TDT4280
Project Part 2: Automated Negotiation

Magne Vikjord
magnevan@stud.ntnu.no

Pernille Wangsholm
pernilwa@stud.ntnu.no

29 March 2015

1 Design

We utilized a *BOA strategy*[1] in our design, where we decouple our agent into 3 individual components; bidding strategy, opponent model, and offering strategy. This approach allows multiple advantages over a unified design, the main one making it easier to swap out and test individual components. With the code structured like this, it is easier to see which approaches to a sub-problem are good and which are bad, as you can easily swap out any one component and re-test your performance.

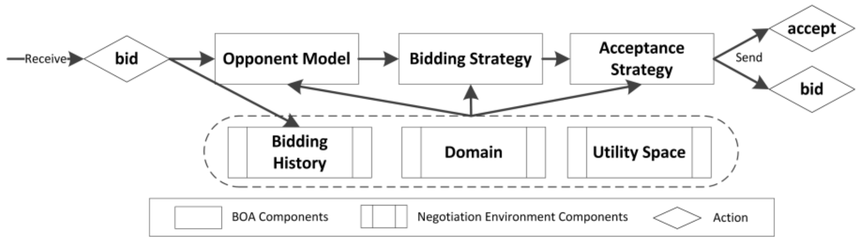
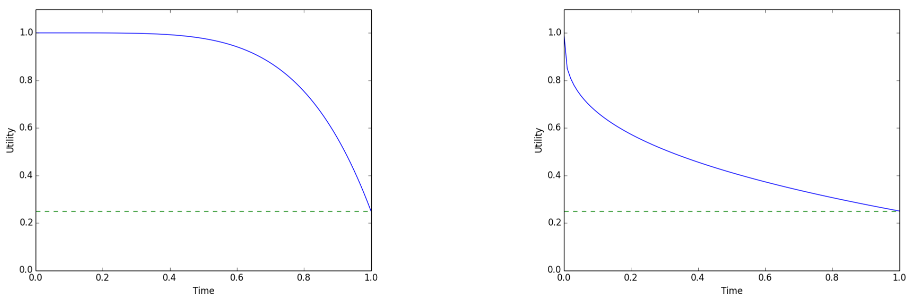


Figure 1: The flow of information in the BOA framework [1]

1.1 Bidding Strategy

The bidding strategy is the part of our agent that is responsible for generating an appropriate bid for each round based on available information. The information used by our agent is the progression in time or rounds towards the deadline of the negotiation, the models of our opponents (section 1.2), as well as a few fixed parameters defining the behavior of the strategy.

For each point in time, the bidding strategy will look to a target utility for it's next bid, moving from the ideal utility of 1.0 to the minimum acceptable utility (reserve utility) U_r. The rate at which this moves is defined by the positive real value S denoting the *concession shape.* If $S = 1$ the shape of the concession curve is linear. For $0 < S < 1$ the target utility is reduced very quickly at the start, similar to the default *conceder* strategy. For $1 < S < \infty$ the target utility is reduced slowly at



(a) $S = 5, U_r = 0.25$ (b) $S = 0.35, U_r = 0.25$

Figure 2: Concession shape for two different fixed parameters

first and then quickly towards the end, similar to the *boulware* strategy.

$$U_{target}(t) = (1 - U_r)(1 - t^S) + U_r$$

Once the agent has a target utility for this round it looks through a range of bids with a utility close to the target, and selects the most appropriate bid in this range. As our bids always get worse for us over time, it is to our benefit to at any point select the bid in the range that is the most beneficial to our opponents, as our opponent will be more likely to accept bids that are also good for them. We therefore pick the bid in the range where we maximise the *worst* utility for our opponent. The reasoning behind this is that a bid needs every opponent to approve of it before it goes through. We want to focus on the welfare of the least fortunate opponent, as they are the most likely to pass on the bid.

In the future, we would like to improve on this so that we look at the history of approved bids for each opponent. We would select the bid where the margin between the utility of this bid for the opponent, and the utility of the worst bid they have accepted is maximized. This would be far better in situations where some opponents are willing to approve of a bid with less utility than others.

In addition to these method, we have a third technique we utilize. If we imagine that the model of our opponents start in a neutral position (every weight is equal), and then gradually converge towards the true model, our initial bids will be based on a model that is incorrect. Without a model that somewhat accurately predicts our opponent, we might as well pick a random bid in the range. In fact picking a random bid might be better, as using a slow moving and not-yet correct model will result in making the same mistake repeatedly, as the agent might wrongly keep thinking certain properties are important to our opponent. If we pick a random bid we will at least try a different variation each time, and might accidentally get lucky and pick a good combination.

Because of this observation, our agent uses a technique similar to simulated annealing[2], where we at each point in time can either pick a random bid (in the range) or the (perceived) optimal bid, and the chance of picking the optimal bid increases over time.

1.2 Opponent Modelling

As mentioned previously (Section: 1.1) our strategy relies heavily on our opponent model. This is a method for learning and creating a model based on the information about our opponent's negotiation profile. The opponent model takes in parameters such as the opponent's set of possible bids, as well as a negotiation trace, and estimates the opponent's utility for this set of bids. The model weights each bid, giving the predicted utility a value between 0.0 and 1.0. This weight is based on an estimate of how important a particular issue is to our opponent. If the opponent suggest many different values for the issue, we assume it to be of lesser importance.

For each round of bidding the opponent model will be updated by the bidding strategy, and as time goes on it will create an increasingly better image of our opponent. While our model is still in it's starting phase, the possibility of a random bid being successful over an estimated one is higher, and will decrease as our model is built. This is in part why we choose to increase the likelihood of using the model instead of the random bid over time.

Given the opponent's bid and the estimated weight from the opponent model, the bidding strategy can determine a counter offer by generating a set of bids with a similar preference.

The opponent model aims to simulate reality by giving more options for the important issues, while giving us more variation in issues that are less important. However, there are some weaknesses to this method.

We observe firstly that if an agent starts accepting offers too soon, our model will not have the full learning benefit. Our agent would also benefit from learning of the utility tolerance for an approved bid, which is something it does not do yet.

Secondly we observe that our model will almost always underestimate the utility of an offer, but this estimation gap will decrease over time. This as the agents we have tested against (especially the boulware agent) has a tendency towards always picking the same combination of issues. This leads us to believe that our model understands which issues are very important, but will underestimate the value of the "next best" options. This is one of the reasons we choose to keep the option of a random bid open, even after some development of our model.

1.3 Acceptance Strategy

Roughly speaking, the objective of a good acceptance strategy is to accept an offer if the result of future negotiations will not lead to a better outcome for the agent. Naturally, the agent does not know the future, so we have to rely on other parameters to decide when this is true.

In this particular negotiation framework, no offer is accepted until all parties have agreed on it, which has enormous implications for what a good strategy is. In real life, if everyone is negotiating an issue such as how to arrange a party and there is a single holdout who refuses to accept, there are generally other options than to keep negotiating. However, in this framework the negotiations must continue. Because of this, it generally pays off to be stubborn, and *accept late*, as we can generally expect our opponents to be willing to accept worse offers over time, and their offers to become better from our perspective.

Another quirk of this negotiation framework is that if the group does not reach an agreement by the deadline, the utility for all parties is 0. Because of this, it is always optimal to accept any bid offered in the final round. Taken to it's logical conclusion, the negotiations here are meaningless, and the first person to offer in the final round will always get it's optimal offer accepted. There is however a few ways to counteract this; one way is to make the round timers fuzzy so that the agents do not know exactly when negotiations will end. A better way is to evaluate agents in a tournament system, where we do a number of test of each agent against each other, in all different orders and with different utility measures. In a system such as this, agents can willingly "crash" the outcome of a negotiation, and refuse to cooperate with a rogue agent who does not try to achieve an outcome which is genuinely agreeable for all parties, such that this kind of greed does not pay off.

This is similar to the repeated prisoners dilemma, where the optimal strategy with two agents against each other is to always defect; the inclusion of a tournament format makes the optimal strategy be closer to what yields the optimal social welfare.

Because of these observations we choose to create a very simple acceptance strategy. In each round, when choosing to accept a bid or offer we always generate a bid using our bidding strategy, and if our utility of the opponents offer is greater than our own bid, we accept the offer.

2 Implementation

Our code is split into 4 classes. There are three classes for each of the BOA components, and the main agent class which handles the communication between them. The main agent class extends `AbstractNegotiationParty` and contains logic to:

- Remember what the current offer is.
- Push the bids into the opponent models as they arrive.
- Ask the bidding strategy for what to offer, given the current time and the model of the opponents.
- Ask the acceptance strategy if a bid should be accepted or not.

The class for the bidding strategy contains three methods

`generateBid` This method generates a bid based on the current time and the current models of the opponents. The logic is elaborated upon in section 1.1.

`getInitialBid` Gets the optimal bid for this agent, and is only called once at the beginning.

`utilityGoal` Gives you a number between 0.0 and 1.0 telling the strategy what kind of utility it should aim for in this round of bidding. The details are described in section 1.1, however the actual implementation is a bit different because we wanted a `concessionShape` between 0.0 and 2.0 to give a smooth transition between (max-)conceder and boulware.

The class for the opponent model contains three methods.

`addApprovedBid` Takes a bid and adds it's data to the program using a basic frequenct heuristic.

`evaluateBid` Given a bid, how high is the utility for this model of the opponent.

`json` Gives a json representation of the current model. We used this output in conjunction with separate programs to further analyze the performance of our models.

The class for the acceptance strategy is extremely simple and contains only a single method `isBidAcceptable` where it evaluates whether the utility of our counter offer is higher than the proposed bid.

3 Tests

To better analyze the performance of our agent, we wrote some custom software to read and analyze the outputs of each run. We were especially interested in seeing the bids made by our agent isolated from the bids of the opponents. We were interested in seeing how the opponents utility of our bids changed over time, which bids they accepted and what utility our agent thought the bids had versus the actual value of the bid.

In the graph (figure: 3 b) where we isolate our bids from the opponents, the thick lines represent the real utility of our bids from our opponents. The transparent area of the same color as the lines represent the difference between our agent's perceived utility of the bids versus the real utility, and the blue squares tell us if the bid was accepted by the opponent.

3.1 Party domain with our agent, boulware and conceder

Parameter	Value
Rounds to consensus	124
Distance to Pareto	0.00
Distance to Nash	0.10
Social Welfare	2.39

Agent	Profile	Final utility
Our agent	party_1	0.84833
Boulware	party_2	0.85649
Conceder	party_3	0.69241

Table 1: Final result of a negotiation between our agent, a conceder and a boulware agent in the party domain.

For the purposes of keeping this report succinct we opted to not show the graph for every test we ran, and instead just show the result of the final configuration we ended up with. We have a few more test included with the delivery in the folder *tests*.

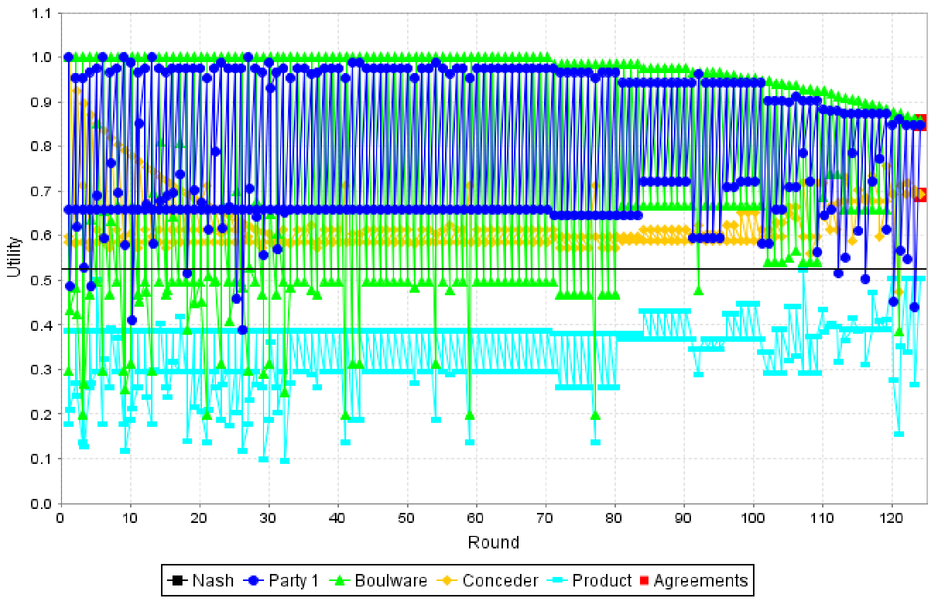
As seen in the tables above, this test was run with one instance of our agent, and one instance of boulware and conceder. The general lessons learned by performing this test was that a utility curve that leaned more towards the greedy (boulware) side of things tended to end up with a final higher utility, so we increased that until we had a result we were happy with.

Initially our opponent model was quite a bit off, but by tweaking it significantly we were able to end up with quite a good result, as you can clearly see by the area under the curve decreasing over time. Initially we had a model where the weight of each issue was scaled with the inverse of the variance within the issue. This turned out to not work very well, so we changed it to a system where we just increased the weight of each category by some variable `lrate` every time the opponent did not change their stance on an issue.

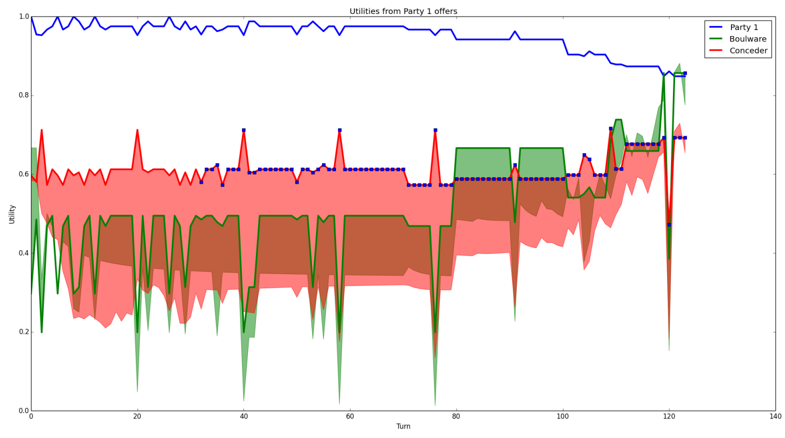
The weight of each item within an issue is set as:

$$\frac{\text{number of times this item has been chosen}}{\text{total number of bids made}}$$

We found that by initializing the numerator to some initial value we generally got a better learning curve. We suspect that this is because the agents we were testing against tended to choose the exact same bid every time, thus our model would undervalue the weight of all other items apart form the one chosen by the agent.



(a) Overview of the negotiation flow.



(b) The bids of our agent isolated from the rest.

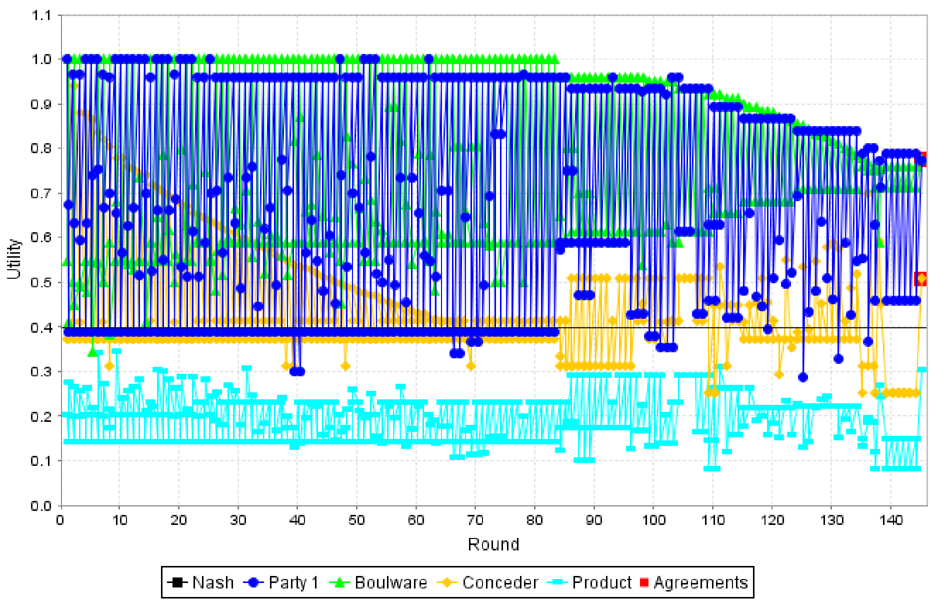
Figure 3: Progression of negotiation between our agent, a conceder and a boulware agent in the party domain.

3.2 University domain with our agent, boulware and conceder.

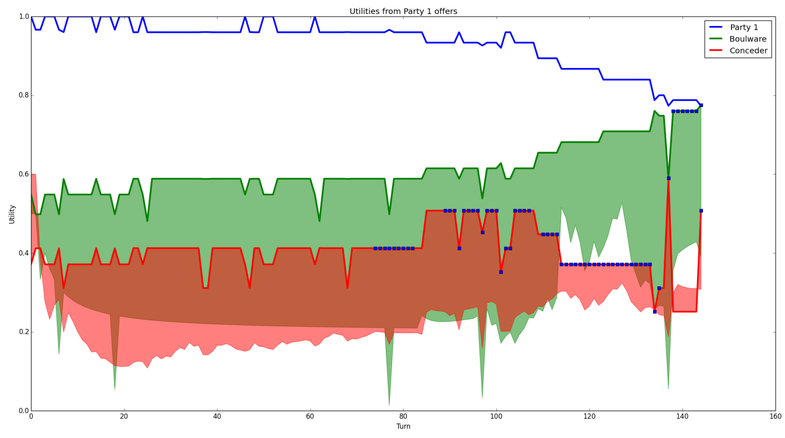
Parameter	Value
Rounds to consensus	145
Distance to Pareto	0.00
Distance to Nash	0.20
Social Welfare	2.06

Agent	Profile	Final utility
Our agent	party_1	0.77335
Boulware	party_2	0.77508
Conceder	party_3	0.50751

Table 2: Final result of a negotiation between our agent, a conceder and a boulware agent in the university domain.



(a) Overview of the negotiation flow.



(b) The bids of our agent isolated from the rest.

Figure 4: Progression of negotiation between our agent, a conceder and a boulware agent in the university domain.

In this test we could see that our agent was able to adapt quite well to a different domain, even though we had done little tuning while testing on this domain. We see the occasional spikes in the utility curve because of the simulated annealing algorithm, however this does not impact the overall performance in the later stages by much.

4 Analysis

From the result of our tests, the overall performance of our agent seem very positive. The final utility for our agent is quite high in the final configuration of both domains, thought we are slightly edged out by Boulware. In testing we found that by making the utility curve decrease slower, we could always win over these agents, but it generally caused the social welfare to go down. The problem with an approach like that is that while it might work very well when it is just put against these “dumb“ default agents, it would not work very well against a more sophisticated lineup of agents, who might have some sort of mechanism to punish very greedy agents. Because of this we chose to focus on the more *fair* approach where we tried to optimize for social welfare instead.

Another more reasonable way we could have increased our individual utility was to also model at which thresholds our opponent seemed willing to accept bids, and take that value into consideration when choosing which bids to make. In this configuration that would have let to us really exploit the conceder strategy, and almost only focusing on making the boulware agent happy. We did however not have time to do proper testing with that technique, and figured that testing this technique against boulware and conceder wouldn not be very meaningful anyway because of their simplistic nature. The result we would get from such testing would likely be overfitting the problem - we would get a result that would exploit that particular duo very well, but wouldn not be very meaningful when put against a wide variety of complex agents.

We can also see that the outcome of the tests yielded a Pareto optimal answer. This tells us that our learning model must be working quite well, because our agent tries to search for a Pareto optimal bid to make, and the fact that the final result is Pareto optimal tells us that it succeeded. We also see that the difference between the perceived utility and the actual utility tends to decrease over time, which is another good sign that the model is working, though not quite as quickly as we hoped it would. To some degree, this can also be explained by the simplicity of the agents we are facing. The conceder agent very quickly starts to accept every offer thrown at it, which causes it to have a very high signal to noise ratio and it would be quite hard for even the best model to tell what issues it actually values. The boulware agent has different problems, because it is so conservative with what bids it makes that it tends to select the exact same bids again and again, causing our model to think that those are the only items of value, and we are unable to tell what it’s “next best“ alternatives are.

5 Conclusion

In the end we were happy with the quality of agent we were able to build in such a limited amount of time. We were able to achieve a high utility, a high social welfare, and a pareto optimal final result. Working further on this agent, we would be able to swap out and test different BOA components and tweak the parameters of the agent, such as concession curve, model learning rate etc, using other machine learning techniques like genetic algorithms and perhaps really push the limits of what it is able to do.

In the future we can imagine that negotiating agents could be used to augment, or in some cases even replace humans in some domains. The main advantage of a such agents being that they are extremely fast, cheap to use and potentially, if the state of the art improves, significantly more precise than a human could ever be. The downside is that an agent like this can only work within it's own well-defined framework, while a human can be much more flexible. For instance, these agents work in a framework where they have no method of communicating their preferences other than by their bids and accepts. In a real life negotiation, the parties would be able to express their preferences more explicitly, they would be able to discuss individual issues separately and they might be able to introduce new terms all together. For instance, in the party-planning domain, one agent could propose that they have their prefered meal but they will in turn pay for it.

There's also the issue that for such an agent to work, you generally first have to manually create a utility heuristic for it to work on, and the agent can be no better than the heuristic you have created for it. This is problematic because it's quite difficult for people to accurately describe their preferences using just a series of weights.

References

[1] Tim Baarslag et al. "Decoupling Negotiating Agents to Explore the Space of Negotiation Strategies". In: *Novel Insights in Agent-based Complex Automated Negotiation*. Ed. by Ivan Marsá-Maestre et al. Vol. 535. Studies in Computational Intelligence. Springer, 2014, pp. 61–83. ISBN: 978-4-431-54757-0. DOI: 10.1007/978-4-431-54758-7_4. URL: http://dx.doi.org/10.1007/978-4-431-54758-7_4.

[2] N. E. Collins. "Simulated Annealing - an Annotated Bibliography". In: *Am. J. Math. Manage. Sci.* 8.3-4 (Jan. 1988), pp. 209–307. ISSN: 0196-6324. DOI: 10.1080/01966324.1988.10737242. URL: http://dx.doi.org/10.1080/01966324.1988.10737242.