Ismail Benaija & Ayoub Bakkoury - SMA

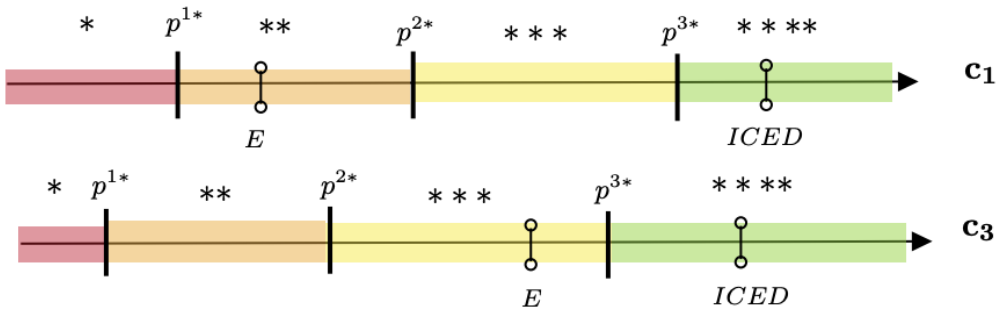
Négociation basée sur l'argumentation pour l'achat d'un moteur de voiture

1. Introduction

L'objectif de ce projet est d'implémenter un système de négociation et d'argumentation à deux agents pour la sélection de moteurs de voitures selon différents critères. Les deux agents A1 et A2, représentant des ingénieurs, devront négocier pour prendre une décision concernant le choix du meilleur moteur. Ces derniers doivent choisir entre deux moteurs ICED (*Internal Combustion Engine Diesel*) et E (*Essence*). Leur choix reposera sur les cinq critères suivants :

- c1: Le coût de production en € ˆ
- c2: La consommation de carburant (L/100 km)ˆ
- c3: La durabilité du moteur. Mesurée sur une échelle de 1 (plus courte) à 4 (plus longue) ˆ
- c4: L'impact environnemental. Mesuré sur une échelle de 1 (faible impact) à 4 (port impact), et que l'on cherche donc à minimiser
- c5: La pollution sonore (dB)

Les agents ont chacun une sensibilité différente à ces cinq critères. Chacun de ces critères est apprécié en 4 zones de profils (de très mauvais à très bon), et dont les seuils varient par conséquent d'un agent à l'autre. Ce sont ainsi des préférences différentes de ces critères qui permettront aux agents de débattre par la suite sur le moteur à sélectionner.



Représentation possible de l'appréciation des critères c1 et c3 pour un agent A (Extrait de l'énoncé du projet)

Le projet a été implémenté sous python à l'aide de la librairie mesa. Le dossier initial est composé de quatre fichiers qui sont les suivants (la structure a été conservée à l'identique) :

- *communication*: Le dossier racine de la communication inter-agents
- *agent*: Contenant l'implémentation des classes d'agents
- *mailbox*: Contenant la classe mailbox ˆ
- *message*: Contenant le message et la classe performative

L'objectif est donc de réaliser une simulation de la sorte (exemple de rendu ci-dessous :

- A1 propose un moteur
- A2 lui demande d'argumenter, puis propose à son tour un autre moteur suite à son argumentation
- Les deux agents exposent leurs préférences et les critères qu'ils privilégient pour leur choix, afin de trouver une zone d'accord possible.
- L'argumentation d'A1 et A2 converge vers un moteur qui satisfait les priorités des deux agents.
- L'agent qui veut ce moteur le propose, et un consensus est trouvé.

```
01: Agent1 - PROPOSE(ICED)
02: Agent2 - ASK_WHY(ICED)
03: Agent1 - ARGUE(ICED <= Cost=Very Good)
04: Agent2 - PROPOSE(E)
05: Agent1 - ASK_WHY(E)
06: Agent2 - ARGUE(E <= Environment Impact=Very Good)
07: Agent1 - ARGUE(not E <= Cost=Very Bad, Cost>Environment)
08: Agent2 - ARGUE(E <= Noise=Very Good, Noise > Cost)
09: Agent1 - ARGUE(not E <= Consumption=Very Bad)
10: Agent2 - ARGUE(notICED <= Noise=Very Bad, Noise> Cost)
11: Agent1 - ARGUE(ICED <= Durability=Very Good)
12: Agent2 - ARGUE(not ICED <= Environment =Very Bad, Environment >
Durability)
13: Agent2 - ARGUE(E <= Durability=Good)
13: Agent1 - ACCEPT(E)
14: Agent1 - COMMIT(E)
15: Agent2 - COMMIT(E)
```

Exemple de rendu de simulation (Source : énoncé du projet)

Pour développer cette simulation, nous avons tout d'abord défini les préférences des agents, les classes d'agent et leurs messages, puis nous avons travaillé sur la partie argumentation en trois étapes.

2. Préférences des Agents

La première étape de ce projet consiste à récupérer les préférences des agents. Nous avons défini une méthode *'most_preferred'* qui, à partir de la liste des moteurs "item_list" et de la fonction *get_score(self, preferences)*, nous renvoie le moteur préféré de l'agent concerné. Nous avons par ailleurs utilisé cette méthode pour récupérer le top 10% des items préférés de l'agent, ou pour savoir si un moteur sélectionné est parmi les 10% préférés de l'agent.
Une fois cette partie terminée, nous réalisons un test unitaire à partir de certains paramètres initialisés dans le fichier source du projet du projet. Nous obtenons en sortie des informations de préférences de l'agent A1 (agent 1) pour les moteurs ICED et E, comme le récapitule l'image ci-dessous :

```
Diesel Engine (A super cool diesel engine)
Electric Engine (A very quiet engine)
Value.VERY_GOOD
True
Electric Engine > Diesel Engine : False
Diesel Engine > Electric Engine : True
Electric Engine (for agent 1) = 362.5
Diesel Engine (for agent 1) = 525.0
Most preferred item is : Diesel Engine
```

Dans le code il existe deux manières de générer les préférences des agents, soit avec le mode general et ca sera aléatoire, soit avec le mode sujet et ca sera les préférences de l'exemple donné dans le sujet.

3. Messages

Dans cette partie, nous définissons la classe Agent, ainsi que la classe messages, utile pour l'échange entre A1 et A2. 5 performatives modélisent les messages possibles entre les agents :

- Proposer un item (moteur) ˆ
- Accepter la proposition
- Commit : pour confirmer qu'un item a été sélectionné (fait office de signature).ˆ
- ask_why: pour demander d'argumenter
- argue: Donner un argument

Nous avons également implémenté, à l'aide des préférences des agents les interactions de base : Propose, Ask_why, et Accept.
Nous avons rencontré quelques difficultés dans cette partie, mais sommes finalement parvenus à les surmonter. En effet, outre des soucis de package à surmonter, nous avons eu du mal à incorporer le double "commit" des deux agents avant de clôturer l'échange. La boucle d'échange ne parvenait pas à s'interrompre, et recevions des erreurs au moment de vouloir implémenter la fonction "argue" seule. Finalement, cela était dû à des bugs sur certaines des fonctions dans les fonctions de notre fichier pw_argumentation, qui ont été mal implémentées.

4. Arguments

Une fois des messages de base implémentés, il s'agissait d'aller plus en détail dans l'implémentation des agents via le développement des arguments entre ces derniers. Nous partons du principe que la base de connaissances des agents est leurs préférences (pour des soucis de simplification). Le premier objectif était ainsi de générer ces arguments. Cela passe par la comparaison de critères, ainsi que la classe *CoupleValue,* qui permettra de

comparer des critères entre eux, ou d'attribuer une valeur à un critère : (Ci = x) ou (Ci > Cj). La classe argument pourra ainsi être définie de la forme : ˆ

- Le nom du moteur pour lequel on argumente en faveurˆ
- Un booléen, qui signale si l'argument est en faveur du moteur ou non
- Une liste de couples de la forme ((Ci , x) et,ou (Ci, Cj).

```
From Ayoub to Ismail (PROPOSE) Electric Engine (A very quiet and ecofriendly engine)
From Ismail to Ayoub (ASK_WHY) Electric Engine (A very quiet and ecofriendly engine)
From Ayoub to Ismail (ARGUE)  Electric Engine CriterionName.ENVIRONMENT_IMPACT' = 'Value.VERY_GOOD'
From Ismail to Ayoub (ARGUE) not Electric Engine CriterionName.PRODUCTION_COST' = 'Value.BAD' CriterionName.PRODUCTION_COST > CriterionName.ENVIRONMENT_IMPACT
From Ayoub to Ismail (PROPOSE) Diesel Engine (A super cool diesel engine)
From Ismail to Ayoub (ACCEPT) Diesel Engine (A super cool diesel engine)
From Ayoub to Ismail (COMMIT) Diesel Engine (A super cool diesel engine)
From Ismail to Ayoub (COMMIT) Diesel Engine (A super cool diesel engine)
```

Exemple de simulation simple de négociation entre agents

Pour la sélection d'arguments, nous avons développé conformément au format de l'énoncé les fonctions permettant de lister les arguments en faveur (GOOD ou Very GOOD) et en défaveur (BAD/VERY BAD) d'un moteur. La fonction support_proposal permet ainsi de sélectionner le meilleur argument lié au sujet du débat.
Pour la dernière partie de relations entre arguments, nous avons simplifié le modèle d'argumentation par rapport à celui défini du sujet, pour pouvoir itérer une première fois le modèle de négociation.

En effet, dans la partie relation entre les arguments nous avons implémenté une fonction attackable qui renvoie un booléen si oui l'argument est attaquable et si non il n'est pas attaquable. Ce booléen intermédiaire permettrait par exemple de conclure la discussion si l'on ne peut pas contredire un argument et qu'il convainc un agent.
Dans le cas où un argument est attaquable, il faut renvoyer un contre argument. Pour le trouver, on utilise la fonction list_attacked_proposal et on trouve les critères pour lesquels l'item a une mauvaise valeur.
Ensuite le contre-argument est construit en ne gardant que les critères qui sont les plus importants pour l'agent, et nous réitérons une boucle.
Enfin, s'il ne reste plus d'arguments, on accepte la proposition. Ce modèle simplifié de génération de contre arguments fait que la simulation est souvent très courte. Les deux agents (nommés Ayoub et Ismail) ne s'échangent que 1 ou 2 arguments avant de tomber sur un accord. Nous avons lancé plusieurs simulations, en essayant de permuter les deux agents, comme ci-dessous:

```
From Ismail to Ayoub (PROPOSE) Diesel Engine (A super cool diesel engine)
From Ayoub to Ismail (ASK_WHY) Diesel Engine (A super cool diesel engine)
From Ismail to Ayoub (ARGUE)  Diesel Engine CriterionName.PRODUCTION_COST' = 'Value.VERY_GOOD'
From Ayoub to Ismail (ARGUE) not Diesel Engine CriterionName.ENVIRONMENT_IMPACT' = 'Value.VERY_BAD' CriterionName.ENVIRONMENT_IMPACT > CriterionName.PRODUCTION_COST
From Ismail to Ayoub (PROPOSE) Electric Engine (A very quiet and ecofriendly engine)
From Ayoub to Ismail (ACCEPT) Electric Engine (A very quiet and ecofriendly engine)
From Ismail to Ayoub (COMMIT) Electric Engine (A very quiet and ecofriendly engine)
From Ayoub to Ismail (COMMIT) Electric Engine (A very quiet and ecofriendly engine)
```

Simulation avec permutation des deux agents

Nous avons exploré différentes pistes pour essayer de développer ce modèle en termes d'échanges d'arguments avant d'arriver à un consensus, mais en vain.

Notre modèle reste tout de même adaptable à une simulation avec plusieurs moteurs, bien que nous n'ayons pas tenté de l'implémenter. Il serait également possible en guise de dépassement d'essayer d'inclure N>2 agents, mais cela nécessite de repenser la structure

de certaines classes que nous avons définies, et de par exemple, mettre en place un ordre de parole pour structurer la négociation entre les agents.