Practical Assignment Group 5

Ajay Adhikari 4627199
Ioanna Birmpa 4632532
Lorenzo Gasparini 4609905
Lars van der Geest 4211669

IN4010 Artificial Intelligence Techniques

January 9, 2017

Contents

1 Introduction

This report contains an overview and justification, as well as a performance analysis for the design of an automated negotiation agent. This automated negotiation agent was made as part of the course Artificial Intelligence Techniques (IN4010).
The different chapters will focus on the various concepts and strategies used in the design of the agent. We will discuss the acceptance strategy, opponent modeling and bidding strategy of our agent and will subsequently focus on evaluating the performance of the implemented strategies. Furthermore we will explain the process of tweaking the different parameters involved to converge to an optimal negotiation agent.

2 Opponent Modeling

In order to make bids that are attractive to both our agent and the opponents, a good estimation of the opponents preferences and strategy is needed. In this chapter we will discuss the way in which we have modeled the opponents during negotiation. The chapter is divided into two parts. The first of two chapters will explain the way in which we handled modeling of issue weights and evaluation values using frequency analysis and the second chapter will discuss the way in which we find out how hardheaded/conceding our opponents are.

2.1 Frequency Analysis

The modeling of opponent issue weights and evaluation values is done using a frequency analysis of the incoming bids, as seen in [1] and [2]. This assumes that the opponents uses a concession-based strategy, meaning that they tend to change less the values of the issues that they care the most about, and they propose more often the issue values that have a higher evaluation value for them. The agent tracks the bid history, containing all the bids made by an opponent.

For the evaluation values, the agent searches the bid history for the amount of times every value in the issue domain was used. The agent then divides these by the maximum number of times a value was used in the same domain in order to have normalized evaluation values with the highest being 1.

$$\text{evaluation value} = \frac{\text{number of times this value used}}{\text{maximum number of times any value used}}$$

The agent then repeats this procedure for each of the issues involved.

In the case of modeling the weights of the opponent for the various issues, the same recorded bid history is used. The agent assumes the weight for every issue to be equal. It then checks the amount of times each issue has changed value. It then assigns a weight to every issue that is computed as follows:

$$w_i = \frac{1}{N} + \frac{S - \Delta_i - 1}{10}$$

where

$$\begin{aligned} w_i &= \text{issue weight for issue i} \\ N &= \text{amount of issues} \\ S &= \text{size of the bid history} \\ \Delta_i &= \text{amount of times the value of issue i changed} \end{aligned}$$

The computed weights are then normalized to add to add up to 1:

$$\tilde{w_i} = \frac{w_i}{\sum\limits_{i=1}^{N} w_i}$$

2.2 Hardheadedness

We decided to give a value to express how hardheaded (or conversely how conceding) the opponent agent is. Our agent tracks the amount of changes in the opponents bids of the last 10 rounds. The agent then computes the following value:

$$1 - \frac{\Delta}{NR}$$

where

$$\begin{aligned} \Delta &= \text{amount of times an issue value changed} \\ N &= \text{amount of issues} \\ R &= \text{amount of rounds considered} \end{aligned}$$

This value will be 1 in the case that the agent never changes his offer and 0 if all issues are changed in every round. Both of these cases will be very unlikely to happen, as we expect most agents to only change 1 or two issue values every round and thus the value for these agents will be somewhere in the range between 0.6 and 0.8. Agents with a value lower than that are more willing to adapt their bids to opponents and/or have a higher utility on a wider spectrum of bids. With this information, we can adapt our own strategy to get a better deal than with a general strategy that does not adapt to opponent strategies.

3 Acceptance Strategy

The first versions of the agent used an acceptance strategy based on the widely used acceptance conditions:

- $\mathrm{AC}_{\mathrm{const}}(\alpha)$: the opponent's bid is accepted when its utility is higher than a fixed given utility
- $\mathrm{AC}_{\mathrm{next}}$: the opponent's bid is accepted when its utility is higher than our upcoming bid
- $\mathrm{AC}_{\mathrm{time}}(T)$: the opponent's bid is accepted when a fixed amount of time has passed.

The strategy combined the conditions so that the bid was accepted only if

$$(\mathrm{AC}_{\mathrm{const}}(\alpha) \wedge \mathrm{AC}_{\mathrm{time}}(T)) \vee \mathrm{AC}_{\mathrm{next}}$$

meaning that the upcoming bid has a higher utility than our upcoming bid or time T has elapsed and the bid has a higher utility than a fixed value.

This approach worked well but it was clear than it was lacking flexibility. It's highly nonlinear meaning that there is a clear distinction in the acceptance behavior before and after time T has elapsed. This poorly represents what can be observed in rationally conducted negotiations between humans: as the time passes and the deadline approaches, each party gradually lowers his expectations in terms of utility, but there is no clear moment in which one decides to start accepting lower valued bids.

To better model the human behavior, we decided to replace the

$$(\mathrm{AC}_{\mathrm{const}}(\alpha) \wedge \mathrm{AC}_{\mathrm{time}}(T))$$

condition with the following:

- $\mathrm{AC}_{\mathrm{min}}(\alpha(t))$: the opponent's bid utility is higher than $\alpha(t)$, where

  $$\alpha(t) = \frac{\log_{10}(t)}{c(t)} + 0.9 \tag{1}$$

  with $t \in [0,1]$ representing the time left and $c(t)$ being the conceding factor, which is time-dependent. The utility has also to be higher than our reservation value, which is set to 0.4. This value is reached only at the very end of the negotiation and is used as a guard towards the acceptance of very low valued bids.

The conceding factor is defined as follows:

$$c(t) = \begin{cases} 13, & \text{if } 0.04 < t \leq 1 \\ 10, & \text{if } 0 < t \leq 0.04 \wedge h \leq 0.6 \\ 7, & \text{if } 0 < t \leq 0.04 \wedge h > 0.6 \end{cases} \tag{2}$$

where h is the maximum hard-headedness of the opponents.

In figure 1 the minimum acceptable utility is represented over time. At $t = 1$ it is 0.9. In the initial phase represented by the blue line the conceding factor is 13. When 96% of the time has elapsed, there is a sudden change to

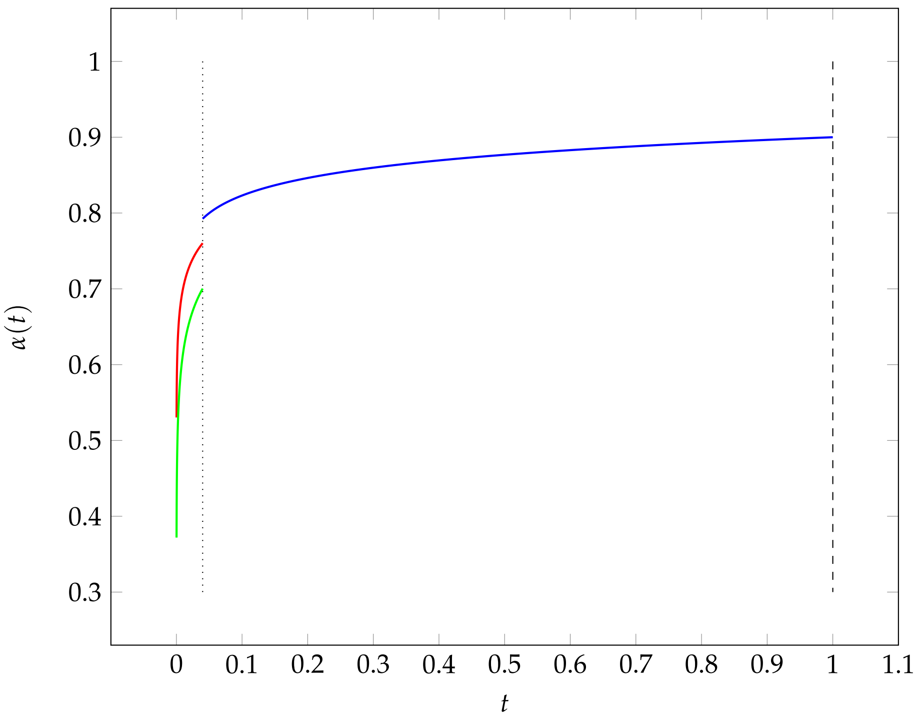

Figure 1: Decrease of $\alpha(t)$ as the deadline approaches

a conceding factor of 10 (red line) or 7 (green line) depending on the hardheadedness of the opponents.

The minimum acceptable utility will reach the reservation value of 0.4 when 0.04% of the time is left.

We decided to keep the AC_{next} condition since proposing a bid with a lower utility for the agent than the bid that was declined does not correspond to the behavior of a rationale agent.

To sum up, our acceptance condition is

$$\text{AC}_{\text{min}}(\alpha(t)) \vee \text{AC}_{\text{next}}$$

4 Bidding Strategy

The overall strategy of our agent is as follows: the agent does not concede at the beginning, starts conceding slowly at 20 percent time and concedes faster at the end depending on the hardheadedness of the opponents.

When 20 percent of the negotiation time has passed, the agent starts to concede and attempts to find bids which are more acceptable for the opponents.

Each turn the agent generates 100 bids randomly which have a utility higher than $\alpha(t)$ (1) for us. $\alpha(t)$ is the minimum acceptable utility depending on the time and the hardheadedness of the opponents. For each bid, it computes the Nash product using the approximate issue values and evaluation values learned from opponent modeling.

A sorted list is created named *bestGeneratedBids* as attribute with maximum size 100, which saves the so far found bids with the best Nash products. The list is sorted with the Nash product of each bid as key. If *bestGeneratedBids* is not full, the best randomly generated bid B is added. And this bid is offered to the opponents. On the other hand, if the list is full, B is added to the list, if B has a Nash product greater than the bid with the smallest Nash product in the list. And then one of the top 5 bids in *bestGeneratedBids* is offered to the opponent. We offer one of the top 5 bids instead of the best bid, because the model of the opponents is based on approximations, meaning that the first bid in the sorted list might not actually be the optimal bid for every other agent.

5 Code Structure

5.1 Opponent

This class is responsible for creating an model of the preference profile of an opponent and their hardheadedness. The following are public functions used by our agents.

- `void addBid(Bid bid)`: This function adds the given bid to the bid history of the opponent. It also triggers the recomputation of the weights of the issues and the evaluation values.
- `double getUtility(Bid bid)`: This function returns the utility of the opponent for the given bid using frequency analysis on the bidhistory.
- `double hardHeaded(int rounds)`: It returns a number between 0 and 1 taking only the last given rounds into considaration. This value will be one in the case that the agent never changes his offer and 0 if all issues are changed in every round.

5.2 Group5

This is the class of our agent. The following are the important data attributes of this class.

- `HashMap<AgentID, Opponent> opponentsMap`: This map saves the models of the opponents.

- `double TIME_OFFERING_MAX_UTILITY_BID`: This variable determines the range of time when our agent only offers the maximum bid at the beginning.
- `double RESERVATION_VALUE`: Our agent does not accept an offer under this utility.
- `List<BidDetails> bestGeneratedBids`: The best bids with the highest Nash products are saved here.

The following are the most import functions excluding the functions we have to override.

- `double getMinAcceptableUtility()`: This function returns the minimum acceptable utility considering the remaining time and the hardheadedness of the opponents.
- `double getConcedingFactor()`: It returns the conceding factor taking the time and the hardheadedness of the opponents into account.
- `boolean isAcceptable(Bid proposedBid)`: This function determines whether the last proposed bid from a opponent is acceptable. The argument is the bid we are planning to propose.
- `Bid generateAcceptableRandomBid(double minimum_acceptable_utility)`: It generates a random bid between the given utility and the maximum utility. Function getBidNearUtility from SortedOutcomeSpace is used to get a bid in that range. This is done to avoid generating a random bid and checking whether it is in the range.
- `Bid generateBid()`: This function returns a bid taking the remaining time, the preference and the hardheadedness of the opponents into account.

6 Experiments

6.1 Parameters optimization

Concerning the conceding factor (2), we tried different values for it. We tried a fixed value of 20 and 7 but as the results in table 1 show, the performance for the agent in terms of average utility was worse than our dynamic strategy.

We also tried removing the initial time in which we keep offering the bid with the maximum utility for us, and again this led to worse performance of the agent.

Scenario	Agent	Avg util.	Avg util. diff.	Avg run time
$c(t) = 20$	**Buyog**	0.8048	0.2062	18.03
	Group5	0.7595		
	Mercury	0.7693		
$c(t) = 7$	**Buyog**	0.7773	0.1888	18.35
	Group5	0.7411		
	Mercury	0.7407		
No ”max utility” time	**Buyog**	0.7975	0.2006	17.98
	Group5	0.7965		
	Mercury	0.7686		

Table 1: Parameter testing

6.2 Results

We ran the final version of our agent in multiple tournaments, against the agents *Atlas*, *Buyog* and *Mercury*.

We ran a tournament with 24 sessions for each possible combination of opponents, one time with the preference profiles from 1 to 4 and another time with the ones from 5 to 8 from the party domain. The negotiation time used was 20 seconds.

Against Atlas and Mercury, our agent is always getting the best average utility.

The table 2 shows the results of the tournaments. The second to last column shows the average difference between the utility of the winner and the utility of the agent that came last. The last column shows the average running time of the negotiations.

It can be detected that the average utility difference does not exceed the value 0.3, showing that the agreement reached is relatively fair to all three

Preference profiles	Agent	Average utility	Avg util. diff.	Avg run time
1-4	Atlas	0.6828	0.2995	11.72
	Group5	**0.7747**		
	Mercury	0.6627		
5-8	Atlas	0.7891	0.2385	9.268
	Group5	**0.8435**		
	Mercury	0.6737		
1-4	Atlas	0.6672	0.1787	18.93
	Buyog	**0.7667**		
	Group5	0.7476		
5-8	Atlas	0.7458	0.2092	18.47
	Buyog	**0.8412**		
	Group5	0.8257		
1-4	**Buyog**	**0.8007**	0.1909	18.34
	Group5	0.7889		
	Mercury	0.7749		
5-8	**Buyog**	**0.8263**	0.2224	17.99
	Group5	0.8254		
	Mercury	0.7575		

Table 2: Results

agents.

Based on the experiments, our agent's utility is higher (and thus wins over the other two agents) when an agreement is reached early during the negotiation. That occurs because the bids that are calculated are always close to the Nash optimal ones. Due to the high level of accuracy of the opponent modeling, the best bids that are saved at the sorted list, are considered fair and optimal for both the opponent agents, so they accept them. This is considered one of the strongest points of our agent.

As far as the weak points are concerned, when the agent is negotiating with hardheaded agents, the results are not always favorable. In our case, the agent *ANAC2015-13-Buyog*, which is considered hardheaded based on our opponent modeling strategy, manages to obtain the maximum average utility at most of the experiments conducted. This situation occurs due to the fact that this agent does not concede until the end of the negotiating time, by which point our own utility value has decreased significantly.

7 Conclusion

Taking everything into consideration, our agent performs relatively well during the multi-party tournament against *Atlas*, *Buyog* and *Mercury*. It adapts an effective opponent modeling strategy based on frequency analysis, in order to realise its opponents preference profiles and offer optimal and fair bids.

In order to support negotiations performed by humans, the time constraint should be less strict. In human-based negotiations, time plays an important role, but it does not necessarily change the behavior of negotiating person. The heavily conceding behavior in the last rounds spotted during our experiments, is rarely observed in real life negotiations, since the deadlines are then considered as soft deadlines.

During the implementation of this assignment, we figured out the difficulties arising from the negotiation process, especially when the opponents' preference profiles are not shared. The examination of the diverse opponent modeling strategies as well as the bidding techniques, brings us one step closer to achieving real-life negotiation.

References

[1] Robert M. Coehoorn and Nicholas R. Jennings. "Learning on Opponent's Preferences to Make Effective Multi-issue Negotiation Trade-offs". In: *Proceedings of the 6th International Conference on Electronic Commerce*. ICEC '04. Delft, The Netherlands: ACM, 2004, pp. 59–68. ISBN: 1-58113-930-6. DOI: 10.1145/1052220.1052229. URL: http://doi.acm.org/10.1145/1052220.1052229.

[2] Koen Hindriks and Dmytro Tykhonov. "Opponent Modelling in Automated Multi-issue Negotiation Using Bayesian Learning". In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*. AAMAS '08. Estoril, Portugal: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 331–338. ISBN: 978-0-9817381-0-9. URL: http://dl.acm.org/citation.cfm?id=1402383.1402433.