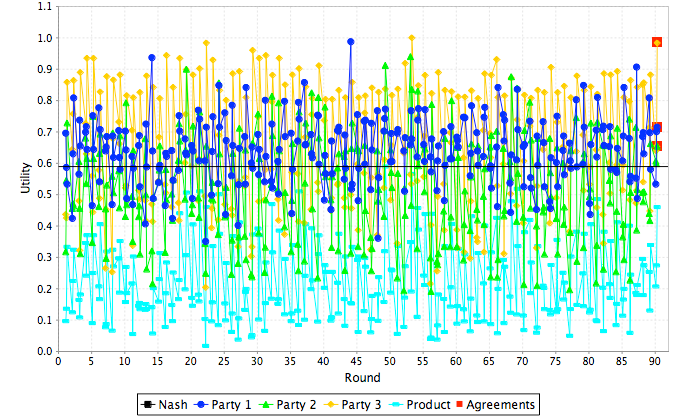
**Artificial Intelligence Techniques   
IN4010**

**Automated Negotiation - Part 1**

October 2015, TU Delft, The Netherlands



**Authors:**

**J. Oostenbrink T. Boumans B. Foppes**

**4169263 4214854 9243144**

### Introduction

bla

### 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.

*Preference profile 1*

My overall motivation for my preference profile is motivated by cost. Areas where costs can be reduced have the highest weight and each 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*

*Preference profile 3*

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 |  |  |
| Food: Finger-food | 4 |  |  |
| Food: Handmade food | 5 |  |  |
| Food: Catering | 1 |  |  |
| *Food Weight* | 0.18 |  |  |
| Drinks: Non-alcoholic | 6 |  |  |
| Drinks: Beer Only | 2 |  |  |
| Drinks: Handmade Cocktails | 8 |  |  |
| Drinks: Catering | 1 |  |  |
| *Drinks Weight* | 0.16 |  |  |
| Location: Party Tent | 7 |  |  |
| Location: Your Dorm | 1 |  |  |
| Location: Party Room | 3 |  |  |
| Location: Ballroom | 2 |  |  |
| *Location Weight* | 0.29 |  |  |
| Invitations: Plain | 6 |  |  |
| Invitations: Photo | 2 |  |  |
| Invitations: Custom Handmade | 5 |  |  |
| Invitations: Custom Printed | 3 |  |  |
| *Invitations Weight* | 0.1 |  |  |
| Music: MP3 | 8 |  |  |
| Music: DJ | 4 |  |  |
| Music: Band | 1 |  |  |
| *Music Weight* | 0.2 |  |  |
| Cleanup: Water and Soap | 7 |  |  |
| Cleanup: Specialized Materials | 2 |  |  |
| Cleanup: Special Equipment | 1 |  |  |
| Cleanup: Hired Help | 1 |  |  |
| *Cleanup Weight* | 0.08 |  |  |
| *Reservation Value* | 0.48 |  |  |

### Basic negotiating agent

Our very first negotiating agent has only one tactic which is too 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, against an agents which only generates random bids and in time starts to concede, it performs very well.

|  |
| --- |
| *public class Group13 extends AbstractNegotiationParty {*  *private double minUtility = 0.8;*  *private double lastBid = 0;*    *private ArrayList<Bid> allowedBids = null;*  *private Random rng = new Random();*  *public void init() {*  *minUtility = Math.max(minUtility, utilitySpace.getReservationValueUndiscounted()); }*    *public void initBids() throws Exception {*  *ArrayList<Issue> issues = utilitySpace.getDomain().getIssues();*  *allowedBids = new ArrayList();*  *for (HashMap<Integer,Value> values : getAllBids(issues, 0)) {*  *Bid bid = new Bid(utilitySpace.getDomain(), values);*  *if (getUtility(bid) >= minUtility) {*  *allowedBids.add(bid); }*  *}*  *}*    *public static ArrayList<HashMap<Integer,Value>> getAllBids(ArrayList<Issue> issues, int from) throws Exception {*  *Issue issue = issues.get(from);*  *if (issue.getType() != ISSUETYPE.DISCRETE) {*  *throw new Exception("Issuetype " + issue.getType() + " not supported”); }*  *IssueDiscrete issueD = (IssueDiscrete)issue;*  *ArrayList<HashMap<Integer,Value>> bids;*  *if (from == issues.size()-1) {*  *bids = new ArrayList();*  *bids.add(new HashMap());*  *} else {*  *bids = getAllBids(issues, from+1); }* |
| *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()));*  *}* |

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.

### Discussion

a

### Conclusion

a