COMP6203: Lab #3

Intelligent Agents
University of Southampton

This lab introduces a more advanced strategy using opponent modelling, and shows how to evaluate your agent against various metrics.

Make sure you have finished the previous lab before starting this one.

Contents

1 Theory Crash Course

There are multiple quality metrics that can help you to analyse your negotiating agent:

1.1 Definitions

- **A negotiation session**: A single run of a negotiation, beforehand the names of agents, which preference profiles that they are assigned to and some other parameters needs to be given to make a single run.
- **A tournament**: Consists of many negotiation sessions. A tournament takes a set of negotiation agents, a domain (with many preference profiles.) and some parameters to determine negotiation sessions such as:
  - Deadline
  - Repetitions (*repetition of a negotiation session*)
  - Data Persistency (*access to data of the previous negotiation sessions*)
  - Protocol

1.2 Quality Metrics

1.2.1 Individual Utility

The utility that is gained from the perspective of an agent. For Agent A, for instance it is:

$$U_j(o^*) = \sum_{i=1}^{n} w_j^i \frac{eval_j(o_i^*)}{max(eval_j(I_i))} \tag{1}$$

where o^* is the accepted offer.

1.2.2 Social Welfare

Implemented in MultilateralAnalysis.java:299:

The social welfare (SW) is the sum of all the individual utilities of all agents. For agreement o^* and m agents, this is:

$$SW(o^*) = \sum_{j=1}^{m} U_j(o^*) \tag{2}$$

1.2.3 Distance to Pareto Efficient Frontier

Implemented in MultilateralAnalysis.java:322:

The Pareto efficient frontier (or simply Pareto frontier) is the set of all Pareto efficient outcomes.

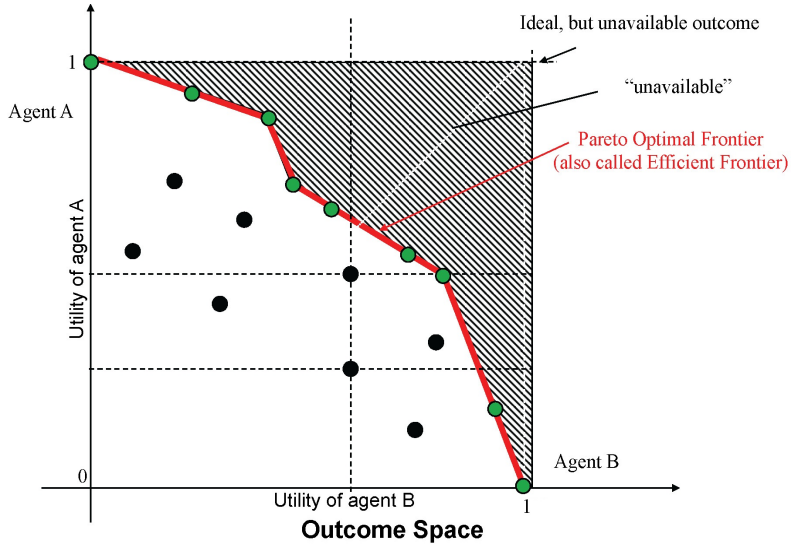

Figure 1: *Outcome space and Pareto frontier*

The distance is calculated by iterating over all the Pareto efficient offers and getting the minimum distance from this set. Formally:

$$ParetoDist(o^*) = min(\{||U(o^*) - U(o_1)||, \ldots, ||U(o^*) - U(o_i)||, \ldots, ||U(o^*) - U(o_k)||\})$$

where o_i is an outcome/offer on the Pareto frontier, k is the number of outcomes on the Pareto frontier, $||U(o1) - U(o2)||$ is the Euclidean distance (see below) between two offers, and $U(b) = (U_1(o), \ldots, U_n(o))$ is a tuple that holds the utility of each agent j.

The Euclidean distance is calculated as:

$$||U(o1) - U(o2)|| = \sqrt{\sum_{i=1}^{n} (U_i(o1) - U_i(o2))^2} \tag{3}$$

1.2.4 Distance to Nash Point

Implemented in MultilateralAnalysis.java:313:

The Nash point is also known as the Nash bargaining solution (NOT to be confused with Nash equilibrium), and this is the point which maximises the product of the utilities minus the

disagreement utility. Formally, the Nash bargaining solution or Nash point is calculated by:

$$o_{nash} = \arg\max_{o} \prod_{i=1}^{n} (U_i(o) - U_i(o_{disagr})) \tag{4}$$

where $U_i(o_{disagr})$ is the disagreement utility, i.e. the utility agent i receives if there is no agreement (this is zero if there is the reservation value is zero). Once the Nash point is found, the distance is calculated by the Euclidean distance between the accepted offer and the Nash point.

2 Opponent Modelling

The negotiation strategy from the previous lab is very simple and does not result in very Pareto efficient agreements. A typical approach to improve this, and which is used in many negotiation strategies, is to model the utility function of the opponent based on the bids received, and then use this model when proposing offers. This model gets updated as more bids are received.

In this lab we will use a simple approach from the Johny Black agent, as explained in their paper). The aim of this lab is to become familiar with reading research papers on this topic, so you can investigate other approaches yourself and also think of your own approach. To meet the learning outcomes, for your final agent, you should try and use other approaches from the literature to show that you have read and understood the academic literature and, ideally, use a combination of approaches. Also you are encouraged to come up with your own solutions. The full proceedings are also available here

Read *Section 2.3* of the Johny Black agent paper and try to understand how they implement the opponent model. Then do the following steps:

1. First create a double integer array which keeps track of the frequency of the issue values. That is, for each issue, and each value you want to count how many times you received an offer with that value (see Tables 1 and 2 in the paper as examples). Make sure this works for any number of issues and values.

See the previous lab for how to access information about the domain (how many issues, how many discrete values for each issue).

Note that the frequency table can get pretty large when the domain gets larger. In the final competition there will be at least 1 large domain, so you will need to adjust your implementation to cope with such large domains and not run out of memory or time. So will you need to be a bit more clever about this, but for the purpose of the lab you can just use a “brute-force” approach.

2. Now start populating the frequency table. That is, for each incoming bid, update the frequency table.

To find out more about how the Bid object works, check the Javadoc, i.e. https://secure.ecs.soton.ac.uk/notes/comp6203/genius/doc/genius/core/Bid.html

Note that you need to match the value to what you have in your table. There are various ways to do this, and requires some basic programming skills and

understanding of Java.

3. Check that your approach works by running the agent and displaying the output.
4. Now that you have the table, use the approach from the paper to compute the estimated opponent utility (see Section 2.3 in the paper).
5. Check again that it works. Initially, with no information, each of the opponent bids should have equal utility. As you receive more bids from your opponent, the utility estimation should gradually improve.

3 Improve the Offer Generating Strategy

We are now going to improve the simple offer generating strategy from the previous lab by using our opponent model. The aim is to obtain Pareto efficient offers. How can we achieve this?

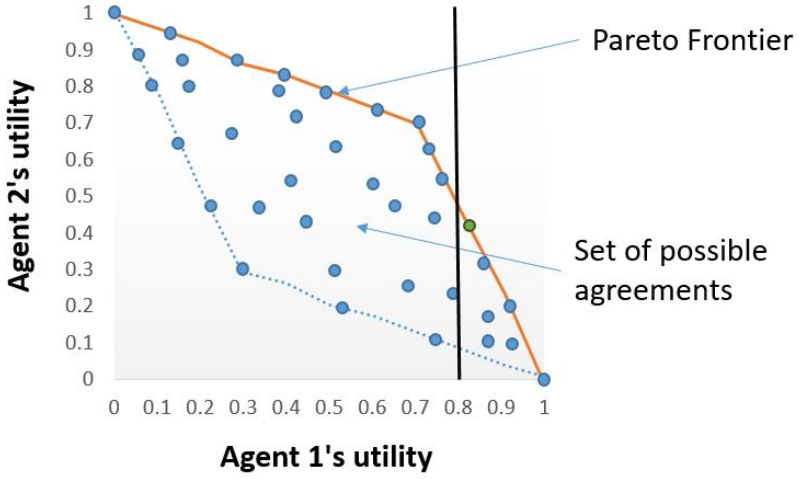

Figure 2: *Finding a Pareto efficient point*

Consider the setting in Figure 2. Suppose that Agent 1 is making an offer, and the current target utility is 0.8. That is, it is willing to generate and accept offers with utility 0.8 or more. This is indicated by the black vertical line in Figure 2. Note that all the offers to the right of the line meet this threshold. Now, in order to find a Pareto efficient offer, all the agent needs to do is to find an offer which meets the threshold *and* has the highest utility for Agent 2. If the opponent model is perfect, this is necessarily a Pareto efficient offer (try and understand why this is the case). In the example, this would be the green point on the frontier. Obviously, in our case, the opponent model is not necessarily correct, and so the best one can hope for is that, by using this approach, the offer still is approximately Pareto efficient.

To implement this approach, there are 2 options:

Option 1: Loop through all the bids which have a utility equal or better than the target utility for the agent, and then choose the one from that set which has highest utility for the opponent according to the opponent model. Ideally, you need to generate a sorted list of offers when initialising the agent. A simpler but less efficient approach is to loop through all offers to see which ones meet the threshold. However, this becomes very inefficient with large outcome spaces.

Option 2: Alternatively, an even simpler approach is using sampling with the same function that selects the random bid above a threshold as in the previous lab. But instead of selecting

only 1 random bid, sample n bids (where n can be set to any size), and from those bids choose the one which the highest opponent utility.

Implement one of the two approaches.

Test your approach, and see if it improves the result.

4 Bonus Task: Nash Bargaining Solution

Recall that the concession strategy sets a minimum threshold to be reached by the deadline. Now, instead of using an arbitrary minimal threshold, calculate the Nash bargaining solution offer using the estimated opponent utility, and set the minimum threshold to the utility you have at the Nash bargaining solution point (for your own agent). Make sure you update this information, as you update the opponent model (i.e. as you receive new offers).

5 Evaluating your Agent

Now run a tournament with your agent against some other agents, and analyse the log results. See how and whether your agent improves, as you improve the strategy.

1. Run the tournament, either using the GUI or the IDE
2. Locate the logs: e.g. `genius/log/tournament-*.(xml|log)`
3. Open `tournament-*.logStats.xml`, you can find the mean results of the metrics that we mentioned.

Try to get familiar on how to plot these results (MS Excel, Numbers, matplotlib, Matlab)

In the previous year students created a python script to parse the log files. The code can be found here: https://gitlab.com/MarsCapone/comp6203-database. If using this code or any other existing code, please acknowledge this in your reports to avoid plagiarism issues.

To create figures, e.g. for your report, it recommended you use either Excel or Python's *matplotlib* library. See below for example code of how to plot a scatter plot.

```
import matplotlib.pyplot as plt
# for instance:
# X = [0.3, ...,]
# Y = [0.6, ...,]
# TODO: get X and Y, plot:
plt.plot(X, Y, "yo") # yo = yellow dot
# TODO: calculate pareto efficient frontier points as X_p and Y_p, plot:
plt.plot(X_p, Y_p, "bs") # bs = blue square
plt.xlabel("Agent 1")
plt.ylabel("Agent 2")
plt.show()
```

END OF LAB 3