Projet - COCOMA M2 ANDROIDE - Taxi Allocation

Étudiants :
Baris KAFTANCIOGLU : 28711733
Paul-Tiberiu IORDACHE : 28706827

Encadré par :
Nicolas MAUDET
Aurélie BEYNIER

8 février 2025

Table des matières

1 Introduction **2**

2 Partie 1 : Génération de tâches en ligne et structure générale de notre code **2**
2.1 Classe des Tâches . . . 2
2.2 Classe des Taxis . . . 3
2.3 Classe de l'Environnement . . . 3
2.3.1 Algorithmes d'Ordonancement . . . 4
2.3.2 Choix des Méthodes d'Ordonancement . . . 4
2.3.3 Allocation des Tâches . . . 5
2.4 Intérface graphique . . . 5
2.4.1 Paramètres de la Fonction . . . 5

3 Partie 2 **7**

4 Partie 3 **7**
4.1 Heuristiques d'Insertion dans la Classe des Taxis . . . 7
4.1.1 Heuristique de Prim . . . 7
4.1.2 Heuristique d'Insertion . . . 7
4.2 Classe de l'Environnement et Allocation des Tâches . . . 8
4.2.1 Protocole PSI . . . 8
4.2.2 Protocole SSI . . . 8
4.2.3 Protocole SSI avec Regret . . . 8

5 Etude de performances **8**
5.1 Performances de la partie 2 . . . 8
5.2 Performances de la partie 3 . . . 9
5.2.1 La taille de grille . . . 9
5.2.2 Le nombre des tâches par taxi . . . 10
5.2.3 Le nombre des taxi . . . 11
5.2.4 Interprétation des performances de la partie 3 . . . 12
5.3 Comparaison des performances entre les parties 2 et 3 . . . 13

6 Conclusion **14**

1 Introduction

Ce projet vise à étudier un problème d'allocation de tâches dans un contexte multi-agents, appliqué à des flottes de taxis. Les trajets (ou tâches) à effectuer sont attribués de manière dynamique, c'est-à-dire qu'ils arrivent de façon progressive au fil du temps, sans être connus à l'avance. Ce type de problème a été récemment étudié dans diverses variantes au sein de la littérature.

On suppose que les véhicules évoluent dans un plan orthonormé, où les positions de départ et d'arrivée des trajets sont des coordonnées valides dans cet espace. Chaque trajet est défini par une position de départ et une position d'arrivée, et son coût correspond à la distance euclidienne séparant ces deux points. Tous les T pas de temps, un nouvel ensemble de trajets est proposé aux taxis, qui doivent alors se répartir ces tâches. Il est à noter que les taxis peuvent ne pas avoir terminé l'exécution des trajets acceptés précédemment.

Le coût total associé à un taxi correspond à la somme des distances des trajets qu'il effectue, à laquelle s'ajoute la distance parcourue pour relier la destination d'un trajet à la position de départ du trajet suivant. L'objectif est de minimiser la somme des coûts de l'ensemble des trajets et des déplacements intermédiaires, tout en garantissant l'exécution de toutes les tâches.

Dans ce projet, nous nous concentrons sur l'étude et la comparaison de différentes méthodes décentralisées pour résoudre ce problème d'allocation en ligne des tâches entre les taxis. Le travail se décompose en trois parties principales. En première partie, on met en place un environnement de simulation permettant de générer des scénarios d'allocation de tâches dans un espace de taille finie. Ensuite, on modélise et résout le problème d'allocation à l'aide de DCOP (Distributed Constraint Optimization Problems). Enfin,on analyse différents protocoles de coordination multi-agents pour résoudre ce même problème.

Pour visualiser les codes source et les résultats de nos expériences, vous pouvez consulter notre github.

2 Partie 1 : Génération de tâches en ligne et structure générale de notre code

Pour effectuer des simulations sur le problème d'allocation en ligne des trajets entre taxis, nous commencerons par définir la structure de notre code, qui est basée sur trois classes principales :

- **La classe des tâches** : Représente les demandes de trajets à allouer aux taxis.
- **La classe des taxis** : Modélise les taxis disponibles pour effectuer les trajets.
- **La classe de l'environnement** : Gère l'interaction entre les tâches et les taxis, ainsi que l'allocation des trajets.

2.1 Classe des Tâches

La classe des tâches est un élément fondamental du système, représentant chaque demande de transport au sein de l'environnement.

Caractéristiques principales :

— **Attributs** :
  — Position de départ et position d'arrivée, représentées sous forme de coordonnées dans la grille de l'environnement.
  — Coût estimée de la tâche, qui peut être calculée en fonction de la distance entre le départ et l'arrivée.
  — Une méthode pour calculer la distance entre deux points de la grille.
— **Utilisation dans le système :**
  — Les tâches constituent les unités de travail à attribuer aux taxis.
  — Elles interagissent avec les algorithmes d'ordonnancement pour optimiser leur exécution.

2.2 Classe des Taxis

La classe des taxis représente les agents responsables d'exécuter les tâches.

Caractéristiques principales :
— **Attributs :**
  — Une position initiale sur la grille.
  — Une liste des tâches assignées au taxi.
  — Un coût total, calculé comme la somme des distances parcourues pour exécuter les tâches assignées.
— **Méthodes clés :**
  — Des heuristiques d'insertion, détaillées dans la Partie 3 (section 4), permettent d'ajouter efficacement de nouvelles tâches dans la liste des tâches existantes d'un taxi.
— **Rôle dans la simulation :**
  — Les taxis participent activement au processus d'allocation, où ils soumettent des offres pour les tâches en fonction de leur coût estimé.
  — Après allocation, ils exécutent les tâches selon un ordre optimisé.

2.3 Classe de l'Environnement

La classe de l'environnement est responsable de coordonner les interactions entre taxis et tâches.

Caractéristiques principales :

— **Paramètres configurables :**
  — `grid_size` : Taille de la grille (ex : 10x10, 50x50, etc.).
  — `num_taxis` : Nombre de taxis présents dans l'environnement.
  — `task_frequency` : Fréquence de génération de nouvelles tâches au cours de la simulation.
  — `task_number` : Nombre initial des tâches générées (doit être au moins égal au nombre de taxis).
  — `num_iterations` : Nombre total d'itérations de la simulation.
  — `delay` : Délai entre les étapes de simulation pour ralentir l'exécution, utile pour la visualisation.
— **Fonctionnalités clés :**
  — Génération des taxis et des tâches en fonction des paramètres.

— Coordination des processus d'allocation et d'ordonnancement.

2.3.1 Algorithmes d'Ordonancement

L'ordonnancement des tâches est un aspect critique, influençant directement l'efficacité des taxis dans l'exécution des trajets. Nous avons choisi d'implémenter trois méthodes afin de pouvoir les utiliser lors de nos campagnes de tests. Ces trois méthodes ont été sélectionnées de manière à couvrir différents compromis : des solutions optimales en termes de qualité, mais prenant beaucoup de temps, des méthodes très faibles en termes de performance par rapport à la qualité de la réponse, ainsi qu'une méthode intermédiaire permettant de trouver un équilibre entre performance et temps d'exécution. Voici les trois méthodes principales :

1. **optimize_task_order (Version optimale)** :
   — Cette méthode effectue toutes les permutations possibles de l'ordre des tâches assignées à un taxi.
   — Le coût total est calculé pour chaque permutation, et la solution avec le coût minimal est retenue.
   — **Avantage** : Produit une solution optimale garantissant le coût minimal.
   — **Limite** : Complexité exponentielle ; peu pratique pour un grand nombre de tâches.
2. **optimize_task_order_christofides (Approche approximative)** :
   — Utilise une approximation basée sur l'algorithme de Christofides pour résoudre le problème du voyageur de commerce (TSP).
   — Garantit un rapport de 3/2 par rapport à l'optimal.
   — **Avantage** : Solution rapide et relativement proche de l'optimal.
   — **Limite** : Bien que performante, elle ne garantit pas le coût minimal.
3. **greedy_task_order (Méthode gloutonne)** :
   — Sélectionne à chaque étape la tâche la plus proche à partir de la position actuelle du taxi.
   — **Avantage** : Très rapide et facile à implémenter.
   — **Limite** : Peut produire des solutions de faible qualité, loin de l'optimal, notamment pour des configurations complexes.

2.3.2 Choix des Méthodes d'Ordonancement

Pour la simulation, le choix de la méthode dépend des besoins spécifiques du scénario :

— **Scénarios nécessitant une solution optimale :**
   — La méthode `optimize_task_order` est utilisée pour des simulations à petite échelle (peu de taxis et de tâches), où la précision est cruciale et le temps de calcul n'est pas une contrainte.
— **Scénarios avec compromis temps/précision :**
   — La méthode `optimize_task_order_christofides` est privilégiée pour des simulations de taille moyenne. Elle permet de maintenir une bonne qualité de solution tout en réduisant le temps de calcul.

Lors de nos études de performance, nous testerons la qualité de ces approches en évaluant le compromis entre la qualité de la solution et le temps.

2.3.3 Allocation des Tâches

Nous avons implémenté des méthodes d'allocation simples permettant d'évaluer différentes stratégies d'affectation.

— **allocate_tasks_random** : Cette méthode attribue les tâches de manière aléatoire aux taxis disponibles. Bien qu'elle soit peu efficace en termes d'optimisation, elle permet d'établir une base de comparaison avec les autres méthodes.

— **allocate_tasks_opti** : Cette approche consiste à tester toutes les permutations possibles d'affectation des tâches afin de trouver la solution ayant le coût minimal. Bien que garantissant une allocation optimale, cette méthode est coûteuse en termes de complexité computationnelle et est donc utilisable principalement pour des instances de petite taille.

2.4 Intérface graphique

Nous avons constaté que, dans le cas de ce problème, il est très important de disposer d'un outil de visualisation, car cela facilite le suivi des agents. Ainsi, en utilisant Pygame, nous avons également développé une fonction permettant de visualiser le déplacement des agents dans notre environnement pendant l'exécution de leurs tâches.

La fonction `run_simulation` est au cœur du processus de simulation. Elle orchestre les différents composants du système, allant de l'allocation des tâches à l'ordonnancement des trajets des taxis, en passant par la gestion de la visualisation de l'environnement. Cette fonction est flexible et permet à l'utilisateur de choisir entre différentes méthodes d'allocation et d'ordonnancement, tout en offrant la possibilité d'afficher une interface graphique pour suivre l'évolution de la simulation.

2.4.1 Paramètres de la Fonction

— `allocation_method :` Définit la méthode d'allocation des tâches aux taxis. Ce paramètre correspond aux différentes stratégies d'allocation de tâches développées dans la Partie 3 (section 4) du projet.
  — 0 : `allocate_tasks_random`, méthode aléatoire.
  — 1 : `allocate_tasks_opti`, méthode optimale qui utilise la methode d'ordonnancement `optimize_task_order`.
  — 2 : `allocate_tasks_psi`, les enchères parallèles (PSI).
  — 3 : `allocate_tasks_ssi`, les enchères séquentielles (SSI).
  — 4 : `allocate_tasks_ssi_with_regret`, SSI avec regret.

— `ordonancement_method :` Ce paramètre détermine la méthode d'ordonnancement des tâches pour chaque taxi. Selon la valeur de `ordonancement_method`, les tâches des taxis sont réorganisées selon différentes stratégies d'ordonnancement :
  — 0 : `greedy_task_order`, une méthode rapide mais sous-optimale.
  — 1 : `optimize_task_order`, une méthode qui garantit l'ordonnancement optimal mais coûteuse en temps de calcul.
  — 2 : `optimize_task_order_christofides`, une méthode approximative efficace utilisant l'algorithme de Christofides.

— `visualisation :`
  Lorsque ce paramètre est défini à `True`, une interface graphique est lancée via

la bibliothèque pygame, permettant de visualiser l'évolution de la simulation en temps réel. Si `False`, la simulation est exécutée sans visualisation graphique, avec uniquement l'affichage des résultats dans la console.

— `delay :` Ce paramètre définit le délai entre chaque itération de la simulation lorsqu'elle est en mode de visualisation. Il permet de ralentir l'exécution pour suivre le déroulement de la simulation. Le paramètre est ajustable en fonction des besoins de l'utilisateur pour garantir une expérience de visualisation fluide et compréhensible.

— `step_by_step :` Lorsque ce paramètre est `True`, la simulation se déroule pas à pas. L'utilisateur doit appuyer sur la touche `Entrée` pour passer à l'étape suivante de la simulation, ce qui est particulièrement utile pour une analyse détaillée et pour le débogage.

— `verbose :` Si ce paramètre est défini à `True`, des informations supplémentaires sur l'état de la simulation sont affichées dans la console. Ce mode est utile pour suivre l'exécution détaillée du programme sans visualisation graphique.

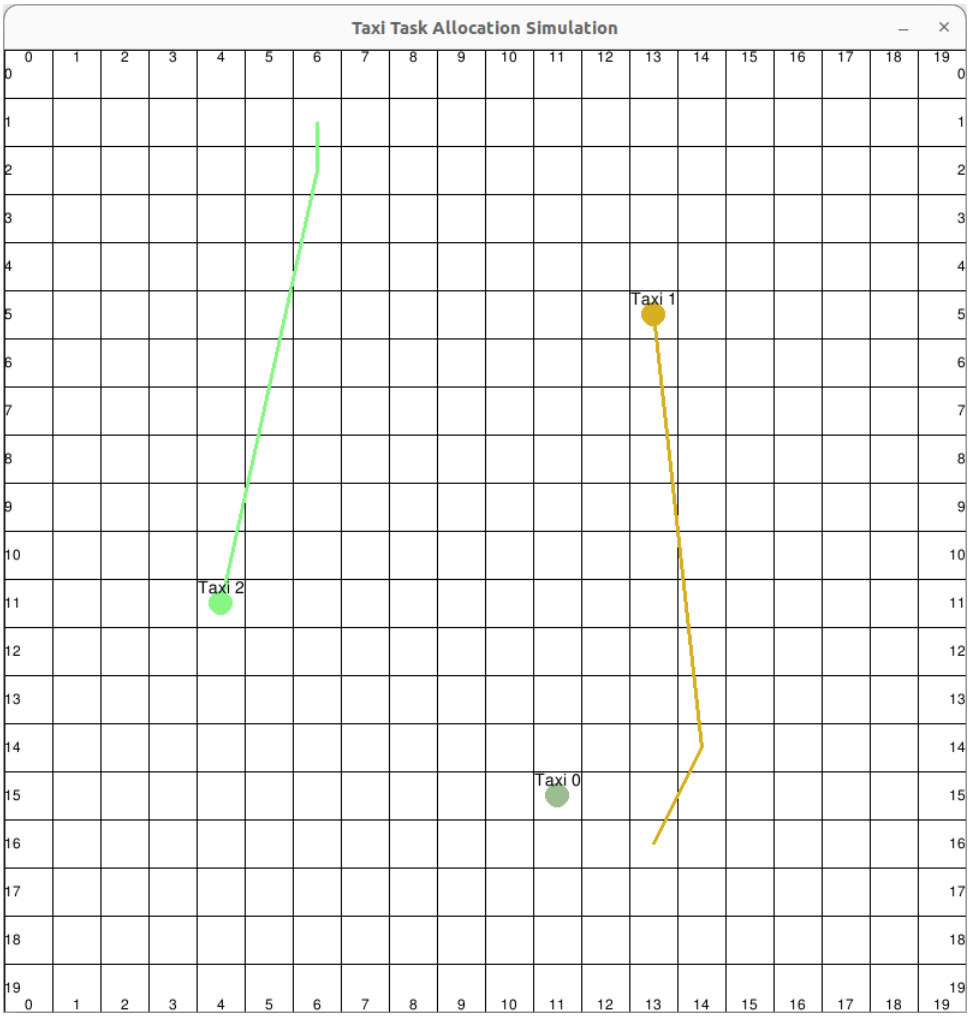

FIGURE 1 – Capture d'écran avec notre intérface graphique dans un environnement de taille 20x20 et 3 taxis

3 Partie 2

On genère le fichier YAML afin de décrire un problème d'affectation de tâches à des taxis, avec une minimisation d'un coût total. Voici une explication des variables et contraintes utilisées :

- **Les variables :** On a une variable pour chaque tâche. Chaque tâche doit être attribuée à un taxi parmi les taxis de problème. Chaque variable de tâche prend exactement une valeur.
- **Préférences des taxis pour les tâches :** Ces contraintes sont extentionnelles, ce qui signifie qu'elles limitent les valeurs possibles pour la valeur des tâches (bien que toutes les options soient possibles ici).
- **Contraintes d'affectation distincte :** Ces contraintes sont de type intentionnelles et imposent un coût supplémentaire si deux tâches sont affectées au même taxi. Ce cout supplémentaire correspond au cout de trandport le point final de premier tâche et point départ de deuxième tâche.
- **Coût des taxis :** Chaque taxi a un coût associé à chaque tâche qu'il effectue. Ce cout est le cout de l'exécution de tâche et le cout de transport de taxi vers le point départ du tâche.
- **Objectif :** L'objectif est de minimiser un coût total, qui dépend du coût d'affectation des tâches aux taxis et des couts supplémentaires si plusieurs tâches sont affectées au même taxi.

4 Partie 3

4.1 Heuristiques d'Insertion dans la Classe des Taxis

Dans cette partie, nous avons ajouté des heuristiques d'insertion à la classe des taxis afin d'optimiser la gestion et l'affectation des tâches aux taxis disponibles.

4.1.1 Heuristique de Prim

- **prim_heuristic** : L'heuristique de Prim est une approche inspirée de l'algorithme de Prim utilisé pour trouver un arbre couvrant minimal. Appliquée à l'affectation des tâches, cette heuristique vise à construire progressivement une solution en sélectionnant la tâche la plus proche en termes de coût cumulatif tout en minimisant la distance parcourue.

4.1.2 Heuristique d'Insertion

- **insert_task_heuristic** : Cette heuristique consiste à insérer une nouvelle tâche dans le planning d'un taxi de manière à minimiser le coût global de la tournée du taxi. L'insertion est faite en évaluant différents points d'insertion et en choisissant celui qui introduit le moindre impact en termes de distance et de temps de trajet.

4.2 Classe de l'Environnement et Allocation des Tâches

L'affectation des tâches aux taxis repose sur différents protocoles de négociation. Nous avons mis en place plusieurs stratégies permettant d'allouer les tâches de manière équilibrée et efficace.

4.2.1 Protocole PSI

— **allocate_tasks_psi** : Ce protocole repose sur une approche économique où chaque taxi soumet une offre pour une tâche donnée en fonction de son coût estimé. L'affectation se fait en tenant compte du prix le plus attractif tout en équilibrant la charge de travail des taxis.

4.2.2 Protocole SSI

— **allocate_tasks_ssi** : Dans cette approche, chaque taxi se voit attribuer un score basé sur plusieurs critères (distance, disponibilité, coût estimé). La tâche est attribuée au taxi ayant le meilleur score, garantissant une répartition optimisée et efficace.

4.2.3 Protocole SSI avec Regret

— **allocate_tasks_ssi_with_regret** : Ce protocole est une variation du SSI qui intègre un facteur de "regret", c'est-à-dire une estimation de l'impact de l'affectation d'une tâche sur les opportunités futures. L'objectif est de minimiser les regrets à long terme en tenant compte des affectations potentielles futures.

5 Etude de performances

5.1 Performances de la partie 2

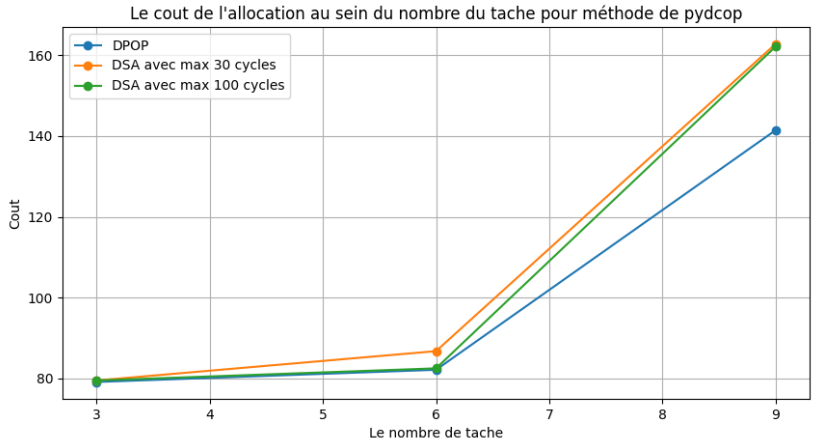

FIGURE 2 – Moyennes des couts d'allocation fait par DPOP et DSA

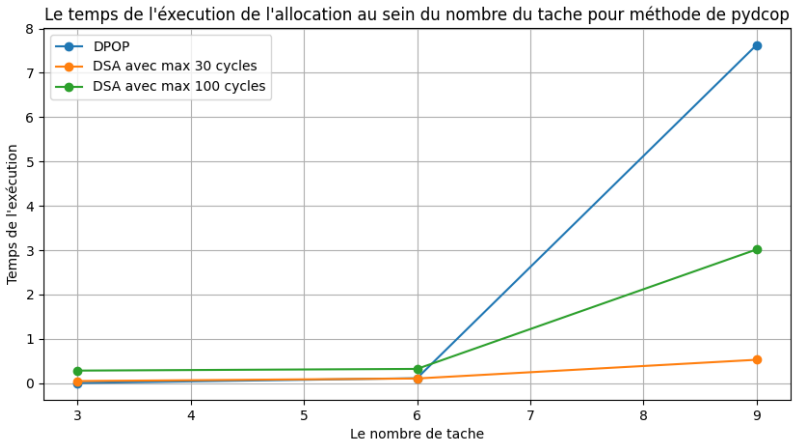

FIGURE 3 – Moyennes des temps d'éxécution d'allocation fait par DPOP et DSA

On a instancié 10 environnements. Dans chaque environnement, on a généré m tâches avec une période de 5 sur 30. Ensuite, on fait l'allocation de ces tâches en utilisant les algos DPOP et DSA. En fin, on calcule le coût et le temps d'exécution pour chaque environnement. Afin de créer une statistique, on fait la moyenne de 10 environnements.

A la lumière de nos graphes, on constate que la performance de DPOP est meilleure que celle de DSA avec max cycle 30 et max cycle 100. Dans l'algorithme de DSA, la performance augmente avec le nombre de max cycle.

En terme du temps de l'exécution, on observe que l'algo DPOP est plus coûteux que DSA avec le max cycle 30 mais beaucoup moins coûteux que l'algo DSA avec max cycle 100.

5.2 Performances de la partie 3

5.2.1 La taille de grille

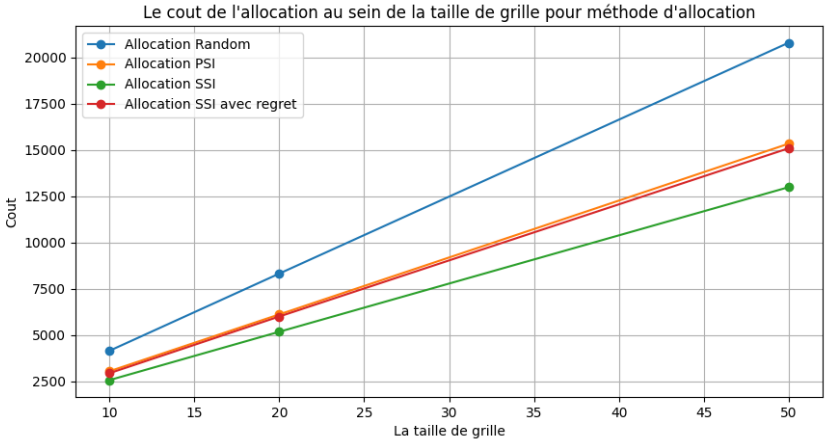

FIGURE 4 – Le coût de l'allocation des différentes méhodes en changeant la grille

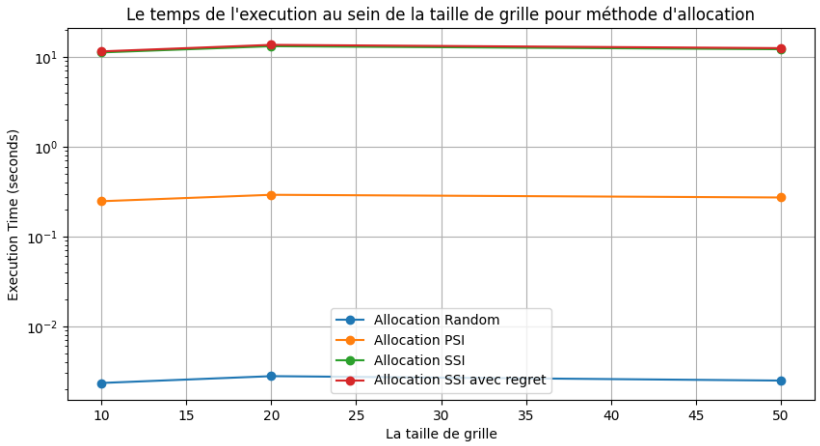

FIGURE 5 – Le temps de l'éxécution de l'allocation des différentes méhodes en changeant la grille

Le côut augmente avec la taille de grille (Les distances et les coûts des tâches peuvent augmenter en mesure de la taille de grille). Le temps de calcul reste stable.

5.2.2 Le nombre des tâches par taxi

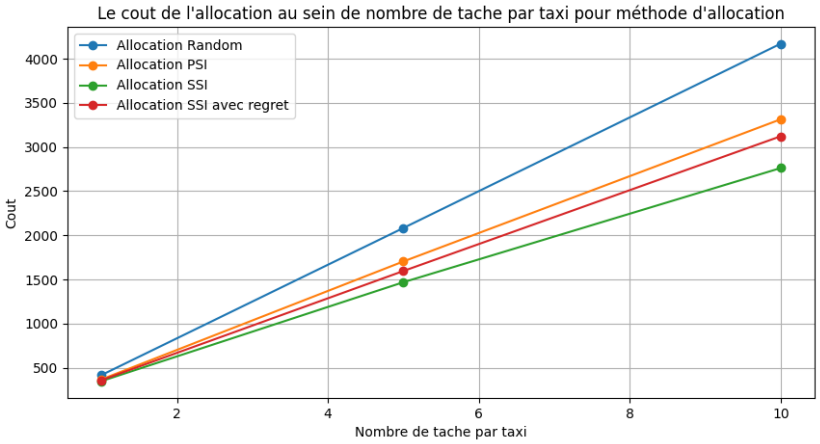

FIGURE 6 – Le coût de l'allocation des différentes méhodes en changeant le nombre des tâches par taxi

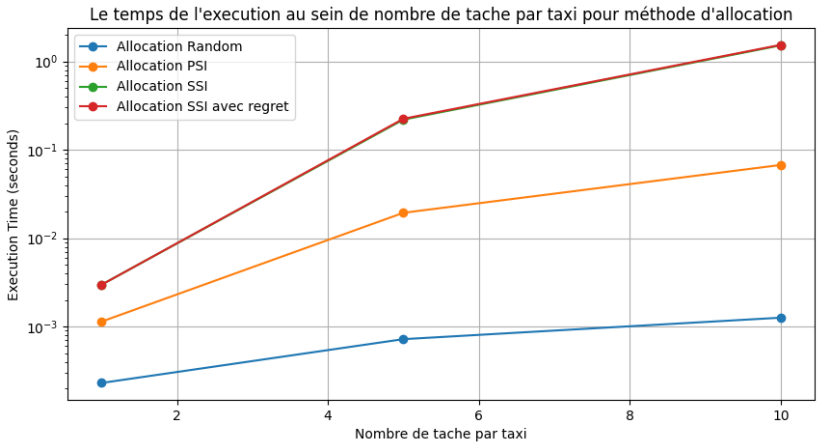

FIGURE 7 – Le temps de l'éxécution de l'allocation des différentes méhodes en changeant le nombre des tâches par taxi

Le côut augmente avec le nombre de tâche par taxi dans une façon linéaire. Le temps de calcul augmente aussi mais d'une façon moins rapide.

5.2.3 Le nombre des taxi

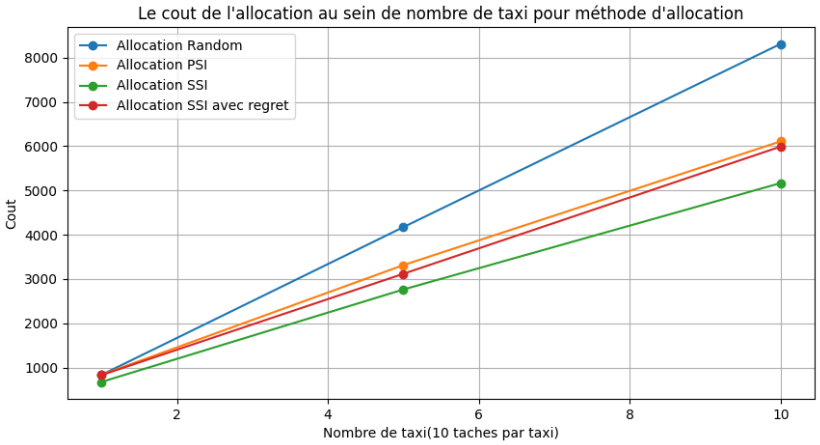

FIGURE 8 – Le coût de l'allocation des différentes méhodes en changeant le nombre des taxis

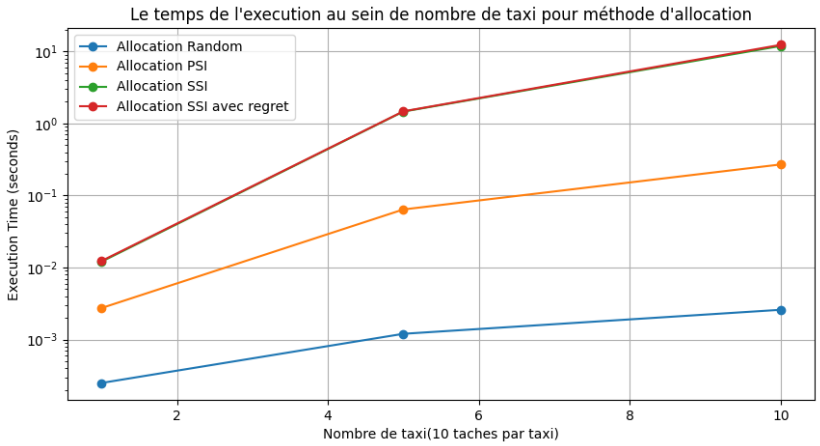

FIGURE 9 – Le temps de l'éxécution de l'allocation des différentes méhodes en changeant le nombre des taxis

Le côut augmente avec le nombre de taxi dans une façon linéaire. Le temps de calcul augmente aussi mais d'une façon moins rapide.

5.2.4 Interprétation des performances de la partie 3

L'allocation aléatoire est la moins efficace en termes de coût. L'allocation SSI est la plus performante, suivie de l'allocation SSI avec regret. L'allocation PSI est meilleure que l'aléatoire, mais moins efficace que les méthodes SSI.

L'allocation aléatoire est la plus rapide mais au détriment de l'optimisation du coût. L'allocation PSI est plus lente que l'aléatoire mais reste beaucoup plus rapide que les méthodes SSI. Les méthodes SSI et SSI avec regret nécessitent plus de temps de calcul, ce qui suggère des algorithmes plus complexes.

Si l'objectif est de minimiser le coût d'allocation, les méthodes SSI (et SSI avec regret) sont les meilleures, malgré un temps d'exécution plus élevé. Si l'objectif est de minimiser le temps d'exécution, l'allocation aléatoire est la plus rapide mais avec un coût élevé, tandis que l'allocation PSI représente un bon compromis entre coût et temps d'exécution.

5.3 Comparaison des performances entre les parties 2 et 3

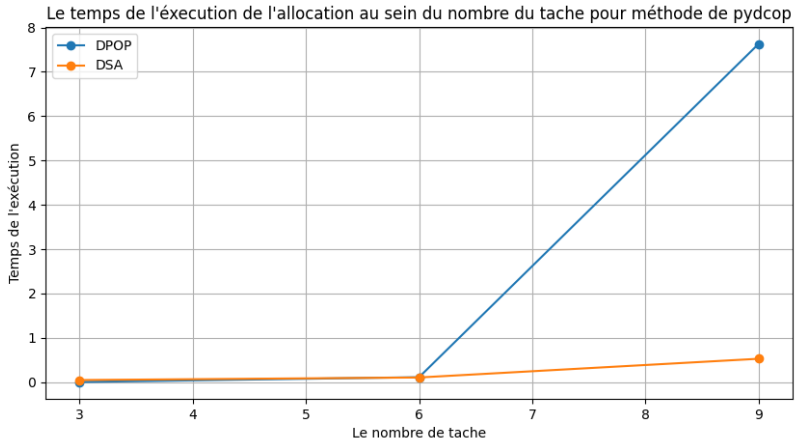

FIGURE 10 – Le temps de l'éxécution de l'allocation des différentes méhodes de pydcop en changeant le nombre des tâches

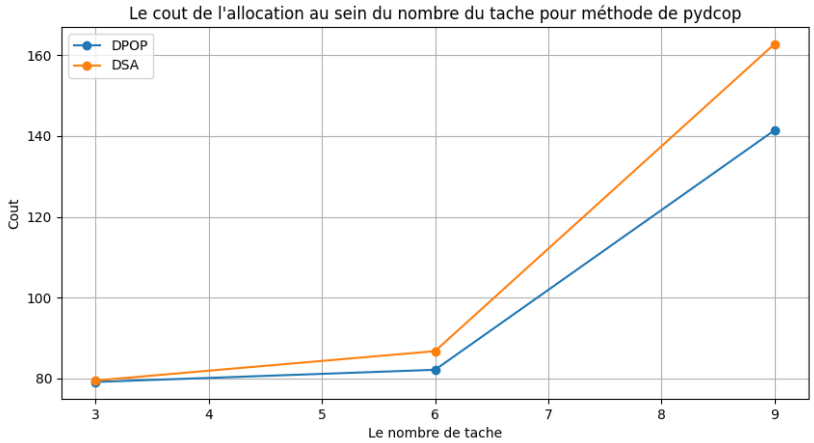

FIGURE 11 – Le coût de l'allocation des différentes méhodes de pydcop en changeant le nombre des tâches

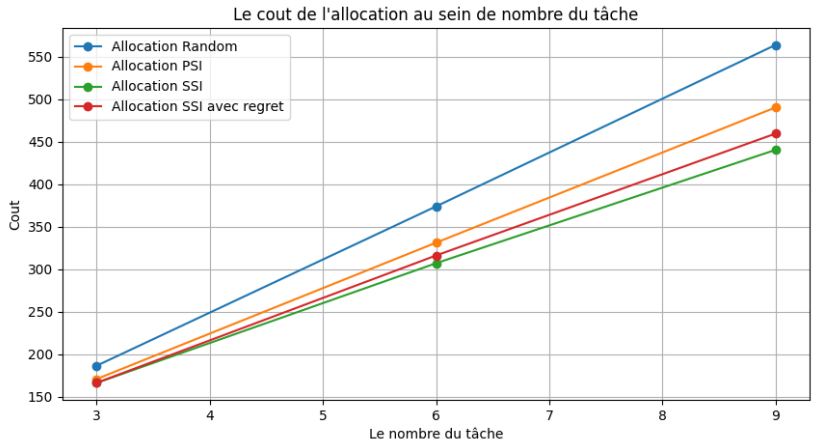

FIGURE 12 – Le coût de l'allocation des différentes méhodes en changeant le nombre des tâches

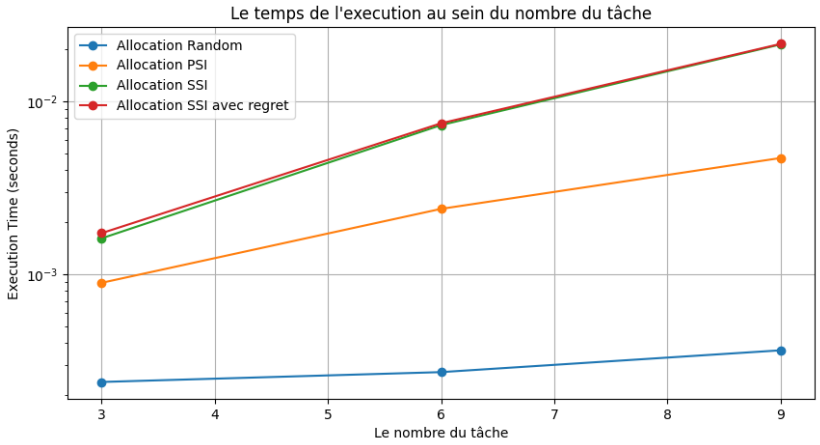

FIGURE 13 – Le temps de l'exécution de l'allocation des différentes méhodes en changeant le nombre des tâches

En terme de temps d'exécution, les méthodes de pydcop sont inefficaces. En d'autre part, les méthodes implementés par nous sont plus rapides. Quand on parle du coût, le méthode DPOP nous garantit d'avoir le résultat optimal mais trop coûteux au niveau tempoorel. Les méthodes SSI et SSI avec regret nous offrent une solution 2-approché. La méthode PSI n'offre aucun garantie pour la qualité de la solution mais elle est meilleure que l'approche aléatoire.

6 Conclusion

Dans ce projet, nous avons exploré l'allocation dynamique des tâches pour une flotte de taxis dans un environnement multi-agents. À travers différentes approches d'ordonnancement et d'affectation, nous avons analysé l'impact des heuristiques et des protocoles de coordination sur les performances du système. Nos résultats montrent que les méthodes basées sur les enchères séquentielles (SSI et SSI avec regret) offrent les meilleures performances en termes de minimisation des coûts, bien qu'elles nécessitent un temps de calcul

plus important. À l'inverse, l'allocation aléatoire, bien que rapide, est significativement moins efficace en matière d'optimisation. Le protocole PSI constitue un bon compromis entre rapidité et coût d'affectation. L'étude des performances selon la taille de la grille, le nombre de taxis et le nombre des tâches par taxi a permis de confirmer la robustesse des méthodes proposées tout en mettant en évidence leurs limites.