.

2. **SimpleLinearOrdering:** This is the given class, therefore not any further explanation is needed.

3. **Node:** Keeps track of each issue value mainly used attribute is mean weight sum which helps to give importance to issue values. Attributes => valueOfIssue (Issue value), weightsum(sum of weights), count, meanWeightSum (counts the number of occurrences of this value)

    4. meanWeightSumComparator: Compares the mean weight sum between two nodes returns 1, -1 or 0

4. **Graph:** Creates a map of issues and nodes, initially empty filled using self_update() method. It creates an empty map using domain knowledge for each issue and its corresponding issue values.

    5. **Self_update():** Used for estimating importance of issue values, using mean weight sum of the nodes (i.e. fills the empty importance map using the mean weight sum attribute of nodes.) Entered list must be ordered with the best bid placed at the last index. For each issue value, update weight is increased by 1 in order to give more chances to likely better bids. The next step is to calculate the weight of each node, this step results in calculating the mean weight sum of each issue. In the next step, each node is sorted according to their mean weight sum. Finally, it standardizes the mean weights by substituting minimum mean weight from each node. It keeps the results in an updated graph.
    6. **getImportance():** Takes bid as an input. Then, it traverses the graph nodes and finds the issue values corresponding with the bid and sum mean weight of each issue value. As a final step, it outputs the importance of input bid.

5. **MyAgent:** Basis of our agent class. Its attributes are listed below:

7. Profile Interface: Profile object, used for communication between GeniusWeb and configuration of relevant information.
8. Party ID: Configured using profile interface is our agents ID.
9. Progress: It provides the required notification for the agent about time, and GeniusWeb interface about which action we took.
10. SimpleLinearOrdering estimatedProfile: Using profile interface has bid ordering and is able to estimate bid utility. Useable for elicit comparisons.
11. **Frequency Opponent Model:** As described in the previous sections, it is used for estimating opponents' preference profile and is able to provide bid estimation; our tests showed that it is able to better estimate opponents utility than simple linear ordering without using elicitation cost.
12. **Graph:** This is described in the previous sections. Used for our agents' preference model calculation using the ordered bids given by the simple linear ordering. It can perform better estimations than simple linear ordering according to our tests.
13. Random: Used for the initial window of the negotiation before our preference profile with all bids are created.
14. AllBidsList bidspace: It contains all the bids available in the domain.
15. my_bids = It includes all the bids available in the domain with their corresponding importances.
16. **opponentBids_and_utility:** It keeps all the bids sent by the opponent, and corresponding utility is not ordered.
17. **offered_Bids:** It contains all the bids offered by our agent.
18. **ordered_bids:** It keeps track of entire bids with an ordered structure according to our importance.
19. **ordered_oppent_bids:** List of the opponents' bids in an ordered fashion according to their utility. (i.e. highest utility first)
20. **satisfying_bids:** Changes in each time window and each size asked. Bids that are between 2 bounds. More explanation will be provided in the remaining parts.
21. best_bid: It corresponds to the best bid in the ordered bid list. (First element)
22. worst_bid: It corresponds to the worst bid in the ordered bid list. (Last element)
23. reservation_bid: The highest utility bid according to our preference profile that is sent by the opponent.

24. index: This does not serve a purpose for the party domain. However, it can be used for larger domains to keep track of the ordering process.
25. counter : If any of the bids are accepted by our bidding strategy, it will offer best consecutive bids using an ordered bids list. This parameter is used for traversing the whole ordered bid list until it reaches end by incrementing its value.
26. **how_much_to_concede:** This is the value that is used to determine concession rate detaily. It will be given in the method explanation.

ii. MyAgent Methods

1. **notifyChange():** It takes information as an input, if information is settings, it configures profile interface, id, progress, bidspace (All available bids), frequency opponent model, estimated profile, graph. If information is an action, 4 options are possible. First one is offering a bid, offered bid is put into lastReceivedBid, and frequency opponent model is updated using this bid, if it is the best received bid, reservation bid is updated. Second option is making a comparison; it helps to update the estimated profile. Third option is our turn. It will be explained in the future methods, else the negotiation is finished.
2. **getCapabilities():** This function returns the protocols which can be used for this agent.
3. **getDescription():** It returns the agent description.
4. **myTurn()**: It returns “accept” if a bid is accepted by our acceptance method, or calls offer_bid method.
5. **offer_bid()**: One of the subparts of our bidding strategy. This function divides the strategy according to time. Further explanation will be given with respect to time.
    a. Time = Initial phase (<0.1) => Offer semi random bids by calling offer_semi_randomBid() method. Create an ordered bid list by calling generate_bids() method, call getMaxBid(), generate ordered bid list, set how much to concede by calling the method, initialize the counter. By using this step, we partially keep our preference profile undefined for our opponent. This results in not letting our opponent exploit our preference profile but also minimizing the error rate of our opponent model by filling it with more bids.
    b. Time = 0.1 => Create ordered opponents bid list.

    c. Time > 0.1 to time to last bid: Offer according to conceding ratio by calling get_threshold_and_bids() method and generate_best_bids_according_to_time() method details will be given in the method explanation.
    d. Time = Last round => Offer the reservation bid.

6. **get_threshold_and_bids():** calls generate_satisfying_bid_list() method according to the threshold set. Threshold is set by number of bids that fall between the upper concede level and lower concede level. Higher satisfying bid size means that the agent conceded more. An example as follows:
    a. Best bid importance = 50
    b. Worst bid importance = 10
    c. range = best bid importance-worst bid importance=40
    d. range / constant = concession range

   This value is updated for each time window and is dynamic, meaning it changes with satisfying bid size. Results in satisfying bids, from these bids our agent is going to make its offer by calling generate_best_bid_according_to_time().
7. **generate_satisfying_bid_list(upperbound, lowerbound)**: Generate bid list that falls between upper and lower bound.
8. **generate_best_bid_according_to_time(bids_to_choose_from) :** Offers the bid that is best for our opponent from satisfying bids. The satisfying bid must not be already offered.
9. **generate_bids():** Generates ordered bid list according to their importance and then sorts the bids. Results in best bid first and worst bid last. For the party domain consisting of approximately 3k bids, this worked fine. But with domains with higher bidspace, a time constraint may need to be implemented.
10. **isGood(Bid bid):** This method changes with time, and will be explained accordingly.
    a. Time < 0.484 => If the agent has ordered opponent bid list, call greater_utility_than_x_number_of_bids()
    b. For the last round accept the given bid, a deal is better than having no deal
11. **greater_utility_than_x_number_of_bids(int number_of_bids_to_consider, Bid last_received_bid):** According to the last ‘x’ bids sent by the opponent, return true if the opponent sent the best bid among last ‘x’ (i.e. 10, 5) bids and the bid has greater utility than 0.7
12. **getMaxBid():** This function returns the highest importance bid.
13. **sortByValue(HashMap<Bid, Double> hm):** Sorts the hashMap according to the value set; best bid first
14. **get_worst_bid():** Basically, this function returns the worst bid.

15. **get_best_bid():** Basically, this function returns best bid.
16. **setHow_much_to_concede():** This function is for determining the conceding factor. It simply makes a calculation by subtracting best bid importance from worst bid importance then dividing this number by some constant.
17. **offer_semi_randomBid():** This function returns a bid that is greater than the threshold value.

3. Explanation Of the Negotiation Strategy

a. Calculating the importance of a bid

This step is done by using graph class. Graph class takes partial bids as an input which are ordered from worst to best.
Give increasing weights ranging from 1 to size of the given bid list after each bid increases this weight by 1.
The second step results in higher weight for the better bids and lower weight for the worse bids. Increase the count by one for the current value.
For each node (issue value) mean weight sum is calculated.
Sort nodes according to their mean weight sums.
For each node, subtract the minimum mean weight sum to standardize the nodes.
These steps results in appointed importance for each node (issue value object)

b. Time-based conceding Strategy

i. Get the best bid importance = Best_bid_imp
ii. Get the worst bid importance = Worst_bid_imp
iii. Calculate the difference = Best_bid_imp - Worst_bid_imp = range
iv. Divide range by some constant = Range / constant = concession rate
v. This constant value is set to 20 by using tests and intuition. Dividing range with 8 would mean considering all bids due to relying on a time based strategy.

c. Upper bound & lower bound calculation

i. Initially upper bound is equal to the best bid importance
ii. Initially lower bound is = upper_bound
iii. How many bids to output must be set i.e 15
iv. Until specified, number of bids can be generated between range [upper, lower]
    1. Increase constant by 0.1
    2. Lower_bound = upper_bound - (constant * concession rate)
    3. If specified number of bids are achieved, then return bids

d. **Choosing bid from upper and lower bound**
   1. From the given upper and lower bound get the possible bids
   2. From these possible bids offer the bid that has the highest utility for the opponent (increases likelihood of acceptance).
   3. Add the offered bid to offered bids list and do not offer again

e. **Choosing how many bids must be between lower and higher bound**
   i. Higher number of bids between ranges means we are making more concessions because we are trying to send the best bid to the opponent. Generally, this means going out from our preference profile.
   ii. Our strategy aims to increase this number as negotiation reaches a sweet point. This point for the party domain is around 0.70 - 0.75, thus our agent purposes to take place around that range.

f. **Upper bound & lower bound update mechanism**
   i. This strategy can be shown successfully on an example
   ii. Set how many bids must be in range = 10
   iii. Upper bound = 50 => This number represents the importance
   iv. Lower bound = 50 => Initial
   v. Number of bids between [upper, lower] = 0
   vi. Decrease lower bound by = upper - (constant * concession rate)
   vii. Constant += 0.1
   viii. Repeat until 10 values fall in range
   ix. Number of bids required to be in that range changes with every time window the minimum number required must be 10 for 200 total rounds. Anything greater than 10 is our concession.

g. **Acceptance strategy**
   i. There are 3 possible outcomes depending on time that comes out from acceptance strategy
   ii. We consider last 10 bids sent by our opponent and if the last received bids utility greater than 9 of them and has higher utility than 0.7, we accept them
   iii. We consider last 5 bids sent by our opponent and if the last received bids utility greater than 4 of them and has higher utility than 0.7 we accept them
   iv. We have reached the last negotiation round and we accept no matter what, because agents are compared according to their mean utility. And no acceptance means 0 utility.

h. Time window division

We did not include any heuristic based time division and our time division tuned for 200 rounds. This can be improved by heuristics but we did not had enough time to implement this part therefore we went for constant time window division, this is one of the parts our agent can be improved.This also goes for concession rate calculation which is tuned for party domain but is able to run in other domains but might not give the desired results

i. Used heuristics

For calculating upper and lower bound, it is estimated that higher bids that fall in this range means we conceded more.
Calculating the importance, it is estimated that bids are considered as whole when giving weights but each of them have their own importance.
Choosing the best bid from upper and lower range, it is expected that a higher utility bid from this range is more likely to be accepted.

4. Strong Points And Weak Points Of Our Agent

a. Strong Points

- Does well under preference uncertainty.
- Does not ask for elicitation cost in any part of the negotiation hence does not decrease its utility.
- By using frequency opponent modeling it able to identify best bid enough times
- Does well against uncomfortable domain preferences meaning opponents preference profile is different from ours.
- Adaptive to time changes in the negotiation

b. Weak Points

- Only takes opponents preferences after it generates possible bits the agent finds satisfying
- Does not send offered bids multiple times which can be problematic given that acceptable bid range widens as time passes.

5. Tests throughout the development phase

a. Giving concrete development improvements is rather hard because of this report being written after the development phase has ended. But is explained below

b. Initially our agent gave bids with decreasing order only meaning it gave best to worst bids in order, opponent modeling was not used, and acceptance strategy was consent at accepting only bids that are higher than 0.9

c. The second improvement we made was conceding at a linear rate, meaning our agent put the bids in order best to worst and let's say for hundred rounds, for the first 10 rounds it offered best 10 bids then it offered 50th best bid to 60th best bid so on and so forth, still does not take into consideration of the opponents bids, and still constant acceptance

d. The third improvement was adding constant upper and lower bound, it generated bids according to this threshold, and offered the best bid among possible bids according to the opponents preferences, the acceptance strategy was to accept the received bid was better than the last 20 bids.

e. The fourth improvement was dynamic threshold appointment meaning generating x number bids between threshold range and among them offer the best one according to the opponent modeling. Acceptance strategy was, for a period accepted if better than the last 10 received bids. After some period accept if better than last 5 bids.

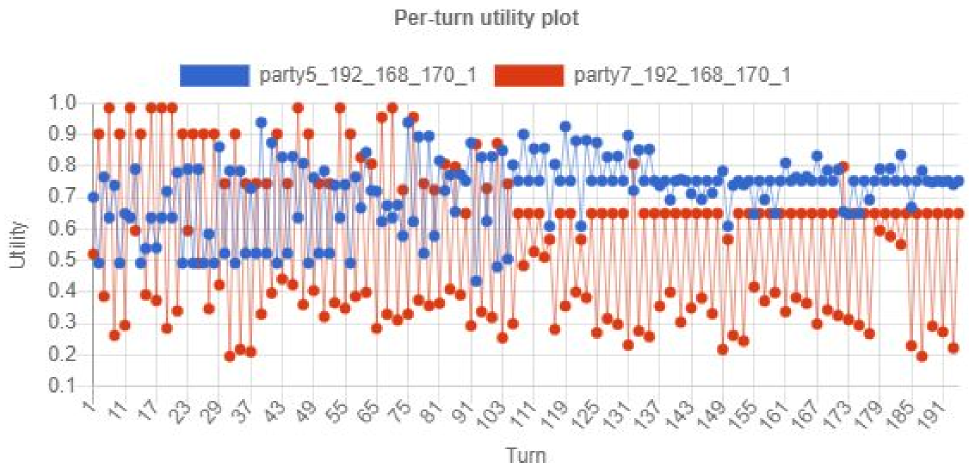
f. The fifth improvement was and last are as follows
    i. Generate bid according to between threshold range and also higher than thresholded opponent utility
    ii. Among them choose the best one and this should not be already offered
    iii. If it is the last round offer reservation bid
    iv. Accept if in current time range better than 10 last bids offered by the opponent
    v. After some time consider last five bids
    vi. If last round accept the offer

g. Some problems we encountered

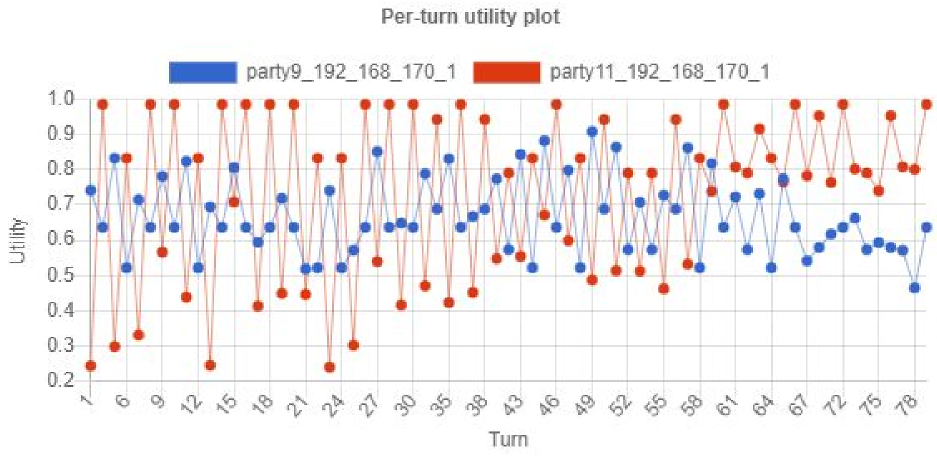
1. Strategy always offers same bids, solution is to do not send already offered bids
2. Accepting too early, increase the number of last bids to consider and set a constant low threshold
3. Not generating enough bids to sent, increase threshold range
4. Conceding too much, give lower constant threshold values to opponent lower utility value threshold
5. Not estimating nash, not general for all domains estimate nash by taking average importance.

6. Test results against agents

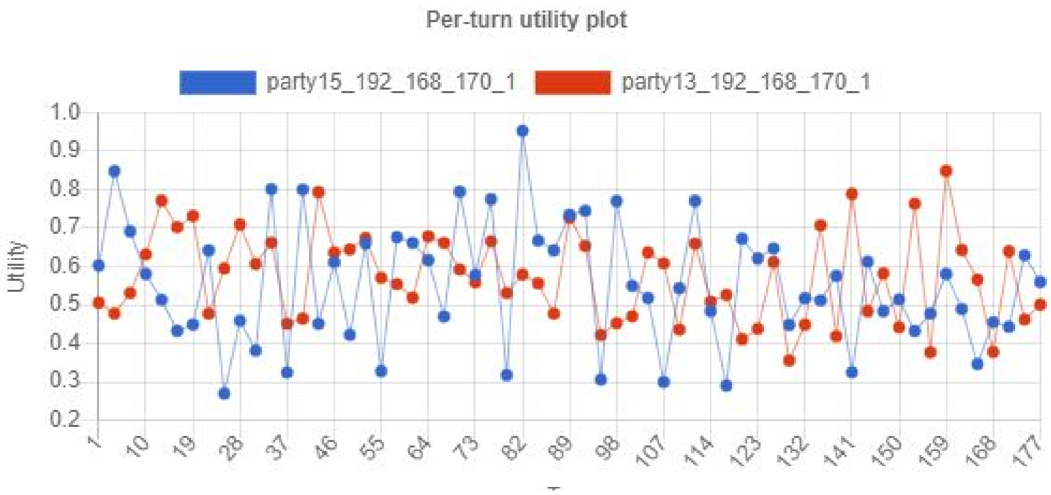
a. Red = AgentGG, Blue = My Agent, Rounds = 200, Profile = Party



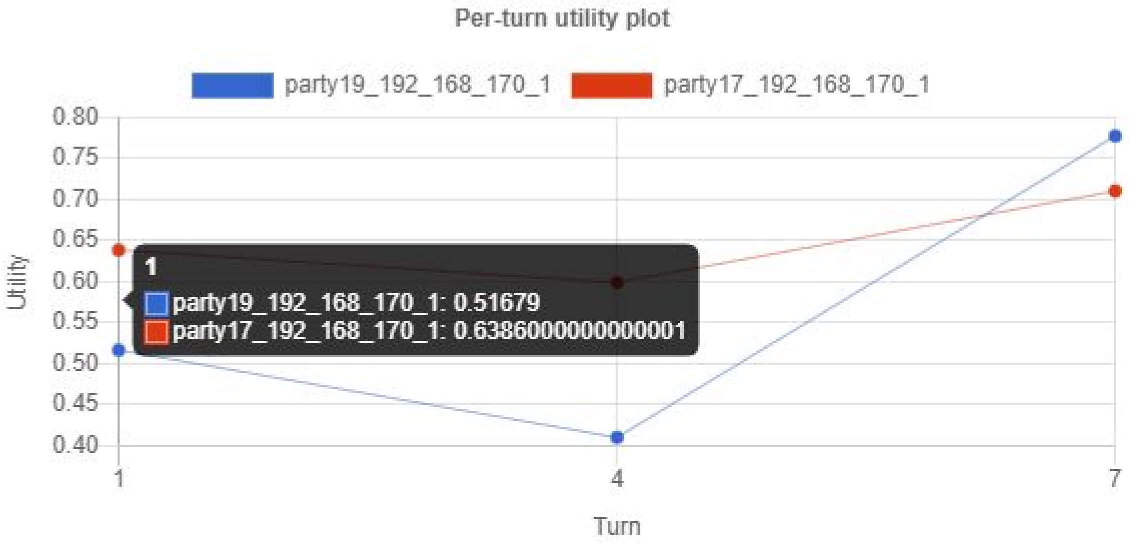
b. Red = AgentGG, Blue = MyAgent, Rounds = 200, Profile = Party



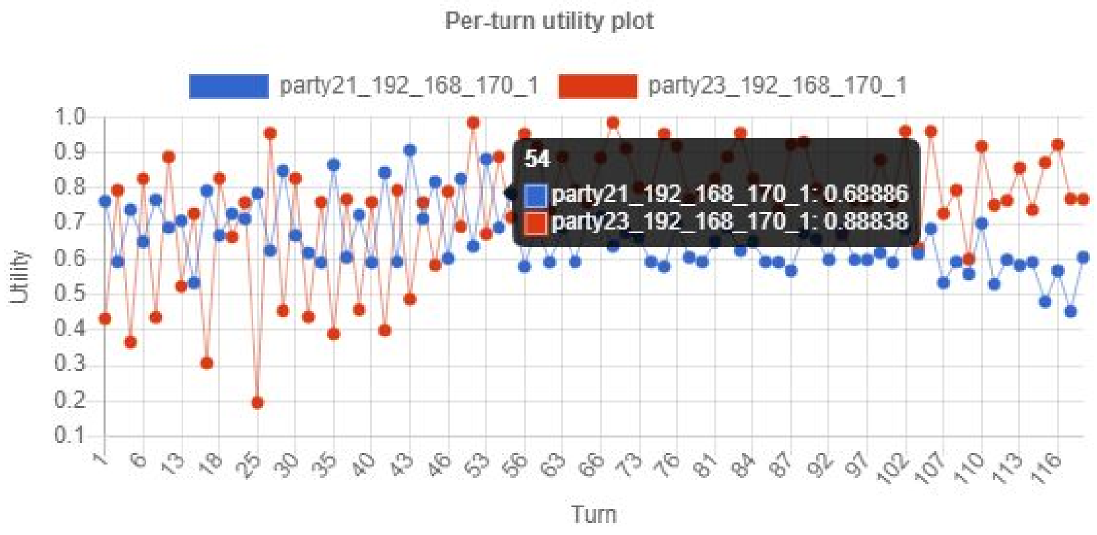
c. Red = SimpleShaop, Blue = MyAgent, Rounds = 200, Profile = Party



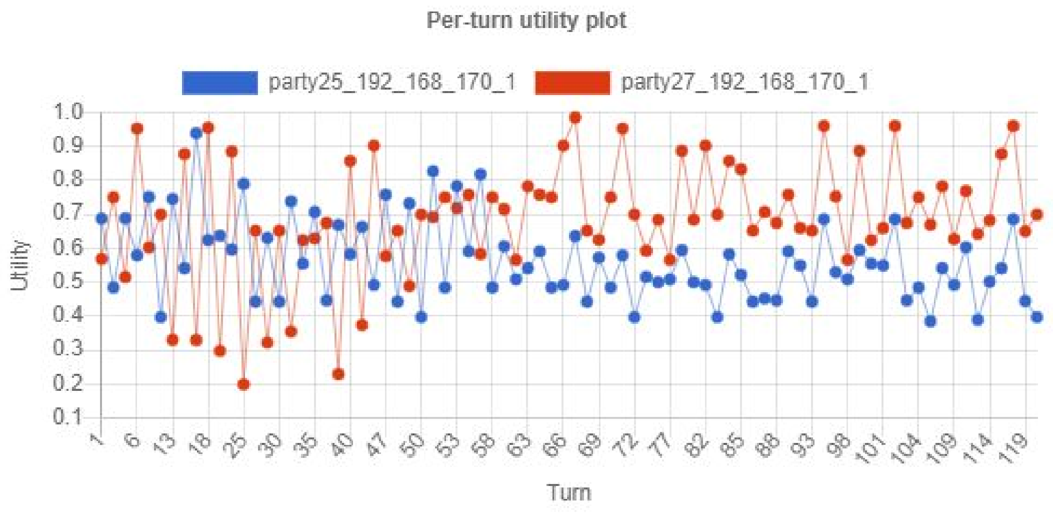
d. Red = SimpleShaop, Blue = MyAgent, Rounds = 200, Profile = Party



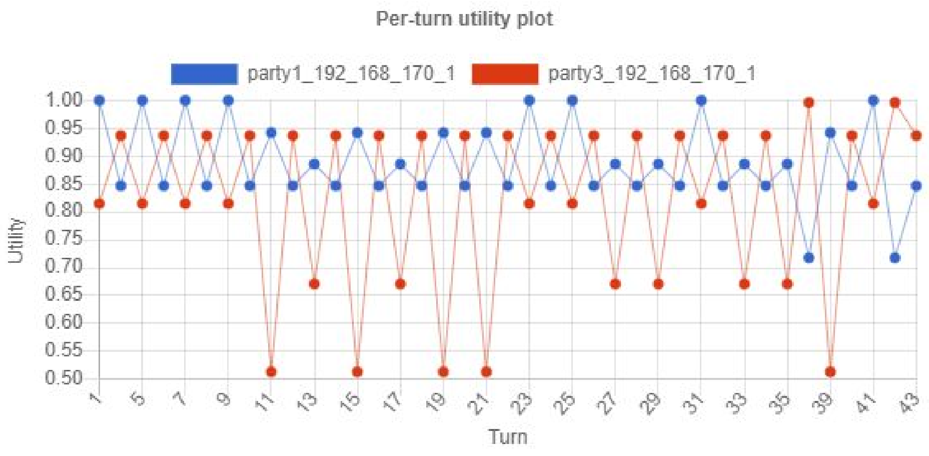
e. **Red = WinkyAgent, Blue = MyAgent, Rounds = 200, Profile = Party**



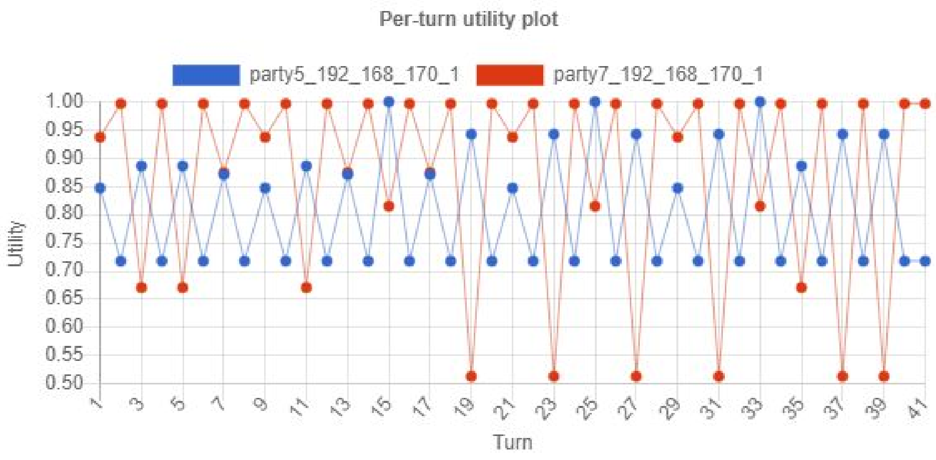
f. **Red = WinkyAgent, Blue = MyAgent, Rounds = 200, Profile = Party**



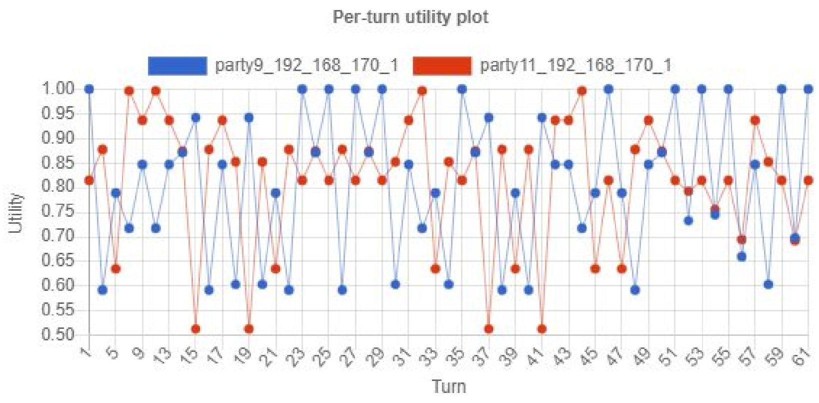
g. **Red = AgentGG, Blue = MyAgent, Rounds = 200, Profile = Laptop**



h. **Red = AgentGG, Blue = MyAgent, Rounds = 200, Profile = Laptop**



i. **Red = MyAgent, Blue = WinkyAgent, Rounds = 200 Profile = Laptop**



7. Improvements

a. Better nash point estimation, our agent is tuned for party domain and in other domains might not give the desired outcome

b. Dynamic size and opponent utility low point estimation should be done not through constant rather it should be done by calculations for each domain

c. For large domains, because of the while(true) loop usage might not terminate

d. Better opponent utility estimation, in our current agent the agent uses normal frequency estimation, this should be rather done by dividing the frequency opponent model through set sized windows and these window divisions should be calculated dynamically.

e. Our acceptance strategy should be extended by a ratio of round, as it stands it is fixed sized.

8. Conclusion

To conclude, in a desired agent, there should be no constant and everything should be calculated through methods but as it stands it would take too much time to code to implement every feature of a truly automated negotiation agent, thereby we calculated as much as we could through this methods but for this agent to be applicable in every domain the constant parameters that we chose should be replaced by the generated ones. **It should be noted that we used AgentGG`s preference profile estimation tools** the tool was actually able to much more but we did not used those parts, the reason was simple linear ordering does not offer much knowledge of our preference profile and it is actually a mathematically simple method, once we tried it for our tests we concluded that we should use AgentGG`s impMap and impUnit tool. We used impunit to its full extent. For impmap we only used it for our preference profile calculations although it offered much more capabilities.