Delft University of Technology

IN4010 Practical Assignment 2

Automated Negotiation

Group 11:
Hidde Coehoorn
Ralf Nieuwenhuizen
Jan-Willem van Velzen

Supervisor:
Reyhan Aydogan

January 20, 2015

1 Introduction to assignment

As part of the TU Delft course IN4010 Artificial Intelligence Techniques completion of a practical assignment on automated negotiation is required. This is the assignment report of group 11, of which the group members are Hidde Coehoorn, Ralf Nieuwenhuizen and Jan-Willem van Velzen.

The subject of the assignment is multilateral negotiation taking place in a multi-issue negotiation domain with discrete issues. The challenge is to design and implement a negotiating agent in GENIUS which can guide the negotiation process and help the parties involved with reaching a satisfying agreement. Various scenarios are created to test the agent, ranging in the degree of conflict between parties.

1.1 Domain

Each group had to define their own negotiation domain, complete with multiple issues. Our group chose to create the 'Purchasing a Car' domain. This domain has a total of five issues, each with a number of issue values ranging from two to four (see Table 1).

Power (hp)	Capacity	Mileage (km)	Fuel consumption (km/l)	Wheels
less than 200	2	Below 10.000	5	3
200 or more	5	Between 10.000 and 20.000	10	4
	7	Above 20.000	15	6
			20	

Table 1: The domain issues and issue values.

In this domain nine preference profiles are created in three sets of three each. Profiles 1, 2 and 3 will negotiate with one another, as will 4,5 and 6 and so on. The first three profiles are negotiating in a collaborative scenario, the second set is a moderate scenario and the last three profiles clash in a competitive scenario.

	Profile	1	2	3	4	5	6	7	8	9
	Issue weight	0.10	0.15	0.20	0.40	0.15	0.40	0.10	0.35	0.50
Power (hp)	less than 200	2	1	2	2	1	2	2	6	1
	200 or more	1	1	5	1	1	5	1	1	4

Table 2: preference profiles 1-9 for issue 1

	Profile	1	2	3	4	5	6	7	8	9
	Issue weight	0.20	0.20	0.20	0.15	0.20	0.15	0.10	0.05	0.10
Capacity	2	3	2	3	3	2	3	6	2	5
	5	2	2	3	2	1	3	4	3	1
	7	1	1	1	1	1	1	2	4	3

Table 3: preference profiles 1-9 for issue 2

	Profile	1	2	3	4	5	6	7	8	9
	Issue weight	0.30	0.35	0.20	0.20	0.35	0.15	0.10	0.05	0.15
Mileage (km)	Below 10.000	3	2	3	2	4	6	6	4	1
	10.000-20.000	2	2	1	5	1	4	3	3	10
	Above 20.000	1	1	1	4	3	1	1	1	25

Table 4: preference profiles 1-9 for issue 3

	Profile	1	2	3	4	5	6	7	8	9
	Issue weight	0.15	0.10	0.20	0.10	0.10	0.15	0.10	0.05	0.05
Fuel consumption	5	2	13	4	2	13	4	5	13	1
(km/l)	10	3	12	3	3	12	3	2	10	20
	15	1	1	1	1	1	1	3	1	18
	20	1	10	2	1	10	2	3	10	17

Table 5: preference profiles 1-9 for issue 4

	Profile	1	2	3	4	5	6	7	8	9
	Issue weight	0.25	0.20	0.20	0.15	0.20	0.15	0.60	0.50	0.20
Wheels	3	3	3	3	4	4	2	8	1	12
	4	2	3	2	3	1	3	7	6	2
	6	1	1	1	1	2	6	1	7	8

Table 6: preference profiles 1-9 for issue 5

2 Agent

Besides that our agent is one awesome badass, this might not be obvious by the look of his humble output. There are a couple of ingenious details that are worth pointing out.

2.1 Tactics

Our agent has a number of Tactics at its disposal to use at any given moment during the negotiation. These Tactics take the current negotiation state into account, and return an Action based on their nature.

The following Tactics are defined:

1. RANDOM: Offers a random bid above the reservation value.
2. BESTNASH: Offers the best Nash bid according to the OpponentModels.
3. NOSTALGIAN: Offers the bid from the negotiation history with the highest utility.
4. ASOCIAL: Offers the bid that has the highest utility.
5. HARDTOGET: Offers a bid with 0.99 times the utility of the previous bid.
6. EDGEPUSHER: Offers a bid slightly beter than the previous bid.
7. GIVEIN: Offers a bid that is slightly worse compared to the last.
8. THEFINGER: Leaves the negotiation.

2.2 Opponent Utility Model

In order to make a prediction about the utilities of the opponent, we keep track of their actions and compile this into usefull information. Every opponent in the negotiation gets their own model.

First of all, we keep counters for all the issue-values in the Domain, and measure how often an opponent includes these values in the bids that it offers. These counts are then used to compute the issue weights and issue-value weights. For every issue, the statistical variance in the issue-value counts is calculated. These are then normalised over all issues, while making sure that no issue has a weight of 0. Lastly, the issue-value weights are made equal to the issue-value counts.

2.3 Opponent Strategy Model

Besides estimating the opponents utility, we can also make a very rough estimation of the opponents tactic. This metric is only used when it's not possible to create a reliable utility model for the opponent. The model does this by taking the opponents bidding history into account, where it records what bids are done by the opponent, but also what the previous bid was that this new bid could be a response to. Every bid that the opponent has done is then compared to two other bids and a distance measure is calculated. The first bid is the previous bid in the negotiation, the other is the previous bid by the opponent. These can be the same, but most likely won't be. If the distance to the first bid is smaller than the distance to the other bid, it is more likely that the opponent's tactic is to alter the previous bid in the negotiation. If not, it is more likely that the opponent is only changing his own bid every round.

2.4 Our own strategies

Apart from the way our agent analyzes its opponents, there is a strategy for its bidding routine as well. We will describe this from the beginning. Furthermore, we would like to state that our agent differentiates between long and short negotiations, because it takes a different strategy for each of them. The critical value is currently set at 20 rounds. All of the described strategies can be seen in figure 1

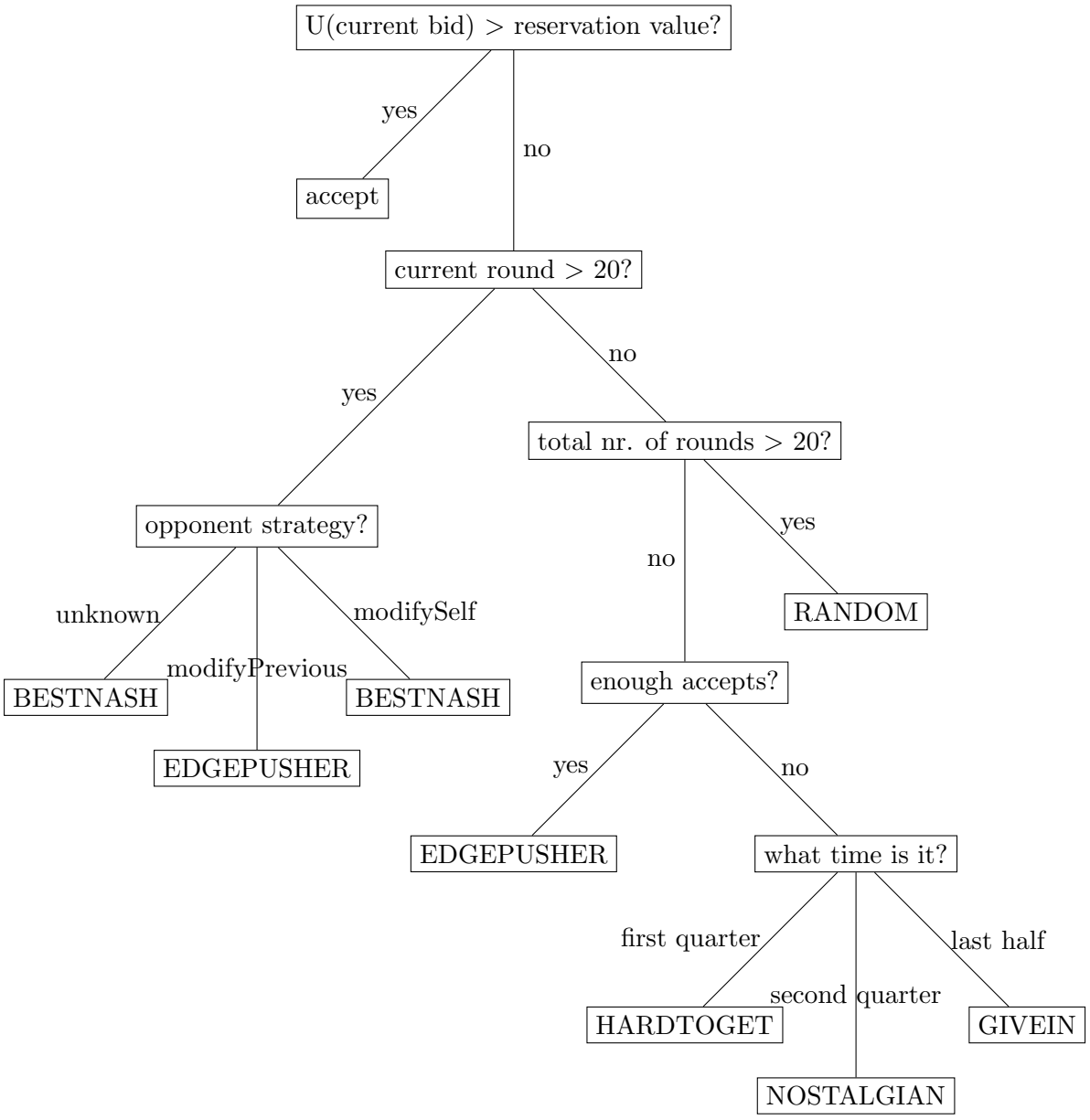

Figure 1: The decision tree of the agent's strategy

2.4.1 The first bid

For the first bid, we really like to make a statement, so we choose the ASOCIAL tactic, which means we offer the best bid for ourselves.

2.4.2 Accepting offers

Apart from several accepting stages in the tactics that will be explained in the next sections, there are certain offers that will be accepted at any time. One possibility is an offer with a utility that is higher than the reservation value. This will always be accepted. This may sound weird, but the reservation value starts off at 0.95, which is not bad at all! This reservation value can be lowered in the final stage (see section 2.4.5), or when the majority of other parties seems to accept an offer. In this last case, the reservation value is multiplied by 0.6, so the offer might be accepted then.

2.4.3 Long negotiation

In a long negation (i.e. there are more than 20 rounds in total), we have the possibility to build a trustworthy opponent model.

As long as we do not have this model yet (20 rounds have not passed yet) we will apply the RANDOM tactic, which means we will do a random offer above our reservation value. The opponent will either accept, which is fine, or offer according to its own strategy, which is also fine, because that means we can determine its preferences.

After 20 rounds we will have a trustworthy model, so we will act according to that. Right now, the most important part of this, is when the opponent is believed to offer bids depending on our own bids. When this is the case, we will apply the EDGEPUSHER tactic, which tries a slightly better offer for us each round, to see whether the opponent might modify that and make a better offer for us to accept. If the opponent is likely to modify his own bids or if the strategy is unknown, we apply the BESTNASH tactic. This picks the best offer from a list of bids sorted on their calculated Nash product, determined by using the opponent models. This bid may change every round due to other offers the opponents might make.

2.4.4 Short negotiation

In a short negotiation (i.e. there are less than 20 rounds in total), we will never have a trustworthy opponent model. We will now apply a tactic based on the time that is left for this negotiation.

In the first quarter of the negotiation, we will apply the HARDTOGET tactic, which means we will offer a bid with a utility around 0.99 times the utility of our last offer.

In the second quarter of the negotiation, we will apply the NOSTALGIAN tactic, which means we will stick to the best bid that has been done during the entire negotiation.

Finally, in the last half of the negotiation, we will apply the GIVEIN tactic, which is very descriptive in itself. We will give in a bit. Each next offer is around the utility of $reservationvalue * utilityoflastoffer$, so each offer is more like to be accepted by the opponents.

As a final remark to the short negotiations, it has to be said that if the majority of people accepted an offer, we will at any time apply the EDGEPUSHER tactic.

2.4.5 The final stage

When the time is running out, we believe that making a bad agreement is always better than making no agreement. Therefore, we will accept any offer when the negotiation is over 95% of time.

To prevent that this final accepted offer is not the worst for us, starting from 85% of time, we are lowering our reservation value. Whereas this is 0.95 before this stage, it is now linearly lowered down to 0.5 at 95% of time. Do you recall that we accept any offer above our reservation value? Well, this works! So most likely, an accepting offer is made before we reach 95% of time, which will most likely be quite good.

2.5 How to read this in the source code

Even though the source code is well documented, it may be hard to figure out where to start, so here we will provide an explanation of its structure.

2.5.1 The negotiation agent / Group11.java

This is the main class of our agent. All the other classes are used from here, and this class is used to start our agent in GENIUS. There are several methods in this class that we will address. Every other method is provided with JavaDoc as well, so it should be clear from the source code what each method does.

- **chooseAction** This method is called by GENIUS each round, and it should return the action this agent is going to perform, which can be an Accept, an Offer, or an EndNegotiation. What this method does for us is checking whether the reservation value should be changed, checking whether the previous offer is acceptable, based on those parameters, and otherwise choose the tactic to apply, based on the strategy described in section 2.4.
- **getActionForTactic** This method is called from the chooseAction method, and basically it just defines the different tactics.
- **receiveMessage** This method is called by GENIUS every time *any* agent performs an action, including this agent. Here we create an opponentUtilityModel for each opponent, and fill it with the offers they make and the offers they accept.
- **getNashUtilityProduct** Based on a list of opponent models, this function determines the Nash product for a certain bid.

2.5.2 The opponent model / OpponentUtilityModel.java

This class is instantiated for each opponent of our agent, and it contains the counters for each issue in the domain. It also uses the OpponentBidHistory and the Statistics classes. We will not describe any method of this class in detail, we will just say it implements the models described in sections 2.2 and 2.3.

- ~~an explanation and motivation of all of the choices made in the design of negotiating agent.~~

- ~~should help the reader to understand the organization of the source code (important details should be commented on in the source code itself).This means that the main Java methods used by your agent should be explained in the report itself.~~
- ~~a high-level description of the agent and its structure, including the main Java methods (mention these explicitly!) used in the negotiating agent that have been implemented in the source code.~~
- ~~an explanation of the negotiation strategy, decision function for accepting offers, any important preparatory steps, and heuristics that the agent uses to decide what to do next, including the factors that have been selected and their combination into these functions.~~

3 Tests we performed

For the testing of our agent we actually employed an agile development technique. This allowed us to get fast results, and use them right away to implement changes.

The real testing was done in the domain described in paragraph 1.1. Our test case consisted of three parts for each run. We started the collaborative scenario (profiles 1, 2, and 3), the moderate scenario (profiles 4, 5, and 6), and the competitive scenario (profiles 7, 8, and 9), and compared the results to each previous run.
Values from the output we considered were:

- Whether the graph looked like we expected;
- Whether or not there was an agreement;
- How soon the agreement was made;
- How far the agreement product was from the nash line;
- Distance to pareto;
- Distance to Nash (which appeared to be different from the difference between the product and the nash line).

When this result was to our liking (i.e. when the solution was still pareto and the distance was closer to Nash), we used to previously implemented change, and went on to the next improvement. When it was not to our liking, or when there was an error, we tried to correct it, or we reviewed our thoughts about the usefulness of this feature.

TODO: Hippe resultaten die we tegengekomen zijn

4 Conclusions

We quite enjoyed discussing about the different actions our agent might take in different situations. As this was very close to real life, we could just think "what would I do?", and discuss about that. It all seemed very intuitive to us. It was also fun to invent and implement our own smart analyzing tools, like the OpponentUtilityModel. This way it really feels like we are making a smart person to analyze the opponent and do the negotiation for us.

In order to be able to support human negotiations in the real world, our agent would need

to understand some sort of input, and provide even more extensive output, to let the human know how he is reasoning. Apart from that, we think this agent is already quite versatile, as it can deal with various situations and always seems to find its way. The testing part may prove otherwise though, we are curious about that.

To really take over the human negotiation, it would be nice to have some form of computer vision as well, to be able to recognize facial expressions in the opponents, and use that to determine their mood and their feelings regarding an offer. Apart from this, it should probably also be able to handle multiple offers at the same time, as human negotiation is unfortunately less structured than agent negotiation.

Report requirements and checklist

10 A4 pages may be enough, 15 A4 pages maximum
The report should include:

- ~~the group number~~
- ~~an introduction to the assignment~~
- ~~a high-level description of the agent and its structure, including the main Java methods (mention these explicitly!) used in the negotiating agent that have been implemented in the source code~~
- ~~an explanation of the negotiation strategy, decision function for accepting offers, any important preparatory steps, and heuristics that the agent uses to decide what to do next, including the factors that have been selected and their combination into these functions~~
- a section documenting the tests you performed to improve the negotiation strength of your agent. You must include scores of various tests over multiple sessions that you performed while testing your agent. Describe how you set up the testing situation and how you used the results to modify your agent
- ~~a conclusion in which you summarize your experience as a team with regards to building the negotiating agent and discuss what extensions are required to use your agent in real-life negotiations to support (or even take over) negotiations performed by humans.~~

The final analysis report should involve an elaborate analysis of your agent's performance from different perspectives (e.g. individual utility gained, social welfare - the sum of utilities of all agents, optimality of the outcome, fairness etc.). NU NOG NIET NODIG