COMP6203: Lab Index

Intelligent Agents
2018 Fall: Semester 1
University of Southampton

1 Lab Sessions

Labs will be held at **16:00-18:00** in **Building 59/ECS Teaching Lab (Level 2)** on Wednesdays.

> If you have questions that you want to ask outside the lab to demonstrators, first:
>
> - Check answered questions from previous years.
>
> If you still couldn't find any answers to your questions:
>
> - Create an issue from this link.

You can download all lab documents combined as a PDF document from here.

October 2018

Monday	Tuesday	Wednesday	Thursday	Friday
1	2	3	4	5
8	9	10	11	12
15	16	**17** Lab #1: Introduction	18	19
22	23	**24** Lab #2: Bids/Issues/Values/Profiles	25	26
29	30	**31** Lab #3: Protocols	1	2

November 2018

Monday	Tuesday	Wednesday	Thursday	Friday
29	30	31	1	2
5	6	**7** Lab #4: Implementing Caduceus	8	9
12	13	**14** Lab #5: Quality Measures	15	16
19	20	**21** Lab #6: Bidding Strategies/Opponent Modeling/Acceptance Strategies	22	23
26	27	**28** Lab #7: Q&A Session	29	30

December 2018

Monday	Tuesday	Wednesday	Thursday	Friday
26	27	28	29	30
3	4	**5** Lab #8: Q&A Session	6	7
10	11	12	13	14
17	18	19	20	21
24	25	26	27	28
31	1	2	3	4

Wednesday 17 Oct 2018	**Lab #1: Introduction** • This lab aims to give an introduction to Automated Negotiation and the simulation environment for negotiation, GENIUS
Wednesday 24 Oct 2018	**Lab #2: Bids/Issues/Values/Profiles** • This lab introduces the concepts: bids, issues, preference profiles and utility spaces in detail.
Wednesday 31 Oct 2018	**Lab #3: Protocols** • This lab introduces the protocols in automated negotiation. The protocols determine the overall order of actions during a negotiation. Parties need to stick to this protocol as deviations from the protocol are caught and penalized.
Wednesday 7 Nov 2018	**Lab #4: Implementing Caduceus** • This lab shows how to implement a previous winner (ANAC2016) of the international automated negotiation competition.
Wednesday 14 Nov 2018	**Lab #5: Quality Measures** • This lab shows how to evaluate your agent against various benchmark metrics.
Wednesday 21 Nov 2018	**Lab #6: Bidding Strategies/Opponent Modeling/Acceptance Strategies** • This lab shows some key components of a negotiation agent and how to start improving them.
Wednesday 28 Nov 2018	**Lab #7: Q&A Session**
Wednesday 5 Dec 2018	**Lab #8: Q&A Session**

COMP6203: Lab #1

Intelligent Agents
Wednesday, 17 October 2018
University of Southampton
Go to Lab Index

This lab aims to give an introduction to *Automated Negotiation* and the simulation environment for negotiation, *GENIUS*. At the end of this lab, you will have a brief idea about the area and keywords such as: `preference profile`, `domain`, `negotiation protocol` and so on.

1 Designing a Negotiation Party

Figure 1: *Organizing a party involves choosing music, food, drinks, catering, etc.*

1. Imagine you have just graduated and plan to throw a nice party for that occasion.
2. The problem is that you do not want to organize everything yourself.
3. Fortunately, a friend of yours has also graduated and wants to throw a party as well.
4. You decide to organize the party together, however you have different preferences. You will now have to make joint decisions about what the party will look like; that is, you will have to *negotiate*.

Being a computer science student, you want to use your skills to make a negotiating party so the party is as close to *your preferences* as possible.

To approach this *negotiation problem*, to be a good negotiator, you may consider:

- Orienting yourself towards a win-win approach.
- Planing and having a concrete strategy.
- Determining which offers you will never accept.
- Creating options for mutual gain.
- Taking the preferences of your opponent into account
- Generating a variety of possibilities before deciding what to do.
- Paying a lot of attention to the flow of negotiation.

1.1 Short Introduction

To approach this problem from a computer science perspective, in the field of *Automated*

Negotiation, this problem is modelled in various ways. To give a brief introduction to these models, assume that there are:

- **A set of agents**: the participants of negotiation.
  - e.g. *negotiating parties*, *humans* or *software agents*
- **A domain**: the context the agents are negotiating in.
  - e.g. *example above: organizing parties*, *negotiating a business deal*, *smart grids*, *negotiating the electric prices*
- **An issue**: a troubling point that requires negotiation.
  - e.g. if domain is organizing party: what food to buy
- **Preference profiles**: a set of profiles, that captures individuals preferences.
  - e.g. If the domain is *party*, assume that Agent A prefers *Pizza* over *Hamburger* for the **Food** issue. Agent B prefers *Hamburger* over *Pizza*.
- **A deadline**: when the negotiation halts.
  - e.g. 1 hour, 2000 rounds (will be explained later)
- **A protocol**: how the negotiation is going to take place.
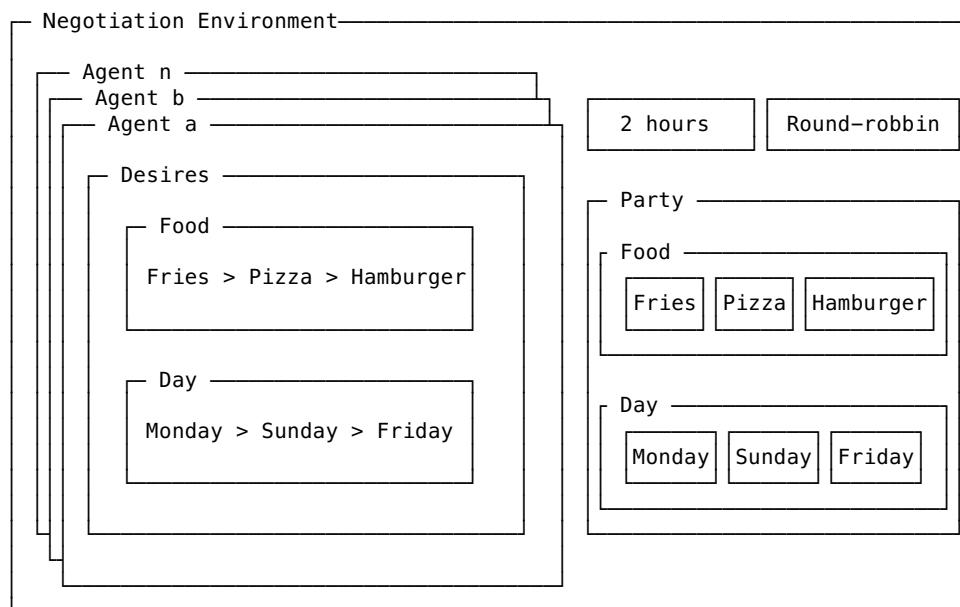 - e.g. Round-robin fashion, a mediator based

Figure 2: *A simple negotiation environment*

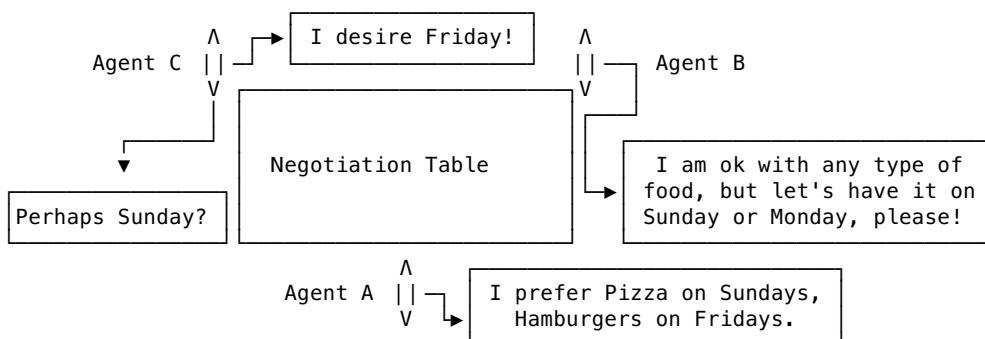> **Practice**: Try to match the terms in Figure 2 with definitions.

Figure 3: *A negotiation session between three agents*

> **Practice**: What will be the best possible scenario that all agents might agree on in Figure 3?

2 Task 1: Set up GENIUS Environment

Throughout this module, you are expected to implement *a negotiating agent*. You are expected to use Java programming language in the automated negotiation simulation platform GENIUS.

2.1 Prerequisites

> ⚠ Make sure that you have downloaded and installed the exact versions of prerequisites. Newer Java versions known to have some issues with GENIUS 9.1.2.

2.1.1 Software Prerequisites

- Java Development Kit 8
- GENIUS 9.1.2

2.1.2 Resources

- Java Cheatsheet *Remember/learn the basics of Java programming language.*
- GENIUS Manual *Official user guide, detailed explanations about the simulation platform*
- *Unofficial* GENIUS 9.1.1 Javadoc *Shows Java API of GENIUS, while you are implementing your agent, you can have a look at the method definitions when necessary here.*

2.2 Installation

There are three different options to follow to set up the simulation environment. If you want to set the environment from scratch follow Option 1. We recommend Intellij Java IDE for implementation. But you can also use Eclipse or Netbeans, if you are more experienced with those.

2.2.1 Option 1

You can watch the video below which explains how to set it up from scratch with using Intellij.

> ⚠ **Things to look out during the video!**
>
> Since the version of the GENIUS is changed, there might be some steps that are slightly different.
>
> - Scenario from the GUI is removed.
> - Use only Tournament (for multiple sessions) and Negotiation (for a single session) options in the GUI.

Figure 4: *How to set up Genius from scratch*

Index of the video:

- 0:00 Introduction & Checking Java Installation
- 0:49 Java IDE Installation
- 1:05 Downloading Genius (Simulation Environment)
- 1:35 Creating a project and compiling ExampleAgent
- 3:30 Running Genius GUI
- 4:20 Adding ExampleAgent to Genius
- 4:35 Running a single negotiation session
- 5:19 Running a tournament (a set of negotiation sessions)
- 5:55 Checking logs of a tournament
- 6:32 Setting up automatic compilation for Intellij

2.2.2 Option 2

You can simply clone (or download as a zip archive) this repository (latest version of Genius included) with version control system `git` by:

```
git clone https://github.com/tdgunes/ExampleAgent.git
```

and import the ExampleAgent folder as a project to Intellij (other major Java IDEs also should work).

3 Task 2: Run GENIUS with ExampleAgent

Follow the steps on the video given above and run ExampleAgent in GENIUS with some opponent agents. You need to validate that your development environment is ready.

COMP6203: Lab #2

Intelligent Agents
Wednesday, 24 October 2018
University of Southampton
Go to Lab Index

This lab introduces the concepts: bids, issues, preference profiles and utility spaces in detail.

1 Theory Crash Course

This section provides the essential theory to understand a negotiation process in GENIUS platform. In the next section, API definitions will be provided.

1.1 Domain: Bids, Issues, Values

A `domain` D specification determines what type of offers that can be made by the agents during negotiation. For instance, agents are to negotiate on what type of laptop to purchase, a domain that describes this negotiation can be:
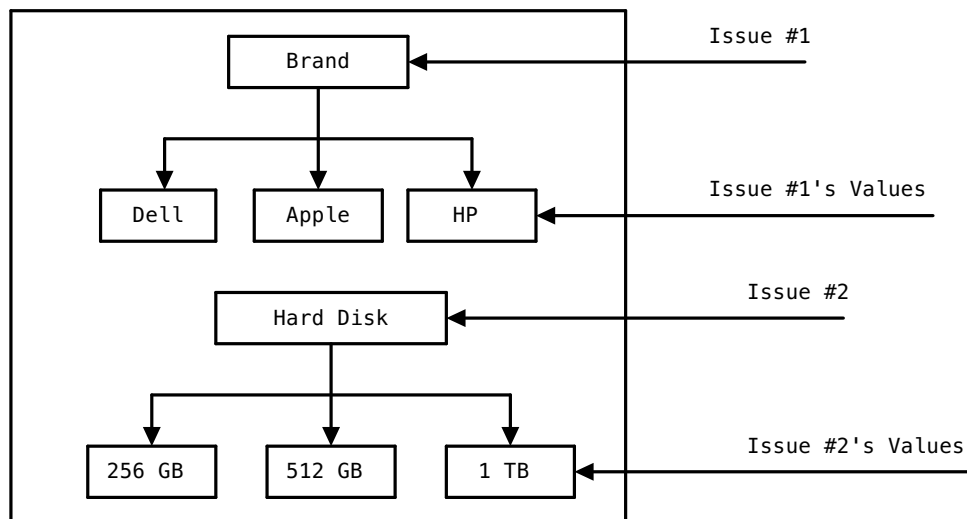

Figure 1: *An example domain for laptop negotiation*

A domain contains *n* issues:

$$D = \{I^1, \ldots, I^n\}$$

Each issue *i* consists of *k* values:

$$\forall I^i \in D \quad I^i = \{v_1^i, v_2^i, \ldots, v_k^i\}_{k=|I^i|}$$

An offer from an agent a that can be formulated as choices c:

$$b_a = \{v_j^i \mid v_j^i \in I^i, 1 \leq i \leq n\}$$

Example: With this notation, to define Figure 1 formally with additional issue *Monitor*:

$$D_{laptop} = (Brand, HardDisk, Monitor)I_{Brand} = (Dell, Apple, HP)I_{HardDisk} = (256GB, 512GB, 1TB)I_{Monitor} = (13, 15, 17)$$

The agent a can generate a bid b as:

$$b_a = (Brand: Dell, \ HardDisk: 1TB, \ Monitor: 13")$$

1.2 Preference Profiles, Utility Spaces

Following the example of the bid, let's assume that there are only two agents: Agent a
and Agent b
negotiating about *which laptop to buy*.

1. Agent a
   makes an offer: b_a = (Brand: Dell, HardDisk: 1TB, Monitor: 13")
2. When Agent b
   observes the offer, the question is: how preferable is the bid b_a
   for Agent b
   ?

Each agent has their own preferences for each issue. In this example, for instance:

- Agent a
  prefers **a more portable laptop with high storage**.
- Agent b
  's only requirement is **a larger screen** in a laptop.

We say each agent has a *preference profile* that represents these preferences. Each participant agent has their own profile. There are many types of different profiles [1], the two we are going to focus are:

- **Additive Utility Space**: The space is additive, when a bid is evaluated, this space can generate a total utility.
- **Partially Ordered Space**[2]: Instead of assigning utility value to bids, this space stores a partial order of values. By partial ordering, the available information about the preferences of an agent is that for some bid X
  is preferred over bid Y
  for a subset of the possible bids.

1.2.1 Additive Utility Space

Formally the utility of a bid is calculated as:

$$U(b_a) = \sum_{i=1}^{n} \omega_i \frac{eval(v_j^i)}{max(eval(I_i))}$$

> ⚠ **Sum of all weights is 1.0**
> .
>
> Formally it is: $\sum_{i=1}^{n} \omega_i = 1.0$

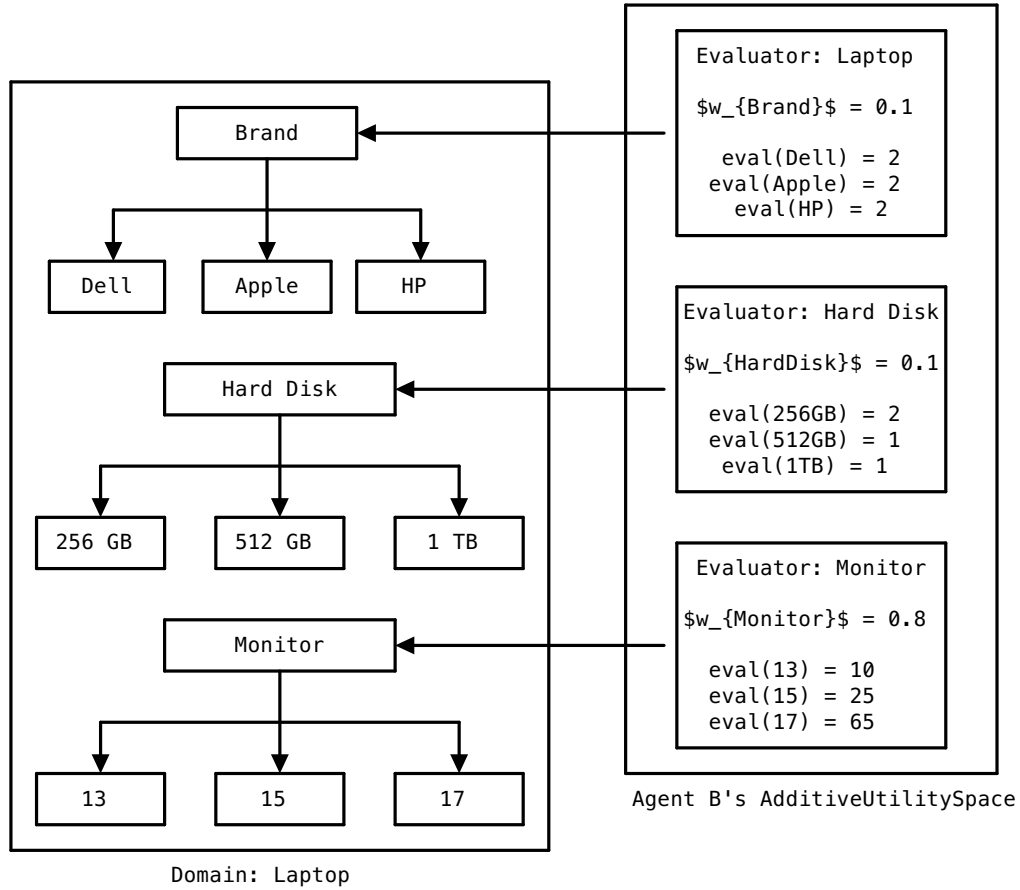Let's again assume, our agent is working in `Laptop` domain (as defined in Equation 4
).

```
                                              ┌─────────────────────────┐
                                              │  Evaluator: Laptop      │
                                              │                         │
                                              │  $w_{Brand}$ = 0.1      │
   ┌──────────────────────────────────┐       │                         │
   │        ┌────────────┐             │◄──────│   eval(Dell)  = 2       │
   │        │   Brand    │             │       │   eval(Apple) = 2       │
   │        └────────────┘             │       │     eval(HP)  = 2       │
   │                                   │       └─────────────────────────┘
   │  ┌──────┐  ┌───────┐  ┌──────┐    │
   │  │ Dell │  │ Apple │  │  HP  │    │       ┌─────────────────────────┐
   │  └──────┘  └───────┘  └──────┘    │       │  Evaluator: Hard Disk   │
   │                                   │       │                         │
   │        ┌────────────┐             │       │  $w_{HardDisk}$ = 0.1   │
   │        │ Hard Disk  │             │◄──────│                         │
   │        └────────────┘             │       │   eval(256GB) = 2       │
   │                                   │       │   eval(512GB) = 1       │
   │  ┌───────┐ ┌───────┐ ┌──────┐     │       │    eval(1TB)  = 1       │
   │  │256 GB │ │512 GB │ │ 1 TB │     │       └─────────────────────────┘
   │  └───────┘ └───────┘ └──────┘     │
   │                                   │       ┌─────────────────────────┐
   │        ┌────────────┐             │       │  Evaluator: Monitor     │
   │        │  Monitor   │             │◄──────│                         │
   │        └────────────┘             │       │  $w_{Monitor}$ = 0.8    │
   │                                   │       │                         │
   │  ┌──────┐  ┌──────┐  ┌──────┐     │       │    eval(13) = 10        │
   │  │  13  │  │  15  │  │  17  │     │       │    eval(15) = 25        │
   │  └──────┘  └──────┘  └──────┘     │       │    eval(17) = 65        │
   └──────────────────────────────────┘       └─────────────────────────┘
            Domain: Laptop                     Agent B's AdditiveUtilitySpace
```

Figure 2: *Preference Profile of Agent b*
(instantiated with AdditiveUtilitySpace)

> **Practice[3]**: Calculate the utility of b_a = (Brand: Dell, HardDisk: 1TB, Monitor: 13")
> for Agent b.

1.2.2 Partially Ordered Space

We expect there will three parameters of such space that will control the generation of bids:

- **comparisons**: a subset of bids that are selected randomly from all possible bids that are assumed to be visible.
- *errors*: *the number of errors in the orders*

> ⚠ **errors** parameter is not used for now, assume that it is always zero.

With regards to our previous example, if the comparisons equal to the number of possible bids (in D_{laptop}
, the number of bids are: $\prod_i^k |I_i|$
, which is 27
), the negotiating agent can observe all the possible bids:

$$B_a^{ordered} = (b_a^1, b_a^2, b_a^3, \ldots, b_a^{27})$$

B_a is sorted by the utility of the each bid starts, from lowest to highest, can be such as:

$$b_a^1 < b_a^6 < b_a^7 \leq \ldots \leq b_a^{21}$$

The list may include bids with same utilities consecutively. Whenever parameter **comparisons** is set to some value, assume that it is 4

. Four bids will be randomly sampled from B_a
. Then, the agent can only observe, for instance:

$$randomly\_sample(B_a^{ordered}, 4) = (b_a^{21}, b_a^{10}, b_a^4, b_a^5)$$

With negotiation sessions that have partially ordered space preference profiels, the negotiating agent needs to reason and predict about the other unknown parts of the space. Also, since the utility of each bid is unknown too, the negotiating agent may need to predict the utilities of the other bids the order alone.

> Genius 9.1.2 includes `UncertaintyAgentExample.java`, which demonstrates the API to access this preference profile.

[1] Genius User Guide also mentions: `ConstraintUtilitySpace` and `NonlinearUtilitySpace`.

[2] It is expected that ANAC 2019 (Automated Negotiation Competition) is going to include partially ordered preference profiles.

2 Task 1: The laptop domain and preferences in GENIUS

At this point, if you are expected to be familiar with the theorical concepts. To not get confused with the implementation, make sure that you understand the topics above.

- For this task, create the same domain as shown in Figure 2 in GENIUS.

> ⚠ There is already a laptop domain included in GENIUS. Use **lab2_laptop** as the name for the new domain.
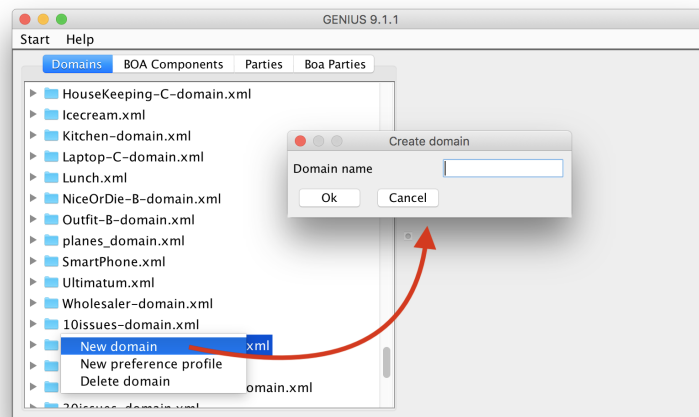
Figure 3: *Right click to the left bar*

2.1 Add the preference profiles to the domain

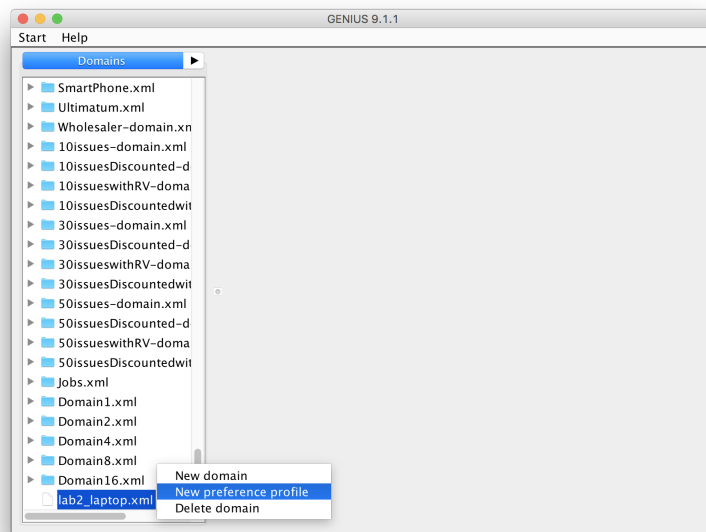After saving the domain with issues create a preference profile for Agent B and an arbitrary profile for Agent A.

Figure 4: *Right click to domain's name*

2.1.1 Creating an uncertain preference profiles

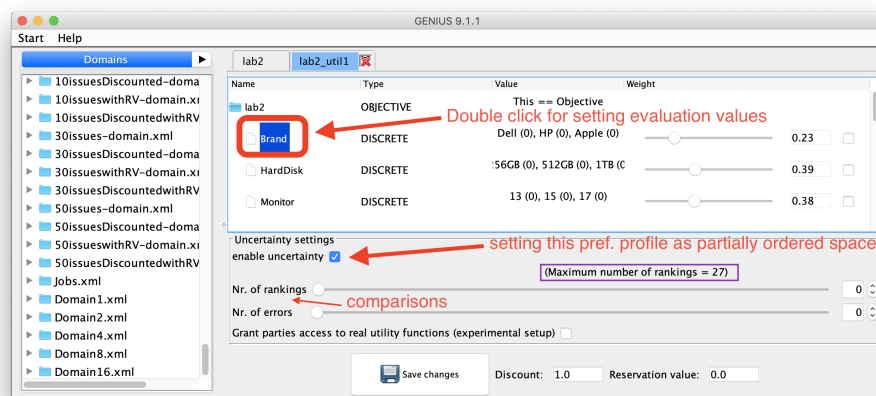You can set the preference profile as uncertain and set *comparisons* metrics:

Figure 5: *Setting uncertain preference profile*

> **Practice:** Why do you think that *maximum number of rankings* (as shown in purple box in Figure 5) is set to 27?

2.2 Double check the created XML

When you create the preference profiles and the domain, have a look at the created XML files. They are by default located at:

```
{PROJECT_WORKING_DIRECTORY}/genius{version}/etc/templates/lab2_laptop
```

> ⚠ **If you are having trouble finding this file:**
>
> Set your working directory where the genius.jar is located. Some links for you, if you are confused:
>
> - What is a working directory in IDE?
> - How to change the working directory of Intellij run/debug configurations?

Check whether `lab2_laptop_util1.xml` is similar as below:

```xml
<utility_space>
    <objective name="lab2_laptop" index="0" description="" etype="objective"
type="objective">
        <issue vtype="discrete" name="Brands" index="1" etype="discrete"
type="discrete">
            <item evaluation="2.0" index="1" value="Dell"></item>
            <item evaluation="2.0" index="2" value="Apple"></item>
            <item evaluation="2.0" index="3" value="HP"></item>
        </issue>
        <issue vtype="discrete" name="Monitor" index="2" etype="discrete"
type="discrete">
            <item evaluation="10.0" index="1" value="13"></item>
            <item evaluation="20.0" index="2" value="15"></item>
            <item evaluation="65.0" index="3" value="17"></item>
        </issue>
        <issue vtype="discrete" name="Hard Drive" index="3" etype="discrete"
type="discrete">
            <item evaluation="2.0" index="1" value="256GB"></item>
            <item evaluation="1.0" index="2" value="512GB"></item>
            <item evaluation="1.0" index="3" value="1TB"></item>
        </issue>
        <weight index="1" value="0.1"></weight>
        <weight index="2" value="0.1"></weight>
        <weight index="3" value="0.8"></weight>
    </objective>
    <discount_factor value="1.0"></discount_factor>
    <reservation value="0.0"></reservation>
</utility_space>
```

Figure 6: *Contents of* `lab2_laptop_util1.xml`

3 Task 2: Accessing the preference profile via API

In this task, you are going to programmatically access the preference profiles. First, you are going to make your agent access to an AdditiveUtilitySpace preference profile that you created earlier.

3.1 Subtask 1: AdditiveUtilitySpace Exercise

Before doing this exercise, it is very important to read below.

> ⚠ **Make sure that uncertainty option of the preference profiles in `lab2_laptop` is disabled.**
>
> Otherwise, you might get `NullPointerException` from GENIUS platform.

> ⚠ **Do not forget to compile your agent after making any changes.**
>
> Otherwise, your agent will not output the preference profile.

> ⚠ **If you see a tiny window when you run GENIUS:**
>
> Have a look at this solution.

> ⊠ **Make sure that your development environment is ready at this point.**
>
> A brief checklist:
>
> - Make sure you have these files:
>   - **ProjectName/**
>   - **genius{version}/**

- etc/
- log/
- some other genius files
- genius{version}.jar
  - **out/**
    - NameOfYourAgent.class
  - **src/**
    - NameOfYourAgent.java
- Make sure that your project has these following settings (to change these `File->Project Structure->Project Settings`):
  - Project language level is set to **8**.
  - Project SDK is set to **Java 1.8**.
  - `genius{version}.jar` is included in Libraries tab.
- Make sure that working directory of run configuration of `genius{version}.jar` is set to `../genius{version}/` folder.
  - You can again have a look at the video from the previous lab.

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

```java
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();
        }
    }
}
```

- Run a single negotiation session and validate if your agent can access the preference profile or not.

> **Try to understand what each line is doing. Add comments to the code above.**
>
> Have a look at the GENIUS javadoc.

3.2 Subtask 2: Partially Ordered Space Exercise

This time, in this space only the preference order of some bids are observed. To access the ranking list of these bids, we use `userModel field` of `AbstractNegotiationParty class`:

> ⚠ **Make sure that uncertainty option of the preference profiles in `lab2_laptop` is now enabled.**

> Otherwise, you might get `NullPointerException` from GENIUS platform.

> ⚠ **Make sure that you remove/comment the code that you inserted from the previous task.**
>
> Otherwise, you are going to get some errors, because the API to access the partially ordered space is different in GENIUS.

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

Also try to access the properties of a bid, using `Bid class`:

```
List< Issue > issuesList = bid.getIssues();

for (Issue issue : issuesList) {
    System.out.println(issue.getName() + ": " + bid.getValue(issue.getNumber()));
}
```

- Run a single negotiation session with partially ordered spaces for laptop domain that you created earlier and check if you can access them by using the code snippets from this section.

[3] *~0.27 si rewsna eht*

COMP6203: Lab #3

Intelligent Agents
Wednesday, 31 October 2018
University of Southampton
Go to Lab Index

This lab introduces the protocols in automated negotiation. The protocols determine the overall order of actions during a negotiation. Parties need to stick to this protocol as deviations from the protocol are caught and penalized.

Contents

1 Theory Crash Course

This section provides the essential theory to understand how agents communicate with each other to handle negotiations. First some definitions:

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

The examples we give on the protocols defined below are going to be in multilateral settings. However, bilateral settings are also possible with these protocols.

1.1 Protocols without a Mediator

The protocols that are mentioned in this section, do not have a third-party agent that handles communication between negotiatiors.

1.1.1 **SAOP: Stacked Alternating Offers

Protocol**

According to this protocol [Aydoğan2017], all of the participants around the table get a turn per round; turns are taken clock-wise around the table. 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.

⚠ Don't forget that most agents aim to maximise their utility during the negotiation

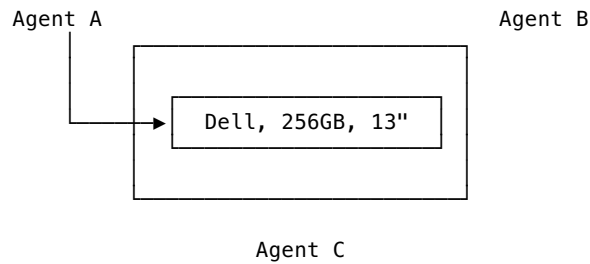To illustrate:

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.
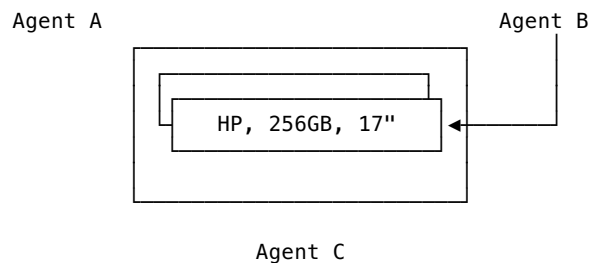

Figure 2: *Agent B makes a counter offer*

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.
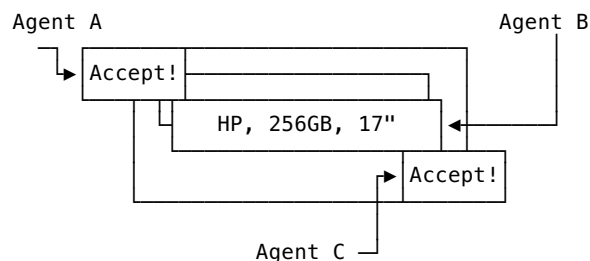

Figure 3: *Agent C and Agent A accept Agent B's offer.*

This protocol is used in the international negotiation competition multiple times. For more about this, check GENIUS's User Guide Section 2.1, ExampleAgent Wiki and a question from previous year.

The next protocols are good to know, but our main focus will be on SAOP.

1.1.2 Alternating Multiple Offers Protocol

There are two phases that consecutively happen in negotiation sessions with this protocol:

- **Bidding Phase** *(every even round (round 0, round 2, ...))*: Each agent submits a bid to the table (i.e. negotiation system).
- **Voting Phase:** *(every odd round (round 1, round 3, ...))* Each agent observes every bid from every agent. Each bid is voted by each agent. Their vote can be accept or reject.

Before bidding phase starts again, if a bid gets accept votes from all parties, the negotiation ends

with this bid.

1.1.3 Alternating Majority Consensus Protocol

This protocol is essentialy same as *Alternating Multiple Offers Protocol*, the main difference is that the protocol keeps track of **the acceptance of offers**. If the negotiation ends without any acceptance of an offer, the offer that got most accepts is used for agreement. As in previous protocol when an offer is accepted before the deadline by everybody, the negotiation ends.

> **For example**, assume that negotiation ends because of the deadline and $(Dell, 256GB, 13")$ is offered and accepted by a bigger majority than other offers, then everybody has to take this offer because of the protocol.

> ⚠ **What will happen if there is a tie in Alternating Multiple Offers Protocol and in Alternating Majority Consensus Protocol?**
>
> - `AlternatingMultipleOffersProtocol`: An agreement is only reached if everybody accepts the bid, otherwise negotiation continues until two conditions: until deadline or an agreement is reached. In this case, there is not a case that there can be a tie.
> - `AlternatingMajorityConsensusProtocol`: From javadoc: "The offer that is supported by the most parties will stay on the table. If a new offer has more supporting parties it overwrites the old offer.". In short, the first offer that has the majority's vote is selected and stored. During the negotiation, if any offer gets a higher vote, that offer gets to be stored.

1.2 Protocols with a Mediator:

The protocols that are mentioned in this section, have a third-party agent that handles communication between negotiatiors. This *mediator* agent hears all of the bids from all parties and the mediator determines which bid to be voted by others. The mediator determines the outcome by observing the votes from other negotiating agents.

1.2.1 Simple Mediator Based Protocol

In this protocol:

- The mediator creates a bid in each round.
- Other negotiating parties, either accept or reject as before. But they can not provide a counter-offer.
- The mediator informs all parties about the outcome of the round.

This continues until the deadline is reached or a bid from mediator agent is accepted by every negotiating agent.

There are several implemented mediators in GENIUS:

- `RandomFlippingMediator`: generates random bids by flipping one issue of the current offer.
- `FixedOrderFlippingMediator`: same as `RandomFlippingMediator` with a fixed-seed random generator.

1.2.2 Mediator Feedback Based Protocol

This protocol extends *Simple Mediator Based Protocol* by changing votes to feedbacks. Instead of negotiating agents provide a binary vote (i.e. [Reject, Accept]), with this protocol they can

provide a feedback (i.e. ['better', 'worse', 'same']) to the mediator.

The mediator generates a different bid in each round as same as before, trying to generate towards to some optimum. The accepted bid is considered as the one that did not get a 'worse' vote from parties.

2 Task 1: Practice SAOP with Paper!

Have a group with three people from the lab and try to negotiate about which laptop to buy according to this protocol.

1. Make everybody in the group to have **a piece of paper** and **a pen**.
2. Everybody needs to write down their preferences as in Additive Utility Space preference profiles in **laptop domain** from the previous lab. Set some weights and evaluation values of your preferences.
3. **Start negotiating according to SAOP for only 5 rounds**.
4. Everybody calculates their own utilities gained at the end of negotiation, write it down.
5. Shuffle the preference profile papers and make sure that everybody gets a unique one.
6. Do step 3 again until everybody used every possible preference profile.
7. Average your negotiating performance. Decide who is the winner.
8. Discuss with your peers about the winning strategy.

3 Task 2: Implement a simple agent for SOAP protocol

3.1 Subtask 1: Half Best & Half Random

Try to write an agent that gives the best offer for itself for the first half of the negotiation time and random offers for remaining time.

Use the snippet from below as a starting point.

```java
import java.util.List;

import genius.core.AgentID;
import genius.core.Bid;
import genius.core.actions.Accept;
import genius.core.actions.Action;
import genius.core.actions.Offer;
import genius.core.parties.AbstractNegotiationParty;
import genius.core.parties.NegotiationInfo;

public class YourAgentsName extends AbstractNegotiationParty {
    private final String description = "YourAgentsName";

    @Override
    public void init(NegotiationInfo info) {
        super.init(info);

    }

    @Override
    public Action chooseAction(List< Class< ? extends Action > > list) {

    }

    @Override
    public void receiveMessage(AgentID sender, Action act) {
        super.receiveMessage(sender, act);
    }

    @Override
    public String getDescription() {
        return description;
    }
}
```

Getting the best bid:

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

Creating random bids:

```java
Bid randomBid = this.generateRandomBid();
```

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

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

3.2 Bonus Subtask: Count the counter-offers and accept messages

Try to count the acceptance and counter-offers from the other agent(s). You may think about storing them in a Map.

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

COMP6203: Lab #4

Intelligent Agents
Wednesday, 7 November 2018
University of Southampton
Go to Lab Index

This lab shows how to implement a previous winner (ANAC2016) of the international automated negotiation competition.

Contents

1 Theory Crash Course

To get you more familiar with how to implement a more sophisticated agent, we introduce `Caduceus` agent, which got the first in 2016 and a slightly different version got the second in 2017.

To learn theory, have a look at:

- The conference paper [Gunes2017]
- The poster
- The presentation

of the Caduceus.

In short, Caduceus incorporates the strategies of multiple agents. We call it experts. These experts can be agents that we know that they are good in negotiation. Metaphorically, it is as if you ask help from multiple friends and try to combine their opinions.

This type of combined approaches exist in:

- Machine Learning
  - Ensemble Learning
  - Algorithm portfolios
- Artificial Intelligence
  - Genetic Algorithms (*slightly similar*)
  - Mixture of Experts

1.1 Caduceus IPO - *Incremental Portfolio Approach*

Essentially, Caduceus has *a set of experts* A.

$$A = \{a_i, \ldots, a_n\} \tag{1}$$

A set of *impact* weights each assigned to the experts.

$$\Omega = \{\omega_i, \ldots, \omega_n\} \tag{2}$$

Therefore, Caduceus' algorithm *IPO variation* a slightly easier to implement version for generating bids is:

- $B \leftarrow \{\}$
- **for each** $i \in A$ **do**

- ◦ $a_i' \leftarrow$ query agent i with action a in time t
- ◦ **if** $type(a_i') = I^{bid}$ **then**
  - ▪ $B \leftarrow B \cup \{a_i'\}$
  - ▪ $\ell_{bid} \leftarrow \ell_{bid} + 1$
- ◦ **else**
  - ▪ $\ell_{accept} \leftarrow \ell_{accept} + 1$
- **if** $\ell_{accept} > \ell_{bid}$ **then return** I^{accept}
- **else return** a bid b' from weighted random sample set B according to Ω

where:

- ℓ_{accept} : the number of experts that generates *accept action*,
  - ◦ i.e. the number of experts that recommend accepting the offer that is on the table
- ℓ_{bid}: the number of experts that generates *bid action*
  - ◦ i.e. the number of experts that recommend making a counter offer
- a_i': action that agent i recommends to Caduceus to take

2 Task 1: Implement Caduceus IPO - *Incremental Portfolio*

⚠ Make sure the expert agents set are receiving the messages from the opponent agents.

⚠ You can pick your experts from previous competitions. But be careful, if you are going to use agents from:

- From ANAC 2016: agents protocol was set to be SOAP.
- From ANAC 2017: agents can access data from their past negotiations form a tournament. Check the details here.
- From ANAC 2019 (upcoming): agents which can work with partially ordered preference profiles will be used, protocol is expected to be bilateral version of SOAP.

3 Task 2: Implement Caduceus CO - *Crossover*

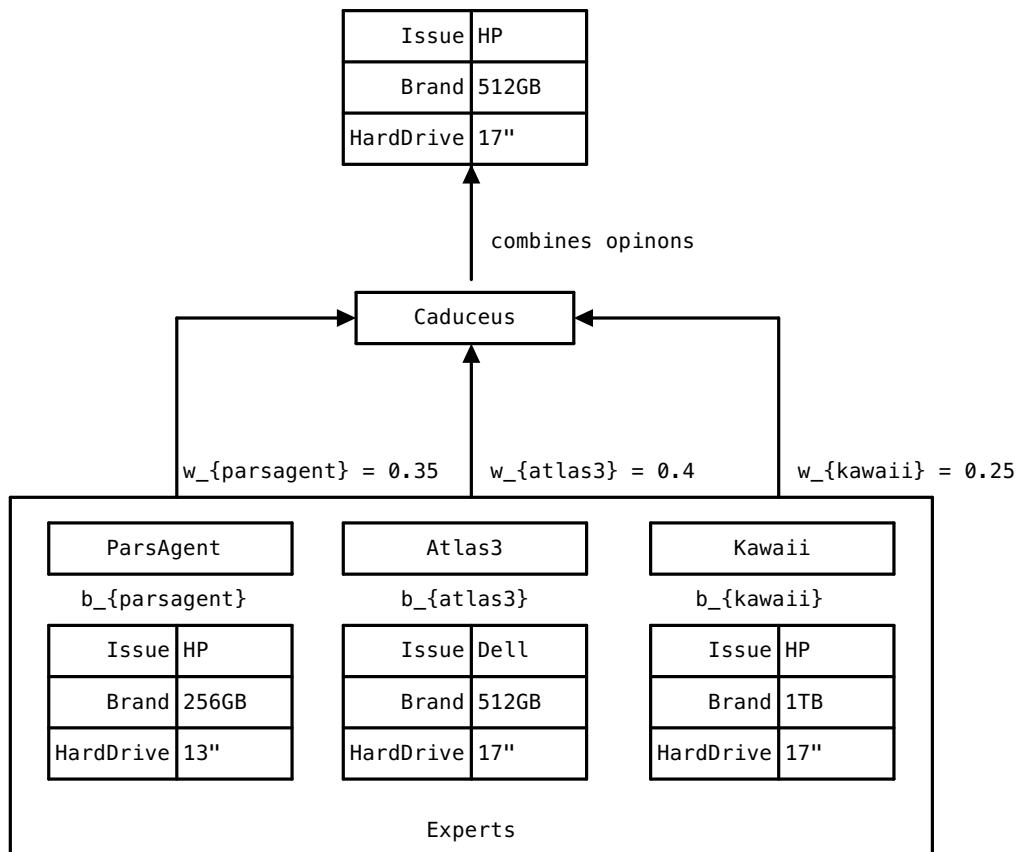Implement *Crossover Strategy* of Caduceus: Algorithm 1.2 without *waiting phase* from the paper.

Issue	HP
Brand	512GB
HardDrive	17"

↑

combines opinons

Caduceus

w_{parsagent} = 0.35   w_{atlas3} = 0.4   w_{kawaii} = 0.25

ParsAgent

b_{parsagent}

Issue	HP
Brand	256GB
HardDrive	13"

Atlas3

b_{atlas3}

Issue	Dell
Brand	512GB
HardDrive	17"

Kawaii

b_{kawaii}

Issue	HP
Brand	1TB
HardDrive	17"

Experts

Figure 1: *CO version of Caduceus combines experts opinions*

The modification that you are going to do to IPO is changing how the bid is generated. In *CO* version pick the values of the issues that are offered the most by the experts.

4 Bonus Task: Further improvements

- Instead of each agent has same impact on the votes, how about using weights to vote?
- Try to come up with an heuristic to find weights of each expert.

5 Bibliography

[**Gunes2017**] Güneş T. D., Arditi E., and Aydoğan R., "Collective Voice of Experts in Multilateral Negotiation," in PRIMA 2017: Principles and Practice of Multi-Agent Systems: 20th International Conference, Nice, France, October 30 – November 3, 2017, Proceedings, B. An, A. Bazzan, J. Leite, S. Villata, and L. van der Torre, Eds. Cham: Springer International Publishing, 2017, pp. 450–458. https://link.springer.com/chapter/10.1007/978-3-319-69131-2_27

COMP6203: Lab #5

Intelligent Agents
Wednesday, 14 November 2018
University of Southampton
Go to Lab Index

This lab shows how to evaluate your agent against various benchmark metrics.

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:

$$U_a(b_{accepted}) = \sum_{i=1}^{n} \omega_i \frac{eval(v_{c_i}^i)}{max(eval(I_i))} \tag{1}$$

$b_{accepted}$ is the accepted bid that is accepted by everybody.

> If in the negotiation environment:
>
> - the protocol is SOAP
> - an agent walks-away
> - or deadline is reached
>
> Everyone gets their reservation value (if discounted, they get discounted reservation value) as their own individual utility.

- **Distance to Social Welfare** (*Defined in MultilateralAnalysis.java:299*): The sum of all the individual utilities of all agents.

$$S(b_a) = \sum_{i=1}^{n} U_a(b_a) \qquad (2)$$

1.2.2 Distance to Pareto Curve

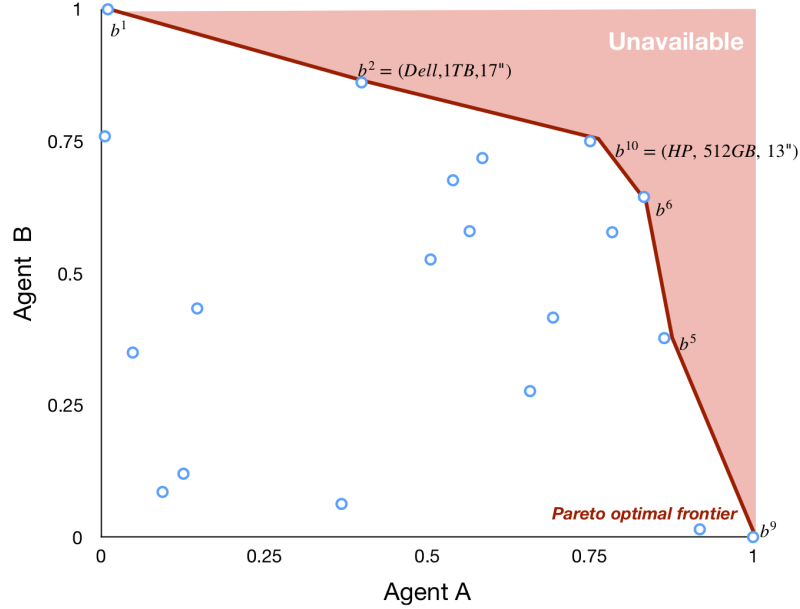(*Defined in MultilateralAnalysis.java:322*): The accepted bid's distance to pareto curve.

Figure 1: *Available bids in the pareto frontier*

"A negotiation outcome is said to be Pareto efficient if there is no other outcome that will make at least one agent better off without making at least one other agent worse off. Intuitively, if a negotiation outcome is not Pareto efficient, then there is another outcome that will make at least one agent happier while keeping eve- ryone else at least as happy." [Jennings2001]

The distance is calculated by getting all the bids which are on the curve. Iterating each bid and getting minimum distance, formally (following example Figure 1):

$$P(b_a) = min(\{||U(b_a) - U(b_1)||, \ldots, ||U(b_a) - U(b_9)||\}) \qquad (3)$$

where $||U(b_i) - U(b_j)||$ is the euclidean distance between two bids and $U(b_i)$ is a tuple that holds each utility that each agent would gain from bid b_i, such that:

$$U(b_i) = (U_1(b_i), \ldots, U_k(b_i)) \qquad (4)$$

1.2.3 Distance to Nash Point

Defined in MultilateralAnalysis.java:313: Nash point is a certain point in (number of agents) dimensional space, such that all parties are in (nash) equilibrium. Assume that b_{nash} is the nash point, let's say Agent A and Agent B are negotiating end. This bid is accepted. This means that Agent A can not improve his utility without hurting Agent B utility and this must apply for the other way around. [Jennings2001]

To see how this point is found, have a look at its implementation (MultilateralAnalysis.java:333) in GENIUS. Once the distance is found, the euclidean distance is again calculated between the accepted bid b_a and b_{nash}.

2 Task 1: Run a simple tournament and plot these metrics

1. Open the GENIUS user interface
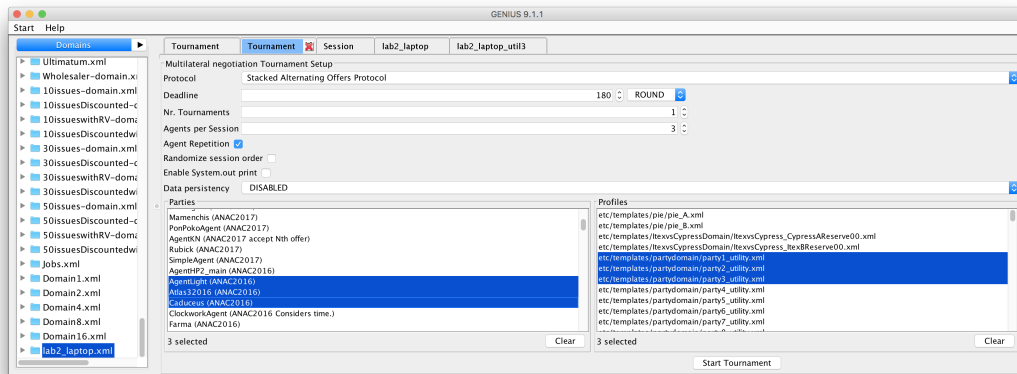2. Create a new tournament with these settings:

Figure 2: *Tournament settings*

3. Locate the logs: e.g. `ExampleAgent/genius-{version}/log/tournament-*.(xml|log)`
4. Open `tournament-*.logStats.xml`, you can find the mean results of the metrics that we mentioned.
   - While you are developing your agent, you can check these for your agents performance
5. Try to get familiar on how to plot these results (MS Excel, Numbers, matplotlib, Matlab)

3 Bonus Task 1: Compute & plot pareto efficient frontier

Compute and draw pareto efficient frontier in a graph such as Figure 1.

4 Bonus Task 2: Parse tournament-*.log.xml, calculate additional metrics

1. You can calculate **the standard deviation** of each these metrics to see how much the mean performance deviates.
   - You can use a scripting language to parse the log files.
   - You can use MS Excel with `tournament-*log.csv` directly with pivot tables.
     - More on this is in Genius User Guide.

2. Try to come up with additional metrics. Some tips:
   - How is your agent performing under:
     - big/small domains many issues and values)
     - discrete, continuous issues
     - and so on.

> **Running tournaments from command line.**
>
> You can define multiple tournaments in an `xml file` and GENIUS can run them. Have a look at the GENIUS User Guide Section 5.3 and here.

5 Bibliography

[**Jennings2001**] Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Wooldridge, M. J., & Sierra, C. (2001). Automated negotiation: prospects, methods and challenges. Group Decision and Negotiation, 10(2), 199-215.

http://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/gdn2001.pdf

COMP6203: Lab #6

Intelligent Agents
Wednesday, 21 November 2018
University of Southampton
Go to Lab Index

This lab shows some key components of a negotiation agent and how to start improving them.

Contents

1 Theory Crash Course

We separate our negotiating agent's grand strategy into smaller strategies:

- **Offering Strategy** (i.e. *Bidding Strategy*): the strategy of what bids to generate

- **Opponent Modeling**: how to best capture/predict the preferences of our opponents

- **Acceptance Strategy**: our strategy of deciding when to accept, when to counter-offer, when to walk away.

- **Opponent Model Strategy**: how to use opponent modeling to select a bid for opponent

There are many papers in the literature focuses on a specific part of the design of an negotiating agent. We recommend you to check the bibliography to existing negotiation strategies.

> GENIUS has a framework called `BOA` to allow mixing different components without implementing your agent from scratch. You can have a look to experiment with strategies.

> ⚠ Early version GENIUS initially allowed bilateral negotiations. Therefore, some components of BOA is focused on only bilateral negotiations. Double check their source code to make sure.
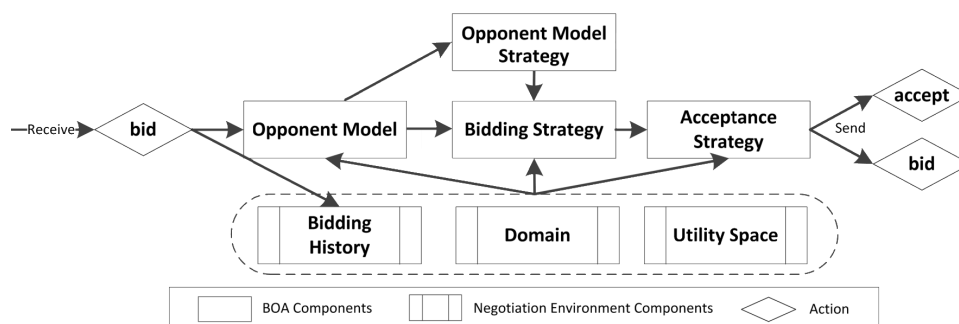

Figure 1: *BOA Framework*

```
opponentModel.init(negotiationSession, omParams);
omStrategy.init(negotiationSession, opponentModel, omsParams);
offeringStrategy.init(negotiationSession, opponentModel, omStrategy,
        osParams);
acceptConditions.init(negotiationSession, offeringStrategy,
        opponentModel, acParams);
```

Figure 2: *BoaParty, the relationships between strategies `negotiationSession` holds values such as: BidHistory, Domain, OutcomeSpace*

1.1   A simple BOA Party:

We can make our simple agent by combining already available strategies from GENIUS. Assume that these are picked:

1.1.1   Offering Strategy: ChoosingAllBids

"An offering strategy which creates a list of possible bids and then offers them in descending order. If all bids are offered, then the last bid is repeated."

1.1.2   Opponent Modelling: UniformModel

"Simple baseline opponent model which always returns the same preference."

1.1.3   Acceptance Strategy: AC_Const

"This Acceptance Condition accepts an opponent bid if the utility is above a constant."

1.1.4   Opponent Model Strategy: BestBid

"This class uses an opponent model to determine the next bid for the opponent, while taking the opponent's preferences into account. The opponent model is used to select the best bid."

> Exercise: try to define the behaviour of our BOA agent that uses all these component above.

2   Task 1: Create this BOA Agent from GENIUS User Interface

1. Go to the `BOA Parties` tab in GENIUS.
2. Right click and click `add new item`.
3. Observe all available components.
4. Pick the ones we mentioned above.
5. After creating the agent, have a couple of runs to make sure it works.

3   Task 2: Define this BOA Agent programmatically

You need to use `BOAParty` class to extend your new agent.

```java
import genius.core.boaframework.BoaParty;
import genius.core.boaframework.SessionData;
import genius.core.parties.NegotiationInfo;
import genius.core.persistent.PersistentDataType;
import negotiator.boaframework.omstrategy.NullStrategy;

import java.util.HashMap;

public class SimpleBoaParty extends BoaParty {
    public SimpleBoaParty () {
        super (null , new HashMap < String , Double > () , null ,
                new HashMap < String , Double > () , null ,
                new HashMap < String , Double > () , null ,
                new HashMap < String , Double > ());
    }
    @Override
    public void init ( NegotiationInfo info ) {
        SessionData sessionData = null ;
        if ( info . getPersistentData ()
                . getPersistentDataType () == PersistentDataType. SERIALIZABLE ) {
            sessionData = ( SessionData ) info . getPersistentData (). get ();
        }
        if ( sessionData == null ) {
            sessionData = new SessionData ();
        }
        negotiationSession = new NegotiationSession( sessionData ,
                info . getUtilitySpace (), info . getTimeline ());
        opponentModel = new MyrequencyModel ();
        opponentModel . init ( negotiationSession , new HashMap < String , Double >
());
        omStrategy = new NullStrategy( negotiationSession );
        offeringStrategy = new MyBiddingStrategy ( negotiationSession ,
                opponentModel , omStrategy );
        acceptConditions = new AC_Next ( negotiationSession , offeringStrategy , 1,
                0);
    // we have init 'd all params here , don 't call super init
    }
    @Override
    public String getDescription () {
        return " Simple BOA Party ";
    }
}
```

4 Task 3: Change *Opponent Modelling* approach with frequency based approach

As you noticed from the implementation of the `UniformModel`, it returns equally evaluations of each bid. Instead of this dummy component, write a frequency based approach that looks at the bid history and makes better predictions about opponents preference profile.