Multi-Agent Systems Assignment

# Designing a Negotiation Agent

*Group 13*
Xin Lan *6690165*
Shaoya Ren *6850413*
Siyang Qian *6349072*
Jiayuan Hu *6876587*

Utrecht University

## 1 Introduction

Negotiation exists in all aspects of our lives. It involves two or more entities who compromise to reach an agreement through the interaction and communication to gain profits, while optimizing their own utilities as much as possible[3]. Everyone of us has experienced negotiation. However, limited by the personal negotiation skills and influenced by emotions, the negotiation outcomes are not always ideal. Besides, in some situations, we are unable to negotiate with other people or parties directly. On one hand, negotiation is necessary, on the other hand, human negotiators are easily influenced by many factors. Therefore, more and more autonomous negotiation agents constructed through programming are required to represent us in the negotiations with others. Researchers from different fields have focused on autonomous negotiation agents since the 1990s.

### 1.1 Related work

In 1999, Faratin et al.[6] constructed a distributed negotiation model which coordinates both agent interactions and individual agent decisions. In 2001, Jennings et al.[7] analyzed automated negotiation from prospects, methods and challenges and proposed a generic framework for classifying and viewing automated negotiations has been developed. In 2008 Lin et al.[8] presented an automated agent design for bilateral negotiation. Since 2010, the The Automated Negotiating Agents Competition has been held once a year. Many excellent negotiation agents were created through this competition, including the agents we used as references for this assignment, such as TheNegotiator[5], HardHeaded[10], and IAMhaggler[4]. In 2014,T. Baarslag et al.[2] introduced BOA structure to construct negotiation agent, which is used in our design.

### 1.2 Problem description

The main goal of this assignment is to design and implement a negotiating software agent. In the assignment, the party domain serves as an example. Two agent should be able to make joint decisions about what a party will look like. Note that negotiation is not a competition, which means, in a successful negotiation, there is not a winner who takes all and a loser who gets nothing. There are two tasks we do need to consider in this assignment. The first task involves creating a preference profile that captures our own preferences with regards to the party domain. The result will be a formal preference function that maps each possible outcome of a negotiation to a utility number in the range of 0 to 1. The second task involves thinking about a strategy used to perform the negotiation itself. In negotiation we need three strategies: an offering strategy (what to bid when), an acceptance strategy (when to accept or reject offers, and when to stop negotiating), and an opponent model (how will your opponent like this offer).

### 1.3 Our initial idea

**Performance measurement** Firstly, we expect our agent to be as greedy as possible while still avoid not reaching an agreement. Secondly, we think that it is important to have a low computational complexity. Finally, time of agreement is also considered as a measurement

**Environment** Environment is all the factors an agent will face to during the negotiation. For our negotiation agent, environment consists of three components: domain, bidding history and utility space. Domain is a set of issues containing information of all the possible bids. Every agent has its preference over each issue known as agents' preference profiles. These profiles form an utility space. All the bids should be in this utility space. Domain information is accessible to every agent while utility space is not, because preference profile is private information, which means that the agent only has access to its own utility value for each bid. Bids offered by agents before will be partially recorded known as bidding history. An agent can speculate opponent's preference profile through opponent modelling.

**Actuators** Our negotiation agent only has two actions: Accept if the bid satisfies the acceptance condition and Offer bid if not.

**Sensors** Sensors get information from the environment. In our case, in each negotiation round, the agent gets the bid from the opponent and then take actions based on this information. Both agents will fully observe the bid history from both sides. However, the preference profile will keep unknown to opposite agent. Still an agent can use bid history to predict opponent's profile.

## 2 Agent structure and negotiation strategy

### 2.1 Bidding strategy

Our bidding strategy is mostly based on the idea from *NegoEngine*[9], but with adapted settings and parameters. The strategy consists of two tactics, one is time dependent tactic and the other is behaviour dependent. To combine these two tactics and obtain our target utility, we multiply the utility values from these two tactics with different weights and sum them up.

In this paper, R. Ros and C. Sierra calculate the utility value for each issue separately. However, in our situation, especially when we only consider discrete issues, to find such a target for every issue will lead to a huge deviation for the overall utility. For example,

if we have total 100 issues and 2 values for each issue, and we want to find a bid whose utility is near 0.6, then we will choose the value that gives higher utility for every issue if calculate them *separately*. Obviously this method gives us a bid with utility 1. But apparently we can use some combinations of those issues, i.e. changing some values of the issues to the lower one, to receive a bid of utility 0.6. Therefore, we will calculate the utility value that we want to offer in the next round as a bid instead of for each issue. And we will expand the range if none bid can be found near this target.

**Code explanation** Firstly, the init() method of Group13_BS class will simply initializes our offering strategy class. Then it will get outcome space, and attempt to get the best bid and worst bid and their utility value within this domain. Since we do not have parameter for user to change, the parameters shall remain empty.
The determineOpeningBid() method, which determines the first bid offered by our agent, returns the best bid in the domain. One of our performance measures is getting higher utility, so we do not expect to reach an agreement with opponent quickly.
After the first round, the determineNextBid() method decides which bid to be offered to opponent. It generates the best bid in the domain. If the last bid from opponent is empty, it returns the best bid. Otherwise, it calls getMyBid(Range range) method with parameters lower bound, minimum utility value, and upper bound, maximum utility value to get the bid we are going to offer.

### 2.1.1 Time-dependent tactic

For time dependent tactic, the main idea is that in order to reach an agreement before time ends, the agent will concede as the time passing. Let $x^t_{A\to B}$ denote the utility value of the bid in this tactic we, agent $A$, are going to offer to opponent, agent $B$, at time $t$:

$$x^t_{A\to B} = min + (1-\alpha(t))(max - min) \qquad (1)$$

The value $min$ and $max$ are the lower bound and upper bound of our utility value in a certain negotiation domain. $\alpha$ is a function that depends on current time $t$ and a parameter $\beta$:

$$\alpha(t) = (\frac{t}{t_{max}})^{\frac{1}{\beta}} \qquad (2)$$

The parameter $\beta$ determines how our agent concedes. If $\beta > 1$, then $\alpha$ will grow faster. So we can see from the first equation that our utility value will concede rapidly from the beginning as time passing. If $\beta < 1$, our utility value will concede very slowly at beginning and only concede rapidly when time is about to end. If we set $\beta$ as 1, then the utility value decreases linearly. In our design, we set $\beta$ as **0.01** because we don't want our agent to concede rapidly at most of the time since it will be forced to accept bid from opponent and perform poorly if the opponent takes aggressive strategy. Also, because the agentHardheaded [10] outperforms other agents in the ANAC competition, it is reasonably for us to also play hard till the very end.

**Code explanation** The method getTimeDependentTargetUtility(Range range) is called to compute the utility value with time dependent tactic. It is a simple implementation of equations (1) and (2).

### 2.1.2 Behaviour-dependent tactic

For behaviour dependent tactic, the main idea is to reproduce behaviour of opponent and try to follow its utility curve in order to reach an agreement. An agent tends to make an agreement of higher utility with itself or agent with similar behaviour.
Let $x^i_{A\to B}$ denote the utility value of the bid in this tactic we, agent $A$, are going to offer to opponent, agent $B$, at round $i$:

$$x^i_{A\to B} = \begin{cases} min, & P \le min \\ max, & P \ge max \\ P, & otherwise \end{cases} \qquad (3)$$

This parameter $P$ reflects the behaviour of the opponent. We use $Relative\ Tit-For-Tat$ method[9] to perform the imitation. We made a bit change to the formula because the formula given in the paper contradicts with how they describes it. They say that this tactic is designed to imitate the opponent's behaviour, but they use $\frac{x^a_j[t_{n-2\delta}]}{x^a_j[t_{n-2\delta+2}]}$ which wrongly calculate the ratio. The formula for $P$ is given below, where $x^i_{B\to A}$ stands for the utility value of the bid offered by the opponent at round $i$.

$$P = \frac{x^{i-1}_{B\to A}}{x^{i-3}_{B\to A}} x^{i-2}_{A\to B} \qquad (4)$$

We calculate the ratio of utility value that the opponent changes 3 steps ago and multiply with the utility value of the bid before last bid of our own. This way, we generate the target utility value of our next bid using this tactic by reproducing opponent's behaviour 3 steps ago. Intuitively, this means that if the opponent increases its utility, we also increase ours. If the opponent concedes, we also concede with almost same speed. $P$ value is returned as our target utility value for this tactic if only it is in our utility range.

**Code explanation** In getBehaviourDependentTargetUtility(Range range) method, firstly we set the $P$

value as maximum possible utility value in the domain. If we do not have a updated opponent model, which means we are not able to imitate opponent behaviour, this method will simply call getTimeDependentTargetUtility(Range range) method and return its return value. In another word, if the opponent model is not providing information, we will entirely use time dependent tactic. If opponent has not offered more than 2 bids yet, current $P$ value will be returned because it is not possible to calculate the variation yet. Then the $P$ value will be calculated by the formula in this section. Finally, the $P$ value bounded by minimum and maximum utility is returned.

#### 2.1.3 Combination strategy

After calculating utility for both tactics, we combine these two return values in a linear way by giving them different weights. Let $u_{time}$ denotes time dependent utility and $u_{behaviour}$ denotes behaviour dependent utility. After doing some experiments with different weight, we find that time-dependent utility barely decrease at most of rounds since we set $\beta$ considerable small, so we set weight for time dependent part relatively small so that the utility can still change according to the opponent behaviour. We use the fixed weight as follows:

$$u_{target} = 0.2 \times u_{time} + 0.8 \times u_{behaviour} \qquad (5)$$

By using this bidding strategy, we take both time and opponent behaviour into consideration. The behaviour dependent tactic part keeps track of opponent utility change so that we know when we can concede and when to play hard. The time dependent tactic part keeps track of the time left and concede a lot when there is few time left so that we can guarantee reaching an agreement before the secession ends.
After getting our target utility value by using this strategy, among all the bids that are around our target utility value, in our case $[u_{target}-0.01,\ u_{target}+0.01]$, we try to offer the bid that has the highest utility value for opponent. Meanwhile if we can not find such bid around our target utility value that can also give opponent the utility value higher than 0.5, then we will expand this range by decreasing lower bound by 0.01 and increasing upper bound by 0.03. This is a very effective way to shorten the negotiation time and reach an agreement early because we try to make our bids closer to Pareto efficiency, not just consider our own utility.

**Code explanation** In getMyBid(Range range) method, after calculate the value for $\alpha$ and time dependent utility and behaviour dependent utility using strategies that is discussed above, we can easily have the final target utility value. Then we initialize a Range object called targetRange, and a PriorityQueue called bidCandidate, which is strictly sorted by their utility using comparator BidDetailsStrictSorterUtility(). We will keep searching bids that is within this range and gives opponent fine utility as well, and keep expand this range, till we find such bids and add them into bidCandidate. After this search, the head of this priority queue, namely the bid with highest utility within the range, is polled from the queue and is returned as our counter offer.

### 2.2 Opponent model

The opponent model that we applied in our agent is based on the one in HardHeaded agent[10]. It is a kind of frequency model, which means it aims to build a model of the opponents' preferences assuming linear additive utility functions and steady concession towards lower utilities. For that, the frequency model uses the frequency of negotiation issue values as an indicator of both negotiation issue and value weights. The model was implemented with the following assumptions: (1) the opponent steadily restricts the offers proposed to a possibly moving and decreasing utility range; (2) the opponent prefers to explore the negotiation space rather than repeating the same offer(s) multiple times; (3) Opponents tend to concede less on their favorite issues, keeping them remain unchanged.
Briefly, the model in HardHeaded works as follows. To estimate the weight of an issue value (e.g. Chips-and-Nuts, Finger-food, Handmade food, Catering for "food"), the frequency model computes how often each issue value appears in the opponent's bids. The weight of the issue value is then normalized by the most repeated issue value. For example, assume that Chips-and-Nuts, Finger-food, Handmade food and Catering appear 40, 30, 20, 10 times respectively. In that case, the estimated issue weights will be V(Chips-and-Nuts) = 40/40, V(Finger-food) = 30/40, V(Handmade food) = 20/40 and V(Catering) = 10/40. The frequency model analyzes how often the value of an issue changes. At the beginning, the assumption is that each value has the same importance. For example, in the "food" issue there are 4 values, the weight of each issue is set as 0.25. For each successive pair of offers made by the opponent, if the value of an issue did not change, then the model increases the weight of that issue.
It is true that in the first negotiation rounds most agents do not tend to vary the value for those issues that are the most important to them. The reason for this is because most agents start by demanding the best offers for themselves, and these offers entail very little changes in the most important issues. However, as the negotiation process going, agents may concede. At some point, it is possible to reach one's own aspirations by varying the values for the most impor-

tant issues and maximizing less important ones. In fact, this behavior can be witnessed in many state of the art agents who trade-off issues to achieve one's own aspirations. Therefore, the assumption that opponents tend to concede less on the most preferred issues may hold for hardheaded agents or agents that do not steadily concede, but it may result bad estimations in other scenarios.

There are two main goals that an opponent model should estimate in a linear additive function scenario: a vector of weights $\hat{\mathcal{W}} = (\hat{w_1}, ..., \hat{w_n})$, which represents the estimation of the importance given by the opponent to the different, and an estimation for every possible valuation function $\hat{V_i}(issue)$. Our model does both estimations.

The classic frequency approach considers that negotiation issues that remain the same between pairs of offers are normally those that are the most relevant. Despite the fact that this may be true in the beginning rounds of the negotiation, as the negotiation proceeds opponents may decide to concede on the most preferred issues and achieve its aspirations with less important issues. A classic frequency approach may produce a wrong model by this type of behavior, which is applied usually in many state of the art agents. As a countermeasure to this behavior, we introduce an issue weight update rule whose effect decays over time(normalized). This update rule will be used to update weights whose value did not change during the negotiation process.

$$New\_\hat{w_i} = Old\_\hat{w_i} \times (1 - t^{\beta}) \tag{6}$$

**Code explanation** Method init will initialize the opponent model by setting the valueAddition (the value to be added to weights of unchanged issues before normalization) and all value weights to one (they are normalized when calculating the utility).

Method updateModel(Bid opponentBid, double time) is the main part of this opponent model. It gives each issue a same weight at first, then repeatedly updates weights according to frequency that issues appear in bid history. As for each issue value that has been offered last time, a constant value(1 in our agent) is added to its corresponding ValueDiscrete.

Method getBidEvaluation(Bid bid) takes a bid as an input, using opponent model to estimate the opponent utility for the bid.

### 2.3 Acceptance Strategy

The acceptance strategy of our agent is mainly adapted from acceptance strategy of TheNegotiator[5], combined with some ideas of HardHeaded[10] and IAMhaggler2011[4]. Generally speaking, the acceptance strategy of our agent takes both negotiation time and opponent's performance into consideration. The fixed rounds and fixed real time set in one negotiation would be transformed into standardized time, ranging from 0 to 1. Referring to TheNegotiator, the whole negotiation process is divided into three phases. The difference is that we adjust the time distribution for each phase. The first phase takes 0.5 standardized time. The second phase takes 0.45 standardized time and the third phase takes 0.05 standardized time. Besides, we take advantage of opponent performance to predict the type of the current opponent. Adar[1] divides the opponent model into three categories, namely, greedy, similar to own agent and tend to cooperate. Here, we indicate two opponent types, greedy and not greedy, as our agent is designed as greedy on the whole. This design aims to help our agent get as higher utility as possible in a tournament. In all phases, we use the following strategy adapted from $AC_{const}(\alpha)$[11].

$$AC_{ouragent} \overset{def}{\iff} \alpha U_A(x^t_{B \to A}) \geq \beta \tag{7}$$

$\alpha$ denotes the accept multiplier and $\beta$ represents the acceptable utility of our agent in an negotiation. $\alpha$ and $\beta$ are different corresponding to different time phases and opponent types. From phase 1 to phase 3, $\alpha$ increases from 0.9 to 1.1. $\beta$ is determined by the utility space and reduces with time. The utility space includes all utilities our agent could gain from all possible bids generated by our own. The reduction speed of $\beta$ is different when our agent face with different types of opponent. And in the last 3 moves, the value of $\beta$ is the minimum utility our agent could gain from the opponent's offers in the whole negotiation history. In phase 3, the time used in each round is stored to calculate the left moves. The details of our acceptance strategy would be explained in the following paragraphs.

**Phase 1** In phase 1, the main aim of our agent is to predict and determine the type of the opponents. Therefore, the acceptance strategy in this phase is quite strict. $\alpha$ is set as 0.9, and $\beta$ is the maximum utility our agent could gain from the utility space, which makes the agreement is difficult to reach. Meanwhile, the opponent type is decided through evaluating the utilities opponents could gain. The default opponent type is greedy, indicated by number 2. In each round, the opponent utility is predicted and stored, taking advantage of the opponent model. The average of last 15 opponent utilities is calculated in each round. If the average utility is lower than 0.7, then the opponent is regarded as not greedy, which is indicated by number 1. However, if the average utility is higher than 0.7, it is still too early to regard the opponent as greedy, because it is possible that the opponent would concede quickly in the following negotiation process. Therefore, for these 15 values, we take the average

value every five, resulting 3 values. If the 3 values satisfy the following function, then we regard the opponent as not greedy.

$$utility_{current} < 0.9 \times utility_{previous} \quad (8)$$

**Phase 2** In phase 2, $\alpha$ is set as 1 and $\beta$ varies with time. The reduction speed of $\beta$ is different corresponding to different opponent types. When our agent is faced with greedy opponent, the reduction speed is faster than when it is faced with not greedy opponent. $\beta$ is calculated through following functions. In the functions, $t_c$ denotes the current time and $t_{pi}$ represents the time of $i$th phase. $max$ and $min$ represent the maximum utility and minimum utility of the utility space. Equation (9) is used when opponent is greedy and equation (10) is used when opponent is not greedy.

$$\beta = max - ((t_c - t_{p1})/t_{p2}) \times (1/8(max - min))) \quad (9)$$

$$\beta = max - ((t_c - t_{p1})/t_{p2}) \times (3/8(max - min))) \quad (10)$$

**Phase 3** In phase 3, time is rather limited for our agent to cooperate with others. To make sure our agent reach as much as agreements as possible, the acceptance strategy is relatively loose. The value of $\alpha$ is 1.1 and $\beta$ is calculated through the following function.

$$\beta = max - ((t_c - t_{p1} - t_{p2})/t_{p3}) \times (5/8(max - min))) \quad (11)$$

Besides, when there are only 3 rounds left, our agent would accept the opponent's offer as long as the utility our agent gained through this offer is higher than the lowest utility our agent could gain through the opponent's offer in the whole negotiation history. Because if we can not reach an agreement, there will not be any utility we receive at all, which is far worse than a lower utility. The acceptance strategy in this period is defined as following.

$$1.1 \times x^t_{B \to A} \geq min_{previous} \quad (12)$$

The method to calculate the left moves adopted the ideas of the method used in TheNegotiator. The number of left moves is calculated with the average time used in last three moves.

$$Left\_moves = (1 - (t_{current}))/t_{average} \quad (13)$$

**Code explanation** Actions determineAcceptability() is the main function of acceptance strategy, determining whether our agent accepts the opponent's offer. Parameters used in main function are calculated by calling predefined functions. calculateCurrentPhase(time) is used to determine the current phase of the negotiation process. In this function, time spent on last 3 moves that used in last phase is also stored. calculateThreshold(time) takes advantage of calculateMaxToMinThresholds() to calculate the threshold for different phases and different opponents. predictOpponentType(time) is called to predict the type of opponent. It makes use of getBidEvaluation() from opponentModel to acquire the opponent's utilities in the negotiation history. calculateMovesLeft() is used in the third phase in order to determine the number of left moves. calculateMovesLeft(), calculateThreshold(time), calculateMaxToMinThresholds() and calculateCurrentPhase(time) are adapted from the ideas of TheNegotiator,HardHeaded and IAMhaggler2011. predictOpponentType(time) is programmed by ourselves. We also called the functions from negotiationSession,for example, getUtilitySpace(), getOpponentBidHistory(), getTime() and so on.

### 2.4 Preference uncertainty

We also make our agent able to handle uncertainty situation. As for the acceptance strategy, if our last and opponent last bid do not exist, agent will immediately reject. And only if the negotiation time has reached the last 10% of it can our agent have to possibility to accept. This idea is that, we intend to learn more information about our and opponent utility space. At the last 10% time, if the bid from opponent is better than 90% of the bids that we can rank, we will accept this offer. Meanwhile, if the bid from opponent ranks higher than the bid of our last offer, we will also accept this offer. As for the offering strategy, we will keep offer the bid with higher ranking from our bid ranking space.

## 3 Agent performance improvement

In this section, we will discuss how did we improve our agent performance and what are the strong and weak points of it.

### 3.1 Testing situation set up

During the tests aiming to improve our agent, we think it is better to concentrate on a specific domain at the beginning, test our agent generically after its performance is satisfying on that domain.
We choose the party domain which contains 6 discrete issues with 3 to 6 values respectively. The protocol we use is Stacked Alternating Offers Protocol. The deadline will be set to different number, i.e. 60, 120, or 180 rounds, or 5, 10, or 15 seconds, in some part of the test in order to see whether our agent can handle long-term negotiation. However, it will be set as 60 rounds at the beginning.
As for the opponent, we will choose BoulwareNegotiationPatry and ConcederNegotiationParty at the be-

ginning. Then the agents from ANAC 2011, for example AgentK2, HardHeaded, BRAMAgent, etc. will also be considered as potential opponent. At last, all agents above, including our agent itself, will be tested in tournaments to test their ability.

### 3.2 Improvement progress

**Bidding strategy** For our bidding strategy, there are three important parameters, $\beta$, weight for time dependent utility $weight_{time}$, and weight for behaviour dependent utility $weight_{behaviour}$. No doubt the two weights should add up to 1:

$$weight_{time} + weight_{behaviour} = 1 \qquad (14)$$

Firstly, the initial value of $\beta$ is 0.9, which is less than 1, because we are greedy and intend to receive a high utility. The initial value of $weight_{time}$ and $weight_{behaviour}$ is both 0.5, because we are not sure which tactic should play a vital role than the other one.
However, when using this parameter setting against BoulwareNegotiationParty, our agent only received 0.502 and the opponent received 0.899, which can be read from Figure 1.

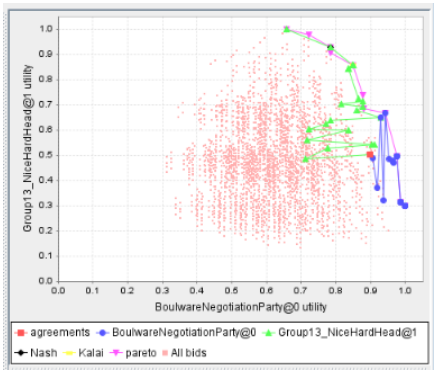


Figure 1: Negotiation against Boulware

This can be easily explained if we plot the time dependent utility value w.r.t time under different $\beta$ value in Figure 2. For convenience, we assume minimum and maximum values are 0 and 1. From the bottom, the $\beta$ values of each curve are 1, 0.8, 0.6, 0.4, 0.2, 0.1, and 0.01 respectively. It is clear that even with 0.1, the target utility value still drops rapidly and is lower that 0.7 before 90% time have passed. Hence, we choose 0.01 as our $\beta$ value because it hardly decrease until the last 4% of session time. The new negotiation outcome against BoulwareNegotiationParty is shown in Figure 3. We received utility of 0.856 and the opponent received 0.848.
As for the weight part, since we already set $\beta$ as 0.01 and from Figure 2 we know the time dependent utility does not drop before 96% of time passes. Therefore,

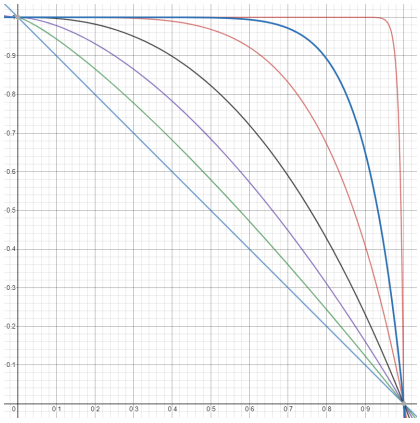

Figure 2: Time-Utility curve with different $\beta$

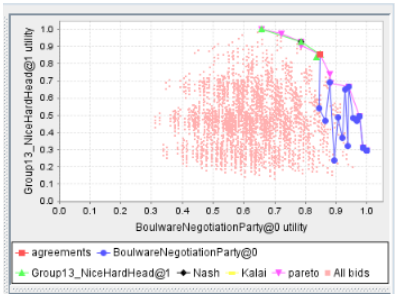


Figure 3: Negotiation against Boulware with $\beta = 0.01$

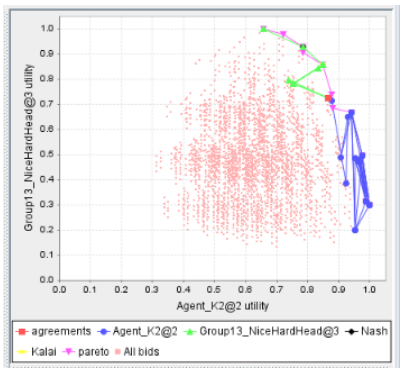


Figure 4: Negotiation against AgentK2 with weight for time dependent utility of 0.2

it is reasonable to lower the weight of time dependent. Otherwise we may never reach an agreement before the very end. For the experiment, we choose AgentK2 as our opponent to repeat same negotiation. We received 0.478 for our initial weights. But if lower the time dependent utility weight to 0.2, we received 0.697 which is shown in Figure 4. However, it does not seems significantly useful to continue to lower this weight as we received same outcome using time dependent utility weight of 0.1.

**Acceptance strategy** For our acceptance strategy, we mainly focus on the adjustment of parameter

acceptMultiplier. The acceptMultiplier for the three phases are 0.9, 1.0 and 1.1. We also tested other values for our agent, however 0.9, 1.0, 1.1 leads to higher results for our agent. We test the performance through tournaments. We put our agent with BoulwareNegotiationParty, ABMPAgent2, Bayesinan Agent, FunctionalAcceptor, Optimal Bidder Simple, and Immediate Acceptor into one tournament. At first, we tested with 0.3, 0.4, 0.5, the average utility of our agent is 0.678534. Secondly, weajusted the acceptMultiplier into 1.5, 1.6, 1.7, the resulting average utility is 0.660692076. The average utilities are similar, which means the medians of these two sets of values may lead to better results. Therefore, we chose 0.9, 1.0, 1.1 and the average utility is 0.722076, increasing 0.1 compared with the other two sets. Therefore, we conducted further experiments with acceptMultiplier round this set. However, the changes in average utility is tiny. At last, we chose 0.9, 1.0, 1.1 as acceptMultiplier for our agent. Besides, when calculating the left moves in phase 3, we decided to calculate average time from phase 2 rather than phase 3 as we designed originally. It is because, when the negotiation rounds or time is relatively less, it is too late to start calculating this average time in phase 3. In this situation, there are maybe only 2 or 3 moves in the whole phase 3. Therefore, we decided to calculate average time for left moves from phase 2.

## 4 Answers to assignment questions

### 4.1 Single negotiation in party domain

In this subsection, we have our agent negotiate against itself, HardHeaded, Gahboninho and IAMhaggler2011 on the party domain, the results are shown below.

First, we have our agent negotiate against itself. The result reaches a Pareto efficient outcome. The utility value for two parties are very close: 0.85 and 0.86 respectively. The reason for this outcome is that our agent takes the opponent's utility value into consideration and offers bid with the highest opponent's utility based on our target utility. This makes our bids closer to the Pareto frontier. So when our agent negotiates with itself, it is highly possible that the outcome is on the Pareto frontier.

When negotiating with agent HardHeaded, the utility value that our agent gets is 0.72 and agent HardHeaded gets is 0.95. The outcome is neither Nash outcome nor Pareto efficient. However the distance to these two outcomes is close, which are 0.07 and 0.02 respectively. It is reasonable that agent HardHeaded gets such a high utility because it bids between a very narrow utility range and barely lower the range in most of the time. From the bidding history, we see that our agent also bids high and doesn't concede much at the beginning since we follow the opponent's change rate. However, as time passes, our acceptance threshold decreases. Also because the opponent is tough, our acceptance threshold decreases more rapidly. Therefore, we will accept sooner than HardHeaded, that's why it gets higher utility than us.

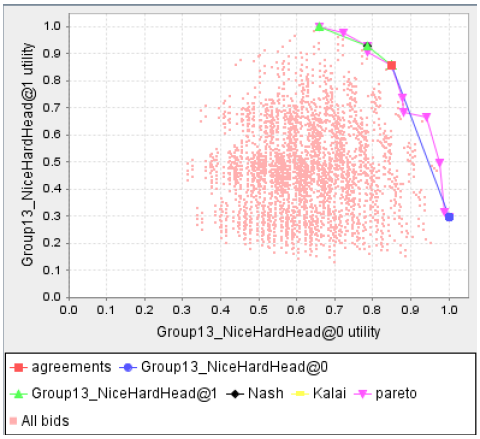


Figure 5: Negotiation against itself

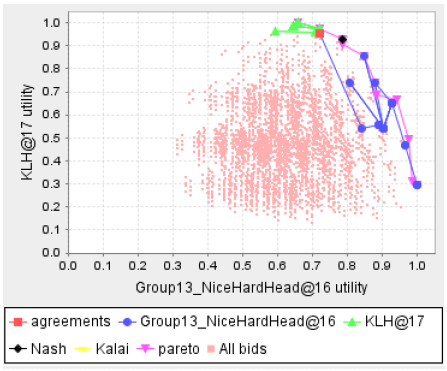


Figure 6: Negotiation against HardHeaded

When negotiating against agent Gahboninho, the utility value that our agent gets is 0.66 and agent Gahboninho gets is 1. In this negotiation, the opponent never concedes because it finds us having the intention to find a compromising outcome and also we don't have discount factor. So from bidding history, agent Gahboninho only offer bids with its own utility value as 1. As for our agent, the utility value of our bids decrease a little because of time and then stay at a relatively fixed value until our acceptance threshold drops to a certain value. This is similar to the negotiation against agent HardHeaded. Here the outcome is Pareto efficient. When negotiating against, we can always get a Pareto efficient outcome since the opponent will always get utility 1 and we get corresponding utility value.

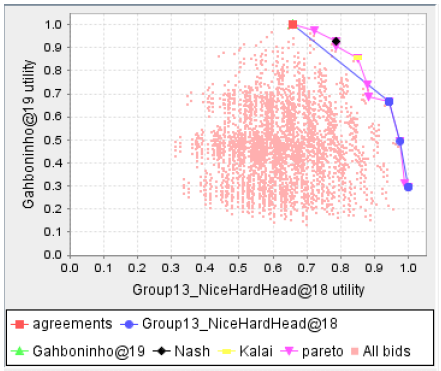


Figure 7: Negotiation against Gahboninho

Compared with two negotiation result figures above, we can see that when negotiating against agent IAMhaggler2011, the opponent's bids don't have a obvious pattern. This is because IAMhaggler2011 bids randomly between a certain range. Because the opponent's behaviour is random, so our agent's offer also follow the same change rate, which makes out offer looks random too. In this negotiation, we get a utility value of 0.82 and the opponent get a utility value of 0.74. The distance to Pareto is 0.06.

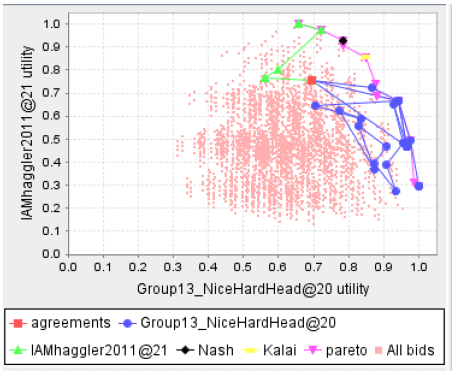


Figure 8: Negotiation against IAMhaggler2011

### 4.2 Acceptance strategy test

As for a small tournament, we select five other agents from ANAC 2011, including TheNegotiator, ValueModelAgent, Gahboninho, BRAMAgent, and HardHeaded, along with our agent to perform in Party domain. Our agent will be equipped with different acceptance strategies in different tournament while other agents will remain the same. The results are shown in Table 1.

| | Our AS | K2Agent | BRAM |
|---|---|---|---|
| Avg. Util. | 0.667(1st) | 0.394(4th) | 0.584(3rd) |
| Dist. to Pareto | 0.007(1st) | 0.521(4th) | 0.201(2nd) |
| Dist. to Nash | 0.291(1st) | 0.692(4th) | 0.411(2nd) |
| Social welfare | 1.572(1st) | 0.831(4th) | 1.316(2nd) |
| NiceTFT | ValueMode | Gahboninho | HardHeaded |
| 0.634(3rd) | 0.283(6th) | 0.297(6th) | 0.285(6th) |
| 0.029(2nd) | 0.691(4th) | 0.000(1st) | 0.691(5th) |
| 0.320(2nd) | 0.843(4th) | 0.667(4th) | 0.843(4th) |
| 1.533(2nd) | 0.568(4th) | 1.297(2nd) | 0.568(5th) |

Table 1: Our agent performance using different AS

It is still not reasonable to say our acceptance strategy outperform other's acceptance strategies. But it is clear shown in the table that our acceptance strategy is the most suitable one for our agent itself considering all performance measures. When using our own acceptance strategy, it received the highest utility among others and won the best social welfare, while reached the closest points near both Pareto efficient and Nash outcome. Among those other acceptance strategies, NiceTitForTat performances well for our agent almost approached the same outcome as our own strategy.

### 4.3 Our opponent model vs frequency model

To test our opponent model against HardHeaded frequency model, we start tournaments against 5 agents from ANAC 2011. Also to make this more statistically convincing, we will test both model 10 times to see if there is a significant difference between these utility values we received. In this case, other measurements will not be considered. The results are shown below.

| Tournament No. | Our OM | Frequency Model |
|---|---|---|
| 1 | 0.664 | 0.648 |
| 2 | 0.648 | 0.648 |
| 3 | 0.648 | 0.645 |
| 4 | 0.667 | 0.645 |
| 5 | 0.648 | 0.650 |
| 6 | 0.648 | 0.667 |
| 7 | 0.648 | 0.670 |
| 8 | 0.670 | 0.648 |
| 9 | 0.642 | 0.645 |
| 10 | 0.650 | 0.648 |

Table 2: Average utility value with different OM

We also performed $t$-test to see the significant level between them. The $P$-value we calculated is 0.5255, meaning there is no significant difference between our opponent model and frequency model w.r.t. the utility value. This is because we indeed made little change to the original frequency model. We just implement its idea in our own way.

### 4.4 Preference uncertainty inspection

We had our agent play against TheNegotiator in Party domain with profile uncertainty enabled under differ-

ent degrees of preference uncertainty, namely 10, 20, and 50.

| Utility | Our agent | TheNegotiator |
|---|---|---|
| Degree = 10 | 0.765 | 0.616 |
| Degree = 20 | 0.765 | 0.616 |
| Degree = 50 | 0.907 | 0.551 |

Table 3: Performance under preference uncertainty

We can see from Table 3 that, as the degree grows, our agent tends to receive a higher utility value. It is reasonable because with larger degree, our agent is able to detect more bid and their ranking order. Then our agent will offer a bid with larger utility than small degree situation.

### 4.5 Generalization capability

In order to test our agent's generalization capability, we have our agent negotiate against with itself, HardHeaded, Gahboninho and IAMhaggler2011 respectively in different domain. The results is shown below. In the table, util.1 denotes our utility and util.2 denotes opponent's utility. The agent names in the column denotes the opponents. We run the negotiation in ItexvsCypress, Smart_Grid and Laptop domain.The tables below are shown in the same order.

From these tables, we can see that in ItexvsCypress domain, our performance is worse than the other two domains and also worse than party domain. Because we didn't reach an agreement with Gahboninho, and the utility we get from negotiating against HardHeaded is low, the average utility in this domain is lower than others. However, our distance to Pareto is still small (except the one that we didn't reach an agreement). In the other two domains, our agent performs well like in the party domain. Our utility is relatively low when negotiating against tough type agents: HardHeaded and Gahboninho. When negotiating with compromising agents we always reach a fair agreement which is Pareto efficient. Therefore, from the results we get, we can say our agent can generalize well in other domains.

| | util.1 | util.2 | Dis.to Pareto |
|---|---|---|---|
| NiceHardHead | 0.711 | 0.392 | 0.102 |
| HardHeaded | 0.301 | 0.894 | 0.000 |
| Gahboninho | 0.000 | 0.000 | 0.886 |
| IAMhaggler2011 | 0.615 | 0.695 | 0.023 |
| Average | 0.409 | | 0.252 |

Table 4: Agent performances in ItexvsCypress domain

| | util.1 | util.2 | Dis.to Pareto |
|---|---|---|---|
| NiceHardHead | 0.894 | 0.833 | 0.000 |
| HardHeaded | 0.667 | 1.000 | 0.000 |
| Gahboninho | 0.720 | 0.832 | 0.128 |
| IAMhaggler2011 | 0.894 | 0.833 | 0.000 |
| Average | 0.794 | | 0.032 |

Table 5: Agent performances in Smart_Grid domain

| | util.1 | util.2 | Dis.to Pareto |
|---|---|---|---|
| NiceHardHead | 0.874 | 0.874 | 0.000 |
| HardHeaded | 0.815 | 1.000 | 0.000 |
| Gahboninho | 0.815 | 1.000 | 0.000 |
| IAMhaggler2011 | 0.941 | 0.852 | 0.000 |
| Average | 0.861 | | 0.000 |

Table 6: Agent performances in Laptop domain

### 4.6 Strong and weak points

After determine the values for each parameter, we start different tournaments to quantify the performance of our agent. We select BoulwareNegotiationPatry, ConcederNegotiationParty, and all agents from ANAC 2011 and choose Party domain.

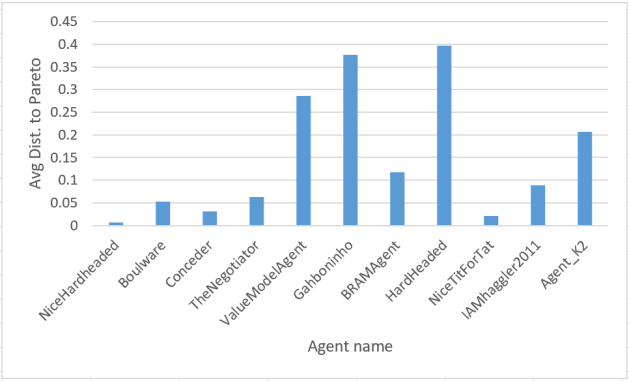


Figure 9: Average distance to Pareto

The most out-standing strong point of our agent is that our distance to Pareto is always small. When comparing the average distance to Pareto of different agents in the tournament, our value is the lowest. The comparison is shown in the Figure 9. Our agent is the first column NiceHardHeaded.

Furthermore, from the results of us negotiating against different agents,we can see than if the opponent is neither completely tough nor conceding type and has the intention to reach a compromising agreement, we can always get a Pareto efficient with them. And the utility distribution is close to 50/50, Which means we always reach a fair agreement with them.

| | Our utility | Opponent utility |
|---|---|---|
| Boulware | 0.856 | 0.848 |
| TheNegotiator | 0.856 | 0.848 |
| BRAMAgent | 0.856 | 0.848 |
| NiceTitForTat | 0.856 | 0.848 |
| IAMhaggler2011 | 0.856 | 0.848 |

Table 7: Fair negotiation results against other agents

| | Our utility | Opponent utility |
|---|---|---|
| Gahboninho | 0.297 | 1.000 |
| HardHeaded | 0.485 | 0.954 |
| Agent_K2 | 0.861 | 0.675 |

Table 8: Negotiation results against tough agents

Below is a table showing the negotiation results of our agent negotiating against these agents.
When negotiating against a conceding type agent which accepts easily like Conceder, we can get as much as utility as we can. For example, we negotiating against agent Conceder, we get utility value of 1.
Our weak point is that we accept sooner when negotiating against tough agents which don't accept easily unless under situation like high factor discount or that time is about to end. Therefore, when negotiating with these agents, the opponent will keep bidding high and our acceptance threshold keeps decreasing (in a more rapid rate since it detects that the opponent is not compromising) until we accept the offer. In this case, our agent doesn't perform well. Below is a table showing some examples.
To conclude, our agent performs well if the opponent's acceptance criterion is not very strict and accept sooner than us. If the opponent doesn't accept easily, we will accept sooner. As long as the opponent keeps bidding high, we will eventually accept opponent's offer once our acceptance threshold drops to a certain level. However, no matter how opponents bid, we always offer bids that are close to Pareto. So the outcomes we get have minimal average distance to Pareto. It is noted that because our agent is also behaviour-dependent, if the opponent keeps decreasing it's bidding utility, our offer utility will also keep dropping unpleasantly. This is the disadvantage of being behaviour dependent. But also because we are behaviour dependent, if the opponent keeps bidding high, we can also prevent our utility from dropping too soon.

## 5 Conclusion

In this part, we want to talk about aspects that we find can be improved during the experiments. First, in terms of our bidding strategy, we use a fixed weight for time-dependent tactic and behaviour-dependent tactic. However, during experiment, we find our utility drops too fast when opponent concedes fast. If we can change the weight in different time phase, the strategy would be more flexible. For example, we can gradually decrease behaviour-dependent weight so that the utility doesn't drop too much because of opponent's behaviour and avoid our opponent taking advantage of our strategy.
For acceptance strategy, the entire acceptance strategy of our agent is tough when facing with not greedy opponents to acquire higher utilities and less stubborn when facing with greedy opponents to achieve as more agreements as possible. In our design, we adopted ideas of functions from several previous negotiation agents, for example function to calculate current phase. We also constructed a function to predict the type of our opponents. However, the prediction method is relatively simple. We only compared the averages of predicted opponent utilities to decide the opponent type. Although we are able to gain high utility in a tournament, it would be better to use a more complex method to predict opponent type.
For opponent model, in many recent studies, frequency modeling has been shown to outperform some other opponent modeling techniques like Bayesian approaches. The reason is weaker assumptions on the opponent's behaviours. Nevertheless, frequency models still rely on some underlying assumptions that may not be true in many agents. In our agent we have presented an improved approach to opponent modeling in automated negotiation. This new approach smooths the effect of some of the assumptions in the classic frequency model, which is decayed weight update to avoid incorrect updates when the opponent starts conceding on the most important issues; But there is still something that we can work on in future. When the opponent offers the same offer repeatedly, the relevant issue values importance will witness an unbalanced growth. In that case, the growth of issue values importance should be slowed somehow.

**Team contribution** We divide our task based on BOA structure because each component can be programmed individually. Also, improving each opponent individually can also improve the agent's performance as a whole. So we divide the task this way. Siyang Qian and Shaoya Ren are responsible for opponent model and acceptance strategy respectively. Since bidding strategy is the most important component, Xin Lan and Jiayuan Hu are in charge of this part. We also assign different report parts based on the same distribution. Xin Lan and Jiayuan Hu also take the evaluation part of this report and answered those question asked on the assignment. The other two people finish the introduction and conclusion. During the time, we kept communicating with

each other about what functions others need in their components and discussing each component's strategy in order to keep everyone informed of the whole picture.

## References


[1] Mai Ben Adar, Nadav Sofy, and Avshalom Elimelech. Gahboninho: Strategy for balancing pressure and compromise in automated negotiation. In *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pages 205–208. Springer, 2013.

[2] Tim Baarslag, Koen V. Hindriks, Mark Hendrikx, A. S. Y. Dirkzwager, and Catholijn M. Jonker. Decoupling negotiating agents to explore the space of negotiation strategies. In *Novel Insights in Agent-based Complex Automated Negotiation*, 2012.

[3] Muhamad Hariz Bin Muhamad Adnan, Mohd Fadzil Hassan, Associate Professor Dr Izzatdin Aziz, and Irving Paputungan. Protocols for agent-based autonomous negotiations: A review. pages 622–626, 08 2016.

[4] Enrico H. Gerding Colin R. Williams, Valentin Robu and Nicholas R. Jennings. Iamhaggler2011: A gaussian process regression based negotiation agent. In *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pages 209–212. Springer, 2013.

[5] ASY Dirkzwager, MJC Hendrikx, and JR De Ruiter. Thenegotiator: A dynamic strategy for bilateral negotiations with time-based discounts. In *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pages 217–221. Springer, 2013.

[6] Peyman Faratin, Christopher Sierra, and N. Jennings. Designing responsive and deliberative automated negotiators. 01 1999.

[7] Faratin P. Lomuscio A. et al. Jennings, N. Automated negotiation: Prospects, methods and challenges. page 199–215, 2001.

[8] Raz Lin, Sarit Kraus, Jonathan Wilkenfeld, and James Barry. Negotiating with bounded rational agents in environments with incomplete information using an automated agent. *Artificial Intelligence*, 172:823–851, 04 2008.

[9] R. Ros and C. Sierra. A negotiation meta strategy combining trade-off and concession moves. Autonomous Agents and Multi-Agent Systems, vol. 12, pp. 163–181, 2006.

[10] Daphne Looije Thijs van Krimpen and Siamak Hajizadeh. Hardheaded. In *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pages 223–227. Springer, 2013.

[11] Catholijn Jonker Tim Baarslag, Koen Hindriks. Effective acceptance conditions in real-time automated negotiation. *Decision Support Systems*, 60:68–77, 2014.