Artificial Intellegence - Negotiation Assignment

Fanny Lie (4088204) Sander Swart (4001699) Edward Teng (4085094)

January 2015

Contents

1 Introduction

For this assignment, the GENIUS skeleton 2014 project is used on Eclipse framework. This project consists of a standard agent script and the GENIUS GUI. With the GUI, different type of negotiations are available. In our case, Multi-Party Tournament and Multi-Party Negotiation are of importance. The main purpose of this assignment is to design a negotiation agent for multiparty negotiation. The agent will then be tested on three different scenarios against other agents.

During this assignment three scenarios, each with three profiles will be created using the GENIUS GUI. The first scenario is collaborative which means that it is very likely to find a mutually acceptable bid since all three parties have a somewhat similar preference profile. The second scenario is moderate, which means the chance to find a mutually bid is lower than the first scenario, thus the preferences of the agents are somewhat different. The last scenario is competitive, which means the preferences of the agents are more conflicting than the other scenarios as on multiple issues the preferences conflict with one another.

This paper has five main sections. The first is this introduction. The second is the agent design, where high-level description and explanation of the agent and its structure is given. The third is the negotiation strategy, where the decision functions for accepting and bidding are explained. The fourth is the performance of the agent, where tests and the description on how these tests are used to improve the agent is given. The last section is the conclusion. In this section team experience regarding building the agent is summarized and discussion of the agent is given.

2 Agent design

2.1 The agent's goal

The goal of our agent is to achieve an agreement where the utility of all the parties are acceptable. Defining acceptable is hard, but we defined it in our agent as having a somewhat equal utility for all the parties, with a maximum difference of 0.25. In that case, our own utility is higher. It is possible to offer a bid which a higher utility for one or more opponents than the utility for our own agent, but as long as our own threshold has been met, we will offer the bid. This does not necessarily mean our agent is at an disadvantage because it will only be more likely that the other party will accept. The possibility that the other agent does not accept still remains, so therefore it is better to play it safe. Our agent certainly will not accept any bid that has a utility lower than 0.5.

2.2 Structure

Every time the agent receives a message from a new opponent, a new opponent model will be created for this opponent in order to store every bid this opponent offers. Our agent keeps a list containing the names of the opponents and has a HashMap to store each opponentModel with their corresponding opponent name. In order to do this, a small helper function called `addOpponentModel(String opponent)` has been created. Since a check will be done whether it is a new opponent or not, creating a new opponent model for each opponent will only occur in the first round. Since we create opponentModels, an issueList needs to be given to the constructor of opponentModel. Therefore we initialized the issueList in the constructor of our agent: `issueList = this.utilitySpace.getDomain().getIssues();`. This issueList can be used for the other agents aswell because the issueList for all the agents is the same during a match.

The agent started off with two methods included in the template: `chooseAction(List<Class> validActions)` and `receiveMessage(Object sender,Action action)`. First, the latter will be explained.

When an opponent sends a message containing an Accept action, nothing is to be handled by our agent. When the opponent sends a bid, it is added to the corresponding opponent model. When the agent is required to choose an action, it first determines the threshold based on the number of rounds to go. If there has been an offer by another opponent, the agent checks whether the bid is acceptable - which is the case if the utility of that bid is at least the threshold.

If the bid was not acceptable, we generate a new bid, meeting the threshold of both our the agents own and its opponents' utilities. Checking whether all the opponents' utilities of our tempBid (the bid we want to propose if all thresholds are met) is done in the helper method checkUtils. The key in this function is that, from all the information collected in opponontModel, the estimated utility for our tempBid can be computed by `oppModel.getEstimatedUtility(bid)`. Generating a bid is done in a do-while loop:

```
do{
    if(counter == 100){
        oppThreshold -= 0.05;
        counter = 0;
    }
    tempBid = rbc.getBid(utilitySpace, threshold-0.05, threshold+0.05);
    newUtility = getUtility(tempBid);
    allOpponentsUtilHigh = checkUtils(oppThreshold, tempBid);
    counter++;
}
while((newUtility < threshold-0.05 && !allOpponentsUtilHigh));
```

To make sure the loop will not become infinite, we keep lowering the opponents' threshold a bit until we find one. Of course, if the agent is the first one to bid, meaning the first turn in the first round, it has to generate a bid. In this case, it will generate a bid that would suit the agents preferences best. This is done by calling Utility.getRandomBid(utilitySpace).

3 Negotiation strategy

To make a sensible bid, which makes it more likely for the opponents to accept the bid but more importantly achieving a higher utility value for the agent itself, one needs to know roughly what the other agents' preferences are. Since the agent only knows his own preferences and it is not possible to get the other agents' preferences in a sophisticated way from the start, an opponent model for each opponent is created to learn these preferences. Every time an opponent offers a bid, the bid is stored in the opponent model so that the preferences and weights of the issues can be more accurately estimated when there are more bids done by that opponent.

Generating such a sensible bid is done by generating a bid and checking its utility for both ourselves and our opponents. If both our threshold and the threshold we decide for the opponents (which is a bit lower than our own threshold) is met, this bid will be the one to offer. The threshold is dependent on the how many rounds have already past compared to the total amount of rounds. The closer to the deadline, the lower the threshold.

For instance, if the negotiation has past 40% of the rounds, our agent will have a threshold of 0.75. When generating a bid, the agents wants an utility of at least 0.7 because we take a bit of margin into account. For the opponents, the utility must meet at least the 0.6 threshold.

4 Performance analysis

For each testing session we used four cases for each scenario: the deadline of the negotiation would be 10, 20, 90 or 180 rounds. The three scenarios are collaborative, moderate, and competitive. Collaborative means that the preferences of the negotiating parties are somewhat similar. The competitive scenario contains agents with some contradicting preferences, and the moderate case is lies between the other two scenarios: not really collaborative but not too competitive either.

4.1 First version

Before reaching our current results, we started off with a version where the agent would generate bids with high utilities for oneself and adjust the least important issue (for the agent) to one of the important values (estimated, since the exact preference model is unknown) of one of the opponents. Our agent would be quite picky about the utility threshold (0.8) and would only lower this to 0.6 if the deadline was very near (if only 10% of the total amount of rounds is left) so that there still would be an agreement. The graphical results are presented in Appendix A. As seen in all graphs, there is no real negotiation: the agents do not converge to a solution; they are rather stubborn. For scenario 1 this is quite alright since the preference profiles lie quite close to each other so the agents will accept the bid. In scenario 2, the agents sometimes reach an agreement, and sometimes they do not. For the competitive scenario however, the agent will never come to an agreement as the preferences lie too far apart.

4.2 Final version

To improve this, we decided to define more time-based thresholds, meaning the closer to the deadline, the lower the threshold will become. Also, since the small adjustment of the bid would comply to only one opponent, the other might not agree with it anyway, so we continued generating bids more automatically and only rely on the utilities since if those are accepted by all, the underlying values for the issues must be acceptable.

See Appendix B for the graphical output. The results show that for both scenario 1 and 2, an agreement is reached. For the third scenario, only an agreement is reached when the number of rounds is "high". Since the threshold is lowered when more rounds pass, the negotiation is slightly converging.

5 Conclusion

For negotiation an agreement is preferred over none and to have an agreement a sensible bid has to be made such that the bid is also appealing to the other agents/parties. To be able to make such bids, other agents' preferences should be roughly known and for this an opponentmodel is created. With this model the agent is able to take its opponents' preferences into account when offering a bid. The threshold for the utility of this bid will lower the more rounds will pass. Our agent does not accept an offer with a utility lower than 0.5, which leads to the possibility of reaching no agreement in the competitive scenario. However our agent's bid will always be sensible for all the parties.

6 Appendix A: first test results

6.1 Scenario 1

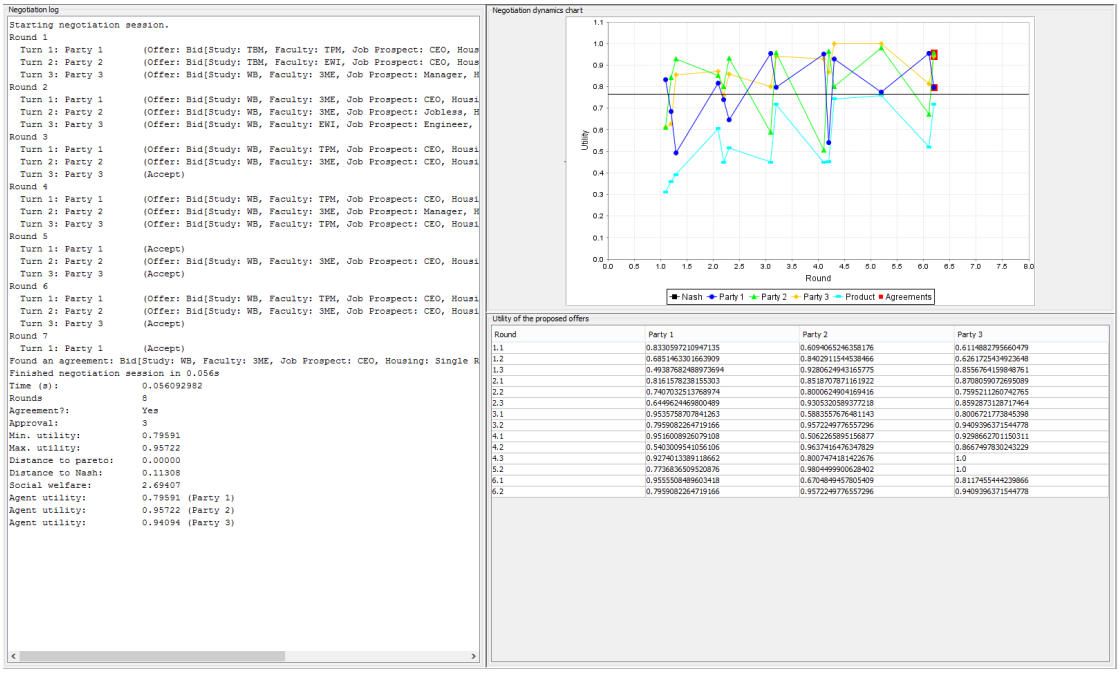

Figure 1: Scenario 1: collabortive - deadline: 10 rounds

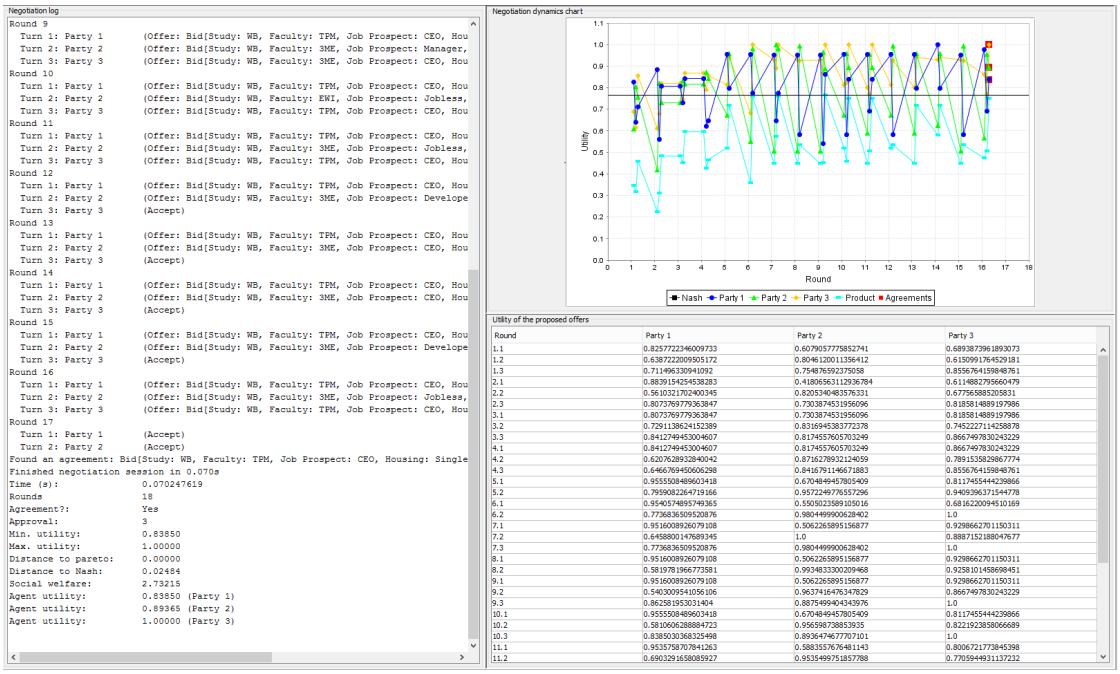

Figure 2: Scenario 1: collabortive - deadline: 20 rounds

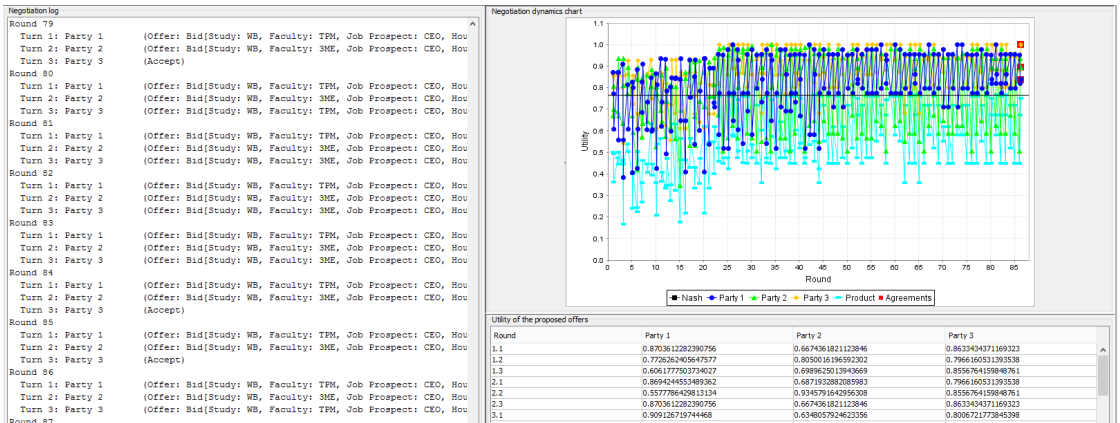

Figure 3: Scenario 1: collabortive - deadline: 90 rounds

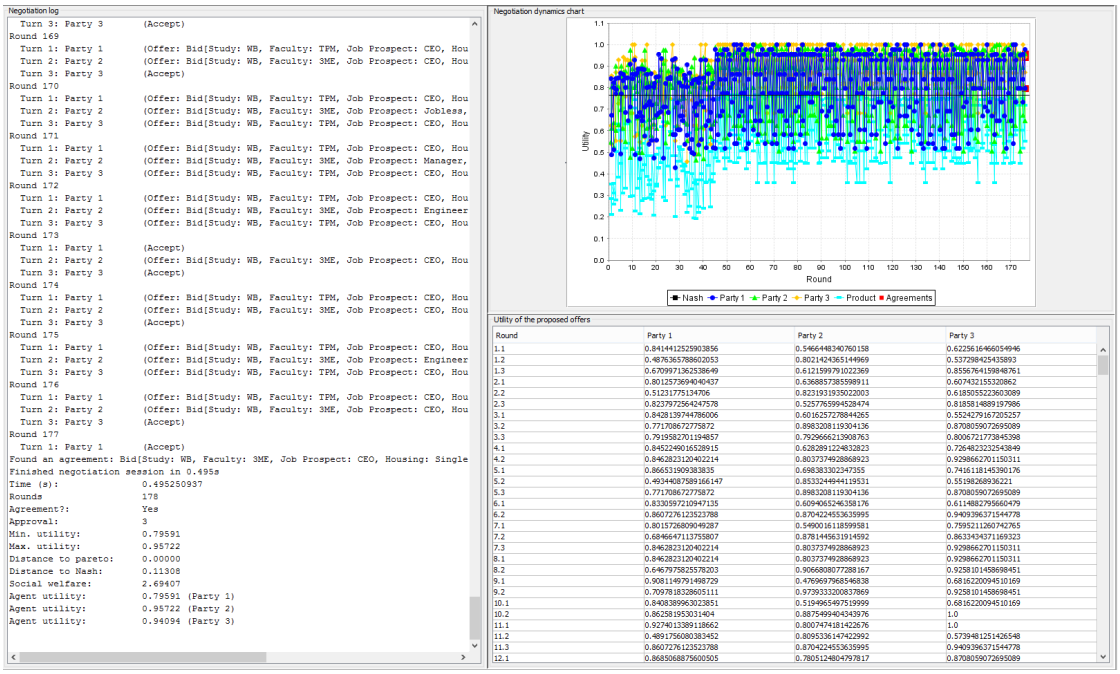

Figure 4: Scenario 1: collabortive - deadline: 180 rounds

6.2 Scenario 2

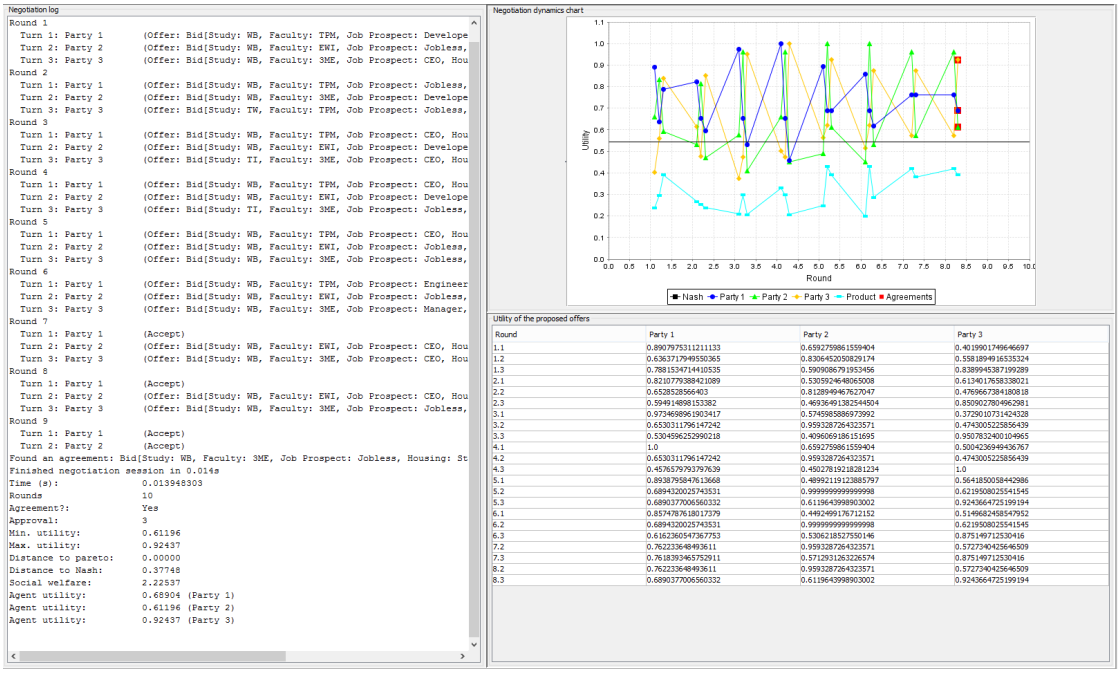

Figure 5: Scenario 2: collabortive - deadline: 10 rounds

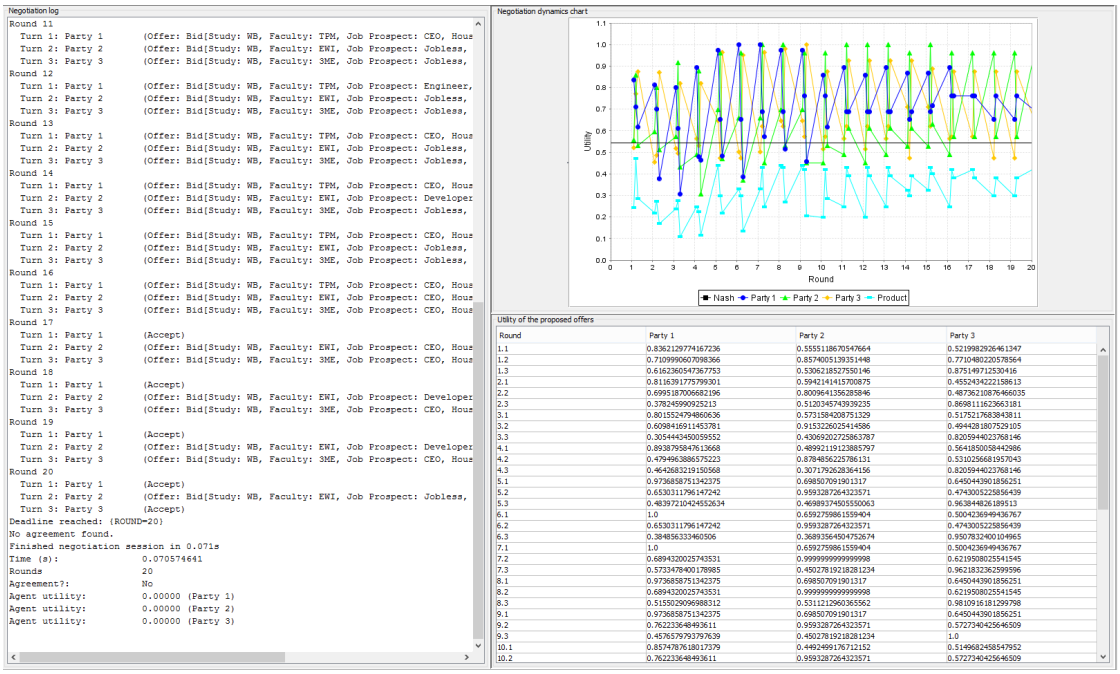

Figure 6: Scenario 2: collabortive - deadline: 20 rounds

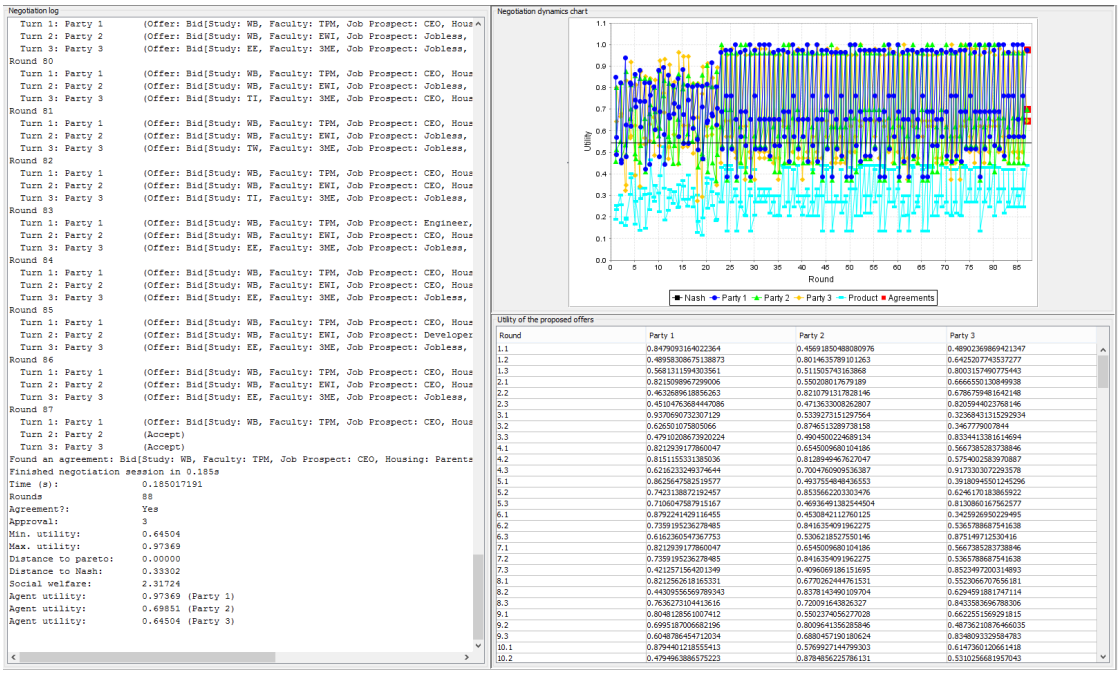

Figure 7: Scenario 2: collabortive - deadline: 90 rounds

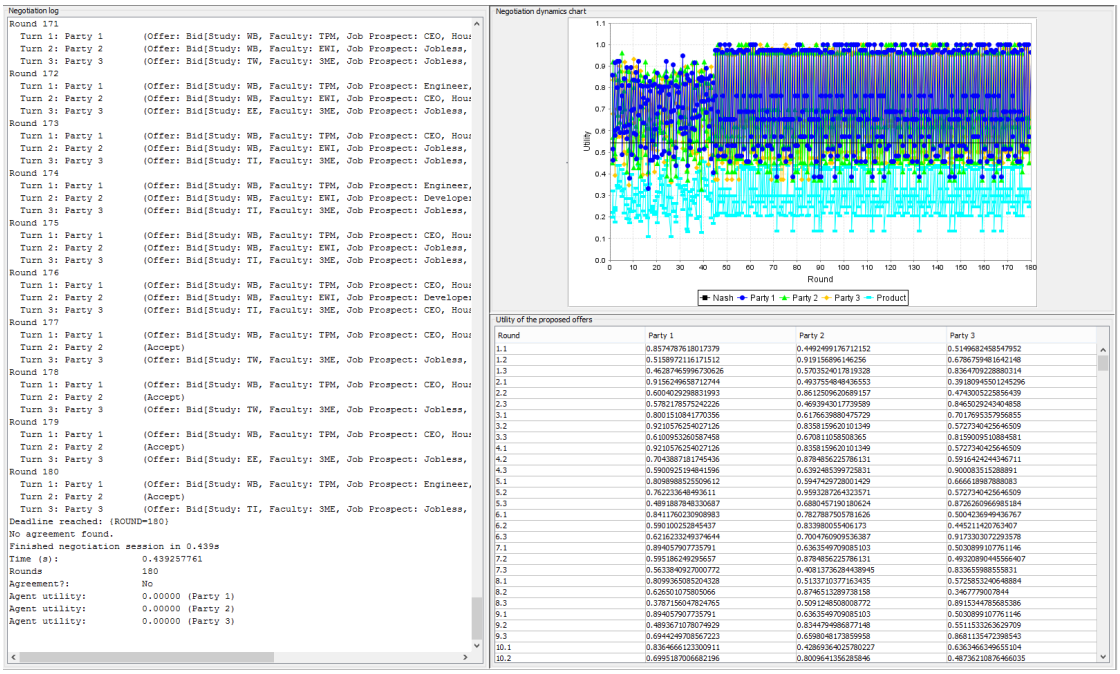

Figure 8: Scenario 2: collabortive - deadline: 180 rounds

6.3 Scenario 3

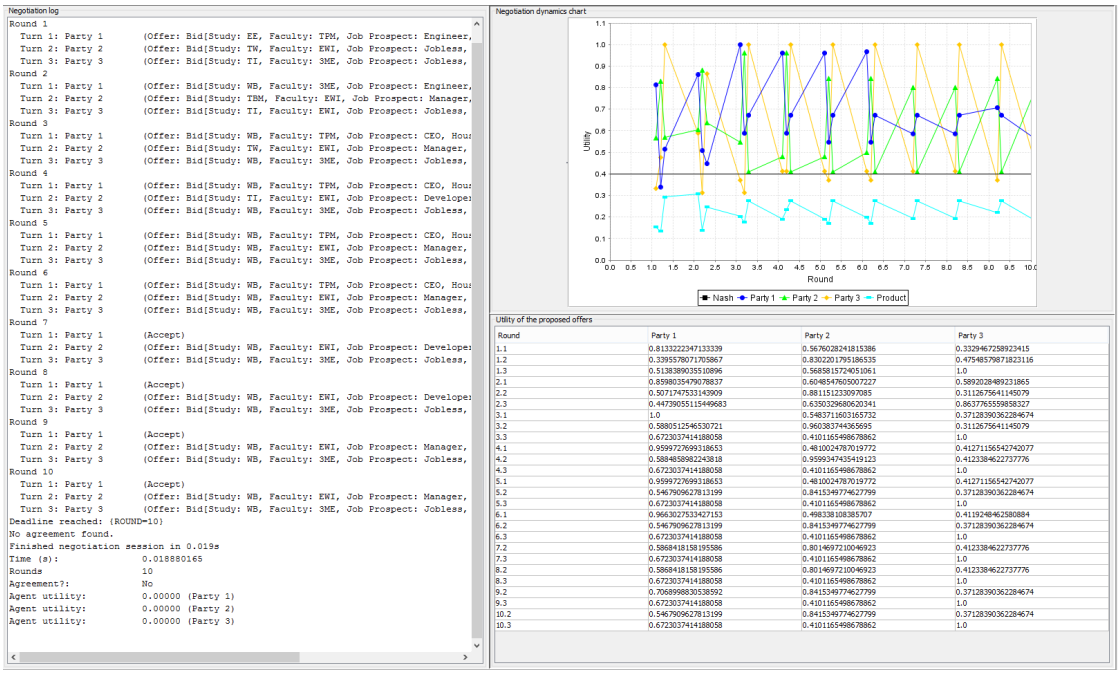

Figure 9: Scenario 3: collabortive - deadline: 10 rounds

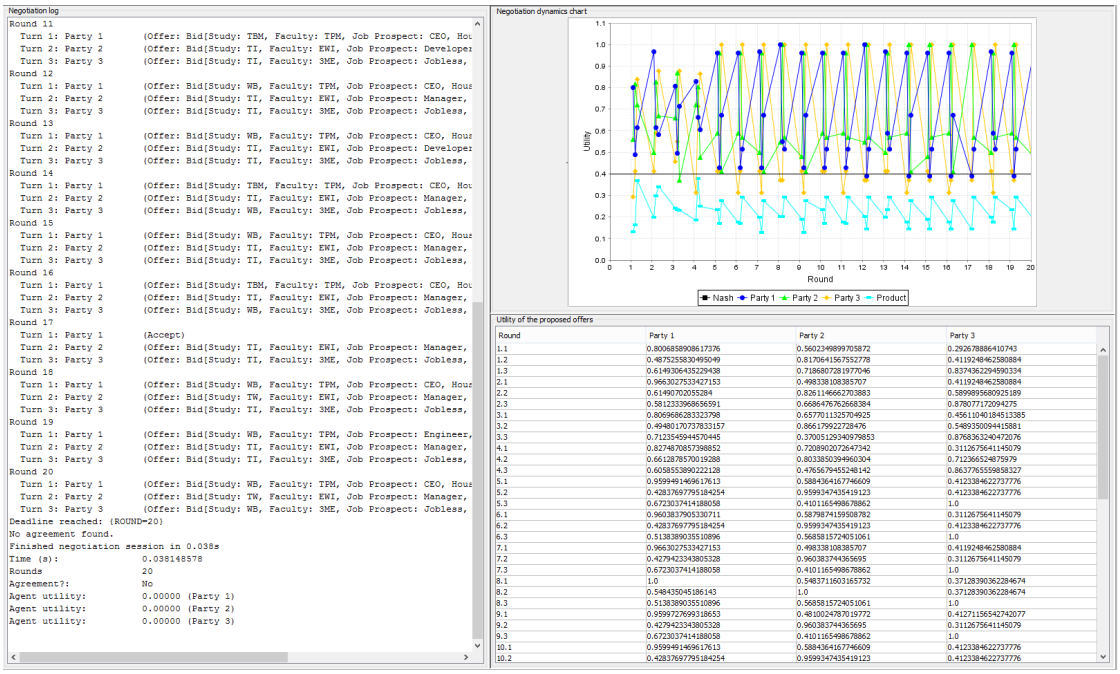

Figure 10: Scenario 3: collabortive - deadline: 20 rounds

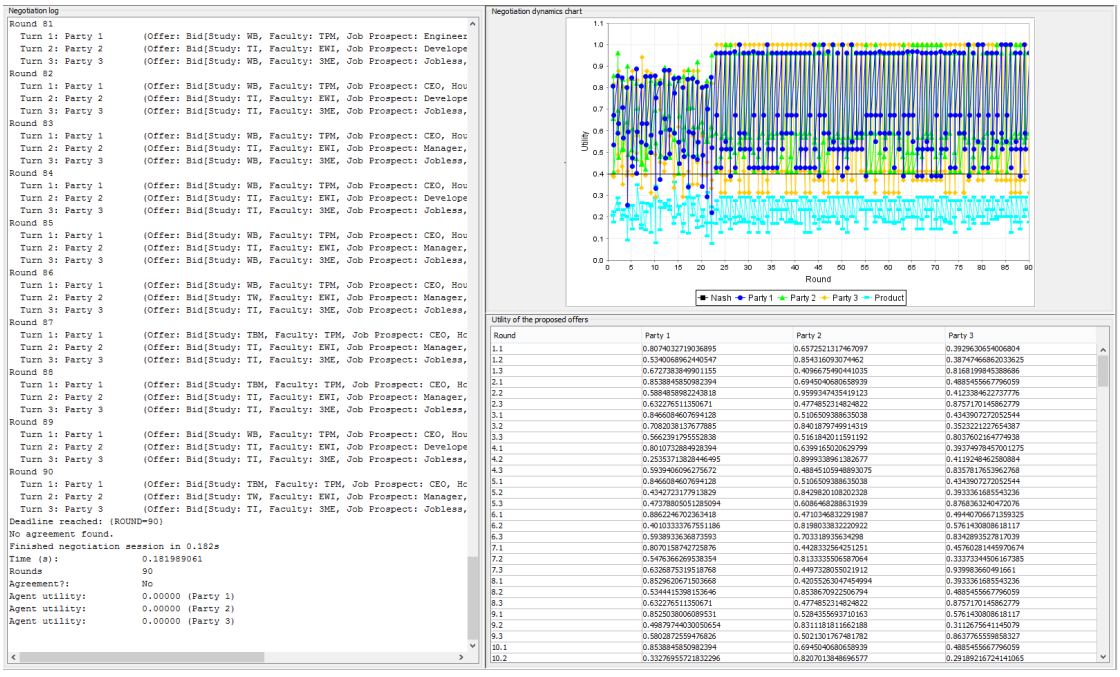

Figure 11: Scenario 3: collabortive - deadline: 90 rounds

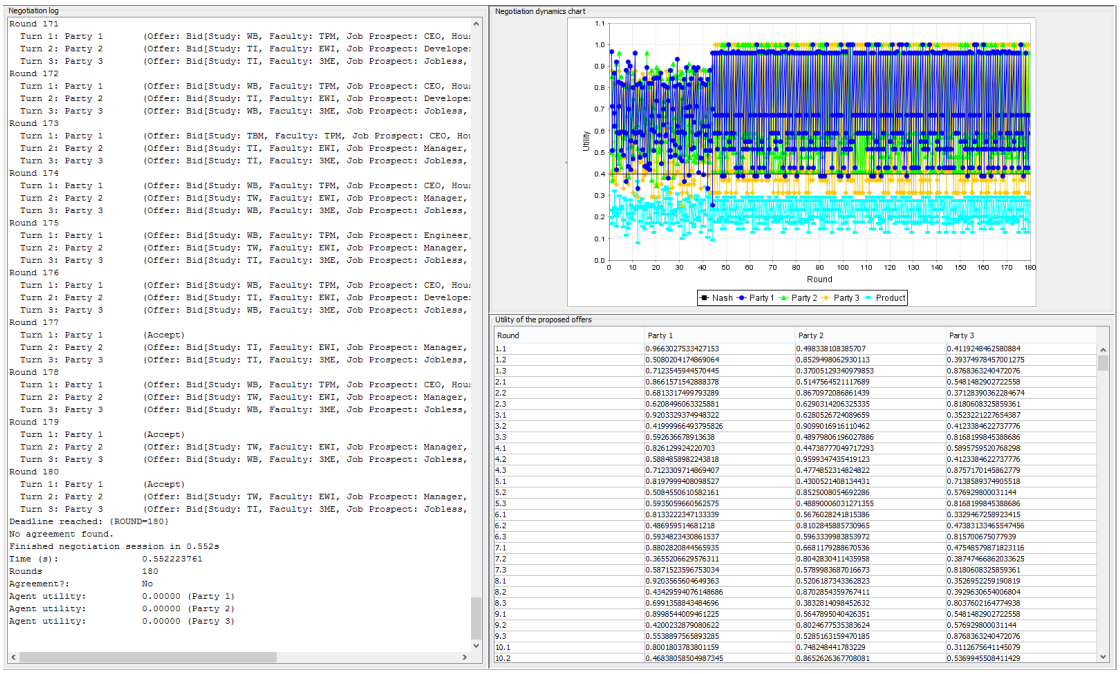

Figure 12: Scenario 3: collabortive - deadline: 180 rounds

7 Appendix B: final test results

7.1 Scenario 1

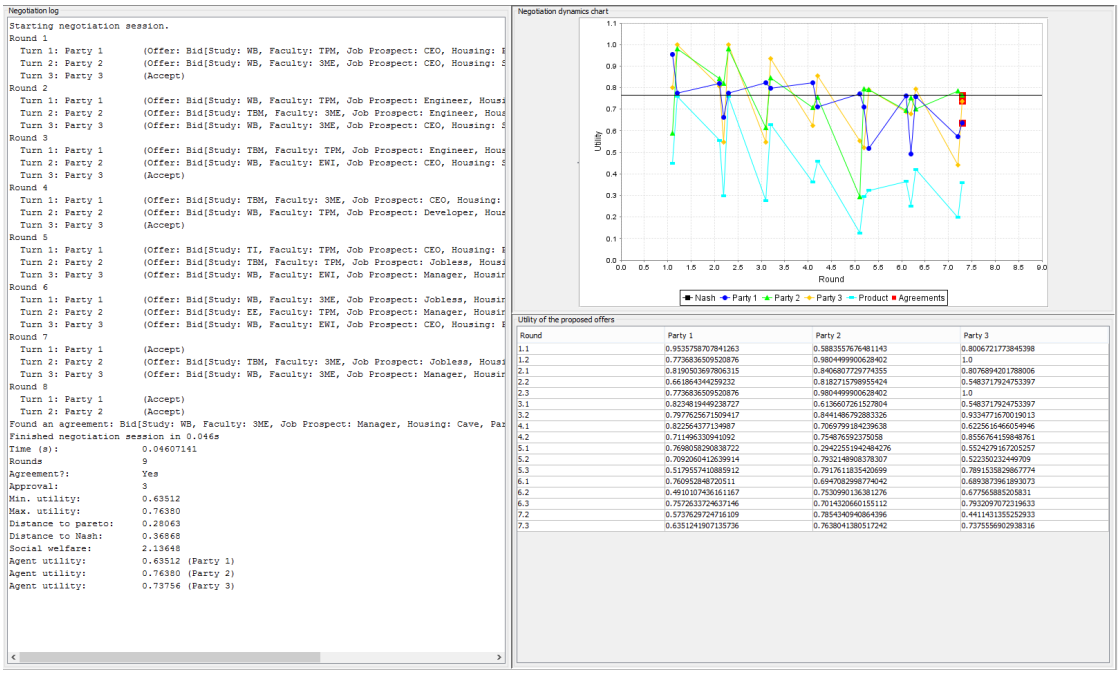

Figure 13: Scenario 1: collabortive - deadline: 10 rounds

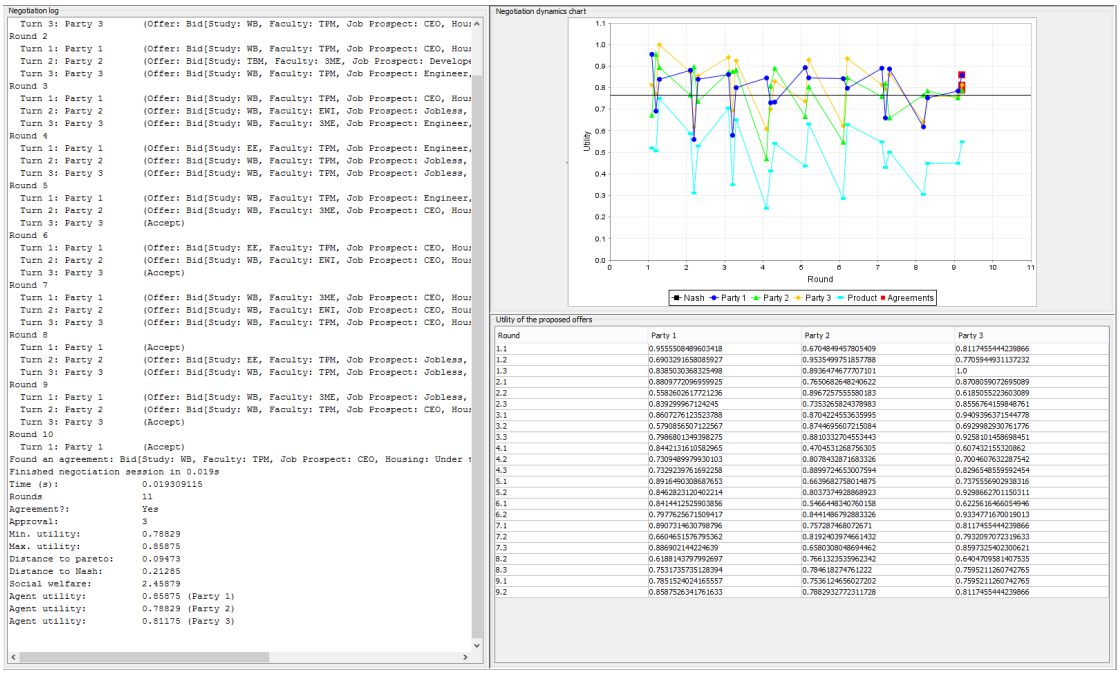

Figure 14: Scenario 1: collabortive - deadline: 20 rounds

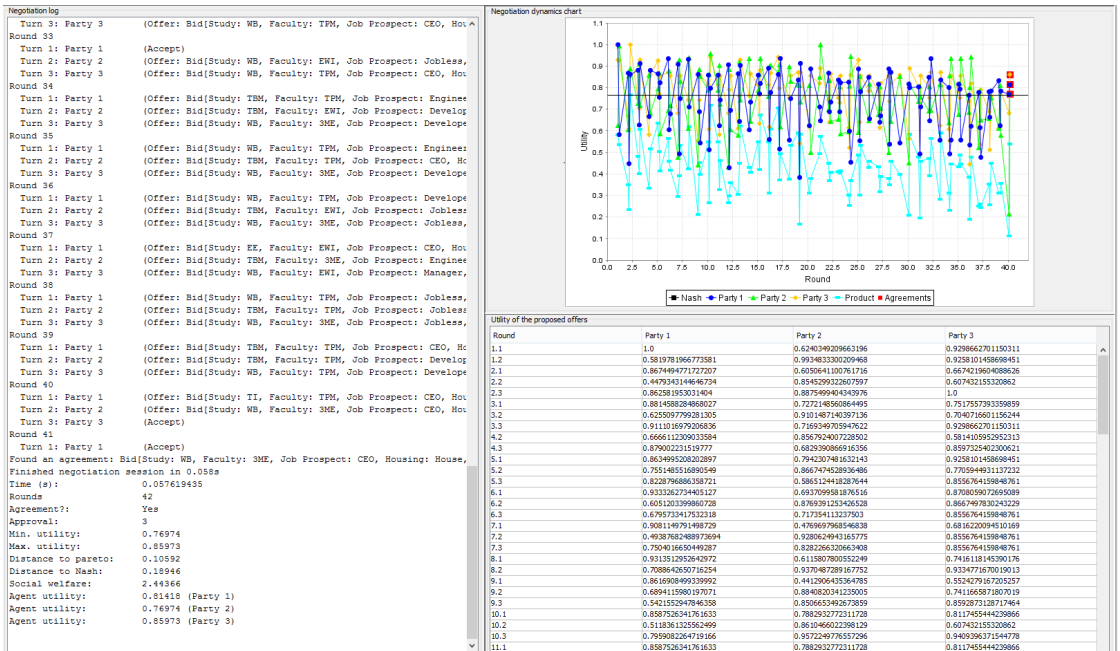

Figure 15: Scenario 1: collabortive - deadline: 90 rounds

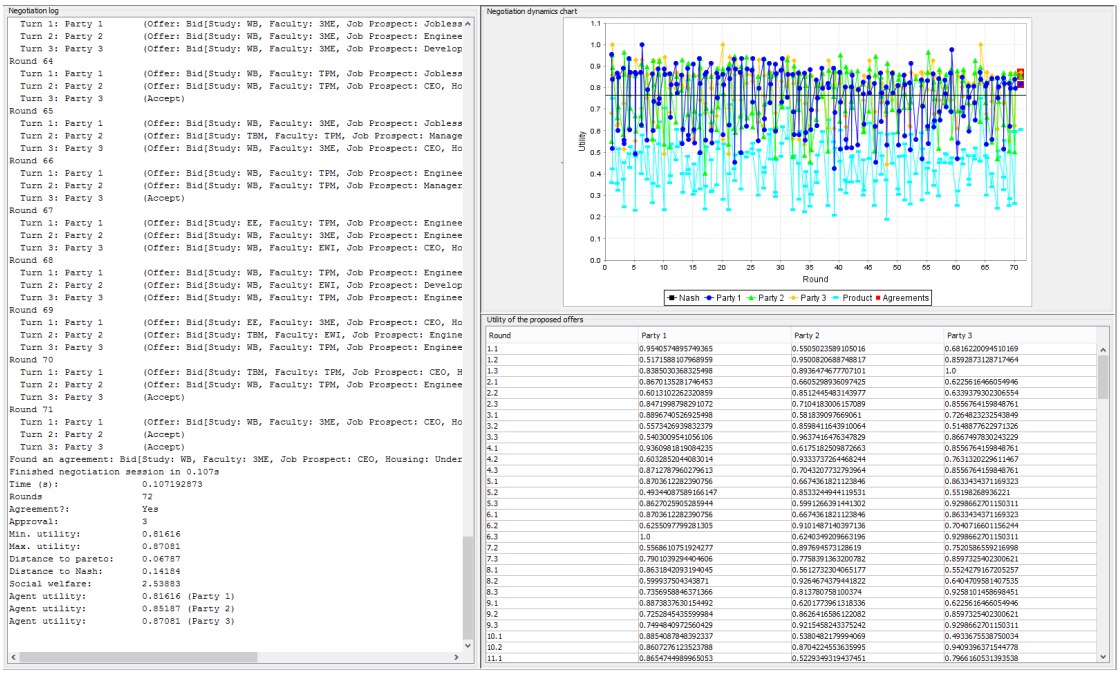

Figure 16: Scenario 1: collabortive - deadline: 180 rounds

7.2 Scenario 2

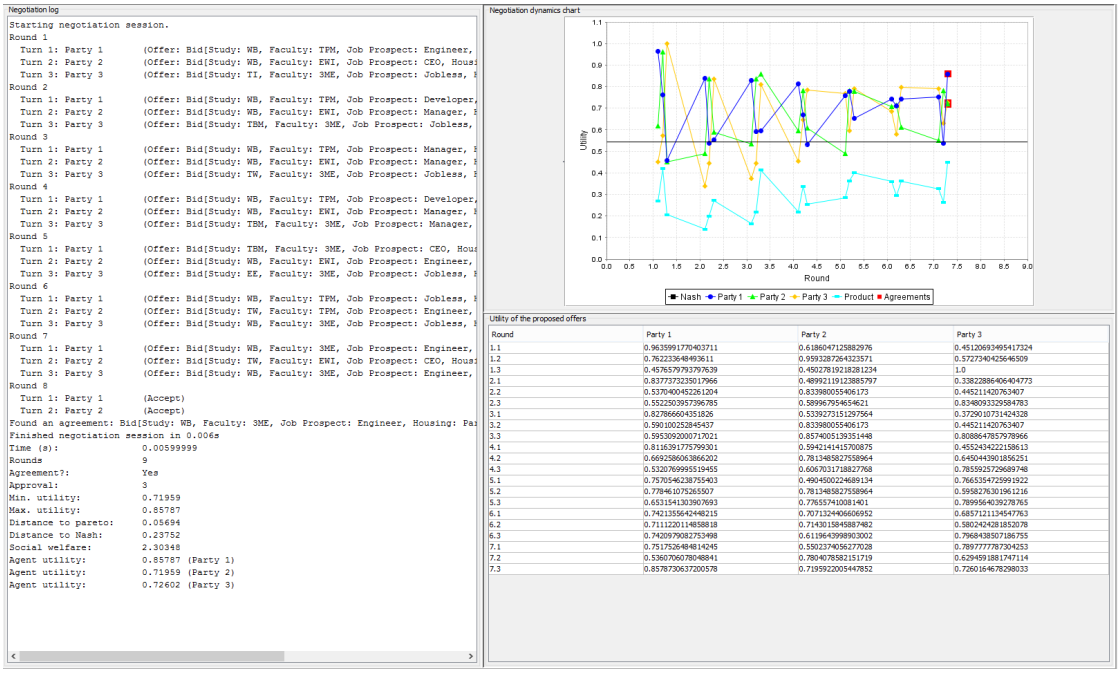

Figure 17: Scenario 2: collabortive - deadline: 10 rounds

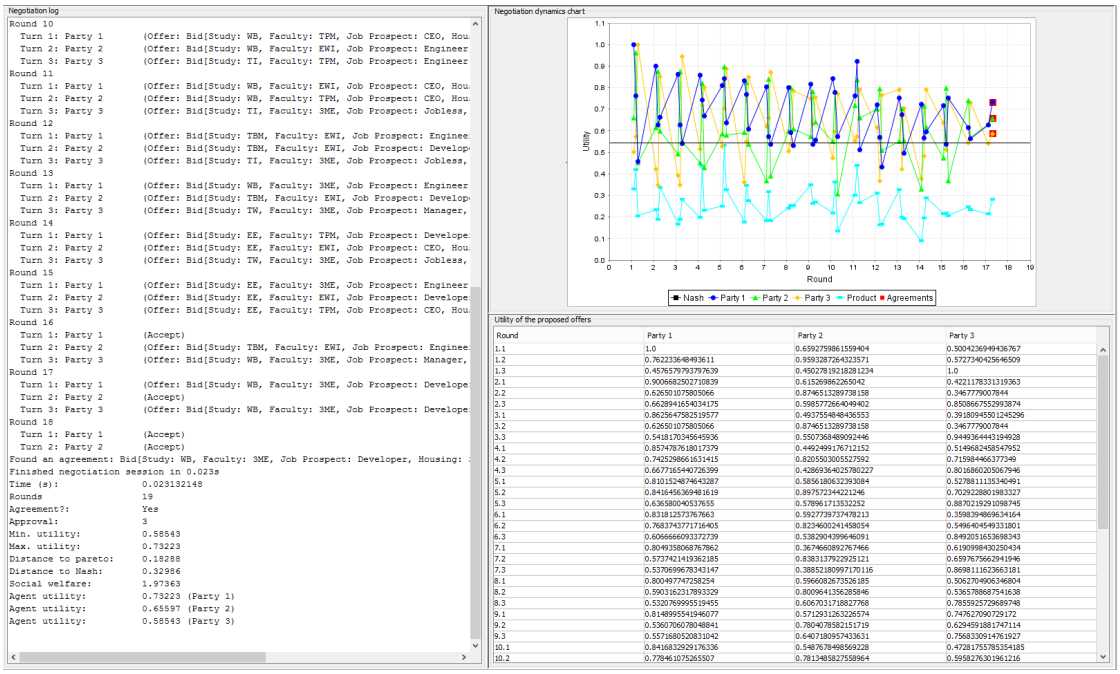

Figure 18: Scenario 2: collabortive - deadline: 20 rounds

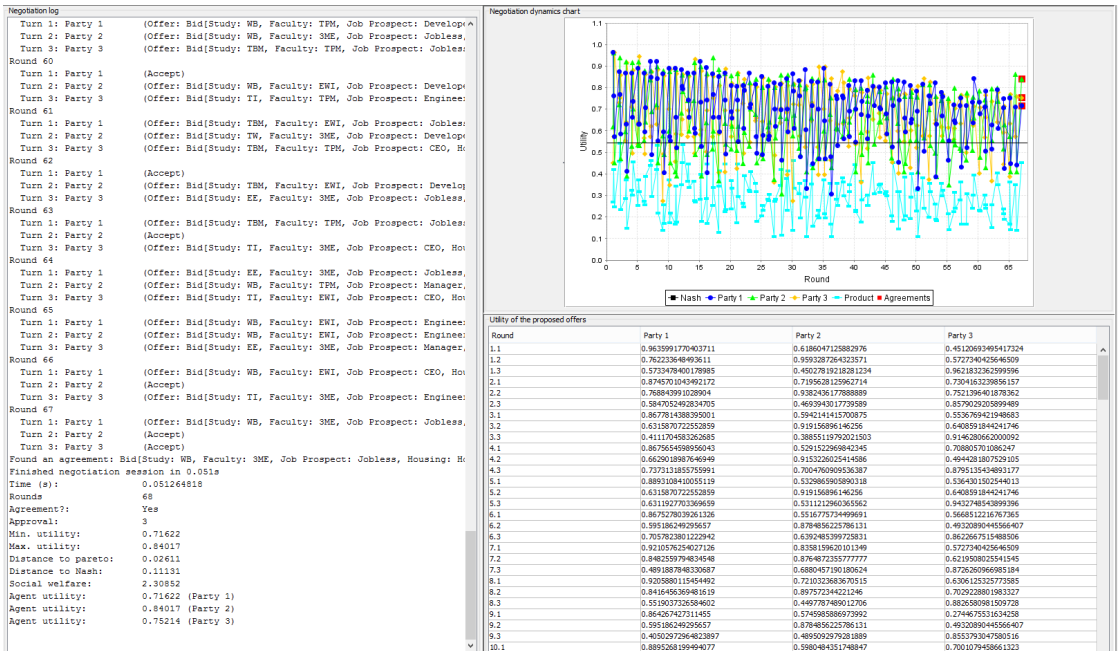

Figure 19: Scenario 2: collabortive - deadline: 90 rounds

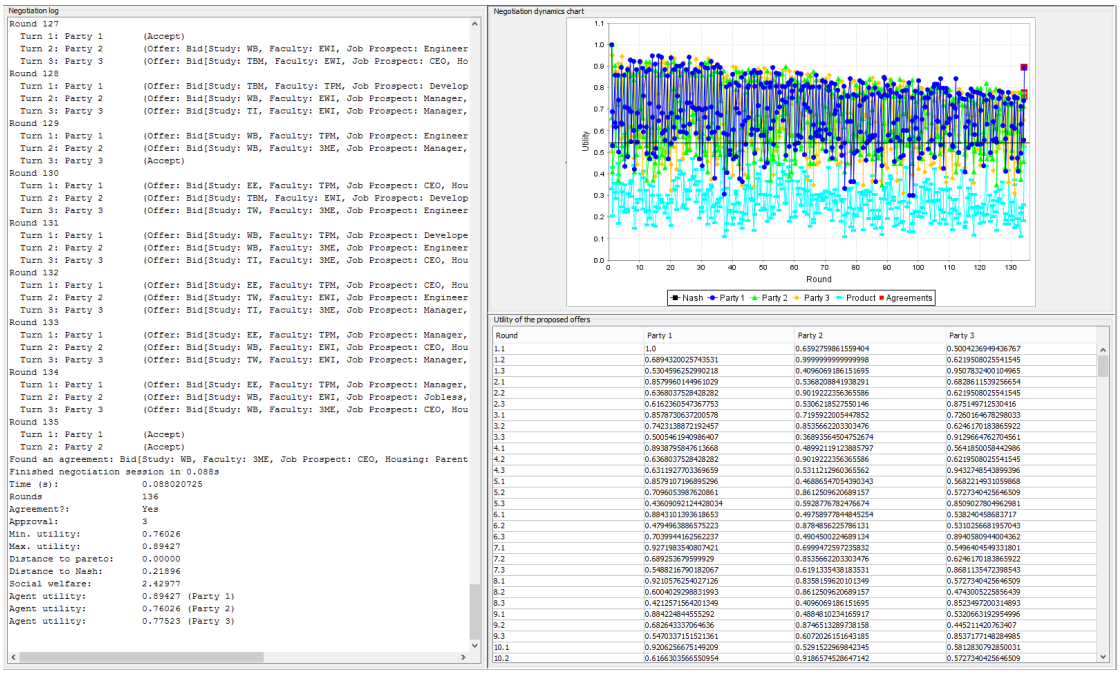

Figure 20: Scenario 2: collabortive - deadline: 180 rounds

7.3 Scenario 3

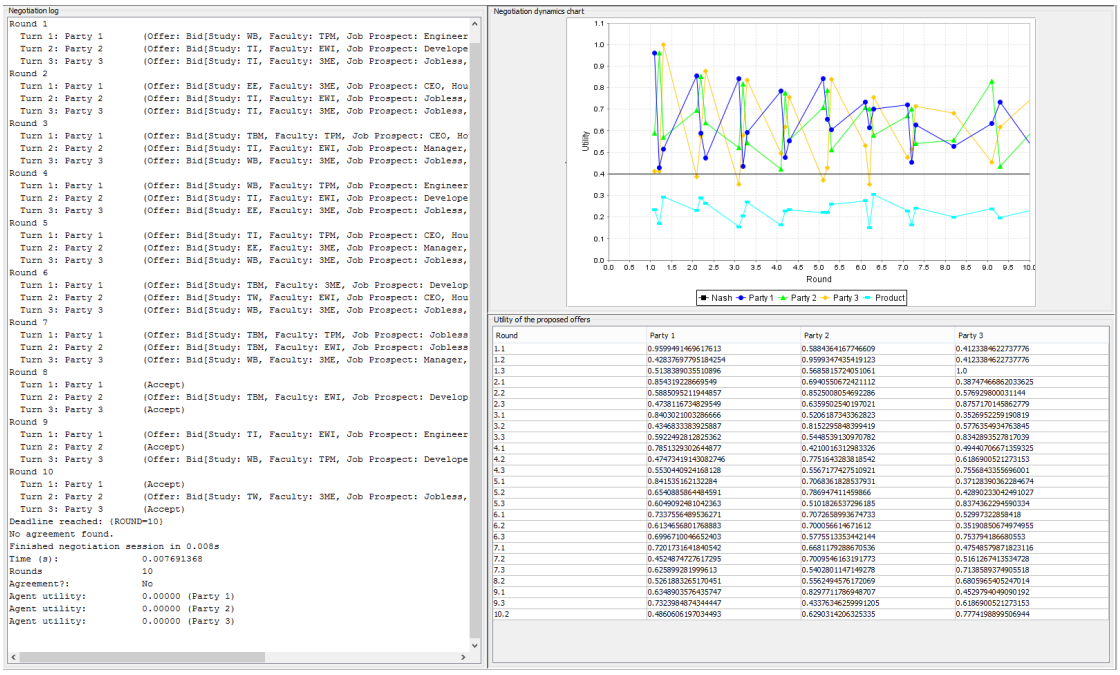

Figure 21: Scenario 3: collabortive - deadline: 10 rounds

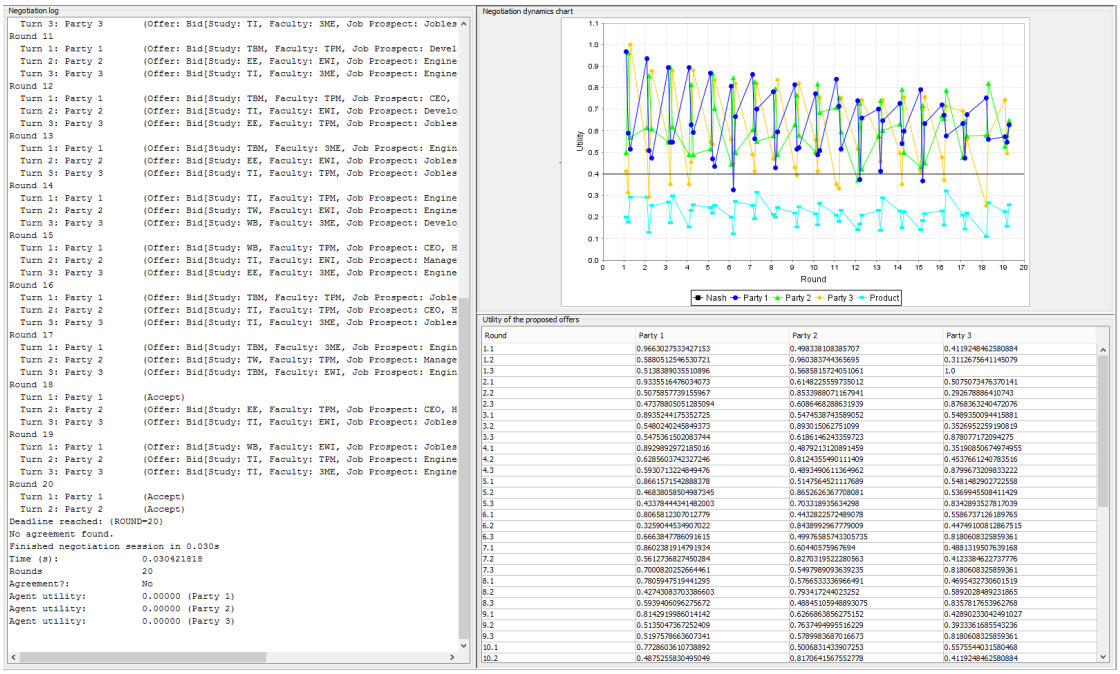

Figure 22: Scenario 3: collabortive - deadline: 20 rounds

Figure 23: Scenario 3: collabortive - deadline: 90 rounds

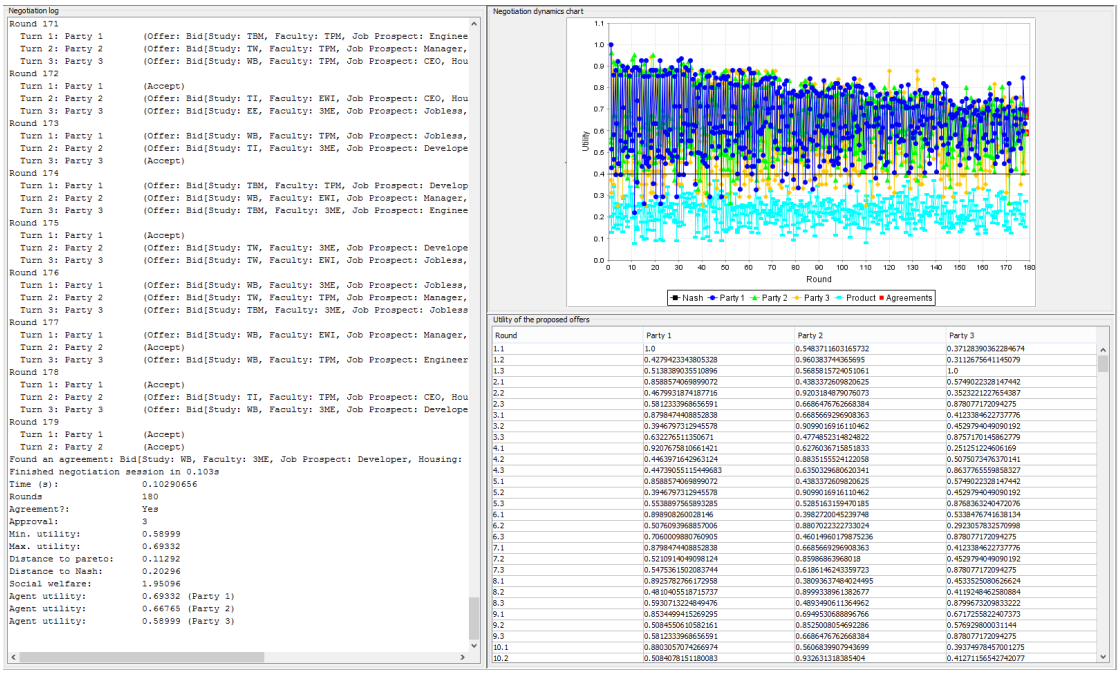

Figure 24: Scenario 3: collabortive - deadline: 180 rounds