# CS451 - FINAL REPORT

Kerem Gure, Baris Ozbas, Cenker Karaors
Group 4

## Table of Contents

# 1. Introduction

In today's world it is becoming an essential for applications to have/use AI techniques in order to maximize its output. In this project we have developed an AI agent that is able to negotiate with the other party on behalf of its user. Even though the domains and the issue weights in this project are fixed a simple modification to such domains will allow for the behaviour explained before. The agent uses SHAOP protocol( Stacked Human Alternating Offers Protocol), also ElicitComparison is used from COB party library provided by GeniusWeb. Throughout the development process of the agent we have used the below tools and coding languages:

- Intellij IDEA
- Visual Studio Code
- OpenJDK 1.8
- Genius Web Framework

This report contains design approaches, algorithms used in developing a negotiation agent on GeniusWeb Framework.

# 2. Design Approach

This section contains the PEAS table and details about the design ideas and approaches taken in the development of the agent.

| Performance Measurement | Environment | Actuators | Sensors |
|---|---|---|---|
| Number of deals made, gained utility in those deals. | Opponent. | Offers, Accept, Reject. | SHAOP |

The above table shows performance measurement, environment, actuators, sensors of the agent. Below these elements are discussed briefly:

- Number of deals made: Since our agent's main aim is to close the deal, the number of deals made shows the ability of the agent to fulfill its aim.
- Gained utility in the deals: How well the agent closes a deal is as important as making the deal itself.
- Opponent: The agent is only aware of the opponent is trying to make a deal with.
- Offer, accept, reject: The agent only has 3 abilities that allows it to talk with the opponent.

- SHAOP: Stacked Human Alternating Offers Protocol

### 2.1. Bidding Approaches

Given x amount of partial bid space to the agent, when an offer is received; firstly, the opponents importance model is updated which is a frequency based modeller that keeps the frequency of each issue-value pair that is received from a bid, these frequencies are later then used when trying to generate a bid to offer. Moreover, the agent tries to calculate the given bid utility from the known bid space, if the bid is not present in the bid space,therefore bid utility could not be calculated, the agent sends an ElicitComparison request with the received offer and the current bid space to get the utility of the bid for the agent. If the utility of the given bid is high enough that is acceptable, the agent then replies with Accept of the received offer.

### 2.2. Offering Approaches

The approach of the agent is to close the deal without cannibalizing its own utility and to generate offers that are thought to be important to the opponent. That is, given x amount of partial bid space to the agent, when the offer received is not acceptable, the agent tries to find a bid that is both suitable for the agent itself and to the other party. For this reason, the agent tries to find a bid in the known partial bid space that its utility is a bit higher than the received bid and the importance of the bid for the opponent is higher than the lowBoundaryOpponentOffer, and if the agent can not find such a bid, it tries to find a bid in AllBidsList via getting each bid from the AllBidsList one by one, making elicit comparison request with those bids to get the utility for the agent, if the utility is greater than lowBoundaryOffer and the importance of this bid for the opponent is greater than lowBoundaryOpponentOffer then the bid is offered. Finally, if no such bid is found the agent offers its highest utility bid.

## 3. Implementation

This section contains details regarding the implementation of the design ideas described in the previous section.

### 3.1. Bidding Implementation

When the session starts an empty opponent model is constructed with all the issue-value pairs and is updated when a bid is received after that the agent checks if this bid in the agent's bid space. If not, the agent makes an ElicitComparison request and updates its bid space with the received bid. After which, the agent compares the received bid’s utility with the lowBoundaryAccept

parameter and if the comparison is equal to or bigger than the lowBoundaryAccept parameter, the bid is accepted.

```
if(!orderedBidSLO.contains(lastReceivedBid) && !(lastReceivedAction instanceof Comparison)) {
                action = new ElicitComparison(me, lastReceivedBid, orderedBidSLO.getBids());
                getConnection().send(action);
                return;
            }
```

The above code snippet shows how the agent chooses to do an ElicitComparison request as the way explained above.

## 3.2. Offering Implementation

Given a received bid that is not acceptable, the agent iterates through the known bid space to find a bid that its higher than lowBoundaryOffer and the importance for the opponent is higher than the lowBoundaryOpponentOffer if no such bid is found in the known bid space then AllBidsList is used to fetch the bids that the agent has not yet experienced after that, these bids utilities are gathered using ElicitComparison. However one may notice that, until such a bid is hopefully found in AllBidsList the agent may spend too much time that would pass the limits of the round.Therefore, the agent calculates how much time it spends on ElicitComparison and if the time surpasses 3 seconds then ElicitComparison is stopped and the agent tries to find a bid in its known space and again if no such bid is found the agent offers the bid with the highest utility.

```
if (isGood(lastReceivedBid))
                action = new Accept(me, lastReceivedBid);
            else {
                while(allBidsCopy.size() >0) {
                    Bid tmp = allBidsCopy.get(0);
                    allBidsCopy.remove(0);
                    if (orderedBidSLO.getBids().contains(tmp)) {
                        System.out.println("INSIDE IF 1");
                        System.out.println("TMP UTILITY: " + orderedBidSLO.getUtility(tmp).doubleValue());
                        System.out.println("RECV BID: " + orderedBidSLO.getUtility(lastReceivedBid).doubleValue());
                        if (orderedBidSLO.getUtility(tmp).doubleValue() > orderedBidSLO.getUtility(lastReceivedBid).doubleValue()) {
                            System.out.println("LAST OFFERED: " + lastOfferedBid);
                            if ((!tmp.equals(lastOfferedBid)) && orderedBidSLO.getUtility(tmp).compareTo(lowBoundaryOffer) >= 0) {
                                System.out.println("NOW OFFERED: " + tmp);
                                action = new Offer(me, tmp);
                                lastOfferedBid = tmp;
                                break;
                            }
                        }
                    } else {
                        action = new ElicitComparison(me, tmp, orderedBidSLO.getBids());
                        getConnection().send(action);
                        return;
                    }
```

The above code snipped shows how the agent decides what to offer to the opponent using its known bidspace and AllBidsList with the help of ElicitComparison requests.

### 3.3. Reasons of Design Approaches

Our vision for the agent is to make deals. Therefore, we have chosen a simple yet effective approach for evaluating received bids. This approach allows the agent to be more flexible with which bids it can accept because it only considers a low boundary of which it can not accept.

As for generating offers we have chosen to with something a bit more probabilistic and which is frequency based modelling and shuffling the bid space when iterating through. The modelling enables the agent to guess what might be important for the opponent in order to generate deals that are most likely to be accepted.

## 4. Experiments

This section contains ideas and aims for the agent's development and it's future work.

### 4.1. Test Setup

We have conducted our test using the genius web app. In our tests, depending on the domain we have chosen to tweak the partial parameter depending on the size of the domain.The reason why we have chosen to alter the partial parameter is that the agent is partially ordered.The tests are conducted on a 12 core CPU is 32 gigabytes of RAM.

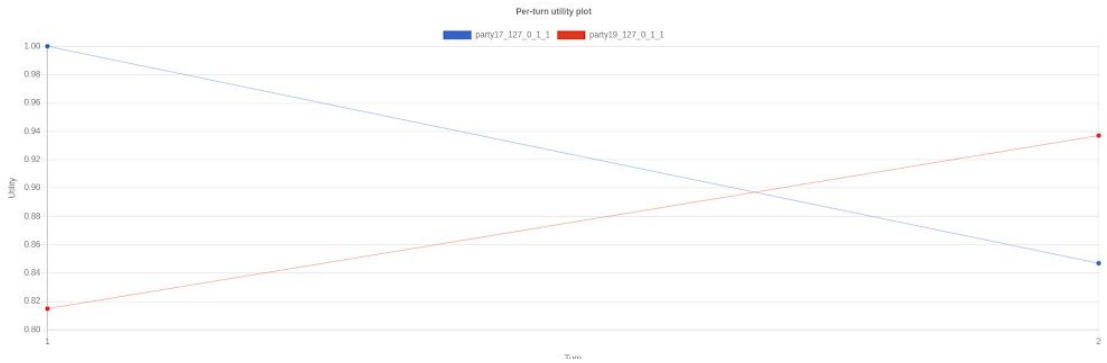
The below sections will go into detail about the tests that we have conducted using our agent and other Agents from the ANAC 2019.

### 4.2. Results

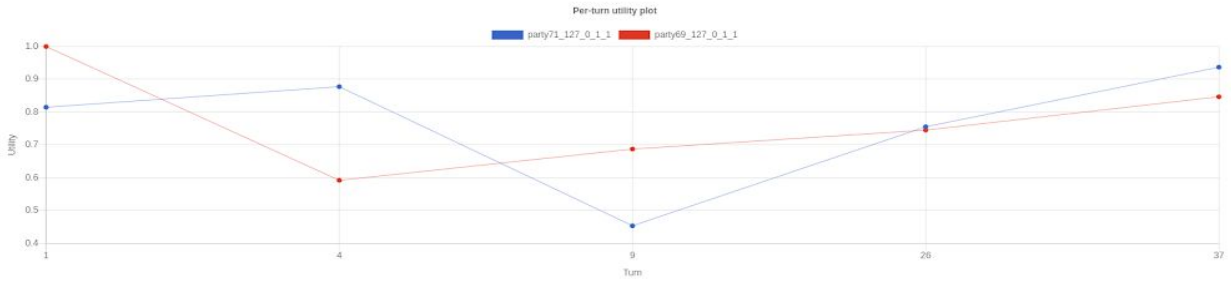
The below sections will go into detail about the results we have gathered from our experiments for each domain. The results include the agent versus itself and the agent versus AgentGG from ANAC2019. Our agents hyperparameters are set and fixed to 0.5 for lowBoundaryOffer, 0.85 for lowBoundaryAccept , and 0.54 lowBoundaryOpponent offer.

#### 4.2.1. Laptop Domain

The laptop domain is a small domain composed of 27 possible permutations. In our test using this domain we have chosen partial to be equal to 10. The results of the tests can be seen below.



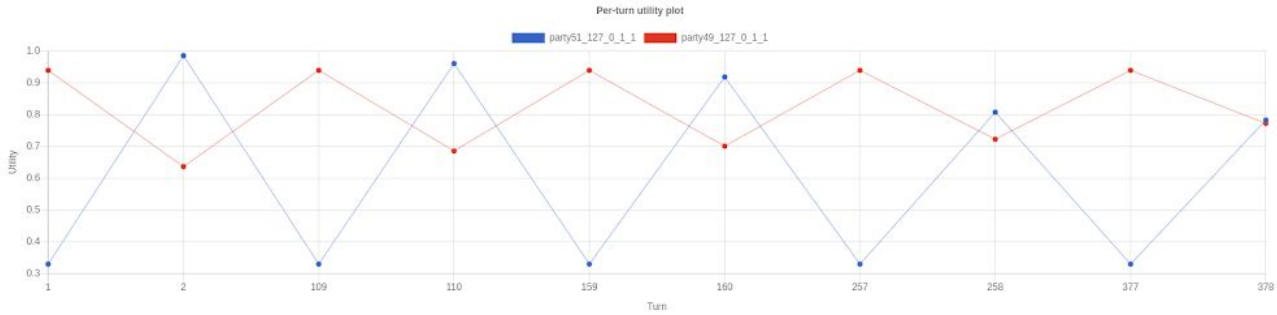

In the figure above you may see that our agents start with their highest bid and after the offer is not accepted by the agentGG, the received offer was good enough for the agent and therefore accepted.



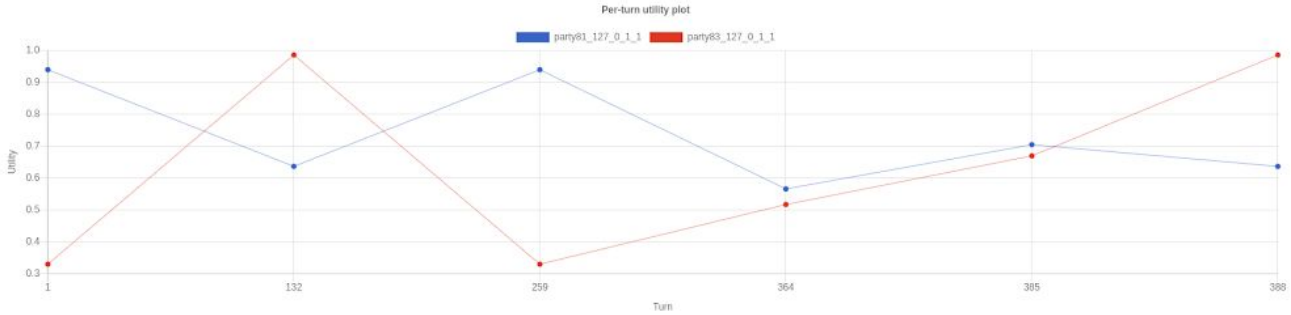

The above figure shows the test between our agents and itself, you can see that agents did some elicit comparison requests in order to figure out the utility of the received bid. The test ended with an acceptance with outcome utilities of %93.7 and %84.7. As one can see, the outcome is pareto optimal because the agents reached a solution that both benefit without cannibalizing the other.

### 4.2.2. Party Domain

The party domain consists of 3072 possible permutations which we classify as a huge size domain. As before, the tests are conducted the agent vs the agent, the agent vs. agentGG with partial = 20. The results of the tests can be seen below.



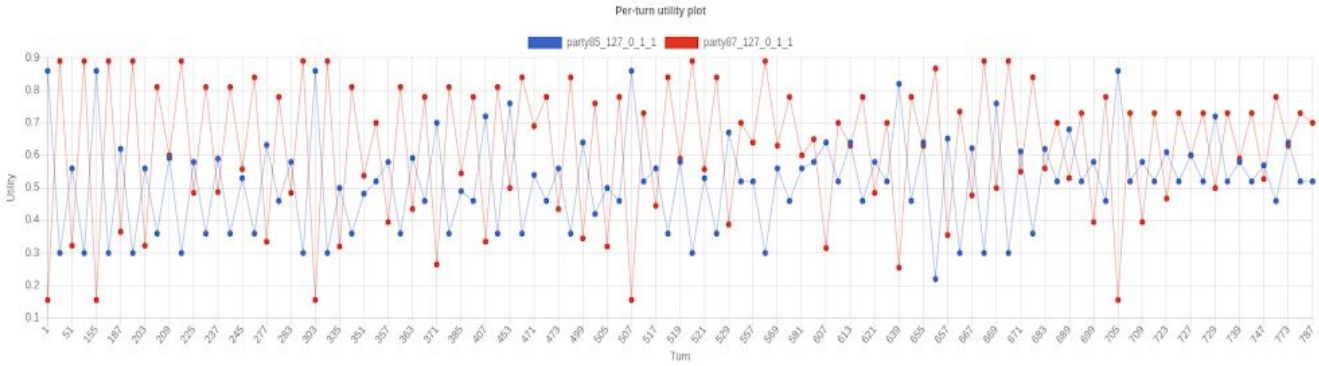

The above figure shows the test between our agent vs agentGG. The test shows that our agent tried to maintain its aim for offering a bid higher than the last received bid however, due to the limited known space of bids, the agent always found the same bid to offer. At the last term the received bid for the agent is accepted also with this acceptance the solution is pareto optimal since both of the agents benefit from it.



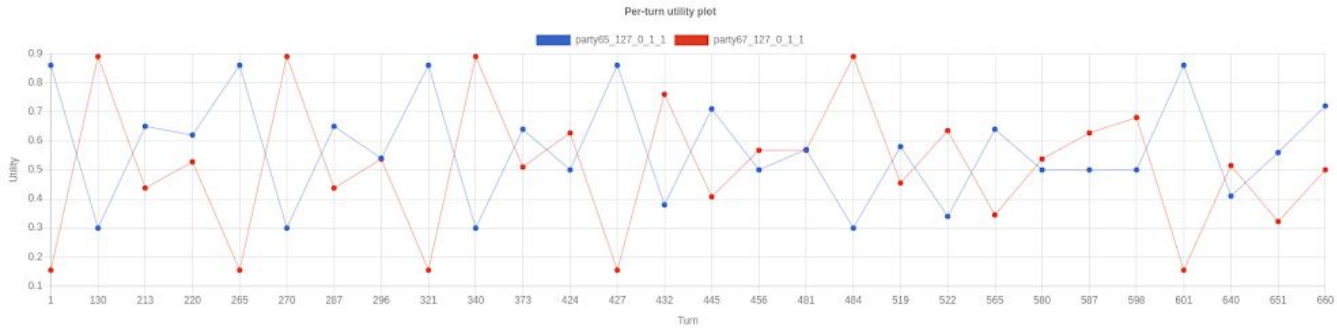

In the figure above the test was conducted in between our agent vs our agent. One can deduce that, the agents did a lot of elicit comparison requests in order to get an idea of the bid space more. In return, the agents' offers had an increase in utility on both of them until to the point that one of the agents offered high with knowing that probabilistically the agent may accept the offer in question.

### 4.2.3. Jobs Domain

The job domain consists of 540 possible permutations which we classify as a medium size domain. As before, the tests are conducted the agent vs the agent, the agent vs. agentGG with partial = 10. The results of the tests can be seen below.




In the figure above the test was conducted in between our agent vs agentGG. As you can see there is a lot of negotiation between the two in a 60 seconds period. One may see that elicitComparisonRequest and majority of them happened at the beginning and the rest is Gaussian Distribution spread throughout the negotiation process. The reason for this is that the aim is to find a close optimal bid that fulfills the above explained criterion. The test ended without an acceptance situation.




The above figure shows the test between our agent vs our agent in a 60 second duration. The test ended with an accept which is a Pareto optimal solution as one can see from the figure.

### 4.3. Improvements

In our early stages of development we have conducted a lot of preliminary tests to see how the agent is performing and how we could improve it more. Our initial and most important improvement based on the results we have gathered was implementing a fix for duplicate offers that is the agent was repeating itself when the received offer was the same back to back. We have caused this situation by implementing a shuffle when we are using AllBidsList. Another improvement we have made is frequency based opponent modelling which was an important step in our agents development in terms of offering bids that have the probability of acceptance.

## 5. Conclusion

The agent that we have created can perform its operations regardless of the domain and the opponent, therefore, the agent is generic. Moreover, the agent can mostly produce Pareto optimal (nash) solutions based on the experiments we have conducted.

One of the strong points of our agent is its high percentage of making the deal, that is out of 30 tests that we have conducted the agent has made 28 deals successfully. Out of these 28 deals, the agent had nearly 57% of these runs with higher utility than the opponent which brings us to the weak points. Although the agent can make deals successfully, the utilities of those deals can be improved.

In conclusion, we have implemented a negotiating agent that has the mission of “making the deal” using the GeniusWeb Framework. The agent uses elicitation approaches to be able to guess the approximate utility of a given bid and implements a simple yet very useful idea for acceptance. Moreover, the agent utilizes an opponent modelling technique to increase its probability of success for “making the deal”. However, making the deal is only the half of it, the other half is a high utility outcome which is where our agent lacks a bit behind but of course this does not mean that our agent don’t perform well, more than the half of the tests that we have conducted, fulfilled both of the parameters of success.