

# Exemple de tri collectif multi-agents

ORTEGA Guillaume, BONNAVAUD Hedwin

Enseignante : HASSAS Salima

## Choix techniques

### Modèle et algorithme

Le sujet de ce TP et l'article associé proposait plusieurs manières d'implémenter le projet. Cela nous a laissé le choix d'autoriser les agents à circuler sur des cases comprenant des objets, et de demander aux agents de choisir de prendre ou de déposer un objet, soit en fonction de sa mémoire, soit en fonction de son voisinage.

Puisque nous avons choisi de réaliser notre projet sous NetLogo, et que dans son environnement les tortues se déplacent de manière continue, nous avons choisi de dissocier les mouvements des termites des positions des objets. Comme si les objets en questions étaient trop petits pour gêner l'évolution des termites.

Concernant les autres choix que nous devons faire, nous ne savions pas quelle méthode était la plus performante concernant la rapidité et l'efficacité du tri. Nous avons donc ajouté des paramètres dans l'interface, afin que l'utilisateur choisisse lui même le mode de prise et de dépôt. Ainsi, nous avons pu tester toutes (ou presque) les combinaisons de paramétrage pour cet algorithme de tri, afin de trouver la meilleure combinaison.

Nous avons également laissé l'utilisateur choisir les paramètres de l'environnement (taille de grille, nombre d'objets, nombre d'agents), mais aussi ceux de l'algorithme (valeur de  $k+$ ,  $K-$ , epsilon, ...).

Enfin, tout en jouant avec ces combinaisons, nous avons pris soin de prendre note des résultats que vous pourrez trouver [ici](#).

### Mesure de l'efficacité

Évidemment, pour connaître l'efficacité d'un ensemble de paramètres, nous ne pouvions pas nous contenter de regarder le déplacement des termites et la position des objets sur l'écran. Nous avons donc décidé d'implémenter une mesure de l'efficacité du tri.

Cette mesure, qui s'actualise et s'affiche régulièrement dans NetLogo, pourrait surement être plus pertinente, mais elle sera suffisante pour comparer l'efficacité de deux ensembles de paramètres. Nous avons pu remarquer que l'évolution de cette valeur évoluait de la même manière que la formation de "tas" dans l'environnement.

Il s'agit de la note de positionnement moyenne sur l'ensemble des cases (ou patch). Ces dernières ont une note qui dépend de leurs voisins.

Si un patch ne contient pas d'objet, on ne lui attribue pas de note. Nous pouvons expliciter cette règle en disant que notre objectif n'est pas de mettre les cases vides ensemble, mais de trier les objets.

La case recevra un malus de 1 pour chaque case vide parmi ses voisins. Nous cherchons ainsi à attribuer une meilleure note à un environnement présentant un seul tas pour chaque type d'objets, plutôt que plusieurs petits tas.

Enfin, elle recevra un malus de 3 pour chaque case voisine contenant un objet différent du sien, et un bonus de 2 si cet objet est du même type que le sien. En effet, si une case compte les deux types d'objet parmi ses voisins proches, on ne peut pas considérer que l'environnement est trié.

Les malus attribués pour des objets différents se doivent donc d'être plus grands que les bonus attribués aux objets similaires.

Nous utiliserons donc cette notation pour évaluer l'environnement à chaque instant en faisant la moyenne des notes de toutes les cases.

Il est important de souligner que cette notation de l'environnement n'influence pas le comportement des agents, mais sert uniquement à évaluer l'avancée de leur travail.

```
;;-----;;
;;          CALCULATE THE CLUSTERISATION
;;-----;;

to set_clusterisation
  let my_color black
  let patches_evaluated 0
  let total_grade 0
  ask patches [
    if pcolor != black [
      set patches_evaluated patches_evaluated + 1
      set my_color pcolor
      ask neighbors4 [
        ifelse my_color = pcolor [ ;; my_color = couleur du patch central, et pcolor = couleur du voisin courant.
          ;; Si le voisin courant a un objet du même type que celui de la case évaluée
          set total_grade (total_grade + 2)
        ]
        [
          ifelse pcolor = black [
            ;; Si le voisin courant n'a pas d'objet
            set total_grade (total_grade - 1)
          ]
          0 ;; Si le voisin courant a un objet d'un autre type que celui de la case évaluée
          set total_grade (total_grade - 3)
        ]
      ]
    ]
  ]
  set clusterisation_value total_grade / patches_evaluated
end
```

Figure 1 : algorithme de calcul du niveau de tri dans l'environnement

## Résultats

Maintenant que nous avons un affichage régulier de l'avancement du tri au cours d'une simulation, nous pouvons utiliser ces informations pour observer et comparer la vitesse et la performance de l'algorithme selon l'ensemble de valeurs choisies pour ses paramètres.

Nous allons les étudier un par un, et pour cette raison, nous conserverons un ensemble de valeurs par défaut, et ne changerons à chaque fois que les valeurs mentionnées.

En utilisant les valeurs conseillées dans le sujet de TP et dans l'article, nous pouvons choisir les valeurs suivantes :

- Epsilon : 0,1
- Taille de la mémoire de chaque termites 10
- K+ : 0,1
- K- : 0,3
- Nombre de déplacements des termites à chaque itérations (sans leur demander de prendre / poser) : 1

Pour plus d'informations sur ces paramètres, vous pouvez vous référer au sujet du TP ainsi qu'à l'article.

Pour ne pas tomber dans des cas particuliers lors de nos simulations, nous triplerons chacune d'elles et n'utiliserons que la mesure moyenne du tri de ces trois simulations.

Enfin, nous ne présenterons pas tous les résultats pour tous les paramètres, mais seulement ceux qui nous semblent les plus pertinents. D'autres graphiques et données de simulations sont rangés dans notre [tableau des résultats](#).

#### Mesure de l'efficacité de l'algorithme en fonction du temps

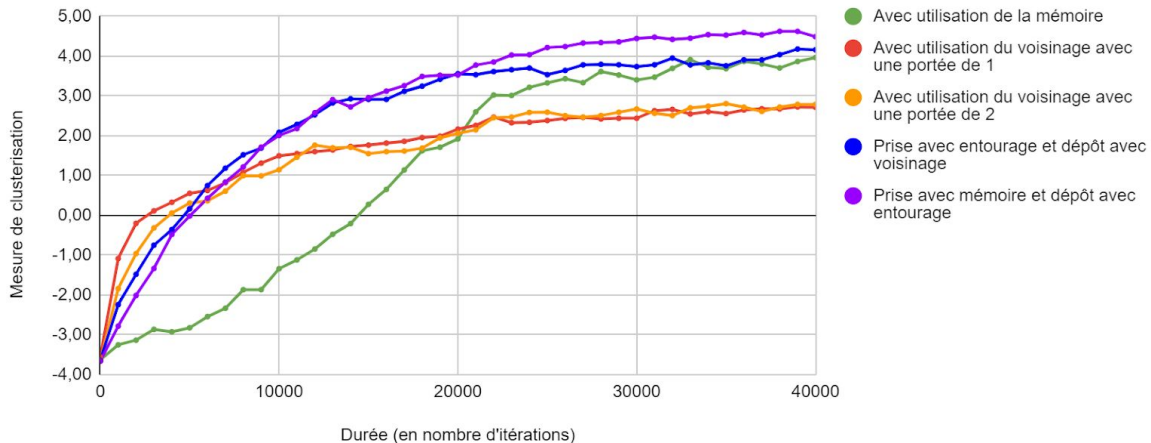


Figure 2 : Évolutions moyennes du tri dans plusieurs simulations.

Afin de répondre à la question qui nous brûle les lèvres depuis la lecture du sujet, à savoir "l'agent doit-il prendre sa décision en fonction de son environnement ou en fonction de sa mémoire ?", désormais il semble clair que le plus optimal est l'utilisation des deux méthodes (cf. figure 2).

Il est important de préciser que les courbes bleues et violettes affichent les évolutions moyennes de 6 simulations chacune (contre 3 simulations pour les autres courbes). De tous nos tests, l'utilisation de la mémoire, pour le choix de la prise d'un certain objet, et du voisinage, pour le choix de son dépôt, reste devant la méthode inverse (courbe bleue).

Mais alors pourquoi l'utilisation de la mémoire est-elle plus adaptée à la prise, et l'entourage au dépôt ?

#### Méthode de prise

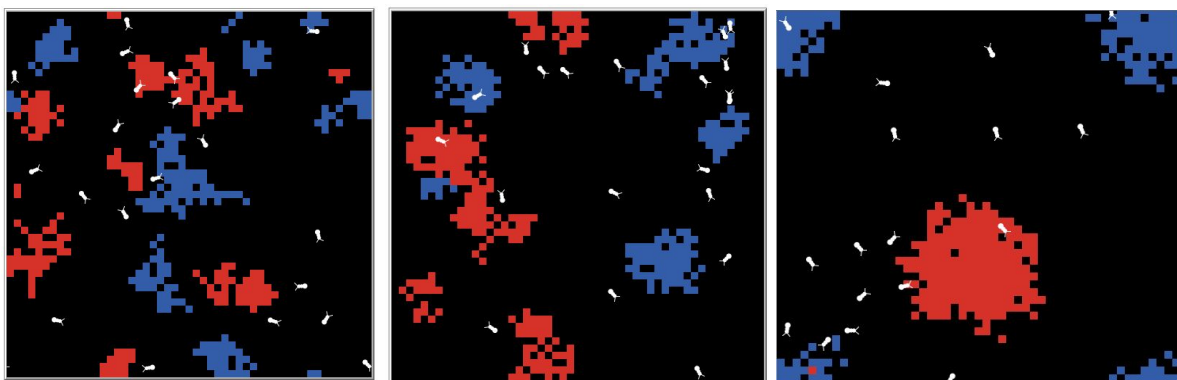


figure 3 : environnements à la fin de l'application de différents algorithmes

Sur ces images, nous pouvons voir le résultat de l'algorithme après 50000 itérations. Avec comme méthode de prise, de gauche à droite : voisinage avec une portée de 1, voisinage avec une portée de 2, utilisation de la mémoire uniquement. La méthode de dépôt est toujours le voisinage dans les trois cas.

Tout en observant ces trois environnements, il est important de rappeler que selon la fonction donnant la probabilité de prise, plus l'objet a été trouvé dans la liste utilisée (mémoire ou voisinage) moins l'agent sera tenté de le prendre.

Ainsi, lorsqu'on passe d'un voisinage de portée 1 à un voisinage de portée 2, un agent au bord d'un petit tas sera tenté de le casser car sa portée ne sera pas replis par le petit tas auquel il fait face. Avec une portée de 2, les tas les plus petits sont donc plus propices à être démantelés que les gros. Ce phénomène favorise donc un regroupement de certains tas.

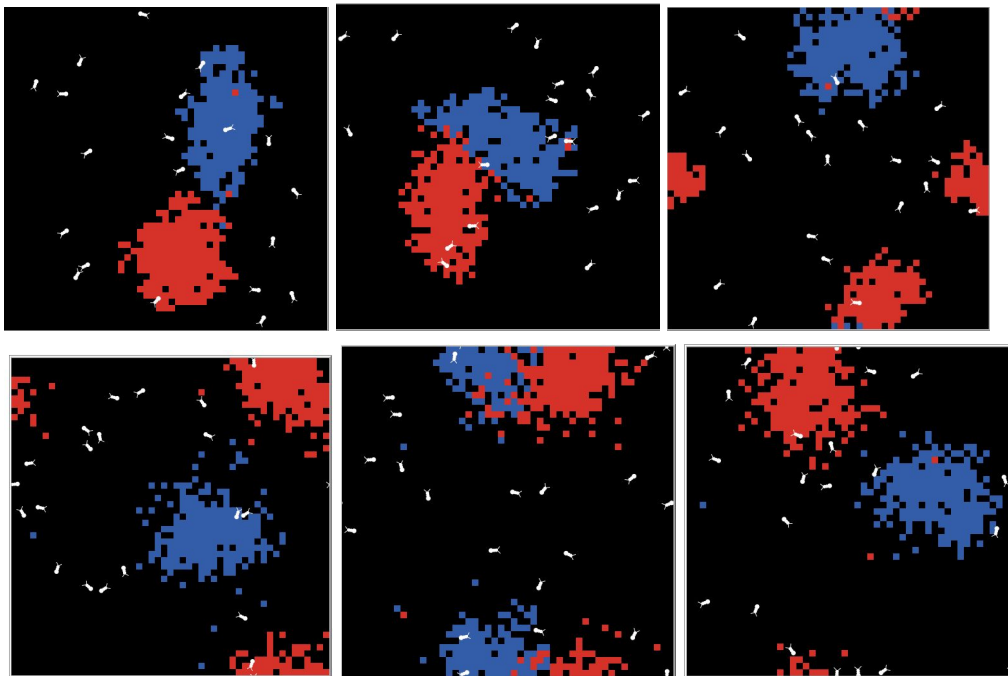
Si on demande à l'agent de ne se fier qu'à sa mémoire, il sera alors moins influencé par des tas volumineux, que s'il utilise une zone d'observation autour de lui. En effet, un agent qui arrive de l'extérieur d'un tas et qui y rentre, aura l'impression de ne n'avoir jamais rencontré ce type d'objet et sera tenté de le prendre. Seul un agent se trouvant déjà à l'intérieur du tas n'y sera pas tenté. La prise est donc grandement facilitée par cet algorithme. Etant donné que la fonction de dépôt a pour rôle de déposer les objets ensemble, il est donc moins probable d'avoir deux tas distincts d'un même type d'objet après stabilisation de l'algorithme.

Cette évolution suffit à montrer la différence observée sur le graphique, car notre fonction d'évaluation donne des malus pour les objets adjacents aux cases vides. Un tas plus gros aura une meilleure note que plusieurs tas plus petits.

## Méthode de dépôt

De la même manière, il est désormais aisé de penser qu'un agent qui utilise sa mémoire pour savoir si il doit déposer l'objet qu'il porte, ne va accepter de déposer son objet que lorsqu'il aura rencontré des objets du même type sur son passage passé. Il aura donc plus tendance à déposer ses objets en sortant et en s'éloignant d'un cluster. Ce comportement amène donc à un éparpillement des objets.

Nous pouvons observer ce comportement sur les résultats de ces 6 simulations.



*Figure 4 : environnements à la fin de l'application de différents algorithmes*

De la même manière, nous pouvons voir sur les images ci-dessus, les résultats d'un choix de dépôt en fonction du voisinage (en haut) et en fonction de la mémoire (en bas). Nous pouvons alors remarquer que les tas sont plus compacts lorsqu'on utilise l'entourage de l'agent pour choisir de déposer un objet.

Nous pouvons conclure de ces observations que le voisinage et l'entourage sont des ensembles de données toutes deux intéressantes lorsqu'il est question de prise d'un objet, en

fonction de quel type de tri nous choisissons (nous pourrions être tentés d'utiliser l'entourage si nous souhaitons plusieurs petits clusters au lieu d'un seul et unique pour chaque type d'objet). Cependant, l'utilisation de l'entourage est bien plus intéressante pour le dépôt car elle permet d'avoir des clusters beaucoup plus nets et plus denses.

## Autres paramètres

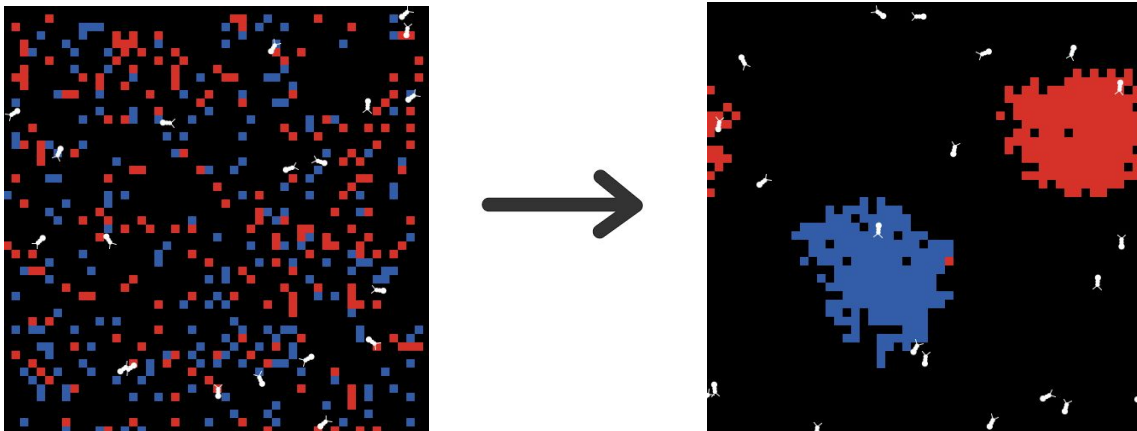
Inclure un taux d'erreur est souvent intéressant car cela peut nous permettre de sortir d'un minimum local si la situation est bloquée. Si inclure un taux d'erreur dans la formule, avoir un taux d'erreur trop grand est très néfaste pour l'efficacité de l'algorithme.

En observant la formule de la prise d'un objet :  $P_{\text{prise}} = (k^+ / (k^+ + f))^2$  nous pouvons remarquer que plus nous augmentons la valeur de  $K^+$ , moins la proportion d'objets du même type dans la mémoire (car c'est bien la mémoire que nous utilisons pour la prise) a un impact sur le choix de la prise ou non d'un objet. S'il semble évident que neutraliser  $f$  a un impact négatif sur l'efficacité du tri, nous avons pu remarquer qu'augmenter un peu  $K^+$  est plutôt positif.

En effet, nous avons trouvé comme valeur optimale (à +/- 0,05 près)  $K^+=0,15$  contre la valeur de 0,1 que nous utilisions jusqu'à maintenant. Pour plus de détails, nous vous invitons à regarder nos [résultats](#), onglet  $K^+$ .

De la même manière, nous pouvons trouver 0.35 comme valeur optimal pour  $K^-$ .

Avec toutes ces optimisations, voici ce que nous obtenons au bout de 50000 itérations sur une grille de taille 50x50 avec 200 objets de chaque type et 20 agents.



## Pour aller plus loin

Une autre méthode pour créer un agent capable de trier des objets, aurait pu être d'entraîner chacun d'eux séparément avec un Q-learning utilisant comme fonction de récompense, la fonction d'évaluation du tri de l'environnement que nous avons présentée avec nos choix techniques.

Si cette fonction pourrait-être améliorée pour mieux évaluer la réalisation de l'objectif par les agents, elle pourrait être suffisante pour gagner en efficacité sur toutes les simulations que nous avons pu faire jusqu'ici.