GASSION Axel INFO-5A
GAUFRETEAU Simon

SMA - TP2 - Partie 2

Lien vers une vidéo d'exécution de notre code : https://youtu.be/UBHjC9sXxDE
Si vous souhaitez l'utiliser, vous pouvez lancer la **MainApplication** depuis un IDE.
L'objet de type **Environnement** possède les paramètres importants :

- **n** & **m** : dimensions de la grille
- **nA, nB, nC** : nombre d'agents de chaque type
- **nbAgents** : nombre d'agents
- **seed** : seed du Random, utilisé pour l'évaluation des résultats

Code et implémentation

Nous avons bien entendu gardé la base implémentée dans la première partie du TP.

En plus de cela, nous avons ajouté 200 objets *C*, représentés en vert sur l'interface graphique, qui sont transportables uniquement par le biais de la coopération entre deux agents.
Nous avons choisi de faire en sorte qu'un seul agent ait l'initiative. Ainsi, lorsqu'un Agent sur un objet *C* décide qu'il doit être déplacé, il va attendre quelque temps de l'aide.
Si un Agent arrive sur une case adjacente, alors il est considéré comme assistant.
L'assistant est retiré de la grille pour éviter certains problèmes, et de manière à ce qu'il soit considéré comme étant sur la même case que l'Agent qu'il aide.

Lorsque l'objet est déposé, l'Agent principal dépose aussi son assistant sur une case adjacente. La mémoire de ce dernier sera similaire à celle de l'Agent principal.

Afin de garder une implémentation assez claire, nous avons dupliqué et ajusté la méthode d'action de l'agent, qui sera à présent **actionV2()**.
Pour ce qui est des paramètres importants, nous avons ajouté les suivants :

- **isAssisting** : Un booléen pour savoir si l'Agent aide ou non un confrère
- **assistantAgent** : Un agent qui est stocké en tant qu'assistant de celui qui déplace l'objet C.
- **isWaiting** : Un booléen pour savoir si l'Agent attend de l'aide.
- **waitForHelp** : Un entier correspondant au nombre de tours d'attente de l'Agent.
- **helpCalls** : Un entier correspondant au nombre de fois où l'Agent a rafraîchi les phéromones environnantes pour attirer les autres.
- **waitRate** : Une valeur servant à calculer une probabilité d'attente ou non d'un agent.

Globalement, l'agent possède maintenant 4 états :

- **Actif** : Il n'attend pas, n'aide pas, ne porte rien; Il se déplace sur la grille.
- **Porteur** : Il est actuellement en train de porter un objet (avec de l'aide ou non).
- **Assistant** : Il est actuellement en train d'aider un autre Agent.
- **En attente** : Il attend de l'aide sur un objet lourd *C*.

Lorsque l'Agent est **Actif**, il possède un comportement quasi similaire à la partie précédente. Il déambule sur la grille à la recherche d'objets à déplacer.
Cependant, il peut maintenant être attiré par des phéromones et venir en aide à un allié. Pour cela, lorsqu'il se déplace, avant de choisir un mouvement aléatoire, il va regarder s'il existe des phéromones dans des cases environnantes. Si c'est le cas, il ira sur la case la plus attirante.

Lorsque l'Agent est **Porteur**, son comportement est simple. Il se déplace jusqu'à trouver un point qu'il juge adapté pour déposer son objet, sans être attiré par les phéromones.

Lorsque l'Agent est **Assistant**, il n'effectue aucune action jusqu'à être libéré par l'Agent qu'il aide, ce qui se traduit par le dépôt de l'objet transporté.

Lorsque l'Agent est **en attente,** il peut agir de deux manières différentes.
Il peut choisir de continuer à attendre. Cela est représenté par un calcul de probabilité qui décroît en fonction du temps d'attente : *Insérer formule*.
S'il attend, il peut décider de mettre à jour les phéromones environnantes. Cela est aussi décidé avec une probabilité, qui décroît en fonction du nombre de fois où il a actualisé ces dernières.
S'il n'attend pas, il abandonne la case et devient **Actif**.

Pour adapter nos fonctions de prise et dépôt d'objets, nous avons décidé d'implémenter un booléen **coop** dont la valeur est *true* lorsque l'objet en question sera sujet à une coopération entre des agents.

Les phéromones sont stockées sur une grille à part, en parallèle.
Les valeurs sont comprises entre 0 (absence) et 1 (case de l'objet).
Les valeurs sont équidistantes, en fonction de la distance de diffusion des phéromones, c'est-à-dire que pour une distance de 1, on a :

1/2	**1/2**	**1/2**
1/2	**1**	**1/2**
1/2	**1/2**	**1/2**

Pour une distance de 2 :

1/3	1/3	1/3	1/3	1/3
1/3	2/3	2/3	2/3	1/3
1/3	2/3	1	2/3	1/3
1/3	2/3	2/3	2/3	1/3
1/3	1/3	1/3	1/3	1/3

Ce choix nous semblait cohérent, et nous permet des réglages manuels pour la couleur des phéromones au niveau de l'affichage graphique.

Enfin, les phéromones s'évaporent après chaque itération :

$$ValeurPhéro = \frac{Valeur\ Phéro}{1{,}1}$$

Si la valeur passe sous un seuil prédéfini : $\frac{1}{28}$ alors on passe la case à 0.

Enfin, nous avons légèrement amélioré l'interface graphique.
Chaque case possède une bordure, représentant la présence d'un Agent ou non.
Si le cadre est noir, un Agent est sur la case.
Si le cadre est Magenta, deux Agents sont sur cette case et portent un objet en coopération.

Aussi, les phéromones sont représentés en nuances de jaune. Plus le jaune est important, plus le phéromone est puissant, plus il est blanc moins il est puissant.

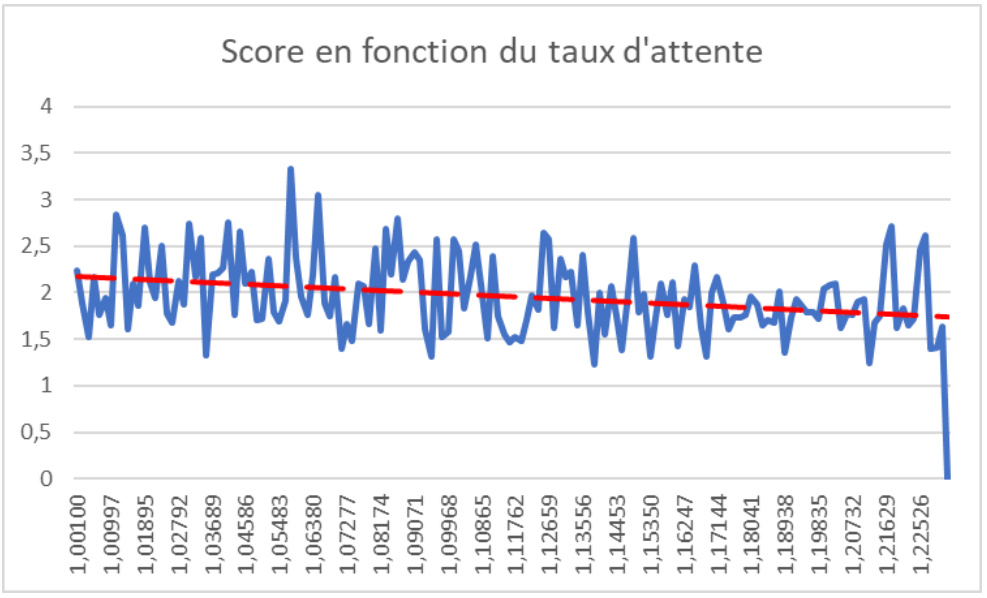
Etude de l'importance des paramètres

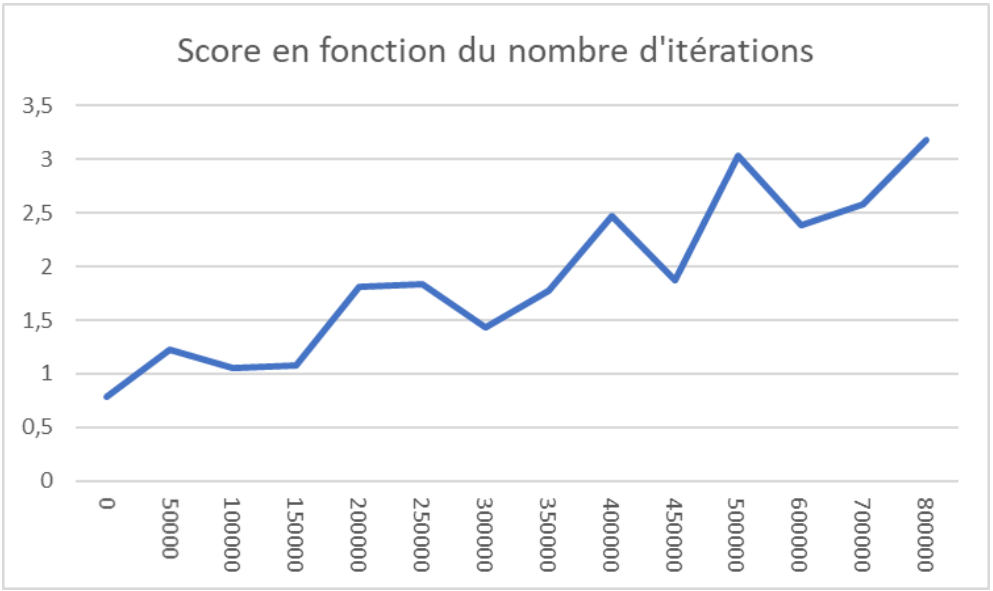
Après l'implémentation de tous ces éléments, nous avons effectué une analyse simple des différents paramètres.
Dans un premier temps, nous avons dû choisir une fonction de scoring pour évaluer la qualité d'un tri. Le sujet étant assez différent des exemples de clustering ou de tri généralement utilisés, nous avons dû implémenter une fonction personnalisée, sa formule vous est donnée ci-dessous :
Score = taille moyenne des cluster / distance moyenne entre clusters de même type

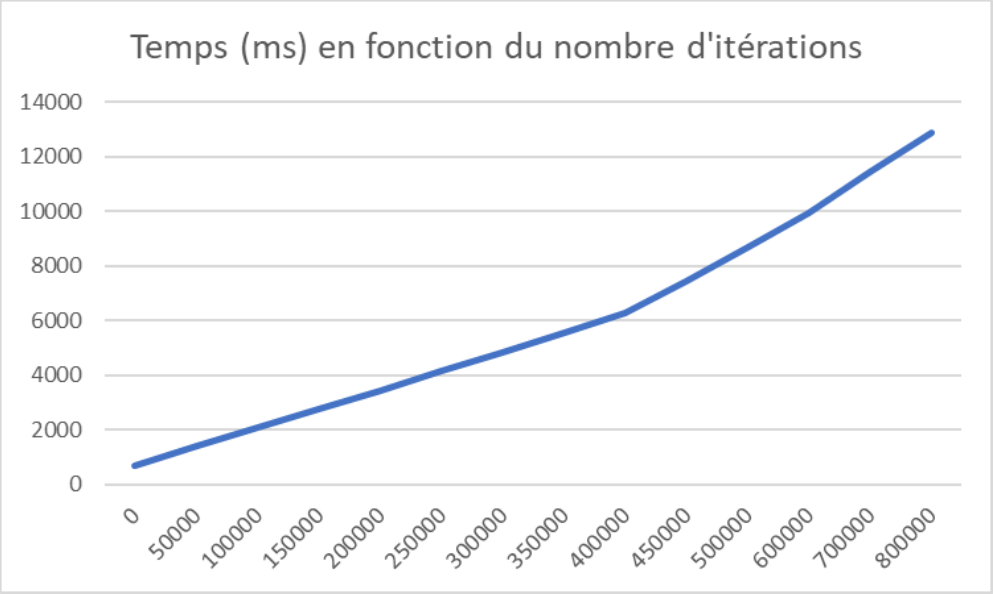
Un bon score maximise la taille des clusters et minimise la distance entre clusters du même type.
Pour pouvoir utiliser cette formule, nous avions besoin de clusters séparés par type, ce que nous n'avions pas avec l'implémentation de base. Nous avons donc implémenté un algorithme simple et rapide de création de cluster.
Cet algorithme se base sur une exploration type DFS (matrice de visite et file FIFO) pour explorer la grille en regroupant les objets de proches. Contrairement aux algorithmes de

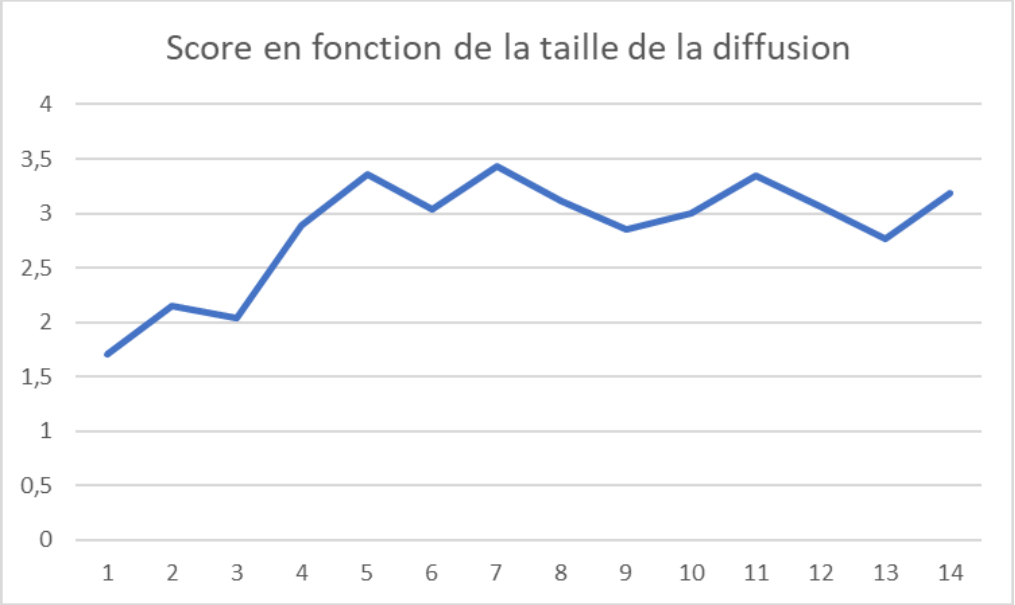
cluster classique, cette implémentation permet de regrouper les objets par **leur type,** ce qui est l'élément le plus important du tri réalisé par les fourmis.

Avec cet algorithme de clustering et le calcul de score, nous avons pû effectuer une analyse des paramètres utilisés par notre implémentation. Voici les résultats de cette analyse :









Toutes les données ainsi que les courbes sont disponibles dans le fichier Excel en annexe de ce document.

Analyse :

Comme on pouvait s'y attendre, le nombre d'itérations est le facteur déterminant de l'efficacité du tri, mais aussi celui du temps d'exécution.
Comme le score et le temps augmentent linéairement, on prendra toujours le maximum d'itérations pour le temps qui nous est accordé pour réaliser le tri.

Concernant le taux d'attente, on peut voir une tendance décroissante quand celui-ci augmente. On choisira donc un taux proche de 1 mais toujours supérieur strictement.

La taille de la diffusion semble donner de meilleurs résultats autour de 6, un nombre qui semble raisonnable par rapport à la taille de la grille (50 lors des tests).