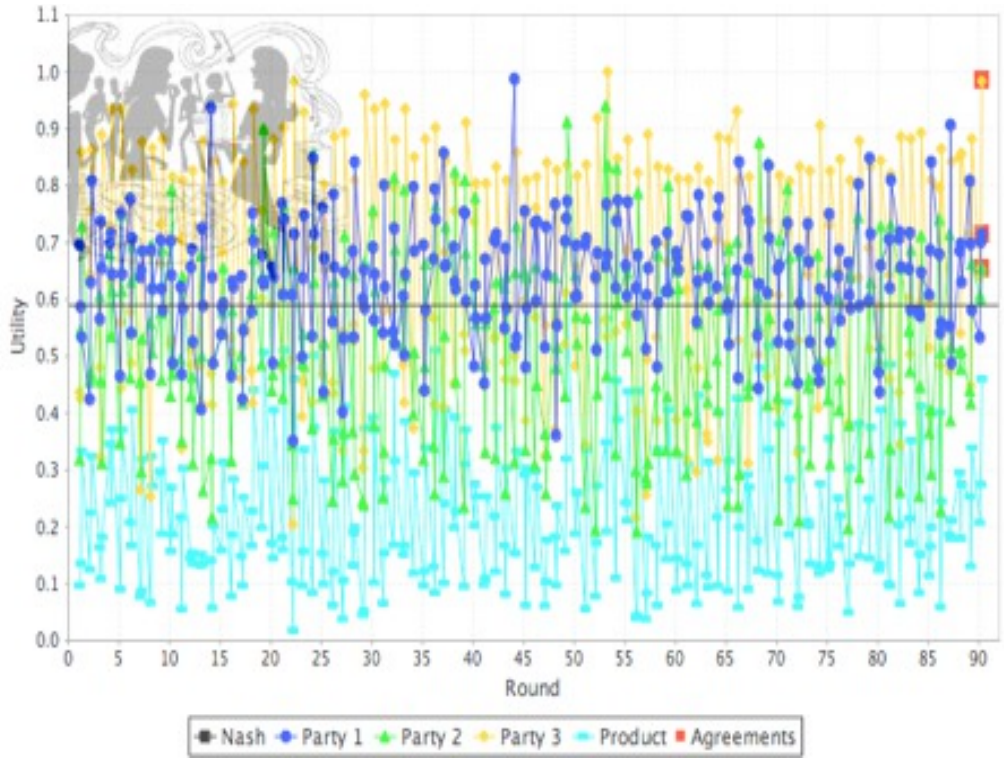
Artificial Intelligence Techniques

IN4010

Automated Negotiation - Part 1 – Group 13

October 2015, TU Delft, The Netherlands



Authors:

J. Oostenbrink
4169263

T. Boumans
4214854

B. Foppes
9243144

Introduction

In our automated negotiation assignment we've just graduated and want to organize a party. Since we have conflicting interests we decided to use GENIUS for automated negotiation. Our preferences are stored in a preference profile and these preference profiles are part of the deliverables. For the negotiating part itself we designed a negotiating agent and experimented with some basic strategies.

Preference profiles

The first task is the create three preference profiles. The three group members each created a preference profile reflecting their interests. The motivation for each profile is described below from the viewpoint of their creator.

Preference profile 1 (party_utils_bernt.xml)

My overall motivation for my preference profile is motivated by cost. Areas where costs can be reduced have the highest weight and within each category such as food, drinks etcetera the options with the lowest cost have a higher evaluation value. For these I prefer to do as much ourselves if possible. Otherwise the cheapest options have higher evaluation values. The categories where mosts cost reductions can be achieved are location, music and food.

In food, cheap and/or handmade have the highest evaluation value whereas catering the lowest as catering is quite expensive in general. For drinks the same motivation holds but I made an exception for cocktails because I simply prefer them strongly over other drinks. Location most likely is the category with the highest cost as hiring a location is expensive. As my dorm cannot hold many people this has the lowest evaluation value. Invitations I don’t care much about but a handmade plain invitation has the most personal touch which I like. Concerning music, MP3 is a very simple and cheap solution. A DJ might be found in our network of friends but a band is too costly to consider. For cleanup, hired help and/or equipment will increase costs therefore simple water and soap has the highest evaluation value.

Preference profile 2 (party_utils_thijs.xml)

The most important issue is the drinks, as a competitive rower I do not drink alcohol and as such non-alcoholic is my preference, handmade Cocktails and catering will probably both allow for some flexibility but beer only is largely out of the question, furthermore some good food is always appreciated. The other issues are less important, however their expenses should be monitored, I do not care about how the invitation goes (handmade cards are nice but plain are easier). I don't really like it when a part is focused more on music then on people so my preference goes to mp3 and for cleaning well we will see about that later but water and soap should do the trick. Lastly I don't particularly like parties so my reservation value is quite high.

Preference profile 3 (party_utils_Jorik.xml)

This whole preference profile is build around the fact I don't drink alcohol. So it's very important to me that there are non-alcoholic drinks available at the party. The only options for Drinks for which there will be non-alcoholic beverages available at the party are Non-Alcoholic and Catering. I prefer Catering, as this means that there will also be alcoholic drinks available for those that do want to drink alcohol. All the other issues don't matter to me that much, although Food is a little more important than the rest of the issues and music is a non-issue for me. That's why Drinks got a weight of 0.8 and the rest of the weight is spread around the other issues. This means that a negotiator should try to get the non-alcoholic drinks options chosen and concede on all the other issues. Because the party isn't important enough for me to go to when there isn't anything to drink, I've put the reservation value at 0.5. So any negotiator should only accept (and offer) bids which contain Non-Alcoholic or Catering for Drinks.

Potential inaccuracies within the preference profiles are due to interpretation. For example there was discussion among us if cocktails contain alcohol by definition or not.

For a general insight in the different preference profiles they are displayed in the table below:

	Profile 1	Profile 2	Profile 3
Food: Chips and Nuts	7	2	1
Food: Finger-food	4	2	2
Food: Handmade food	5	4	3
Food: Catering	1	3	4
Food Weight	0.18	0.195	0.1
Drinks: Non-alcoholic	6	8	4
Drinks: Beer Only	2	0	0
Drinks: Handmade Cocktails	8	4	0
Drinks: Catering	1	6	5
Drinks Weight	0.16	0.599	0.8
Location: Party Tent	7	3	1
Location: Your Dorm	1	6	2
Location: Party Room	3	2	2
Location: Ballroom	2	1	2
Location Weight	0.29	0.089	0.03
Invitations: Plain	6	1	4
Invitations: Photo	2	1	2
Invitations: Custom Handmade	5	1	1
Invitations: Custom Printed	3	1	2
Invitations Weight	0.1	0	0.03
Music: MP3	8	3	1
Music: DJ	4	2	1
Music: Band	1	1	1

Music Weight	0.2	0.103	0
Cleanup: Water and Soap	7	4	2
Cleanup: Specialized Materials	2	3	1
Cleanup: Special Equipment	1	2	1
Cleanup: Hired Help	1	2	3
Cleanup Weight	0.08	0.012	0.04
Reservation Value	0.48	0.7	0.5

Inaccuracies

There are some inconsistencies in how each of us interpreted the values. For example, in profile 2 it is assumed that cocktails can be non-alcoholic, while in profile 3 it is assumed that cocktails are always alcoholic. So while both parties may agree on drinks, their negotiators may not reach this agreement.

Another inaccuracy is that this domain assumes different values can't be combined. e.g. you can't have both beer and non-alcoholic drinks, or finger-food and chips and nuts. In reality we can combine these values, possible resulting in a very good compromise.

Basic negotiating agent

Our very first negotiating agent has only one tactic which is to only accept a bid if it's utility is equal to or higher than the reservation value. If the reservation value is less than 0.8, the minimum accepted utility is 0.8. The code for this very basic agent is shown below.

```
public class Group13 extends AbstractNegotiationParty {
    private double minUtility = 0.8;
    private double lastBid = 0;

    public void init() {
        minUtility = Math.max(minUtility, utilitySpace.getReservationValueUndiscounted());
    }

   @Override
    public Action chooseAction(List<Class<? extends Action>> validActions) {
        try {
            if (validActions.contains(Accept.class) && shouldAccept(lastBid)) {
                return new Accept();
            } else {
                return new Offer(generateBid());
            }
        } catch (Exception ex) {
            System.err.println("Exception in chooseAction: " + ex.getMessage());
            return new Accept();
        }
    }

   @Override
```

```
    public void receiveMessage(Object sender, Action action) {
        super.receiveMessage(sender, action);
        if (action instanceof Offer) {
            lastBid = getUtility(((Offer) action).getBid());
        }
    }

    private boolean shouldAccept(double utility) {
        return utility >= minUtility; }

    private Bid generateBid() throws Exception {
        return utilitySpace.getMaxUtilityBid();
    }

    @Override
    public String getDescription() {
        return "Negotiator Group 13"; }
}
```

Although this is a very basic agent, the performance against agents which only generate random bids and in time start to concede is still very good.

Our first improvement was to generate a set of bids with a utility greater than 0.8. Only bids from this set are used in the negotiation. This agent is a more realistic one but performs about the same as the very first one. The main code for this first improved agent is shown below:

```
    ArrayList<HashMap<Integer,Value>> ret = new ArrayList();
        for (ValueDiscrete v : issueD.getValues()) {
            for (HashMap<Integer,Value> bid : bids) {
                HashMap<Integer,Value> newBid = new HashMap(bid);
                newBid.put(issueD.getNumber(), v);
                ret.add(newBid);  }
        }
        return ret;
    }
}

@Override
    public Action chooseAction(List<Class<? extends Action>> validActions) {
        try {
            if (validActions.contains(Accept.class) && shouldAccept(lastBid)) {
                return new Accept();
            } else {
                return new Offer(generateBid());
            }
        } catch (Exception ex) {
            System.err.println("Exception in chooseAction: " + ex.getMessage());
            return new Accept();
        }
    }


@Override
 public void receiveMessage(Object sender, Action action) {
    super.receiveMessage(sender, action);

     if (action instanceof Offer) {
         lastBid = getUtility(((Offer) action).getBid());
     }
 }

private boolean shouldAccept(double utility) {
    return utility >= minUtility;
}
```

```
private Bid generateBid() throws Exception {
    if (allowedBids == null) initBids();
      return allowedBids.get(rng.nextInt(allowedBids.size()));
}
```

As second bot attempt to negotiate by conceding towards what the other agents want, is done by offering the best deal that concedes one item towards what the other agent wants, the code for generating these offers is shown below:

```
public Bid generateLeastPainfullCompromise(Bid ourLast, Offer currentOther){
        Bid best=currentOther.getBid();
        Iterator<Entry<Integer, Value>> it = currentOther.getBid().getValues().entrySet().iterator();
        HashMap<Integer,Value> ours = ourLast.getValues();
        while(it.hasNext()){
                Entry<Integer,Value> issue=it.next();
                if(!issue.getValue().equals(ours.get(issue.getKey()))){
                        Bid compromise = new Bid(ourLast);
                        compromise.setValue(issue.getKey(), issue.getValue());
                        if(getUtility(compromise)>getUtility(best)){
                                best=compromise;
                        }
                }
        }
                return best;
   }
```

This both works well against the random agents but against a better agent it will quickly concede all the way towards whatever is best for the other agent suggesting that maybe something like tit for tat is needed.

Conclusion

Against a random agent we've discovered that the best strategy is to offer the highest utility bid until it is accepted. However, against better against other methods have to be applied.