CentraleSupélec

Systèmes multi-agents

Projet 2: argumentation-based dialogue for choosing a car engine !

Superviseurs: Belahcène Khaled, Ouerdane Wassila et Sabouret Nicolas

Mots clés et thèmes

Systèmes multi-agents, Python, Mesa.

Michot Albane & Nonclercq Rodolphe

22 Avril, 2023

Table des matières

Abstract

L'objectif du projet est de **simuler un processus d'interaction entre différents agents pour choisir le meilleur moteur** pour une nouvelle voiture, en prenant en compte plusieurs **critères** tels que la consommation, l'impact environnemental, le coût, la durabilité, etc. Le but est de simuler différents comportements d'agents pour aboutir à **la meilleure offre**.
La librairie de modélisation multi-agents `mesa` [1] met à disposition des outils pour créer des modèles d'agents, les faire interagir, et analyser leurs résultats.

1 Introduction

1.1 Description

Un constructeur automobile souhaite lancer une nouvelle voiture sur le marché. Pour cela, **un choix crucial est celui des moteurs** qui doivent répondre à certaines exigences techniques tout en étant attrayants pour les clients (économiques, robustes, écologiques, etc.). Plusieurs types de moteurs existent et offrent donc une large gamme de modèles de voitures : essence ou diesel Internal Combustion Engine (ICE), gaz naturel comprimé (CNG), batterie électrique (EB), pile à combustible (FC), etc. La société décide de prendre en compte **différents critères** pour les évaluer : consommation, impact environnemental (CO2, carburant propre, NOX1...), coût, durabilité, poids, vitesse maximale ciblée, etc. Pour établir la meilleure offre/choix parmi un grand nombre d'options, elle décide de **simuler un processus d'interaction où des agents, avec des opinions et des préférences différentes** (voire des connaissances et des expertises différentes), discutent de la question **pour aboutir à la meilleure offre**. La simulation permettra à la société de simuler plusieurs typologies de comportements d'agents (expertise, rôle, préférences, etc.) à moindre coût et dans un délai raisonnable.

Pour cela, nous avons implémenté **une simulation de dialogue fondée sur l'argumentation entre agents**. Des agents représentant l'ingénierie humaine devront interagir les uns avec les autres pour prendre une décision conjointe concernant le choix du meilleur moteur. Les conflits dans l'interaction surviennent lorsque les agents ont des préférences différentes sur les critères, et l'argumentation les aidera à décider quel élément sélectionner.

1.2 Segmentation

Notre répertoire est segmenté en 18 fichiers `python`, 1 fichier `csv`, un fichier `markdown`, un fichier `.gitinore`, deux fichiers `.txt` pour les requirements et un fichier pdf (Figure 1).

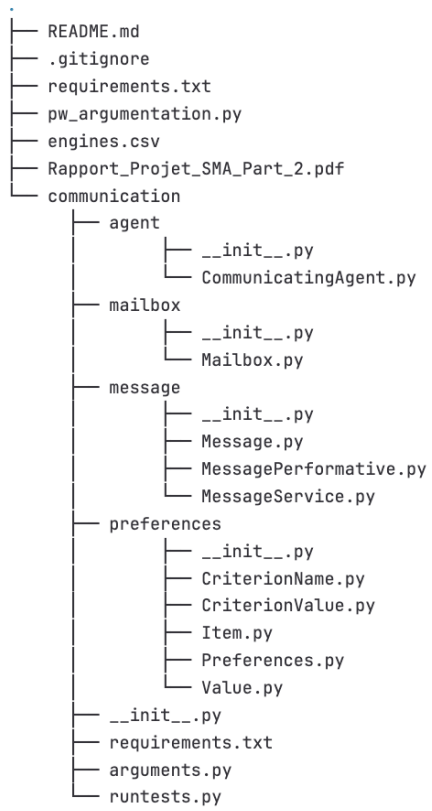

Figure 1: Segmentation du répertoire du projet

Les fichiers suivants a été complétés:

1. pw_argumentation.py qui définit notre classe d'agents et notre modèle pour la simulation d'argumentation.
2. arguments.py qui définit la structure de nos arguments.
3. engines.csv qui sauvegarde les cinq moteurs et leurs valeurs correspondantes pour chaque critère.

L'analyse se focalisera, pour ce rapport, sur la mise en oeuvre du projet et nos choix techniques.
Afin de répondre au mieux au problème, la démarche suivante est proposée :

1. Algorithme implémenté (`pw_argumentation.py`)
2. Mise en oeuvre des profils et préférences (`engines.csv` et `pw_argumentation.py`)
3. Arguments (`arguments.py`)
4. Exemple de dialogue (`pw_argumentation.py` et `arguments.py`)
5. Conclusion

2 Algorithme implémenté

Nous avons repris et implémenté l'algorithme proposé en modifiant certaines parties:

Algorithme 1 *: protocole de communication*

initialization;

- create a set of n agents
- set-up a list of m objects (items) $o_1, ..., o_m$
- set-up a list of k criteria $c_1, ..., c_k$

foreach *agent i in agents* **do**
  define preferences of i (criteria order, a performance table)
end
foreach *$(X, Y) \in$ agents* **do**
  X: propose (o_i);
  Y: receivers-message(propose(o_i));
  if *($o_i \in$ 10% favorite objects/items of Y)* **then**
    if *($o_i = o_j$ | o_j top item of Y)* **then**
      Y: accept(o_i);
      X: commit(o_i);
      Y: commit(o_i);
    end
    else
      Y: propose(o_j)
    end
  end
  else
    Y: ask_why(o_i)
  end
  X: receive-message(ask_why(o_i));
  if *(X has at least one argument pro o_i)* **then**
    X: argue(o_i, reasons);
  end
  else
    X: propose(o_j), $j \neq i$
  end
  Y: receive-message(argue(o_i, reasons))
  if *(Y has a counter-argument(o_i))* **then**
    switch *(if reasons correspond to ($c_i = x$))* **do**
      case *(Y has a better alternative o_j, $j \neq i$, on c_i)* **do**
        Y: argue(o_j, $c_i = y$, y is better than x)
      end
      case *(o_i has a bad evaluation on c_i)* **do**
        Y: argue(not o_i, $c_i = y$, y is worst than x)
      end
      case *(o_i has a bad evaluation on c_j ($j \neq i$) and c_j is an important criterion for Y)* **do**
        Y: argue(not o_i, $c_j = y$ with y is worst than x, $c_j \succ c_i$)
      end
      case *o_j, exist c / $c_j \succ c_i$* **do**
        Y: argue(o_j, $c_j \succ c_i$)
      end
    end
    switch *(if reasons correspond to ($c_i = x$ and $c_i \succ c_j$)* **do**
      case *(Y has a better alternative o_j, $j \neq i$, on c_i)* **do**
        Y: argue(o_j, $c_i = y$, y is better than x)
      end
      case *(Y prefer c_j to c_i)* **do**
        Y: argue(not o_i, $c_j = y$, $c_j \succ c_i$)
      end
    end
    else
      if *has argument* **then**
        Y: argue(o_j, $c_j = y$, y is top 3 criterium on this item)
      end
      else
        Y: accept(o_i);
        X: commit(o_i);
        Y: commit(o_i);
      end
    end
  end
end

Nous avons **rajouté un cas** pour le `counter_argument`: s'il existe un critère non-exploré, tel que $c_j \succ c_i$, il renvoie l'argument en question `o_j, c_j>c_i` (partie orange).

Nous avons supprimé la partie en rouge initialement implémentée. En effet, dans notre architecture, il nous était plus difficile de comparer les préférences entre elles. Nous avons donc remplacé la partie en rouge par la **recherche d'arguments valorisant l'item préféré** de l'agent (partie orange).

3 Mise en oeuvre des profils et préférences

Nous considérons deux agents A_1 et A_2.

Ils doivent se mettre d'accord pour choisir un moteur parmi les cinq suivants, les agents ont ainsi différents choix pour argumenter et trouver un compromis:

- ICED : moteur diesel
- E : moteur électrique
- ESS : moteur essence
- H : moteur hybride
- HYD : moteur hydrogène

Les agents prennent en compte cinq critères différents:

- c_1: coût de production (€, la valeur la plus basse étant la meilleure)
- c_2: consommation (€/100km, la valeur la plus basse étant la meilleure, nous changeons ici les unités en €/100km à la place de L/100km pour une meilleure visualisation de la métrique)
- c_3: durabilité (mesurée sur une échelle qualitative allant de 1 (meilleure performance) à 4 (performance la plus faible), la valeur la plus faible étant la meilleure)
- c_4: impact environnemental (mesuré sur une échelle qualitative allant de 1 (faible impact sur l'environnement) à 4 (fort impact sur l'environnement), la valeur la plus faible étant la meilleure)
- c_5: degré de bruit (mesuré en dB, la valeur la plus faible étant la meilleure)

À titre d'illustration, les performances des cinq moteurs, pour chaque critère, sont décrites dans la Table 1:

Engine	c1	c2	c3	c4	c5
ICED	12 000	6.0	4	4	65
E	25 000	2.0	1	3	50
ESS	8 000	5.0	3	4	69
H	16 000	4.0	2	2	66
HYD	50 000	3.0	1	1	62

Table 1: Données sur les moteurs

Leurs préférences ont été générées à partir de **profils**, qui eux sont **aléatoires**, attribués à chacun des agents.
On prend aléatoirement les profils afin que les agents aient des **préférences différentes** et ainsi une **discussion**.
Nous avons rajouté la fonction `sorted_item_list` dans la classe `Preferences` afin de faciliter l'écriture du code pour `is_item_among_top_10_percent` et accélérer le processus de décision côté agent.

4 Arguments

Nous avons créé un fichier python `arguments.py` pour **structurer les arguments** en rajoutant en plus des phrases des **metadata**, comme par exemple le type d'argument.
La structure de la classe `Argument` permet de **renvoyer une phrase correspondant à l'argument**, en plus de faciliter l'utilisation de ces derniers.
Nous avons également rajouté un moyen de **comparer les arguments entre eux**, en particulier afin de voir s'ils sont égaux pour éviter de réutiliser plusieurs fois le même argument.

5 Exemple de dialogue

La Figure 2 est un exemple de **dialogue entre deux agents et cinq items** (*engines*). Les agents parlent chacun à leur tour et propose ou argumente sur les critères.
S'ils **n'ont plus d'arguments**, ils peuvent **proposer un autre moteur** (au maximum 3 moteurs peuvent être proposés).

```
ordre de préférence croissant de agent_1:  ['HYD', 'ICED', 'E', 'H', 'ESS']
ordre de préférence croissant de agent_2:  ['HYD', 'ESS', 'ICED', 'H', 'E']
From agent_2 to agent_1 (PROPOSE) E (A very quiet engine)
From agent_1 to agent_2 (ASK_WHY) E (A very quiet engine)
From agent_2 to agent_1 (ARGUE) Le E a une bonne performance sur Durabilité
From agent_1 to agent_2 (ARGUE) ESS est équivalent à E du point de vue de Durabilité
From agent_2 to agent_1 (ARGUE) E est meilleur que ESS du point de vue de Consommation
From agent_1 to agent_2 (ARGUE) ESS est meilleur que E du point de vue de Production
From agent_2 to agent_1 (ARGUE) Par rapport au critère Environnement le E a une meilleure performance que ESS
From agent_1 to agent_2 (ARGUE) Le ESS a une bonne performance sur Bruit
From agent_2 to agent_1 (ARGUE) Par rapport au critère Bruit le E a une meilleure performance que ESS
From agent_1 to agent_2 (ARGUE) Le ESS a une bonne performance sur Durabilité
From agent_2 to agent_1 (ARGUE) E est équivalent à ESS du point de vue de Durabilité
From agent_1 to agent_2 (ARGUE) Je préfère ESS pour Environnement
From agent_2 to agent_1 (ARGUE) Le E a une bonne performance sur Bruit
From agent_1 to agent_2 (PROPOSE) H (A well-balanced engine)
From agent_2 to agent_1 (ASK_WHY) H (A well-balanced engine)
From agent_1 to agent_2 (ARGUE) Je préfère H pour Bruit
From agent_2 to agent_1 (ARGUE) E est meilleur que H du point de vue de Bruit
From agent_1 to agent_2 (ARGUE) Par rapport au critère Production le H a une meilleure performance que E
From agent_2 to agent_1 (ARGUE) Par rapport au critère Consommation le E a une meilleure performance que H
From agent_1 to agent_2 (ARGUE) Par rapport au critère Durabilité le H a une meilleure performance que E
From agent_2 to agent_1 (ARGUE) Par rapport au critère Environnement le E a une meilleure performance que H
From agent_1 to agent_2 (ARGUE) Le H a une bonne performance sur Environnement
From agent_2 to agent_1 (ARGUE) Je préfère E pour Durabilité
From agent_1 to agent_2 (ARGUE) Le H a une bonne performance sur Consommation
From agent_2 to agent_1 (ARGUE) Je préfère E pour Bruit
From agent_1 to agent_2 (PROPOSE) E (A very quiet engine)
From agent_2 to agent_1 (ACCEPT) E (A very quiet engine)
From agent_1 to agent_2 (COMMIT) E (A very quiet engine)
From agent_2 to agent_1 (COMMIT) E (A very quiet engine)
```

Figure 2: Exemple de dialogue avec deux agents et cinq items

6 Conclusion

Ce projet nous a permis de mettre en pratique les nouvelles connaissances apprises sur les systèmes multi-agents.

Dans un futur travail, il serait **intéressant de mettre en place plusieurs agents négociant simultanément** et d'implémenter un modèle qui permette aux agents de **retenir les préférences de chacun des agents** et trouver le meilleur compromis sur l'ensemble des critères.

Une autre ligne directive intéressante à tester serait de **prendre en considération les sentiments/caractères de chacun des agents**.
Ainsi pour chaque agent, un profil le caractériserait (de manière aléatoire) parmi, par exemple, les cinq profils suivants élaborés:

1. **L'écologiste** se définit comme l'acheteur soucieux de l'environnement. Les critères c_2 et c_4 seront ceux qu'il considère les plus importants avec une **préférence pour le critère** c_4.

2. **Le voyageur** se définit comme l'acheteur qui parcourt de longues distances régulièrement. Les critères c_3 et c_5 seront ceux qu'il considère les plus importants avec une **préférence pour le critère** c_3 .

3. **Le débrouillard** se définit comme l'acheteur qui doit absolument acheter une voiture mais qui a un petit budget d'entrée. Les critères c_2 et c_3 seront ceux qu'il considère les plus importants avec une **préférence pour le critère** c_1 car il cherche avant tout à pouvoir s'offir une voiture peu chère.

4. **L'économe** se définit comme l'acheteur qui souhaite économiser de l'argent que ce soit sur les coûts de carburant à long terme ou sur le coût d'achat. Les critères c_1 et c_2 seront ceux qu'il considère les plus importants avec une **préférence pour le critère** c_2 car il cherche avant tout à économiser sur le long terme.

5. **Le citadin** se définit comme l'acheteur vivant dans un quartier où les réglementations en matière de bruit et d'impact environnemental sont strictes. Les critères c_4 et c_5 seront ceux qu'il considère les plus importants avec une **préférence pour le critère** c_5.

References

[1] Kazil, Jackie, Masad, David Crooks, Andrew, Octobre 2020. *Utilizing Python for Agent-Based Modeling: The Mesa Framework.*