Intelligent Agents Report

Student ID
Group Number:11
Email

ABSTRACT
In this paper, I describe the details of our group's negotiation agent's design, including how to evaluate our utility value, how to build a model of opponent, how to get a reasonable bid and when to receive a bid and so on. Then, I will analysis the results in the tournaments and give some suggestion about our group's agent to make it be improved.

Keywords
Negotiation agent, performance uncertainly, artificial intelligence.

1. INTRODUCTION
The tournament uses the GENIUS framework to run. And it is bilateral negotiation between 2 agents. The protocol of the tournament is Stacked Alternating Offers Protocol, i.e.in every turn each agent only can do one action from the actions below: offer a new bid, accept the last bid or walk away. Each agent will repeat the process until there is an agreement or the negotiation time is run out or one agent walks away.

In addition, in the tournament, each agent has preference uncertainty, which means that each agent has uncertainty about their own preferences, but they will have a ranking of a limited set of outcomes. The agent can evaluate its utility value through these outcomes.

In our negotiation agent, we build a object to estimate opponent's preference and use a function to estimate our agent's utility value. And with the time changing, our agent's action will change. The detailed information about our group's agent's will be showed in the next section.

2. ACTION STRATEGY
As mentioned above, our agent's action will change with the time changes.

Firstly, because we have not enough information about opponent's utility function and our utility, so in the first 30% time, our agent will generate random bid and do not receive any bid to collect some bids to build models to estimate opponent's utility and our utility.

Then, from 30% time to 95% time, if the bid we received is more than 0.8 of our utility we estimated, our agent will accept the bid. If not, we will offer a bid which is in a Nash solution.

At last, if 95% time is run out and there still is not an agreement, we will try to decrease the expected received utility value. The value will be generated depending on the time. In other word, the expected received utility value will change with the time changes to make the negotiation get an agreement at end.

This is all of the action strategy of our agent. And some details of the methods we used in our agent will be explained in the next section.

3. Method
3.1 Opponent Modeling
In order to estimate opponent utility. We build a class named **AgentModel**. It has many varaiates. For example,the estimated weight of every issues, the number of issues and so on. It has two functions: **UpdateModel**() and **predictUtility**().Then let us talk about there two function. In my agent, I build a two-dimensional array named **frequency**, its first dimension is the number of issues, and the second dimension is the maximum number of values in one issue. And the elements of the array is the frequency of every bids. The array will update when it receives a bid. **UpdateModel** function will update the estimated weight of every issues of the model through the latest update array frequency.The design of UpdateModel function mainly comes from []. About **predictUtility** function, it will compute the input bid's utility value through the estimated weight of the model.The details of these two functions are as follows.

3.1.1 UpdateModel()
In every turn, we will use updateModel() to update the estimated weight of opponent. In the function, there are two sections. The first is generating estimated values of options. The second is generating estimated weight.
When generate estimated values of options, we use a loop to traversing **frequency**. Then we order the options for an issue by their frequencies, and use the equation[1] below to compute the the value predicted for every option.

$$Value_{Option} = \frac{Sum_{issue} - Order_{Option} + 1}{Sum_{issue}} \quad (3\text{-}1)$$

For the equation, $Value_{Option}$ means the value of the option. $Order_{Option}$ means the order of the option in an issue. Sum_{issue} means the number of option that included in an issue.

When generating estimated weights of every issues, firstly we sort the issues through their options' maximum frequency. And sort them in the order from small to large. For example, if issue A has three options, their frequency are 9,2,1; Issue B has three options as well, their frequency are 4,3,5. Because 9 is the biggest frequency of issue A's options and 5 is the biggest frequency of issue B's options, 9 is bigger than 5. So we assume that the weight of issue A is bigger than that of issue B. Then the order of issue A is lower than that of issue B. When we get the order of every issues, we will use it to generate estimated weights. And we generate them through the equation [2,3] below:

$$Weight_i = \frac{2 * Order_i}{N_{issue}(N_{issue}+1)} \quad (3\text{-}2)$$

For the equation, $Weight_i$ is the estimated weight of issue i, $Order_i$ means the order of issue i, N_{issue} means the sum number of issues.

3.1.2 predictUtility()

The input of the function is a bid. And the output of it is the estimated utility value of the bid for opponent. When we update the estimated weights and values, we will use the equation below to compute the opponent's utility value.

$$Utility_{opponent} = \sum_{issue\ i} (Weight_i \times Value_{Option}) \quad (3\text{-}3)$$

3.1.3 Example Model

We now provide an example scenario to illustrate our modeling approach. The frequencies of offered or accepted options in the issues by the opponents for our example is presented in table below.

Table: The frequency of options for issues for opponent

	Option 1	Option 2	Option 3
Issue A	4	5	3
Issue B	10	2	0
Issue C	1	7	4

In this section, we only computer the issue A's options' value to make a demo. For issue 1, because the frequency of option2 >option 1> option 3.So

$$Value_{Option1}=\frac{3-2+1}{3}=\frac{2}{3}.$$

$$Value_{Option2}=\frac{3-1+1}{3}=1.$$

$$Value_{Option3}=\frac{3-3+1}{3}=\frac{1}{3}.$$

Then let us computer the estimated weights of issues. To begin with, we should sort them in order from small to large. And sort them through their options' maximum frequency. Because 5 is the issue A's options' maximum frequency, 10 is the issue B's options' maximum frequency and 7 is the issue C's options' maximum frequency, and 5 < 7 < 10, So we assume the order of issue A,B,C is that issue A, issue C, issue B. After getting the order of issues, we will use the equation(3-2) to compute the estimated weights.

$$Weight_{issueA}=\frac{2\times1}{3\times(3+1)}=\frac{1}{6}.$$

$$Weight_{issueB}=\frac{2\times3}{3\times(3+1)}=\frac{1}{2}.$$

$$Weight_{issueC}=\frac{2\times2}{3\times(3+1)}=\frac{1}{3}.$$

Let's computer a bid's utility of opponent, we assume the bid is (option 1, option 3, option 3), the opponent 's utility of the bid is:

$$Utility_{opponent}=\frac{1}{6}\times\frac{2}{3}+\frac{1}{2}\times0+\frac{1}{3}\times\frac{1}{3}=\frac{1}{9}.$$

3.2 Estimate own utility

About our agent, there is a very easy function to estimate our agent's utility. GENIUS framework offer an interface that ***getBidOrder()*** function to get a bid ranking array, the output of the function is a list bid from low utility to high utility. Our method to estimate own utility mainly depend on this interface. When we need to estimate a bid's utility, we will research whether the bid is contained in the list bid which is gained from ***getBidOrder()*** function. If it is contained in the list bid, we will find its position in the list, and estimate the bid's utility through the position. As the position is lower, its utility score becomes higher. The equation is as follows. If the bid is not contained in the list bid, we return 5.5 as the bid's utility.

$$Utility_{own} = \frac{Order_{bid}}{Sum_{bid}} + 0.1 \quad (3\text{-}4)$$

For the equation, $Order_{bid}$ means the position that the estimated bid in the list gained from ***getBidOrder()*** function. Sum_{bid} means the number of bid in the list mentioned above.

3.3 Offer bids

After the estimation of utility function of opponent and our agent has been done, our agent will choose a bid which is apply nash solution. It was shown by Nash that the only solution that satisfies all of these properties is the product maximizing the agents' utilities [4]. So, I use a equation below to compute whether a bid is a good bid.

$$Score_{bid} = \frac{1}{\max(0.1,|U_{own}-U_{opponent}|)} + U_{own} \quad (3\text{-}5)$$

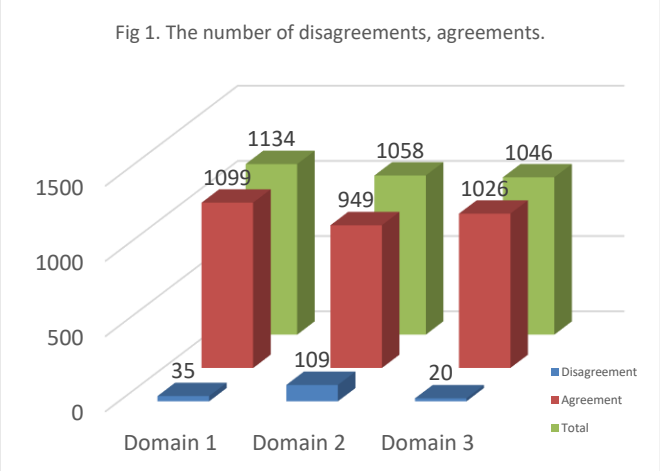
For the equation above, $Score_{bid}$ means the score of the bid, U_{own} means the estimated utility score of our agent, $U_{opponent}$ means the estimated utility score of opponent.

Before our agent propose a bid, we will calculate the score of the last bid of our agent, and generate a random new bid, compute the score of the new bid. If the new bid's score is higher than the last bid's score, we will use the new bid as our next bid. If not, we will repeat the process until have a new bid whose score is higher than the last bid's score.The pseudocode for selection of a bid is shown below.

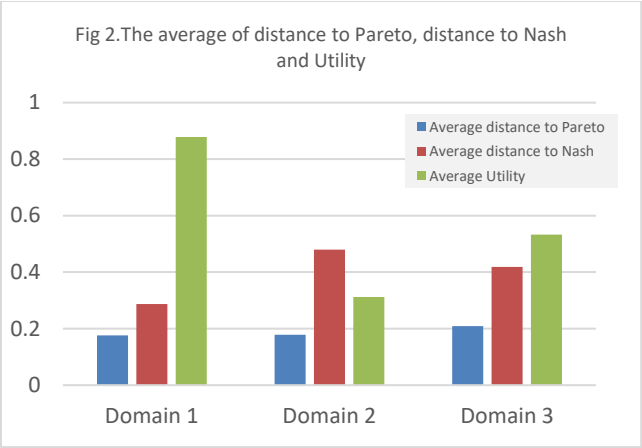
```
1.  While(true)
2.  {   oldScore=computeScore(theLastBid);
3.      newBid=generateBid();
4.      newScore=computeScore(theNewBid);
5.      If(newScore>oldScore){
6.          theLastBid=newBid;
7.          return theLastBid;}
8.  }
```

4. RESULT ANALYSIS

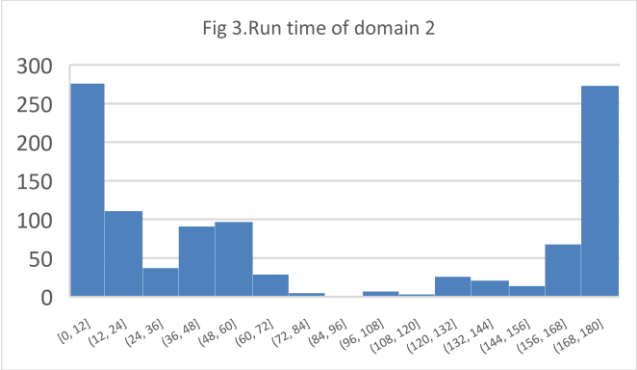
Firstly, I draw a bar chart to show the number of disagreements, agreements and total negotiations.

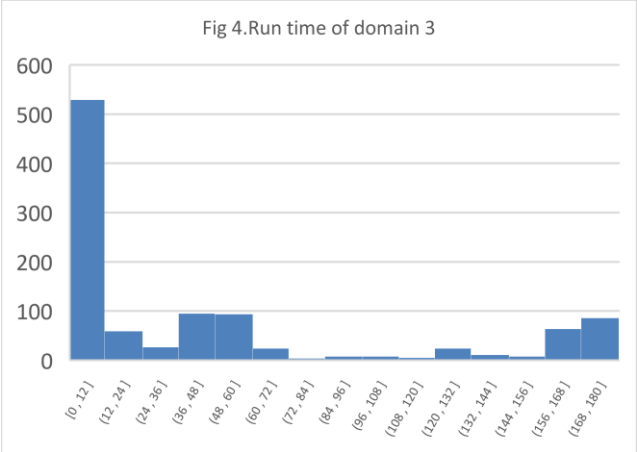


The picture above shows that most of the negotiations reach an agreement. It is a good thing. But I know the score of my agent is a little low. Then I draw a picture to show the average utility, the distance to Pareto and the distance to Nash.



Through Fig 2 we can see that the average utility of domain 1 is ok, but that of domain 2 and domain 3 are lower. I feel very strange about it, then I check the negotiation record to find some answers about it. The histogram of run time of domain 2 and domain 3 are shown below.





Then I found plenty of the negotiation is end in the first few seconds and last few seconds, and the utility of these negotiation are very low. Therefore, I turned my eyes to my code. I found there are mainly two Bugs in it, which can make my agent's utility low.

One is that I did not consider the situation of low and medium performance uncertainty, I only makes my agent solve the high performance uncertainty. So at the beginning of a low or middle uncertainty negotiation, my agent still assume it is high performance uncertainty. It will still offer random bids. If the bids offered by us are high utility to opponent but low utility to my agent, the opponent will accept it and my agent will unhappy. I think this is the reason why most of the negotiation is end in the first few seconds in domain 2 and 3, and the utility of our agent are low.

The other Bug is about the accept strategy. As I mentioned in Section 2, when 95% time run out and there still is no agreement, my agent will decrease its expected utility, and the utility will decrease. And the equation of changing expected utility is shown below.

$$Utility_{expected} = 1 - 0.5 \times Time \qquad (4\text{-}1)$$

So when time is close to 1, the expected utility will be close to 0.5. And now let us see section 3.2. In section 3.2 I mentioned that if the tested bid is not contained in the bid list which is gained from ***getBidOrder()***, I will assume its utility is 0.55. Based on these two actions, the Bug is shown: when time is close to 1 and my agent receive a bid which is not contained in the bid list, it will assume its utility is 0.55 and just accept it, because 0.55 >0.5. And we don't know the bid's utility for our agent. I think this is another reason leading to the low utility.

5. CONCLUSION AND FUTURE WORK

Based on the discussion in section 4, I know that there are two Bugs in my agent, which can make our agent's utility very low. I think fixing these two Bugs can make my agent get a better score in tournament.

In addition, the evaluation function of estimating our agent's utility can be changed to make our agent's performance improve. The function we used in our agent is too simple.

6. REFERENCES

[1] Yucel, Osman, Jon Hoffman, and Sandip Sen. Jonny Black: A Mediating Approach to Multilateral Negotiations. *Modern Approaches to Agent-based Complex Automated Negotiation.* Springer, Cham, 2017. pp. 231-238.

[2] Siqi Chen, Jianye Hao, Shuang Zhou and Gerhard Weiss. Negotiating with Unknown Opponents Toward Multi-lateral Agreement in Real-Time Domains. Modern *Approaches to Agent-based Complex Automated Negotiation.* Springer, Cham, 2017. 219-237.

[3] K. Hindriks, D. Tykhonov, 2008, Opponent modeling in auomated multi-issue negotiation using bayesian learning, in AAMAS'08 (2008), pp. 331–338.

[4]Raz Lin, Sarit Kraus, Jonathan Wilkenfeld, James Barry. Negotiating with bounded rational agents in environments with incomplete information using an automated agent.

APPENDIX :

```
package group11;
import java.util.*;

import agents.uk.ac.soton.ecs.gp4j.util.ArrayUtils;
import genius.core.Bid;
import genius.core.actions.Accept;
import genius.core.actions.Action;
import genius.core.actions.Offer;
import genius.core.issue.Issue;
import genius.core.issue.IssueDiscrete;
import genius.core.issue.Value;
import genius.core.issue.ValueDiscrete;
import genius.core.parties.AbstractNegotiationParty;
import genius.core.parties.NegotiationInfo;
import genius.core.uncertainty.AdditiveUtilitySpaceFactory;
import genius.core.utility.AbstractUtilitySpace;


public class Agent11 extends AbstractNegotiationParty {
    private class AgentModel {
        private double Issues [][];
        private double Weights [];
        public int frequency [][];
        private int N_issues;
        private int N_values;
        private int maximum_freqs [];
        private int weight_ranking[];
        private int value_ranking[];

        //build a class to estimate opponent's utility
        public AgentModel(int num_issues, int num_values, int [][] freq, List<Issue>
domain_issues) {
            Issues = new double [num_issues][num_values];
            Weights = new double [num_issues];
            frequency = freq;
            N_issues = num_issues;
            N_values = num_values;
            maximum_freqs = new int[N_issues];
            weight_ranking = new int [N_issues];
            value_ranking = new int [N_values];
        }
        //update the model
        public void updateModel() {
```

```
            int i,j,k;
            for(i = 0; i < N_issues; i++) {
                maximum_freqs[i] =
Collections.max(Arrays.asList(ArrayUtils.toObject(frequency[i])));
                if(maximum_freqs[i] == 0) {
                    maximum_freqs[i] = 1;
                }
                weight_ranking[i] = N_issues-i;
                j = i - 1;
                while(j != -1) {
                    if(maximum_freqs[i] > maximum_freqs[j]) {
                        weight_ranking[i] = Math.max(weight_ranking[j],
weight_ranking[i]);
                        weight_ranking[j]--;
                    }
                    j--;
                }
                for(j = 0; j < N_values; j++) {
                    value_ranking[j] = N_values-j;
                    k = j - 1;
                    while(k != -1) {
                        if(frequency[i][j] > frequency[i][k]) {
                            value_ranking[j]++;
                            value_ranking[k]--;
                        }
                        if(frequency[i][j] == frequency[i][k]) {
                            value_ranking[k]--;
                            value_ranking[j] = Math.min(value_ranking[k],
value_ranking[j]);
                        }
                        k--;
                    }
                }
                for(j = 0; j < N_values; j++) {
                    Issues[i][j] = value_ranking[j]*1.0/N_values;//Computer estimated
values
                }
            }
            for(i = 0; i < N_issues; i++) {
                Weights[i] = 2.0*weight_ranking[i]/(N_issues*(N_issues+1.0));//Compute
estimated weights
            }
        }
        //predict a bid's utility value of opponent
```

```
        public double predictUtility(Bid bid) {
            double U = 0.0;
            for(Issue issue: bid.getIssues()) {
                U = U +Issues[issue.getNumber()-1][((IssueDiscrete)
issue).getValueIndex((ValueDiscrete)(bid.getValue(issue.getNumber()))))]
*Weights[issue.getNumber()-1];
            }
            return U;
        }
    }
    private AgentModel agentmodel;
    private Bid lastReceivedOffer=null;
    private Bid myLastOffer;
    Bid curr_bid ;
    private int number_of_issues;
    private java.util.List<Issue> domain_issues;
    private java.util.List<ValueDiscrete> values;
    int freq [][];
    int turn=0;


    public void init(NegotiationInfo info) {
        super.init(info);
        curr_bid = generateRandomBid();
        domain_issues = this.utilitySpace.getDomain().getIssues();
        number_of_issues = domain_issues.size();
        System.out.format("Domain has %d issues\n ", domain_issues.size());
        int max_num_of_values;
        max_num_of_values = 0;
        for(Issue lIssue : domain_issues) {
            IssueDiscrete lIssueDiscrete = (IssueDiscrete) lIssue;
            values = lIssueDiscrete.getValues();
            if (values.size() > max_num_of_values) {
                max_num_of_values = values.size();
            }
        }
        freq = new int [domain_issues.size()][max_num_of_values];
        agentmodel = new AgentModel(domain_issues.size(), max_num_of_values, freq,
domain_issues);
        for(int i = 0 ; i < domain_issues.size() ; i ++){
            for(int j = 0 ; j < max_num_of_values ; j++){
                freq [i][j] = 0;
            }
        }
```

```
    }

    @Override
    public Action chooseAction(List<Class<? extends Action>> possibleActions) {
        turn+=1;
        if(lastReceivedOffer == null) {
            System.out.println("\nOffering random Bid at the beginning");
            myLastOffer = generateRandomBid();
            lastReceivedOffer=myLastOffer;
            return new Offer(this.getPartyId(), myLastOffer);
        }else {
            lastReceivedOffer= ((Offer) getLastReceivedAction()).getBid();
            update_freq(lastReceivedOffer);
            agentmodel.updateModel();
            double test=agentmodel.predictUtility(lastReceivedOffer);
            System.out.printf("The test of predict is %f",test);
            if (getTimeLine().getTime() < 0.3) {
                return new Offer(getPartyId(), generateRandomBid());
            } else if (gettheUtility(lastReceivedOffer)>0.8) {
                return new Accept(this.getPartyId(), lastReceivedOffer);

            } else if (getTimeLine().getTime() > 0.95 &&
gettheUtility(lastReceivedOffer)>= this.getTargetUtility()) {
                return new Accept(this.getPartyId(), lastReceivedOffer);
            }
            myLastOffer =generateBid();
            return new Offer(this.getPartyId(), myLastOffer);
        }
    }

    private void log(String s) {
        System.out.println(s);
    }

    @Override
    public AbstractUtilitySpace estimateUtilitySpace() {
        return new AdditiveUtilitySpaceFactory(getDomain()).getUtilitySpace();
    }
    public double getTargetUtility() {
        return 1-0.5*getTimeLine().getTime();
    }

    //estimate own utility
    public double gettheUtility(Bid bid){
```

```
        List<Bid> bidOrder = userModel.getBidRanking().getBidOrder();

        if (bidOrder.contains(bid)) {
            double percentile =  bidOrder.indexOf(bid)
                    / (double) bidOrder.size();
            return percentile+0.1;
        }else{
            return 0.55;
        }
    }

    public void update_freq (Bid curr_bid_freq ){

        java.util.List<Issue> bid_issues;
        java.util.HashMap<java.lang.Integer, Value>    bid_values;
        IssueDiscrete lIssueDiscrete ;
        ValueDiscrete lValueDiscrete;
        bid_issues = curr_bid_freq.getIssues();
        bid_values = curr_bid_freq.getValues();


        for (Integer curr_key : bid_values.keySet()){
            lIssueDiscrete = (IssueDiscrete) (bid_issues.get(curr_key -1));
            lValueDiscrete = (ValueDiscrete) bid_values.get(curr_key);
            freq [curr_key-1][lIssueDiscrete.getValueIndex(lValueDiscrete.getValue())]
++;
        }

    }

    private Bid generateBid(){
        double T , T_old, oldScore , newScore , myUtility, predicted=0 ;
        Bid new_bid;
        int random_issue;
        HashMap<Integer, Value> curr_bid_value = new HashMap<Integer, Value>();
        HashMap<Integer,Value>     bid_values;
        Random randomnr = new Random();
        for(int k = 1 ;k<600  ; k++){
            myUtility = this.gettheUtility(curr_bid);
            predicted= agentmodel.predictUtility(curr_bid);
            oldScore = (1.0/Math.max(0.1, Math.abs(myUtility - predicted)))+
(myUtility);
            bid_values = curr_bid.getValues();
            random_issue = randomnr.nextInt(number_of_issues);
```

```
            for(Issue lIssue : domain_issues){
                IssueDiscrete lIssueDiscrete = (IssueDiscrete) lIssue;
                if( random_issue != lIssue.getNumber() ){
                    curr_bid_value.put(lIssue.getNumber() ,
bid_values.get(lIssue.getNumber()));}
                else {
                    curr_bid_value.put(lIssue.getNumber() ,
lIssueDiscrete.getValue(randomnr.nextInt( lIssueDiscrete.getNumberOfValues())));
                }
            }
            new_bid = new Bid(utilitySpace.getDomain(), curr_bid_value);
            myUtility = this.gettheUtility(new_bid);
            predicted = agentmodel.predictUtility(new_bid);
            newScore =(1.0/Math.max(0.1, Math.abs(myUtility - predicted))) +
(myUtility);
            if(this.gettheUtility(new_bid) < 0.9 && this.gettheUtility(new_bid) >
getTargetUtility() ){
                if(newScore > oldScore ){
                    curr_bid = new_bid;
                    break;
                }
            }
        }
        return curr_bid;
    }
    @Override
    public String getDescription() {
        return "Negotation agent of group 11 which can deal with uncertain
preferences";
    }
}
```