MOSYMA

Un Agent de négociation bilatérale avec Genius

Le but de ce mini-projet est de réaliser un agent négociateur performant en se basant sur l'agent "RandomWalk" proposé en TP. Cet agent utilise l'architecture BOA (voir cours) qui est implémentée dans Genius. Les sources de cette implémentation sont dans les packages *negotiator.boaframework*. Les principales classes à voir sont :

- negotiator.Bid ;
- negotiator.actions.Action ;
- negotiator.actions.Offer ;
- negotiator.bidding.BidDetails ;
- negotiator.boaframework.SortedOutcomeSpace et negotiator.boaframework.OutcomeSpace
- negotiator.Agent ;

Pour implementer un agent avec Genius, vous devez également regarder les classes suivantes (cette description est reprise du manuel d'utilisation de Genius) :

- BidDetails is a structure to store a bid and its utility.
- BidDetailsTime is a structure to store a bid, its utility, and the time of offering.
- BidHistory is a structure to keep track of the bids presented by the agent and the opponent.
- BidIterator is a class used to enumerate all possible bids. Also refer to SortedOutcomeSpace.
- BidSpace is a class which can be used to determine the Pareto-optimal frontier and outcomes such as the Nash solution. This class can be used with the opponent's utility space as estimated by an opponent model.
- Pair is a simple pair of two objects.
- Range is a structure used to describe a continuous range.
- SortedOutcomeSpace is a structure which stores all possible bids and their utilities by using BidIterator. In addition, it implements efficient search algorithms that can be used to search the space of possible bids for bids near a given utility or within a given utility range.
- UtilitySpace is a representation of a preference profile. It is recommended to use this class when implementing a model of the opponent's preference profile.

1 Framework BOA

2 RandomWalker

Dans ce mini-projet nous utilisons le framework BOA (voir cours et Figure 1).

Un exemple d'implémentation de cet agent est :

```
package agents;

import java.util.HashMap;
```

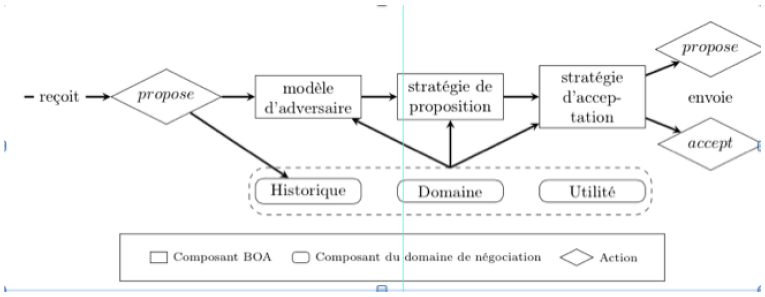

FIGURE 1 – Architecture d'Agent BOA

```
import java.util.List;
import java.util.Random;

import negotiator.Agent;
import negotiator.Bid;
import negotiator.actions.Accept;
import negotiator.actions.Action;
import negotiator.actions.ActionWithBid;
import negotiator.actions.Offer;
import negotiator.issue.Issue;
import negotiator.issue.IssueDiscrete;
import negotiator.issue.IssueReal;
import negotiator.issue.Value;
import negotiator.issue.ValueReal;
public class RandomWalker extends Agent{

 private static final long serialVersionUID = 1L;
 private Random rand;
 private Bid lastBid;

    @Override
    public void init(){
        super.init();
        rand = new Random();
    }

    @Override
    public Action chooseAction() {
        Bid bid = utilitySpace.getDomain().getRandomBid(rand);
        Action action = new Offer(getAgentID(), bid);
        return action;
    }

    public void ReceiveMessage(Action opponentAction){
        if(action instanceof Offer)
        lastBid = ((Offer) opponentAction).getBid();
    }
}
```

```
}
```

Cet agent doit "override" les trois méthodes ReceiveMessage, init, et chooseAction. ReceiveMessage traite l'action (Accept, Reject ou Offer) de l'adversaire. chooseAction définit la réponse à l'offre reçue. Ceci est un exemple d'implémentation de cette méthode :

```
public Action chooseAction() {
    Action action = null;
try {
if (actionOfPartner == null) {
action = chooseRandomBidAction();
}
if (actionOfPartner instanceof Offer) {
Bid partnerBid = ((Offer) actionOfPartner).getBid();
double offeredUtilFromOpponent = getUtility(partnerBid);
// get current time
double time = timeline.getTime();
action = chooseRandomBidAction();
Bid myBid = ((Offer) action).getBid();
double myOfferedUtil = getUtility(myBid);

// accept under certain circumstances
if (isAcceptable(offeredUtilFromOpponent,myOfferedUtil,time)) {
action = new Accept(getAgentID());
}
}
  } catch (Exception e) {
e.printStackTrace();
action = new Accept(getAgentID()); // best guess if things go wrong.
}
return action;
}
```

La method *isAcceptable* implémente une fonction d'acceptation P_{accept} :

P_{accept} = u - 2ut + $2\Big(t - 1 + \sqrt{(t-1)^2 + u(2t-1)}\Big)$ $2t - 1$(1)où u est l'utilité de la proposition faite par l'adversaire (fourni par utilityspace), et t ile temps défini comme une fraction du temps global.

Chaque classe d'agent utilise plusieurs classes et attributs tels que :

- UtilitySpace utilitySpace : Le profil de préférences du scénario alloué à l'agent,
- Timeline timeline,
- double getUtility(Bid bid) : our avoir l'utilité d'un bid,
- void init() : Informe l'agent quand une nouvelle session de négociation commence,
- void ReceiveMessage(Action opponentAction) : Informe l'agent quel action l'adversaire a effectué,
- Action chooseAction() : Retourne l'action que l'agent propose d'effectuer,
- String getName() : retourne le nom de l'agent

3 Agent CarefulAgent

Cet agent utilise des concessions simples en exploitant la liste des bids possibles qu'il définit dans la méthode *init* suivante :

```
private List<Bid> acceptableBids;

    @Override
    public void init() {
super.init();
acceptableBids = new ArrayList<Bid>();

BidIterator iter = new BidIterator(this.utilitySpace.getDomain());
while (iter.hasNext()) {
Bid bid = iter.next();
try {

if (getUtility(bid) >= utilitySpace.getReservationValue() && (Math.random() <= getUtilit
this.acceptableBids.add(bid);

} catch (Exception e) {
e.printStackTrace();
}
}

}
```

Une fois testé, cet agent pourrait être amélioré pour avoir une approche adaptative. Par exemple, il pourrait adapter son mécanisme de concession au temps de la négociation restant.

4 ImitatingAgent

Dans un premier, cet agent imite son adversaire en s'appuyant sur une analyse des offres reçu.

5 AdaptiveAgent

Cet agent mémorise l'historiques des différentes négociations avec un agent et les réutilise. Il peut ainsi utiliser différentes stratégies et adapter ensuite le choix de la stratégie en s'appuyant sur son historique.