COMP6203: Lab #2

Intelligent Agents
University of Southampton

In this lab you will create a simple negotiation strategy in Java using the GENIUS platform.

Prerequisites

Before you start this lab, ensure that:

- You have completed Lab 1
- GENIUS is downloaded
- Java 1.8/JDK 8 is installed
- Either Eclipse or IntelliJ are installed

Contents

1 Theory Crash Course

Before we proceed with the practical component, we have a closer look at the SAOP negotiation protocol. Although in this assignment we will only consider bilateral negotiations, the SAOP protocol has been developed to handle both bilateral and multi-party negotiations.

- **Bilateral**: Only two agents are participating in the negotiation session.
- **Multi-Party**: Usually refers to more than two agents participating in the negotiation session.

SAOP: Stacked Alternating Offers Protocol

According to this protocol [Aydoğan2017], all of the participants get a turn per round; turns are taken in a specified order, and the order remains the same in each round. The first party starts the negotiation with an offer that is observed by all others immediately. Whenever an offer is made the next party in line can take the following actions:

1. Make a counter offer (*thus rejecting and overriding the previous offer*)
2. Accept the offer
3. Walk away (e.g. *ending the negotiation without any agreement*)

This process is repeated in a turn taking clock-wise fashion until reaching an agreement or reaching the deadline. To reach an agreement, all parties should accept the offer. If at the deadline no agreement has been reached, the negotiation fails and ends without an agreement.

Don't forget that most agents aim to maximise their utility during the negotiation rather than walking away and getting nothing (There can sometimes be a benefit to walk away, but only if there are discount factors and reservation values, as explained later).

To illustrate, suppose we have 3 agents: Agent A, Agent B and Agent C.

Round 1:

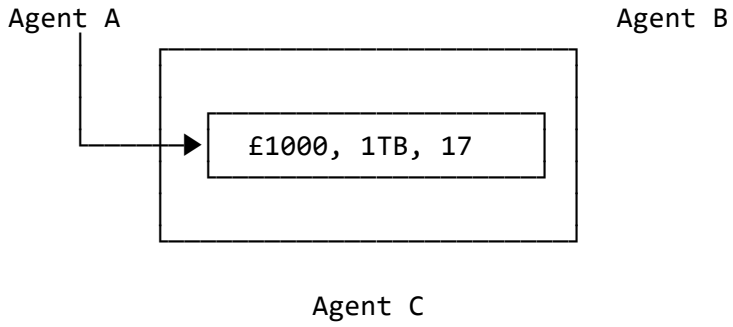


Figure 1: *Agent A makes an offer*

The second party can *accept this offer*, *make a counter offer* or *walk away*. Let's assume that Agent B makes a counter-offer.

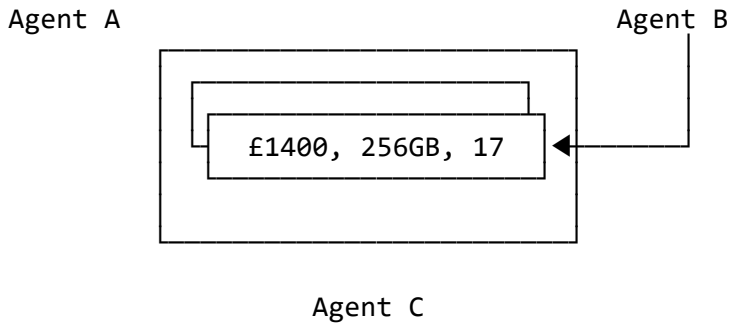


Figure 2: *Agent B makes a counter offer*

Assume that Agent C and Agent A accept the offer on the negotiation table. Since they all agree on this offer, the negotiation ends with this offer.

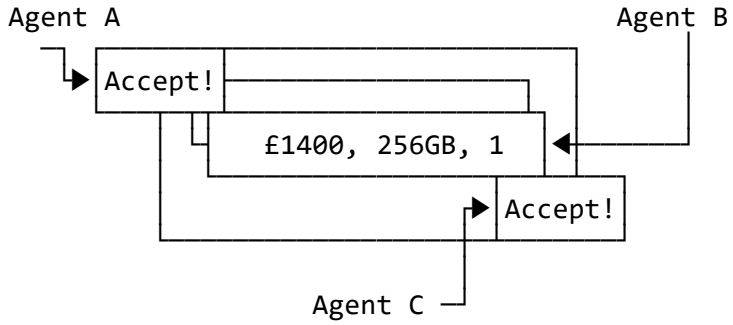


Figure 3: *Agent C and Agent A accept Agent B's offer.*

This protocol has been used in the international negotiation competition and we will also use this protocol in the assignment. For more about this, check GENIUS's User Guide Section 2.1, ExampleAgent Wiki and a question from previous year.

However, it is important to realise that there are many other possible negotiation protocols. Please see the GENIUS user guide for information about other protocols.

Negotiation sessions are parametrized with two additional parameters: *time pressure* (*discount factor*) and *reservation value*. These influence the agents performance throughout the negotiation.

Reservation Value

`Reservation Value` is the utility of *walking away*. In other words, it is a real-valued threshold below which a rational negotiating agent should not accept any offers.

Each agent may have a different reservation value. They are defined within a preference profile.

Time Pressure

Each negotiation session has a predefined deadline, it can be in real time as seconds or it can be in terms of rounds. The simulation environment GENIUS normalizes the time t such that $t \in [0, 1]$.

Apart from a deadline, a negotiation session may also feature a **discount factor** δ, which decreases the utility of the offers under negotiation as time passes. It is bounded in range $\delta \in [0, 1]$. The utility of an offer o at the normalized time t for agent j, $U_j^t(o)$, is calculated as:

$$U_j^t(o) = U_j(o) \cdot \delta^t \tag{1}$$

where $U_j(o)$ is the utility of the offer without the influence of the discount factor.

Reservation values also gets discounted in the same way. Hence, as negotiation proceeds this reduces the value of a disagreement and walking away might be a good strategy if there is no scope for an agreement.

2 Task 1: Running GENIUS in Eclipse or IntelliJ

2.1 Set up the Java environment in Eclipse or IntelliJ

Integrated Development Environment (IDE)

It is highly recommended that you use a Java IDE, such as Eclipse or IntelliJ, for developing your agent. If you are already experienced with Java, then you can use any IDE you are most comfortable with. Otherwise, it is recommended that you use Eclipse since this is the one used in most of the examples.

For **IntelliJ**, follow the instructions in the video tutorial. NOTE: The GENIUS version has been updated since the video was made, and so you will notice some subtle differences.

For **Eclipse** the steps are as follows:

- Create a New Java Project
- Choose the GENIUS working directory
- Make sure the execution environment JRE is version 1.8 (if not you can change this later)
- Press NEXT
- If a `src` directory does not already exist, select `Create new source folder`. Use `src` and select Finish.
- Still in the `New Java Project` window, go to the Libraries tab. Select `Add JARs` (on the right hand side). Go to the genius folder and select `genius-9.x.xx.jar`.
- Press FINISH.

In order to change the Java version in Eclipse, go to `Project -> Properties -> Java compiler`, select `Enable project specific settings`, and set the version to 1.8.

In order to add the genius JAR file to the build path in Eclipse, go to `Project -> Properties -> Java Build Path -> Libraries`

In IntelliJ, project settings can be changed by choosing `File->Project Structure->Project Settings`

2.2 Creating a Simple Agent

Next, follow the steps to create a new agent.

- In your project, create a new package entitled `groupn`, where *n* is replaced with your group number (e.g. `group1` for group 1).
- From the folder `bilateralexamples`, move or copy the file `RandomBidderExample.java` into the previously-created package.
- Go to the RandomBidderExample source code in the editor. Change the package name (first line) to the correct package name (e.g. `group1`).
- Change the class name to `MyAgent` (TIP: In Eclipse, do this by choosing right-click and refactor so it updates any dependencies).
- *(optional)* In the `getDescription()` method, change the description of the agent to something specific to your agent (e.g. "I am the greatest negotiation agent")

In Eclipse, the end result should look something like this (for `group1`):

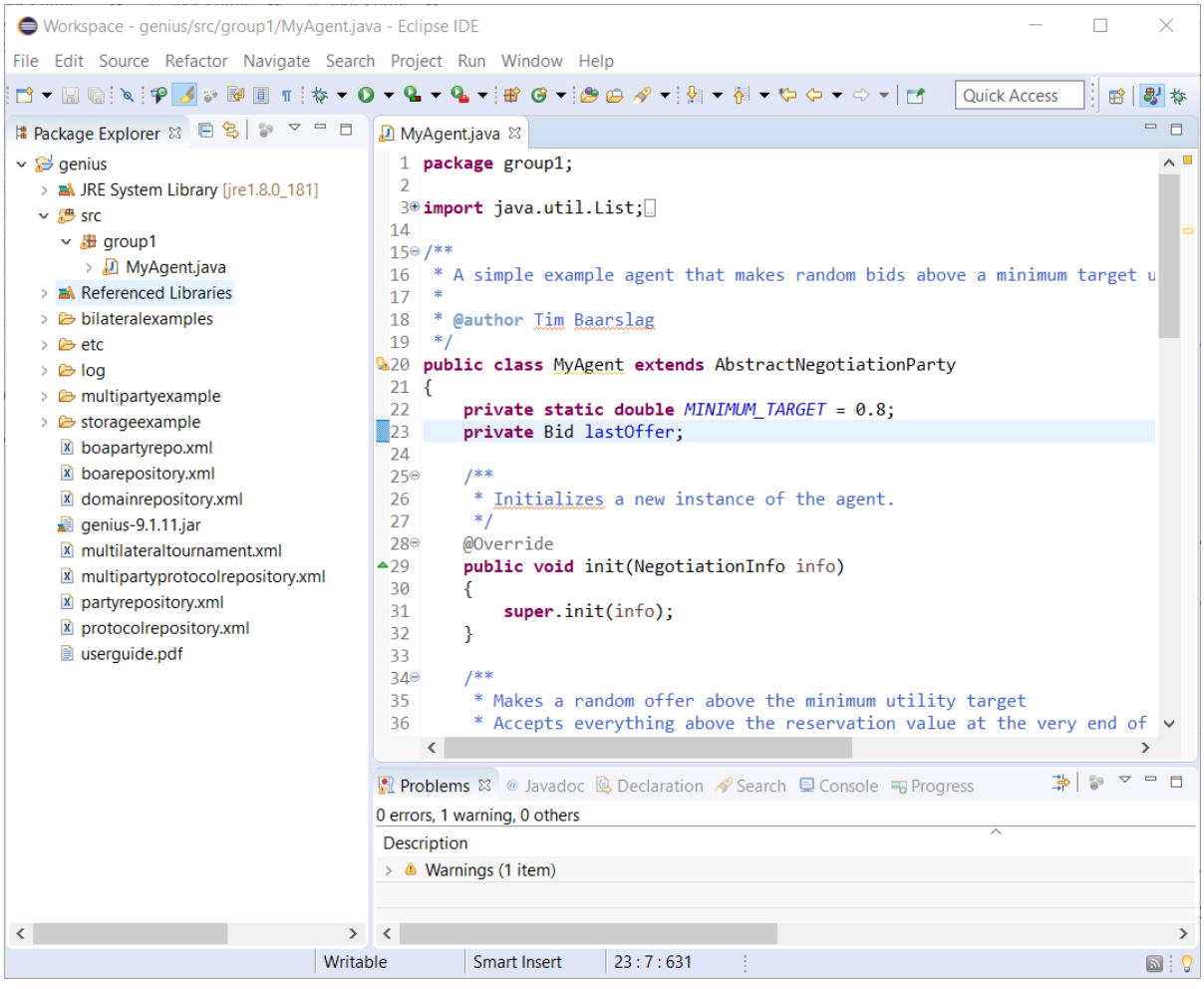


Figure 4: *Eclipse screenshot.*

2.3 Running your Agent using the GENIUS interface

Make sure the code is compiled every time you make a change, e.g. by enabling Build Automatically (recommended) or by building the project manually every time.

First, you need to set up the IDE environment so you can run GENIUS within the IDE (rather than double clicking on the JAR file as we did in the last lab). Otherwise we will not be able to see any console outputs. To do this you need to run the class `genius.Application`. To do this in Eclipse:

- Go to the `Run configurations` of the project (e.g. under the Run menu item, or by right clicking on the project, choosing Run As).
- Right-click on Java Application and choose `New configuration`
- Set the main class to `genius.Application` and set the name to Application. Press `Apply`. It should look like Figure 5.
- Select run to start GENIUS

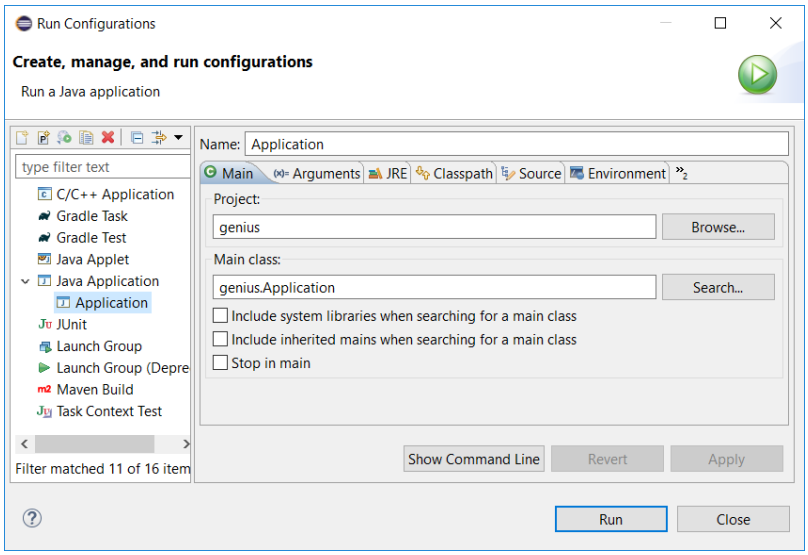


Figure 5: *Creating a run configuration in Eclipse.*

Note that Eclipse will save these configurations, and to run again you can simply press the play button and selecting the appropriate run configuration.

Now we are ready to add the party in GENIUS:

- In the GENIUS interface, select the `Parties` tab.
- Right-click on any party, and select `Add new party`
- Go to the directory where the `.class file is stored` (in Eclipse, this is the `bin` directory; in IntelliJ, this is the `out` directory), go to the `groupn` subdirectory, and select `MyAgent.cass`
- Check the party appears in the list (it will appear at the end)
- Start a new negotiation session and use your agent

If you make changes to your agent you don't need to restart GENIUS. Furthermore, it has now stored the information about your agent in the `partyrepository.xml` file. This means it will remember your agent next time you start GENIUS. Note that, instead of adding an agent using the interface you can also add an agent by directly modifying the xml file. Note that, every time you modify an xml file, GENIUS needs to be restarted for changes to take effect.

3 Modifying the Negotiation Strategy

Javadoc and Additional Resources

The GENIUS Javadoc describes all the public classes, methods and variables. You can link the Javadoc in the IDE to your project, so it shows the explanation when e.g. hovering over a class or using code completion (CTRL+Space).

You can download the GENIUS javadoc zip file from here: https://secure.ecs.soton.ac.uk/notes/comp6203/genius/doc.zip. Unzip the file inside the GENIUS workspace. Then, in Eclipse, you can set the javadoc by going to Project Properties → Javadoc Location.

Alternatively, you can check out the javadoc HTML directly here: GENIUS javadoc.

Furthermore, you can download the entire GENIUS source code from the GENIUS website.

Finally, further resources for help are available here: https://github.com/tdgunes/ExampleAgent/wiki

⚠ **Do not forget to compile your agent after making any changes.**

Otherwise, your agent will not output the preference profile.

3.1 Understanding the UtilitySpace

An object which implements the UtilitySpace interface contains the domain and the agent's preference profile, i.e. the issues, values, weights and evaluations. Note that an agent knows only his own weights and evaluations, and not that of the opponent.

In this task, you are going to explore this object/class. In particular, since we are dealing with additive utilities, we are interested in an implementation of the UtilitySpace called AdditiveUtilitySpace. The following code accesses the domain and displays the issues, etc. Try and understand each line of code, using the Javadoc as a reference for the various methods. You will need to become familiar with the classes and methods to develop your own agent.

- Copy paste the code snippet *inside* `init(NegotiationInfo info)` method of your agent (*after* super.init(info)):

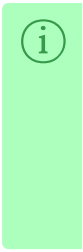
```
AbstractUtilitySpace utilitySpace = info.getUtilitySpace();
AdditiveUtilitySpace additiveUtilitySpace = (AdditiveUtilitySpace) utilitySpace;

List< Issue > issues = additiveUtilitySpace.getDomain().getIssues();

for (Issue issue : issues) {
    int issueNumber = issue.getNumber();
           System.out.println(">>   "   +   issue.getName()   +   "   weight:   "   +
additiveUtilitySpace.getWeight(issueNumber));

    // Assuming that issues are discrete only
    IssueDiscrete issueDiscrete = (IssueDiscrete) issue;
                EvaluatorDiscrete    evaluatorDiscrete    =    (EvaluatorDiscrete)
additiveUtilitySpace.getEvaluator(issueNumber);

    for (ValueDiscrete valueDiscrete : issueDiscrete.getValues()) {
        System.out.println(valueDiscrete.getValue());
                                  System.out.println("Evaluation(getValue):    "    +
evaluatorDiscrete.getValue(valueDiscrete));
        try {
                                System.out.println("Evaluation(getEvaluation):   "   +
evaluatorDiscrete.getEvaluation(valueDiscrete));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



You will get several errors once you have copied the code since you need to import several classes. In Eclipse you can do this easily, by placing the cursor at the end of the class name where the error is, and then pressing CTRL+Space (you can use this for code completion). This will give you several options, and typically choosing the first one. Pressing enter will import the corresponding class.

- Make sure you fix any errors. Run a single negotiation session with your agent and check the console for the output produced.

Try to understand what each line is doing. Add comments to the code above as appropriate.

3.2 A Simple Concession Strategy

We will now modify the negotiation strategy. The negotiation strategy consists of 3 possible actions:

- Accepting the opponent's offer
- Generating a (counter) offer (which automatically means rejecting an opponent offer)
- Ending the negotiation.

To do so there are two main methods in the NegotiationParty interface that need to be implemented:

- `receiveMessage` will receive the offer. This offer will then need to be store for later use.
- `chooseAction` is used for selecting one of the 3 actions above.

Note that, if the agent is the first to make an offer, then the first action should be to generate an offer. Before proceeding any further, have a look at how these two methods are implemented in your simple agent and make sure you understand the given code.

Next, we introduce a simple concession strategy. The concession strategy determines the target utility level at which to produce and/or accept offers.

1. Use a simple linear approach to set the target utility, which starts with the highest possible utility (i.e. the utility of the best possible offer), and then concedes up to a threshold when the time limit is reached. Set this threshold using a variable (e.g. it can be the average between the highest and lowest possible utility within the set of possible offers). Feel free to use a slightly more advanced approach, e.g. Boulware or Conceder strategy (see slides on negotiation lecture and/or literature). Some code to help you is provided below:

Getting the best bid (note that you need to catch the exception, which is thrown when there are no bids):

```
private Bid getMaxUtilityBid() {
    try {
        return utilitySpace.getMaxUtilityBid();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

Accessing normalized time $t \in [0, 1]$:

```
double time = getTimeLine().getTime();
```

2. Select a random offer which has a minimum utility threshold using the code already provided (we will improve this strategy later).

3. Offer that bid to your opponent and accept any offers received by the opponent which have a utility equal or greater than the target utility.

4. Test your solution.

Note that using the random offer approach is not very efficient. Think about how this could be improved (no need to implement it yet).

3.3 Running Tournaments

Running individual negotiation sessions for testing and evaluating your agent can be a tedious task. Instead, it is possible to run multiple negotiations in one go, and this can even be done directly from the command line without using the GENIUS GUI. In GENIUS, a *tournament* consists of multiple negotiations, involving multiple agents and multiple profiles. GENIUS will automatically generate all possible configurations and you can even repeat tournaments multiple times, all with 1 click of a button. For example, suppose you want to compare 3

different strategies and 2 different profiles, then this will generate 3*3=9 individual negotiation sessions.

To see how this works, first run a tournament using the GENIUS interface by selecting Start → Tournament. Select individual agents, e.g. Conceder, Bouldware and your agent. Furthermore, select at least 2 profiles from e.g. the Party domain. Run the tournament and see what happens.

Note that, when running a tournament, any System output will be supressed and no longer shown, and there is no record of the individual bids exchanged either (unlike in the case of single negotiation sessions). This is to prevent large files. If you nevertheless want to create your own separate log even for tournaments, this can be done by e.g. saving any information to a separate file.

Now you are going to run a tournament without using the GUI and directly in Eclipse.

- Edit the file `multilateraltournament.xml` using a text editor and change the parties to Bouldware, Conceder and your agent. To find the classpath for the Boulware and Conceder, look into the file `partyrepository.xml`. The class path for your agent will be `groupn.MyAgent` (replacing n with your group number).
- Add a new Run Configuration, this time with the Main class being `genius.cli.Runner`.
- In the Arguments tab of this run configuration, include the following 2 Program Arguments: `multilateraltournament.xml log/log1`. This means it will use the tournament setting from the file `multilateraltournament.xml` and will save the log to the file `log1.csv` in the `log` directory.
- Press Run (making sure you run the new configuration and not the previous one)
- Check it worked and analyse the logs.

⚠ If you have not already done so, make sure you read the coursework assignment in detail (find the link on the course website) before the next lab. Also, it is recommended you have a look at the GENIUS user guide which contains much of the information covered in the past labs.

You are now ready to start creating your own negotiation strategy.

END OF LAB 2

4 Bibliography

[**Aydoğan2017**] Aydoğan R., Festen D., Hindriks K.V., Jonker C.M. (2017) Alternating Offers Protocols for Multilateral Negotiation. In: Fujita K. et al. (eds) Modern Approaches to Agent-based Complex Automated Negotiation. Studies in Computational Intelligence, vol 674. Springer, Cham https://link.springer.com/chapter/10.1007/978-3-319-51563-2_10#citeas

Produced using Markdeep. Copyright University of Southampton