Delft Technical University

IN4010 Artificial Intelligence Techniques

Automated Negotiation Group 19

T. Cherici
M. de Vries
T. Resink

October 23, 2015

1 Introduction

In this report, the development of a negotiation agent by group 19 of the course Artificial Intelligence Techniques will be discussed. In this assignment, a negotiation agent has to be developed that is able to contest with agents of other groups. The goal is to design an agent which can get a high utility, preferably not on the expense of others. An emphasis will be put on the philosophy behind the strategies, the development process and the results. It will be subdivided in three tasks: The preference profiles, the design of a simple agent, and the improvement process towards the final agent.

2 Task 1, Preference profiles

In this chapter, three manually created preference profiles will be discussed. As suggested in the assignment description, three preference profiles were created with an own 'identity'. This is necessary if we want to give the negotiation agents a real challenge, in a situation where all parties have significantly different interests. In this case, one of the preference profiles has a high weight for the type of music as this virtual personality finds music of great importance during a party. Another preference profile is mainly concerned with the 'marketing' of the event. This person is thus more interested in the invitations.
A short description of the preference profiles is given below.

2.1 Brief profile descriptions

Profile_1
This profile find music important, and is especially fond of band music. He does not really care about the clean-up. He is moderately interested in the food and the location. He prefers handmade food and a party room.

Profile_2
This profile finds the choice of food the most important, he would love to have catering on the party. He would also love to have a custom printed invitation. The choice of beverage is also something he cares about: non-alcoholic beer is what he wants, although a regular beer is also an option. If possible the party should be in a party tent and he would like some water and soap to clean up the mess. Music clearly doesn't interest him.

Profile_3
This virtual identity is highly concerned with the location of the feast, she would like it to be in a party tent. She is moderately interested in the clean-up but not in doing it herself, she would like to have a hired help. She would prefer a DJ and handmade cocktails although she will still have a good time if they are not present. She's happy that there is a party and does not care about the invitations. Since she will eat after the party she does not at all care about the food. Although, if there would be a catering service she wouldn't mind.

2.2 Profile validation

In this section all profiles will each be used by three different agents in a negotiation session to see if the interests are far enough apart. The results of negation sessions with these agents and profiles are shown in figures 1-3.

In the simulations of the preference profiles the following agents will be used:

1. ANAC2015-1-Phoenix Profile 1
2. ANAC2015-2-CUHK Profile 2
3. ANAC2015-3-ParsAgent Profile 3

These agents were chosen at random for the validation. Three negotiation sessions with a deadline of 100 rounds and the SAOPMN protocol.

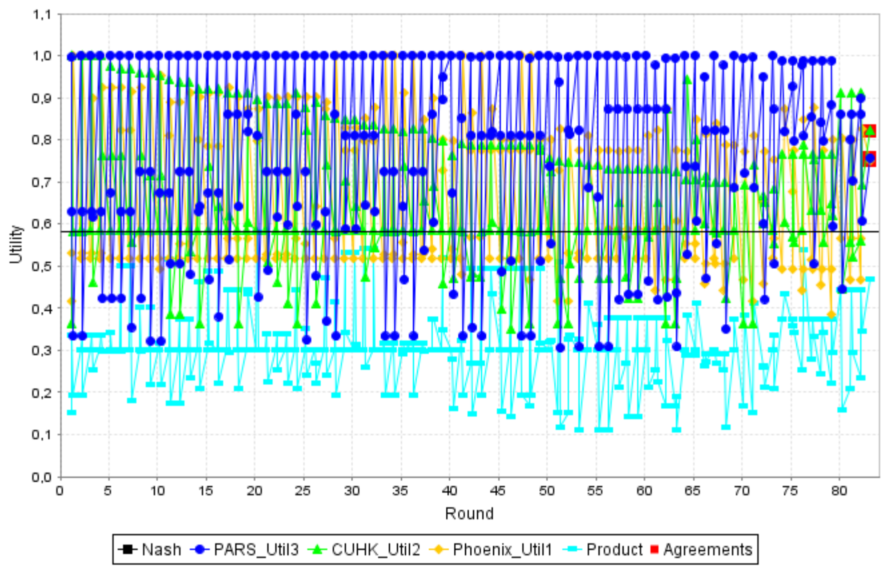

Fig. 1: Session 1, Utilities[0.75740;0.82201;0.75115]

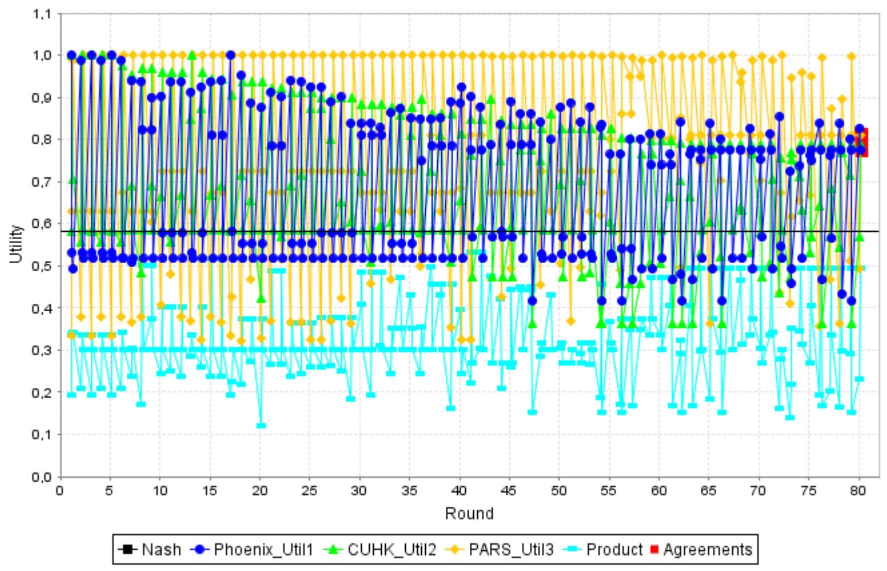

Fig. 2: Session 2, Utilities[0.77431;0.78736;0.81057

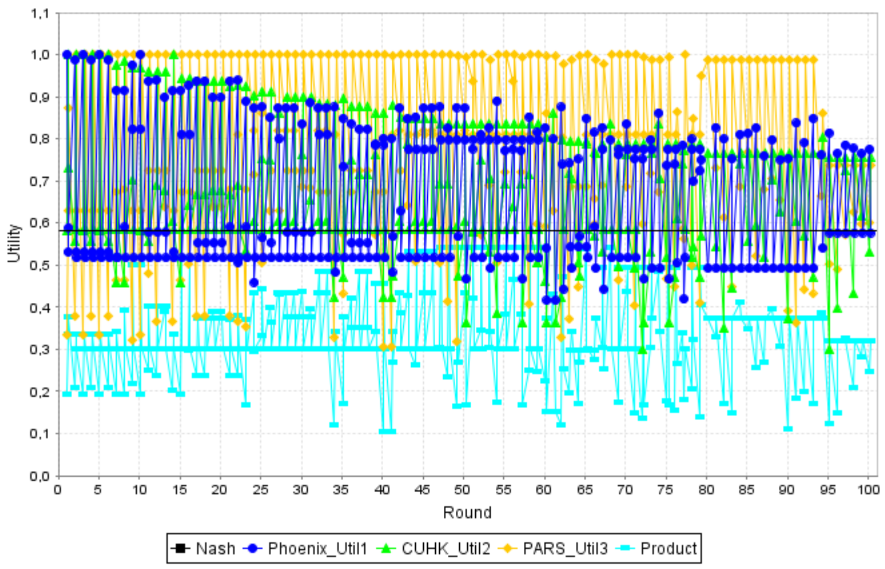

Fig. 3: Session 3, Utilities[0,0,0]

From the graphs can be deducted that the profiles are distinct enough to provide a good testing simulation for the agents. The final utilities of the agents are not very high, in the third round there isn't even reached an agreement.

3 Task 2, Simple agent design

In this chapter the design of the simple agent will be discussed.

3.1 Design goal

The goal of this task is to create an agent capable of defeating 2 random agents in a to be determined domain. This means that the agent should be able to obtain a higher utility than its adversary. Since the first opponents of the agent will be 'Random Walkers',i.e. a negotiation agent which offers random bids, the designed agent will not use opponent modelling to try and predict the behaviour of the opposing agents. Instead there will be focused on constructing basic and functional bidding and acceptance strategies.

Opponent Modelling Acceptance Strategies
The following acceptance strategies were taken into consideration:

1. Annealing
2. Hill Climber
3. Custom: A base utility which deviates based on independent time and utility functions.

These types of basic acceptance strategies have their distinct advantages and disadvantages. For example the paper by Klein[**?**] suggests that the sum of the utilities is higher in a contest with solely annealing agents than for a group of Hill Climber agents. When a Hill Climber agent is paired with annealing agents however, the Hill Climber receives a higher utility value.
The third and last option considered for the basic agent is an acceptance strategy that bases the lowest utility it will accept on a fixed desired utility, multiplied by normalized weight factors for the time and the utility offered by the opponents. These are determined by independent functions. more on this in section **??**.

Bidding Strategies
For the bidding strategies the following options were considered:

1. Random Walker
2. Time dependent concession strategy
   (a) Conceder tactic: $\beta > 1$, concedes fast and goes to its reservation value quickly.
   (b) Boulware tactic: $\beta < 1$, hardly concedes until the deadline.
3. Trade-Off strategy
4. Behaviour dependent strategies

The amount of known bidding strategies which are useful when dealing with 'Random Walker' based agents is limited to variants of options one and two. This is because the opponent models option three and four use are easily misguided with random bids. Since the opponent bids purely at random, these agents will not perform as well as options 1 and 2 against such an agent.

3.2 Tournament

The multi-party tournament is set up as follows:

- 2 Opponents, both of the random walker type.
  - Bid Utility ≥ 0.6 (according to lecture 3)
- The domain is the party_ domain with known preference profiles.

3.3 Design and motivation

The agent that is created uses the following strategies:

- **Bidding strategy**: Boulware strategy
- **Opponent modelling**: none
- **Acceptance strategy**: annealing strategy

The boulware strategy persistently offers the same high utility, hoping that the random walker accepts it at some point. The annealing strategy is a more social strategy which also takes into account the time left, so that the change of not coming to an agreement is greatly reduced. The simulated annealer calculates the probability it will accept the opponents bid, based on the difference between the offered utilities and the time.

3.4 Analysis of the simple agents' performance

After simulating the agent against the random walkers, we concluded that the annealing strategy is not optimal. The agent had a tendency to accept far quicker than its bids suggested. This was due to the 'randomness' of the annealing strategy.

$$P_{accept} = min(1, e^{-\delta U/T}) \tag{1}$$

It is expected that this problem will increase when playing against more sophisticated opponents, because the random walkers concede a lot when $T \approx 0.9T_{end}$, and other agents will probably not.

4 Task 3, Final agent

In this section, the design and performance of the final agent will be discussed. This agent is not based on the simple agent, although we used the result analysis of the simple agent to decide on certain strategies. In figure 4 the model of our agent is shown.

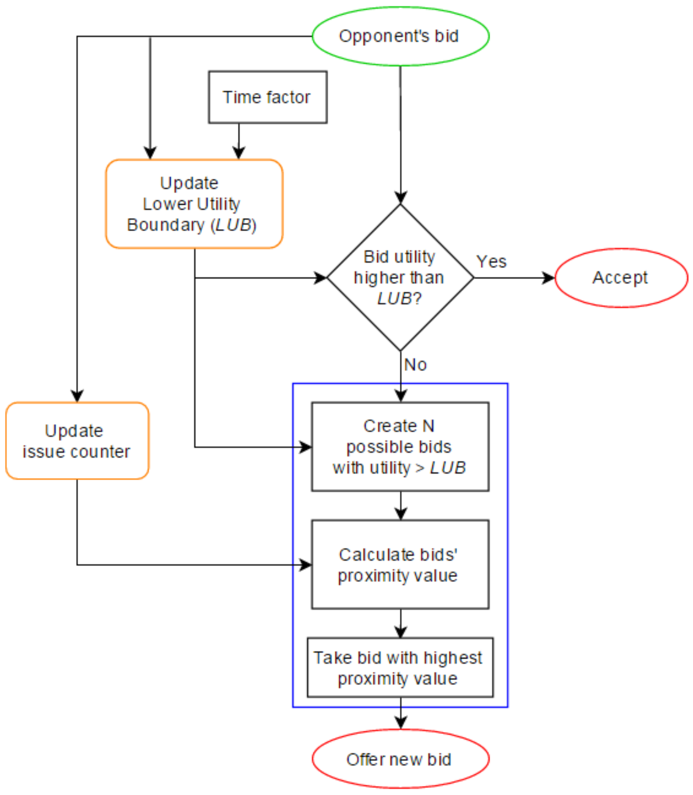
Opponent's bid
Time factor
Update Lower Utility Boundary (*LUB*)
Bid utility higher than *LUB*?
Yes
Accept
No
Update issue counter
Create N possible bids with utility > *LUB*
Calculate bids' proximity value
Take bid with highest proximity value
Offer new bid

Fig. 4: The structure of the negotiation agent

The aim is to design a very social agent, which will go far to reach an agreement with its opponents and who takes everyone's interests into account, while still prioritizing its own preferences.

4.1 Opponent modelling

Opponent modelling is usually approached by creating one model per opponent party, and taking a (possibly weighted or nonlinear) average of all the models when creating a new bid to offer. Since our agent is strongly focused on a social approach there is no difference between opponent agents, and their offerings are handled as a whole. To do this, we make use of the *issue counter*. Every

time an opponent makes an offer or accepts a bid, the counter of the bid's issue arguments (eg. [beer][catering food][ballroom]) all go up by 1.
When considering a new bid to make, the *issue counter* is normalised to create a weight set for all issue arguments (an example is shown in Table 1). The bids utility according to these weights is then calculated, and the obtained value is called the *proximity value.* A bid with a high *proximity value* has issue arguments close to the average offered (and accepted) bid of all opponents, and has thus a higher chance of being accepted by the opposing agents (eg. using the weights of Table 1, the bid:[Wine][Catering][Classical][Ballroom] has a *proximity value* of $prox = 0.5 + 0.3 + 0.4 + 0.6 = 1.8$).

Tab. 1: Opponent model: drinks issue

Soda	Beer	Wine	Cocktails
0.1	0.25	0.5	0.15
Catering	Fingerfood	Chips	Handmade
0.3	0.5	0.15	0.05
Classical	Jazz	Rock	
0.4	0.15	0.45	
Ballroom	Party Tent	Dorm	Own Room
0.6	0.2	0.05	0.15

4.2 Lower boundary utility (LBU)

The Lower Boundary Utility (LBU) is a utility value that is used in both acceptance and offering strategy. The LBU is obtained in three steps discussed below, in sections 4.2.1 and 4.2.2.
Whenever the agent receives a message from an opponent, the LBU is updated. If the opponent's message is an acceptance the value of the bid that the opponent accepted is used for the update, as it is considered as a bid that the message sender would have offered.

4.2.1 Discounted mean utility of opponents' offerings (DMUOO)

In order to get an understanding of how far the profiles of the opponents are from our own, it is necessary to consider the mean utility of the opponents' bids: if the opponents are offering, on average, bids with an utility lower than 0.6 it is quite naive to make bids with an utility of 0.95 and expect the opponents to accept them. On the other hand, if the opponents are bidding in an utility range of 0.8 offering bids with an utility of 0.7 would make no sense.

The $DMUOO$ function is shown below:

$$DMUOO = (1 - \gamma) \cdot DMUOO_{prev} + \gamma \cdot Utility_{receivedBid} \tag{2}$$

Here γ is the discount factor, which defines how heavily the current mean weights compared to the incoming bids:

$$\gamma = \frac{Sensitivity}{TNOB} \tag{3}$$

The *TNOB* value is the Total Number of Opponents' Bids, that is the expected total amount of bids that the agent would receive from the opponents if the negotiation were to continue until the end of the negotiation time. The *TNOB* is calculation based on the number of bids received at the present time, divided by the negotiation time factor (in the range [0,1]). It is updated every bid so that it becomes more accurate further in the process:

$$TNOB = \frac{NOB}{time} \tag{4}$$

The reason to use the *TNOB* is that the discount behaviour must be independent of the total time of the negotiation.

The discount sensitivity factor determines how quickly the *DMUOO* converges to the real mean of the opponents' offerings.
This factor is an important agent behaviour definer: Choosing a high value (10) makes the *DMUOO* quickly reach the real mean, but also gives little time to the opposing agents to model our own profile preferences. Choosing a small value (0.1) makes the *DMUOO* decrease less steeply from the high initial value (set to 0.95) but also makes it less reactive to changes in the average opponents' bids, which increases the risk of never reaching an agreement due to a decrease in willingness to concede.

4.2.2 Time factor

The time function is a function that is implemented in our agent to increase the chance of finding an agreement if the negotiation session is reaching the very end. While the function returns 1 for the most part of the negotiation (thus not influencing the *LUB* value), it decreases greatly when the negotiation session is reaching its end. The time function (β) calculates a normalized factor by taking into account the time left to negotiate. It is thus solely dependent of the total time and the time that passed. It looks as follows:

$$\beta = 1 - \left(\frac{time}{T}\right)^{\alpha} \tag{5}$$

where α is manually selected after looking at the behaviour of the function. This behaviour is shown in figure 6 for $\alpha = 30$, which was selected in the agent.

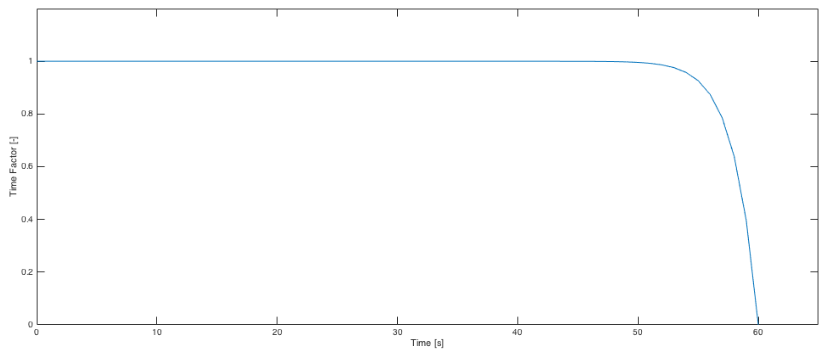

Fig. 5: Behaviour of the time function

4.2.3 LBU construction

The LBU value is then calculated using the above calculated values:

$$LBU = \beta \cdot (\zeta + (1 - \zeta) \cdot DMUOO^{\epsilon}) \tag{6}$$

The $DMUOO$ value is first implemented in a quasilinear function, using the two (fixed) coefficients ζ and ϵ, and then multiplied by the time factor β. The quasilinear function determines how selfish the agent is. Choosing high values of ζ will result in a LBU that is always decisively above the $DMUOO$, and thus will result in a selfish agent that never lowers his utility boundary too much. The fixed value given to ζ is 0.4.
The ϵ coefficient determines how fast the LBU approaches $DMUOO$ for higher values of the latter. In figure 5, this function is plotted with different values of ϵ. The condition for selecting ϵ was that the function would not overstep the linear line from (0,0) to (1,1). This line indicates the boundary where we select a utility equal to the $DMUOO$, and this is undesired.

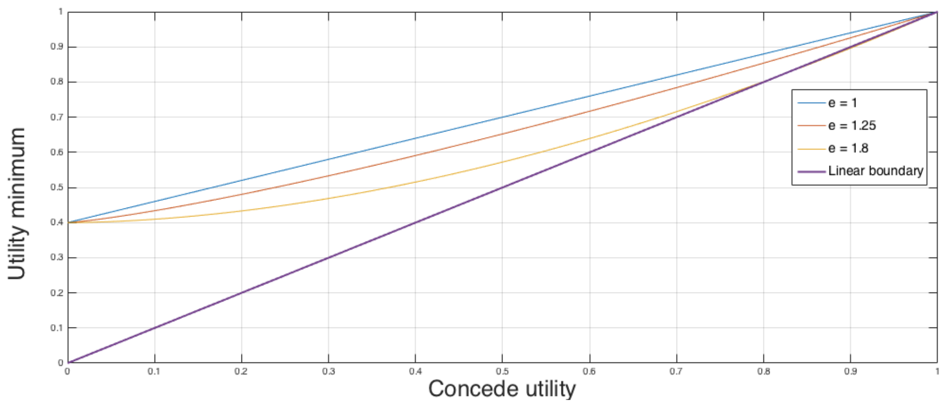

Fig. 6: Minimum utility function

As can be seen in the figure, $\epsilon = 1.8$ touches the linear boundary at $U_{min} \approx 0.75$. We deemed this value too low, and $\epsilon = 1.25$ was thus selected.

4.3 Acceptance Strategy

The acceptance conditions of the agent is simple: accept every bid above the LBU. If the offered bid is not accepted, a new bid is offered (Offering strategy).

4.4 Offering strategy

The creation of a new bid to offer is determined in three steps: Generation of random bids, evaluation of proximity, selection of best bid (In the blue box in Figure 4.

4.4.1 Generation of random bids

At first, n random bids are generated, all with value above the LBU, and added to an ArrayList.
The value n is fixed, and set to 30. A high number results in a long processing time, but a potentially more effective bid selection, while having a small n is fast, but at the cost of reliability.

4.4.2 Evaluation of proximity

For every generated bid, its proximity value is calculated, in the opponent modelling, and added to an ArrayList.

4.4.3 Selection of best bid

As last step, the bid in the array that has the best proximity value is chosen as the bid that is going to be offered to the opponents.

5 Methods description

All the methods used in the agent class are shown below, with a brief description

5.1 initfunc

```
private void initfunc() throws Exception {...}
```

This method is run once at the very beginning of the negotiation (bidNum=-1). While this implementation is not the prettiest, it does work, while the init() function is somehow never run in negoSimulator.jar

This function defines variables and initializes the ArrayLists.

5.2 chooseAction

```
public Action chooseAction(...) {...}
```

This method is run when the agent must take an action (Accept or Offer new bid).
Lines 66-73: If negotiation just started, run initfunc (see initfunc)
Line 74: set turn counter up (happens for every party's turn, so in chooseAction and receiveMessage).
Lines 77-82: Obtain utility of last offer
Lines 87-92: If the offer is acceptable (see isAcceptable) and accepting is an option, update BidHistory and accept
Lines 93-97: Otherwise send a new Offer (see getBid)

5.3 receiveMessage

```
public Action receiveMessage(...) {...}
```

Method runs whenever an opponent chooses an action (accepts or makes a new offer).
Lines 110-117: If negotiation just started, run initfunc (see initfunc)
Line 118: set turn counter up (happens for every party's turn, so in chooseAction and receiveMessage).
Lines 121-132: If received message is a bid, set as lastBid, otherwise take bid of last offering party.
Line 135: Update LBU(see lowBoundUtil_update and section 4.2 starting from the second round.
Line 139: Update the *issue counter* with lastBid.

5.4 isAcceptable

```
private boolean isAcceptable(double offered){...}
```

Checks if the utility of the offered bid is higher than the minimum accepted (reservation value) and if it is higher than the LBU (see lowBoundUtil_update and section 4.2).

5.5 getRandBid

```
private Bid getRandBid(double minUtil){...}
```

This method returns a bid for the given domain, with a utility of at least *minUtil*

5.6 BH_update

```
private void BH_update(int oppID, Bid newBid) {...}
```

Method runs for every action chosen or received, adds the bid *newBid* to the bid history of player with id *oppID*.

5.7 Counter_update

```
private void Counter_update(Bid recBid)...{...}
```

This function updates the *issue counter* with *recBid*.
The counter works by having two concurrent ArrayLists of ArrayLists, one with the string values of issue argument names (eg. "beer","cocktails"... in IssueNames(1); "catering","fingerfood"... in IssueNames(2)) and one with their relative counters (eg. 23,50,... in IssueAmounts(1); 15,68,... in IssueAmounts(2)).
Line 188: for every issue in the domain (eg. food,drinks,location...)
Lines 190-193: If there are no elements in IssueNames(k) then automatically add given bid issue to the names, also add a counter in IssueAmounts(k), and set found to true (this is to avoid making an impossible for loop in line 196).
Line 196: for every issue argument in IssueNames
Line 199: if the bid's issue argument is equal to that in IssueNames
Lines 200-204: Add one counter to the IssueCounter of that issue and set found to true
Lines 207-209: if found is not true, the received issue argument is not yet in the IssueNames array, thus we add it and we add a new counter (set to 1 of course).

5.8 normaliseMean

```
private void normaliseMean(){...}
```

Updates the normalised issues counters, used to calculate the proximity value of a bid.
Line 219: For each known issue (should always be equal to the number of issues)
Lines 220-223: calculates total amount of counters of all issue arguments
Line 224: clear old normalised issue arguments
Line 225: creates a temporary ArrayList of values
Lines 226-228: for each issue argument, return its counter divided by the total amount of counters
Line 229: set the temporary ArrayList as the new normalised issue arguments.

5.9 getBid

```
private Bid getBid(){...}
```

getBid follows the Offering strategy described in section 4.4.
Line 238: get LBU
Line 242: only create a bid array if we are already in the second turn, otherwise just create a random bid above the LBU (Lines 266-268)
Lines 243-244: create an empty ArrayList of Bids and one of Proximity values
Line 246: update the normalised mean of issue counters (see normaliseMean)
Lines 248-257: Create RAND_BID_AMOUNT random bids above the LBU, and calculate their proximity to the normalised mean (see getProx)
Lines 259-263: Find the best proximity value in the array
Line 265: Take bid BidArr(x), where x is the index of the best proximity value in the array
Line 269: Return the found bid.

5.10 getProx

```
private double getProx(Bid bid)...{...}
```

getProx calculates the proximity value of a bid to the normalised Issue counters array.
Line 277: for every issue
Line 278: for every issue argument
Line 279: if the name of the issue value of the bid matches that of the issue argument
Line 280: add its normalised value to the total proximity value
Line 284: return total proximity value

5.11 lowBoundUtil_update

```
private void lowBoundUtil_update(double lastUt){...}
```

Function that updates the LBU given the utility of the last opponent's action. This method is explained thoroughly in chapter 4.2.

6 Agent evaluation

The agent has to be tested in order to see where it stands in the negotiations. For this testing, the Atlas and Mercury agent were suggested. We replaced the parsagent with the phoenix agent.

6.1 General Remarks concerning the evaluation

The evaluation of the agent will take place in the party domain. There are specific reasons to choose this domain. Firstly, this domain will be the domain on which the agent will eventually be evaluated. Secondly, this domain has a good amount of issues with a good amount of options per issue.

6.2 Evaluation Session 1

The first testing session, which was an exploratory session to check if the agent was able to function in a negotiation tournament, was conducted using multiple combinations of profile *Party1_utility-Party4_utility*. It was discovered that profile 3 is a profile which is more 'unique' than the other profiles. This means that the agent with profile 3 will be likely to receive a lower utility than the other agents. The results of these sessions can be seen in the following table:

These results show that the agent is able to generate acceptable utilities when it received preference profile 1, however with preference profile 3 the results are poor. In simulation 3 the agent at least succeeds in generating more utility than the Mercury agent. In simulation 4 the results are abysmal, only 0.598 utility was received. Interestingly only simulation 1 resulted in a Pareto optimal solution.
The cause of the relative low utilities for profile 3 appeared to be the tendency

Tab. 2: Results of Multi-Party negotiation for Iteration 1, 60[s] simulation time

	Sim 1	Sim 2(3x)	Sim 3	Sim 4
Profile 1	Group19: 0.836	Phoenix:-	Phoenix: 0.912	Mercury: 0.773
Profile 2	Phoenix: 0.840	Atlas:-	Mercury: 0.518	Phoenix: 0.888
Profile 3	Mercury: 0.704	Group19:-	Group19: 0.546	Group19: 0.598
Profile 4	Atlas: 0.774	Mercury:-	Atlas: 0.786	Atlas: 0.679
Pareto Distance	0	-	0.106	0.034
Nash Distance	0	-	0.366	0.163

of the agent to concede too early in the negotiation, this can be seen in the following figure:

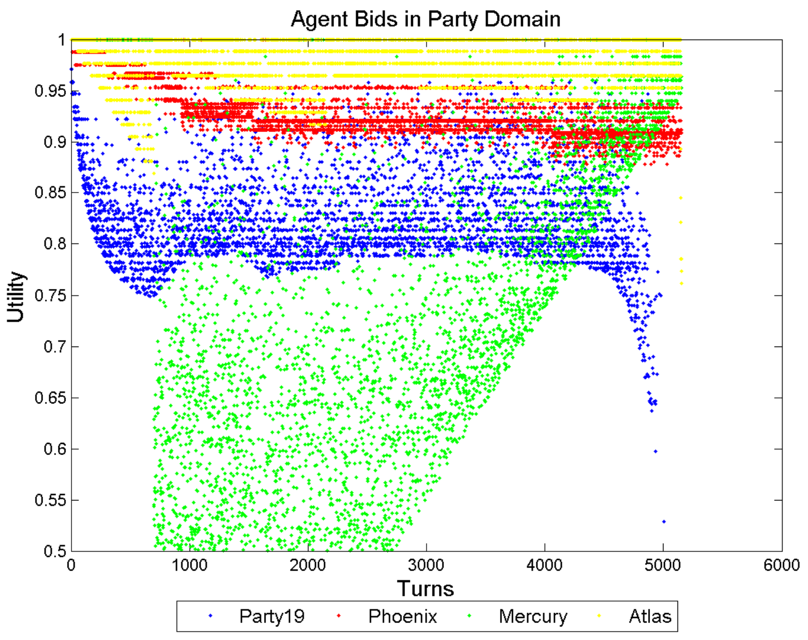

Fig. 7: The bidding utility of each agent during simulation 3

In the figure it can be clearly seen that the agent concedes too fast and maybe will even disturb the modelling of our behaviour by opponents. Furthermore, when no agreement is reached and the simulation is reaching T_{end} the agent will concede to a much lower utility in an attempt to close a deal resulting in a low utility. This near T_{end} behaviour is the social trait our agent.
From the figure it can clearly be seen that the Mercury agent is the most explorative agent, and that atlas and phoenix have a somewhat 'selfish' strategy.

After Evaluation Session 1 it was concluded that the agent should concede less, which will allow the negotiation to shift more in favour of our agent. This was done by changing the *DISC_SENSITIVITY* value from 10 to 1 in an attempt to make the agent tougher in the negotiation.

The team also noticed an error in the implementation of the mean calculation. The agent was calculating the discount value per negotiating turn instead of per received bid. This resulted in a discount value that did not correctly take the amount of adversaries into account when calculating the discount value. This error was fixed in session 2.

6.3 Evaluation Session 2

After implementing the suggested improvements from evaluation session 1, simulation 3 was rerun a multiple of times resulting in the following values:

Tab. 3: Results of Multi-Party negotiation after 2 sessions, 60[s] simulation time

	Simulation 3(2x)
Profile 1	Phoenix: 0.894
Profile 2	Mercury: 0.504
Profile 3	Group19: 0.709
Profile 4	Atlas: 0.786
Pareto Distance	0
Nash Distance	0.342

This particular simulation was run 2 times resulting in exactly the same utility for all parties. The reason for this is probably that the solution is Pareto optimal, meaning that it is an optimal solution.
The bidding information of one of these simulations is shown below:

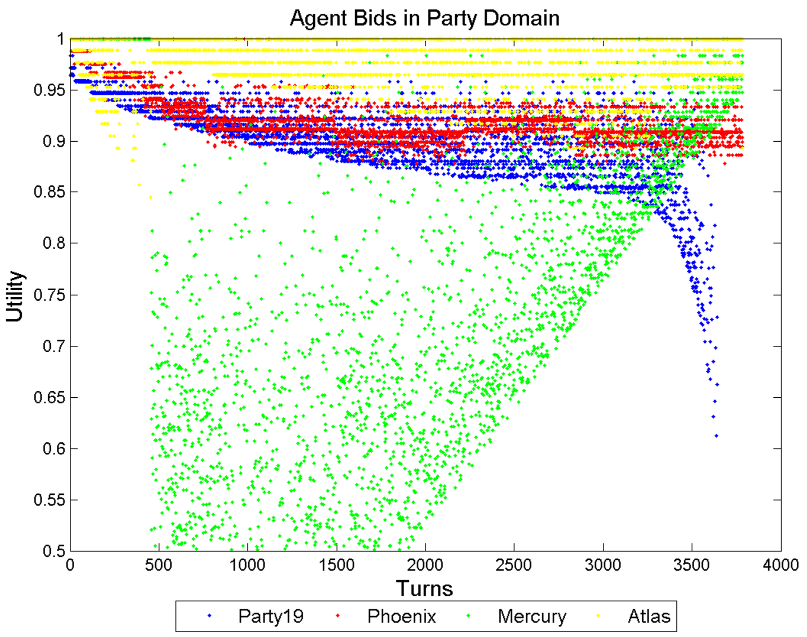

Fig. 8: The bidding utility of each agent during simulation 3 for a discount value of 1

When comparing this figure with figure 7 it is clear that the opponents bidding has not significantly changed. However, the bidding behaviour of our agent has clearly changed. The agent is much less conceding in the first 75% of the negotiation. It remains a social agent and will still concede to a lower utility in an attempt to close a deal. The group concluded that the change of the discount sensitivity improved the result of the agent in this situation and this change will therefor be permanent.

6.4 Evaluation Session 3

In this evaluation session the agents ability to reach the Nash point was tested in the KillerRobots Domain. This domain was chosen because it contains a mix of discrete and integer type issues. The results of the simulation can be found in the table below:

Tab. 4: Results of Multi-Party negotiation for Iteration 3, 60[s] simulation time

	Sim 1(3x)
Profile 1	Group19.1: 0.927
Profile 2	Group19.2: 0.851
Profile 3	Group19.3: 0.847
Pareto Distance	0
Nash Distance	0.042

The results from the simulation, which was run multiple times with similar results, show that the agent is able to come close to the Nash equilibrium point. this proves that the agent explores enough options to reach this optimum. This can be seen from the bidding behaviour as well:

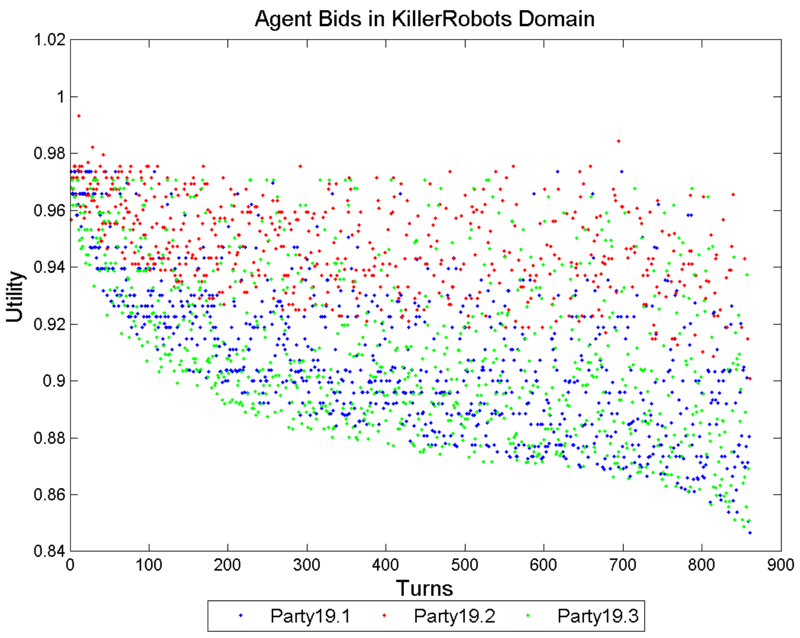

Fig. 9: The bidding utility of each agent during a simulation in the KillerRobots Domain

Observing the figure it becomes clear that the average of the utilities lies closer to the profile of agent 2 than of the other 2 agents. The red agent continuously offers a higher bid for himself than the other agents. Contrary to what one might expect, this does not mean that agent 2 is in the most favourable position. It only means that the mean of the other agents is close to his profile.

6.5 Evaluation Session 4

For this evaluation session the team decided it was time to test the agent using the 'Multiparty tournament' option in genius. For this simulation the four party_domain profiles were once again used. After one tournament consisting of 24 rounds the average utilities were compared between the agents, this led to the following results:

Tab. 5: Average utility after 2x24 simulations of 60[s] in the Party_Domain

	Average Utility T1	Average Utility T2
Phoenix	0.602	0.608
Mercury	0.432	0.431
Atlas	0.452	0.460
Group19	0.439	0.432

The average utilities resulting from the simulations are relatively low because there were quite some combinations of agents and profiles where an agreement was not reached. This explains the low average utility. What is interesting to see is that our agent, although outperformed by the Phoenix agent, is able to function well in such a difficult negotiation, especially when compared with the Mercury and Atlas agents. Despite this it is still a social agent.

6.6 Evaluation Conclusion

After performing multiple evaluations and improvements the agent performs according to our design goal. The agent is social, it will attempt to reach an agreement. At first it will stay close to its own profile's optimum and later it will be more flexible and concede to an agreement which is suitable for all parties. If the time is at it's end, it will do anything to get an agreement, accepting any offer. This design choice makes it virtually impossible for the agent to get the highest utility during a negotiation session. This is of course a direct result of choosing a social agent.

7 Acknowledgements

During the evaluation it became clear that there were a lot of parameters in our agent which could be optimized. This together with the fact that there were no descriptions of the provided anac2015 agents made it difficult to create a suitable test protocol to optimize our agent's performance. The team experimented with different values for key parameters outside the scope of this report, however the changes did not consistently lead to higher utilities and were therefore not implemented or documented.
We did notice that from the tests the Phoenix agent performed best although we suspect that this agent, together with the Atlas agent, are not interested in reaching an agreement at all cost. For example, they sometimes try to bid a utility of 1 in the end of the negotiation session which leads to a failed negotiation.

Almost all agreements were reached in the last few seconds of the negotiation simulations which means that our agent will accept nearly all bids and therefore the 'road' to the final few seconds is quite important for our agent. If the the negotiation converges to our solution in the first 90% of the simulation time the agent is more likely to get a higher utility.

As a last remark we would like to point out that this assignment was a joy to work on and very valuable for our understanding of negotiation agents. Even though our group consists solely of 3me master students with no prior Java experience, we were able to develop a working agent that also does what it's supposed to do.
While the report is longer than 15 pages, the description of the final agent and its evaluation (Task 3 and 4) cover 14 pages in total. We assume that this will not be a problem for the given page limit (15 pages).

8 Preference Profile

Tab. 6: Preference Profile Weights

Preference Profile	Food	Drinks	Location	Invitations	Music	Cleanup
Party_util(1)	0.2	0.1	0.2	0.1	0.35	0.05
Party_util(2)	0.28	0.2	0.12	0.24	0.05	0.12
Party_util(3)	0.01	0.17	0.41	0.4	0.15	0.23

Tab. 7: Food

Preference Profile	Chips and Nuts	Finger-Food	Handmade Food	Catering
Party_util(1)	2	4	8	6
Party_util(2)	1	1	3	5
Party_util(3)	3	2	4	

Tab. 8: Drinks

Preference Profile	Non-Alcoholic	Beer Only	Handmade Cocktails	Catering
Party_util(1)	4	3	1	1
Party_util(2)	4	3	1	2
Party_util(3)	1	1	4	5

Tab. 9: Location

Preference Profile	Party Tent	Your Dorm	Party Room	Ballroom
Party_util(1)	2	1	4	3
Party_util(2)	5	4	1	1
Party_util(3)	4	3	1	3

Tab. 10: Invitations

Preference Profile	Plain	Photo	Custom, Handmade	Custom, Printed
Party_util(1)	4	1	3	3
Party_util(2)	1	1	3	4
Party_util(3)	3	1	2	2

Tab. 11: Music

Preference Profile	MP3	DJ	Band
Party_util(1)	1	2	6
Party_util(2)	3	2	1
Party_util(3)	2	3	2

Tab. 12: Cleanup

Preference Profile	Water and Soap	Specialised Materials	Special Equipment	Hired Help
Party_util(1)	4	3	1	2
Party_util(2)	5	4	3	2
Party_util(3)	1	2	2	2