UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Université de Genève

Travail de Bachelor

Multi-agent systems - The exploration-exploitation dilemma.

Noan Goulard

supervisé par
Prof. Roland Bouffanais

June 10, 2024

Table des matières

1 Introduction

Au fil des années se sont démarquées diverses technologies informatiques pour répondre à des problèmes toujours plus nombreux : Internet of things, Blockchain, Intelligence artificielle et beaucoup d'autres. Parmis ces solutions, l'une d'elles se montre particulièrement digne d'intérêt pour proposer des solutions a des problèmes dans des systèmes distribués. Inspirée de la nature, ou mise en place en imitant celle-ci sans même s'en rendre compte, cette solution est celle des systèmes multi-agents (MAS). Colonies de fourmis, vols d'oiseaux, l'existence de systèmes autonomes composés d'agents distincts s'étend évidemment à notre propre société, créant et résolvant divers problèmes.

Bien sûr, détailler chacun de ces systèmes est impossible, cependant s'intéresser à la capacité de ces MAS à résoudre des tâches dynamiques complexes lorsqu'elles sont distribuées dans l'espace sur un plan général l'est. La résolution de ce genre de problèmes implique le design de stratégies de coopération entre agents, une tâche reconnue difficile et non triviale qui, entre autres, soulève l'important dilemme de l'exploration / exploitation. Une balance entre ces deux caractéristiques de notre système est d'une importance capitale, en se basant sur de récents résultats, pour améliorer l'adaptivité et la flexibilité de ces systèmes multi-agents.

Dans le cadre de ce travail, nous allons ainsi dans un premier temps fournir un état de l'art soulignant les concepts importants concernant les MAS ainsi que les premiers résultats et définitions qui nous aideront à éclaircir notre compréhension du dilemme d'exploration / exploitation. Par la suite, nous aurons l'occasion de se pencher sur un code déjà existant, permettant l'analyse de simulations d'un essaim d'agents dans le cadre de la traque de cibles mouvantes. Cela servira d'introduction à la recherche des leviers impactant la dynamique de notre système et la compréhension de ceux-ci, nous permettant de poursuivre par la recherche de moyens permettant de controler avec plus d'efficacité notre dichotomie exploration / exploitation et d'implémenter ces stratégies de sorte à permettre un equilibrage dynamique entre ces deux états. Nous basant sur ces résultats ainsi que sur notre état de l'art, nous pourrons conclure sur l'efficacité de nos implémentations, leurs potentielles améliorations, et de manière plus générale les perspectives d'évolution et de compléments de recherche à partir de nous résultats.

2 Etat de l'art

2.1 Systèmes multi-agents - Définitions

Un système multi-agents, que pour des raisons de praticité nous appellerons MAS pour suivre le vocabulaire de l'essentiel des études sur ce sujet, est un outil qui comme l'indique son nom est basé sur de multiples agents uniques formant, dans un environnement dépendant de notre problème, un système non centralisé exploitable pour résoudre des problèmes. Les domaines d'application sont divers : logistique, management, communication, santé, villes intelligentes etc (Julian (2019)). Cette source nous permet aussi de définir avec plus d'aisance ce genre de systèmes, souligner l'importance de la communication entre ces agents, la coordination de leurs activités, pour produire un résultat exploitable. Ces agents seront souvent programmés suivant divers frameworks, certains pour s'attaquer à des problématiques plus poussées (Berger-Tal et al. (2014)), avec ou sans, en fonction de nos besoins, une capacité d'apprentissage déléguée à notre système dans son ensemble par son fonctionnement, ou parfois à l'agent en lui même comme le montrent les schémas suivants. Bien sûr, ces schémas minimalistes ne prennent pas en compte nombre d'autres problématiques, entre autres la communication entre nos agents.

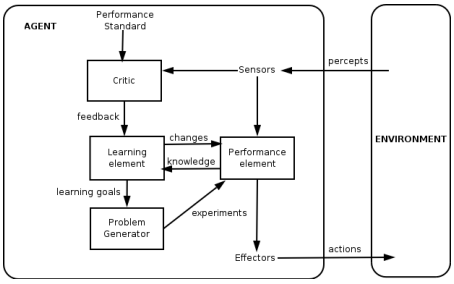

Figure 1: Schéma d'un agent avec Learning
Source : https://en.wikipedia.org/wiki/Multi-agent_system

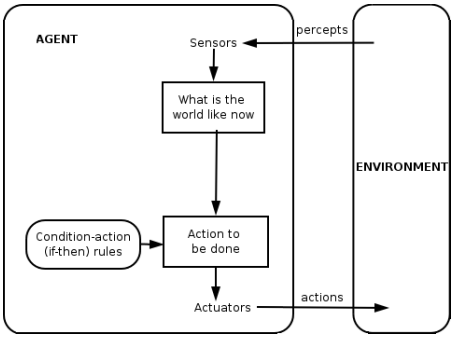

Figure 2: Schéma d'un agent sans Learning
Source : https://en.wikipedia.org/wiki/Multi-agent_system

Il est évident que ce genre de systèmes comporte, en raison de ses capacités, nombre de problèmes à résoudre. Nous aurons l'occasion de nous intéresser à l'un d'eux en particulier par la suite mais pouvons mentionner et garder en tête la problématique du consensus dans le cadre de problèmes le nécessitant (Rezaee and Abdollahi (2015), Zou et al. (2013)), de l'auto organisation du système pour permettre une exploitation correcte de l'environnement dans lequel notre système évolue (Ye et al. (2017)), ou le problème de l'apparition de regrets lors d'un choix effectué par un agent, entre autres lors de l'exploitation d'informations obtenues par d'autres agents (Bowling (2004)). Le design d'un MAS étant particulièrement complexe, nous aurons évidemment plusieurs features différentes à considérer, que ce soit pour le design original, la topologie du système, l'apprentissage ou la gestion des consensus et communications (Dorri et al. (2018)). Tout au long de ce travail, nous aurons ainsi l'occasion de revenir sur une partie des concepts introduits, entre autres mais pas uniquement, dans ces articles.

2.2 Exploration et Exploitation

Si plus tôt nous avons eu l'occasion de relever plusieurs problématiques soulevées par le design des MAS, l'essentiel de notre travail consistera à étudier la question complexe du dilemme exploration / exploitation. En effet, par sa construction, nos agents évoluent dans un système originellement inconnu pour eux, ou, en fonction de nos choix, sur lequel ils possèdent une connaissance limitée qui pourrait être ou non partiellement fausse. Comme le sous-tend notre problématique, il nous est nécessaire de trouver un équilibre entre exploration et exploitation, deux activités souvent mutuellement exclusives (Bouffanais et al. (2022), Berger-Tal et al. (2014)) qu'il nous est nécessaire de définir.

L'exploration est, souvent, la première action que nos agents auront tendance à effectuer. En effet, comme le dit Cohen et al. (2007), l'humain - tout comme l'étudiant effectuant son travail de bachelor - doit commencer par récolter une certaine connaissance avant de pouvoir l'exploiter, parfois aussi abandonner l'exploitation pour récolter des données qui pourraient améliorer ou changer les étapes suivantes, considérations ayant, en fonction du type d'environnement exploité, une autre couche de problématiques. Le problème qui sous-tend la réflexion est bien sûr, comme l'indique l'article, la capacité innée à l'exploration de pouvoir augmenter le bénéfice de l'exploitation qui suivra par la découverte de meilleures options et le besoin à, un moment ou un autre, de commencer celle-ci. Effectuer cette exploration pourra souvent dépendre d'un besoin, d'un problème, mais une solution commune pour l'effectuer est l'utilisation d'une random walk plus ou moins dirigée pour contrôler notre quantité d'exploration. (Bouffanais et al. (2022), Carmel (1999))

L'exploitation, dans son cas, nécessite une définition plus laxiste. Elle dépend par sa définition du problème, dans notre cadre nous allons nous intéresser à du target tracking et définirons de fait l'exploitation en fonction de cette considération, Bouffanais et al. (2022) soulève plusieurs autres tâches, entre le decision making, le mapping, foraging, allocation de tâches et autres. Nécessitant l'exploitation des informations récoltées lors de la phase d'exploration, il est notable que, selon March (1991), une exploitation trop hâtive risque, en fonction du temps alloué à la tâche, de ne se montrer efficace que pour un temps très limité avant de devenir la cause d'un comportement auto-destructeur qu'il est évidemment préférable d'éviter en travaillant l'équilibrage entre les deux possibilités d'action de nos agents.

2.3 Décrire et contrôler l'évolution d'un système

Lors de l'étude du dilemme de l'exploration / exploitation, il est souhaitable de pouvoir nommer avec une certaine précision certaines phases de toute simulation de MAS que nous pourrions étudier. Pour ce faire, penchons nous sur un framework permettant de préciser un peu mieux les comportements de nos agents en s'appuyant sur l'article déjà cité : Berger-Tal et al. (2014). L'article décrit quatre phases distinctes : Knowledge Establishment, Knowledge Accumulation, Knowledge Maintenance et Knowledge Exploitation semblent ainsi rythmer l'existence des agents de notre MAS. L'article entre en détails dans la caractérisation de ces phases, certaines pouvant parfois disparaitre ou varier en durée, approchant notre définition du dilemme. C'est une formulation pratique qui peut permettre de décrire l'état d'agents observés, suivant avec précision notre caractérisation exploration / exploitation en rendant plus claires leurs diverses spécificités.

Pour décrire plus en détails un système, nous aurons aussi l'occasion d'étudier la phase de transition, définie comme l'augmentation soudaine de toute métrique de performance, passant d'une performance faible à forte, lors entre autres de l'augmentation de la densité de notre essaim d'agents, une des caractéristiques qu'il est donc nécessaire de contrôler pour répondre aussi bien aux questions de budget que d'efficacité de notre modèle. (Bouffanais et al. (2023)). Cette phase de transition est donc d'une importance capitale dès lors que nous avons un intérêt pour optimiser l'adaptabilité de notre modèle.

Notons que, lors de la création de notre MAS, la programmation des agents sera aussi une importante source de spécificité pour le modèle, comme l'indiquent les figures 1 et 2. La possibilité d'apprentissage de nos agents est déjà une caractéristique créatrice d'une certaine spécificité pour le système. Un intéressant résultat de Oded Berger-Tal (2012) démontre une capacité à contrôler la façon d'agir de nos agents en modifiant, par exemple, leur optimisme concernant la qualité d'un environnement, qui améliore de manière notable les résultats de l'exploitation, surtout pour un agent avec une mémoire. De tels biais fixés pour des agents peuvent donc améliorer ou non le résultat de l'exploitation d'un agent en fonction de sa relation avec l'environnement. La mise à jour de ces biais et la capacité d'adaptabilité de notre agent devient ainsi une priorité pour améliorer nos résultats, les résultats de tels modèles avec des capacités d'adaptation étant, sans surprise, significativement meilleurs. (Carmel (1999))

Pour clore la sous-section, remarquons que la topologie du système et la communication entre les agents est aussi une problématique qu'il est nécessaire de prendre en compte, par exemple en forçant l'actualisation du réseau de voisins d'un agent ou en gérant ses distances et son nombre de communications (Bouffanais et al. (2022)) de sorte a éviter une perte de qualité de l'information possédée. L'information récupérée par des relais sociaux est en effet à risque de produire des conséquences négatives si non régulée. (Johnstone et al. (2002))

2.4 Se concentrer sur notre problématique

Un état de fait concret soulevé dans une partie de la littérature est la spécificité de certaines solutions implémentant un MAS ne permettant de résoudre qu'une quantité particulièrement limitée de problèmes (Bouffanais et al. (2023)). En effet, nous pouvons sans peine admettre l'existence de nombre de problèmes différents et, de fait, d'algorithmes ou fonctions plus ou moins optimisées au niveau de l'agent pour la résolution de ces problèmes. (Bouffanais et al. (2022)) Il est ainsi particulièrement difficile de trouver une unique réponse pour toute catégorie d'applications trop étendue, même si l'adaptabilité de notre essaim d'agents à l'ensemble des problèmes qu'on pourrait lui présenter serait l'issue la plus souhaitable. Contourner cette problématique étant particulièrement difficile, dans le reste de ce travail comme cela peut être fait dans la littérature, nous aurons l'occasion de revenir sur ce point et préciser la façon dont sont implémentés les agents et la façon dont nous influençons nos caractéristiques pour obtenir un optimum. Généraliser ces problèmes est bien sûr difficile, et pour faire le pont entre l'état de l'art et notre travail, mentionnons encore une fois ici que notre code implémentera un algorithme de tracking de cibles mouvantes ou non, de fait, toute modification de certains paramètres devrait être considérée dans ce contexte.

Finalement, mentionnons que dans le cadre de ce travail nous utiliserons Python (Van Rossum and Drake (2009)) et Julia (Bezanson et al. (2017)) pour effectuer nos simulations et exploiter le code originellement fourni.

3 Etudier notre code

3.1 Introduction

Dans le cadre de ce travail, nous exploiterons essentiellement le code suivant : https://github.com/hianlee/swarm-density-tracking. Celui-ci, écrit en Julia pour l'essentiel et en Python pour l'exploitation des données et la création de graphes, propose une implémentation d'une simulation de target tracking. Pour plus de détails, les librairies Julia utilisées sont essentiellement des librairies standard pour permettre l'utilisation de diverses fonctions utiles au code, en plus de PyCall, dans la mesure où nous utilisons aussi Python. Pour Python, nous utilisons la librairie pandas pour l'ouverture et exploitation des fichiers csv, matplotlib pour nos graphes et numpy pour les opérations mathématiques.

Nous allons nous arrêter, par la suite, sur le détail de ce code afin de comprendre un peu mieux son fonctionnement. Notons ici qu'il consiste en une implémentation d'agents traqueurs, de cibles qu'il sera nécessaire pour ceux-ci de traquer, d'un moyen de communication inter-agent pour communiquer les informations sur les cibles et finalement d'une sauvegarde des données produites dans des fichiers csv. Par la suite, nous allons aussi, quand ils sont importants, relever les paramètres les plus susceptibles d'être mentionnés plus tard afin d'en avoir une définition. Notons dans notre introduction que, de manière générale, nous travaillons avec un ensemble de variables globales utilisées pour créer nos simulations, entre autres les boundaries qui permettent de déterminer la taille de notre simulation, et de fait contrôlent aussi la densité du système.

3.2 Code des agents

Première des deux structures mutables de notre code avec la cible, nos agents possèdent de manière notable une vitesse qui dépend de leur caractérisation comme un agent lent (basique) ou rapide. Possédant une connaissance des limites du plan puisqu'on ne travaille pas dans une structure pour le terrain mais uniquement dans la mémoire de nos entités, ces agents possèdent aussi une mémoire et des rayons représentant la distance couverte par leurs sensors ainsi que la taille de leurs zones minimales et maximales de répulsion.

Une importante caractéristique de ces agents contrôlée par beaucoup de nos fonctions s'appliquant aux agents est la présence de deux modes différents que sont "track" et "explore", dépendant de la connaissance ou non de l'emplacement de la cible et de l'état du timer de ces agents. En effet, le mode de traking est limité par un timer contrôlé par nos paramètres de memory. Ces modes, en plus de contrôler le traking de la cible, permettent de gérer le rayon de répulsion concernant les autres agents, c'est-à-dire le réduire lorsqu'un agent est en train de suivre une cible, et l'augmenter lorsqu'il est dans une phase d'exploration, repoussant donc ses voisins de plus ou moins loin.

Un important paramètre sur lequel nous allons avoir l'occasion de nous pencher est le nombre de voisins. En effet, limité dans notre code par une variable globale, il permet l'utilisation de la fonction integrate_neighbour_info!( ) qui implémente notre protocole de communication inter-agents. Effectuant entre autres le calcul de densité pour chaque itération de notre simulation, son intérêt pour le fonctionnement de celle-ci est évidemment le regroupement d'informations concernant l'emplacement de la cible, chaque agent récupérant à l'exécution de cette fonction les informations de ses k plus proches voisins, entre autres position, densité locale, position de la cible si connue ... Ces informations permettront de guider la mise à jour des informations concernant les cibles pour tous les agents, puis, par la suite leur déplacement.

Finalement, nous pouvons nous intéresser rapidement à la fonction de mouvement des agents, celle-ci contrôlant la vitesse de ceux-ci en calculant la vitesse et le "waypoint", mais surtout, nous l'avons remarqué plus tôt, en prenant uniquement en compte sa propre valeur "boundary". En effet, comme indiqué précédemment, la zone de simulations n'a pas d'existence propre et de fait, c'est une gestion individuelle par les agents qui assure qu'ils ne dépassent pas nos limites imposées de manière globale.

3.3 Code des cibles

Seconde et dernière de nos structures mutables, comme les agents, une cible possède sa propre vitesse maximale ainsi que des boundaries. Possédant comme les agents un "waypoint" pour diriger son mouvement, une cible possède aussi une direction aléatoire. Il est notable que ses mouvements sont aussi gouvernés par des paramètres de détection d'agents à proximité et un timer. La cible possède grâce à sa "sensing_range" une capacité à détecter les agents proches et surtout à savoir se considérer comme traquée ou non. Ses uniques fonctions permettent de gérer son mouvement, et si de manière similaire aux agents la cible doit elle-même prendre en compte les limites de la simulation, l'essentiel de la complexité du code vient de sa façon de gérer le fait d'être traquée. Trois possibilités gouvernent sa "movement_policy ", ne (non-evasive), e (pure evasive) et mix (evasive puis tente de distancer les agents si traquée).

Le cas non-evasif se comporte de manière simple, se dirigeant en direction du waypoint lui étant attribué, le recalculant au besoin, et calculant une direction à suivre, sa vitesse se limitant à la vitesse maximale que nous avons autorisée et permettant de suivre les parties sinus et cosinus du résultat de notre calcul. Dans le cas evasif cependant, nous prenons en compte les agents à proximité de la cible et leur position pour calculer une direction qui prend en compte ces paramètres et cherche à s'éloigner le plus possible de l'ensemble des agents, oubliant de fait de suivre le waypoint. Finalement, le cas "mix" va combiner une approche évasive lorsque notre cible n'est pas ou peu suivie de sorte à limiter l'exploitation de sa position, puis se contentera de suivre le waypoint en espérant distancer les cibles si elle venait à être traquée trop longtemps. C'est cette dernière stratégie que nous utiliserons durant ce projet.

3.4 Fonction main

Cette fonction sert d'exécution principale de notre programme Julia. Elle consiste en une boucle sur une variable qu'on veut tester, dans notre cas différentes tailles d'environnement, et cause la création d'agents lents auxquels on peut ajouter des agents rapides, et la création des cibles. Suit une boucle sur le nombre d'itérations. Remarquons que pour une itération, se suivent dans l'ordre chronologique les évènements suivants : Déplacement de la cible, récupération de la position des agents et cleanup de leur mémoire si nécessaire, communication inter-agent sur la position de la cible puis mouvements des agents. Une fois toutes les itérations passées, on changera ainsi de taille d'environnement après avoir récupéré les valeurs qui nous intéressent.

3.5 Exploiter les résultats

Nous obtenons grâce à ce code 3 quantités différentes que sont le score, la proportion d'exploitation inversible en la proportion d'exploration, et finalement la densité. Définissons rapidement ces trois quantités :

- Le score est défini comme le nombre d'itérations durant lesquelles une cible a été traquée sur le nombre d'itérations fois le nombre de cibles. Notons que deux cibles traquées en même temps sont donc cumulatives.
- La proportion d'exploitation augmente à chaque itération lorsqu'un agent traque la cible. On peut voir cette valeur comme le score du point de vue des agents, et de fait elle décrit si inversée la quantité d'agents ayant participé à l'exploration et non à l'exploitation.
- La densité est définie comme la moyenne de la densité calculée par chaque agent à chaque itération, suivant la formule : $\frac{n+1}{\pi \cdot \text{avg\_neighbor}^2}$ avec avg_neighbor définie comme la moyenne de la distance des n voisins avec lesquels l'agent communique.

Ces informations seront par la suite stockées dans un CSV et exploitées via un code en Python pour produire les graphes que nous aurons l'occasion d'étudier plus tard. Cette dernière partie est l'unique reposant sur du Python.

3.6 Mise à jour du code

En étudiant le code, j'aurai eu l'occasion de relever plusieurs points que j'ai modifiés avant de commencer le travail de recherche. L'un d'entre eux fut un problème de mise à jour des emplacements de cibles par les agents, une fonction exécutée trop tôt ayant eu la capacité de mettre à jour des données internes aux agents avant que le protocole de communication ne se termine. Ce problème corrigé, nous avons pu exploiter l'extension ProfilView afin de confirmer que la fonction la plus coûteuse en temps était la fonction get_neighbour_info(). Utiliser le multithreading de Julia pour la paralléliser aura permis de quasiment diviser par 2 le temps d'exécution de notre code.

Le schéma ci-dessous compare l'ancien code, le code modifié ainsi que le code modifié et parallélisé avec 16 threads en fonction de leurs résultats pour le score et la densité avec 10 000 itérations. Les couleurs l'indiquent, le code parallélisé et modifié se superposent parfaitement : il n'y a aucun problème de data racing avec notre multi-threading.

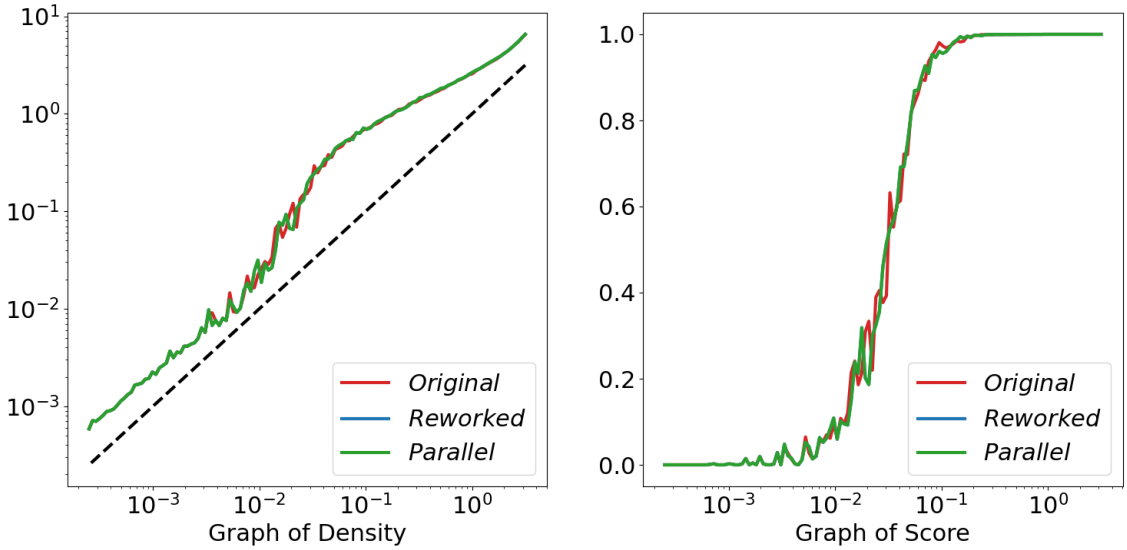

Figure 3: Comparaison de la modification du code.

Concernant le temps d'exécution, nous relevons un temps d'exécution de 2262 secondes pour le code original (38 minutes), 2313 secondes pour le code modifié (39 minutes) et 1224 secondes pour le code parallélisé (20 minutes). On divise quasiment le temps d'exécution par deux en se limitant à paralléliser une boucle unique, permettant une certaine flexibilité du code si on venait à vouloir modifier en détail le reste des paramètres.

4 Identifier des leviers d'action ...

Dans le cadre de notre projet, nous allons nous intéresser de manière plus précise à des MAS avec une considération sur l'hétérogénéité de notre essaim. De manière générale, cela implique la spécialisation de certains agents pour des tâches précises, ou des caractéristiques les séparant en plusieurs groupes. Dans le code étudié, nous avons le cas des agents rapides, avec une vitesse plus grande qu'un agent normal, et un coût supposé plus grand si on venait à les rendre réels. Obtenir une balance entre les agents classiques et rapides est un moyen d'optimiser l'adaptabilité et l'efficacité de nos essaims. Nous tâcherons ainsi par la suite d'expliquer l'effet causé par la modification du nombre d'agents rapides, trouver un moyen d'optimiser leur nombre si possible, mais aussi modifier plus en détails ces agents pour changer leur comportement, et essayer de déterminer quels types d'agents pourraient, si ajoutés dans notre système, augmenter nos performances ou réduire nos coûts, et de quelle manière. Notons que notre limite principale sera le refus total de centraliser les informations de sorte à ne pas produire d'overhead pour les calculs. Nous tenterons aussi de proposer des solutions théoriquement réalisables lors de la modification des agents ou le choix de leurs nombres.

4.1 ... Exploitant le code déjà existant.

Plusieurs valeurs existent déjà et sont exploitables dans le cadre de ce projet. ?ous allons définir chacune d'entre elles et souligner leur intérêt, la spécificité des implémentations les concernant ne sera relevée que plus tard.

1. La première valeur déjà définie est la densité locale. Dépendant des n voisins les plus proches d'un agent à un instant t, c'est une valeur qui permet d'indiquer la proximité relative de ceux-ci. Décrite plus tôt, cette valeur peut être exploitée pour décrire partiellement l'entourage d'un agent et peut ainsi guider ses choix de mise à jour de ses valeurs.

2. La seconde de ces valeurs déjà définies est le mode, séparé entre "tracking" et "exploring", il permet de décrire l'état d'un agent en fonction de sa relation à la cible, nous remarquerons qu'il présente un intérêt pour décrire l'entourage d'un agent au niveau du ratio d'exploration local.

3. La dernière de ces valeurs est k, soit le nombre de voisins considérés lors de la section de communication du code. Nous remarquerons que k est une valeur particulièrement importante qui se trouve capable d'influencer de manière considérable les agissements de l'ensemble des agents.

Nous décrirons plus tard leur importance lors de l'implémentation, observons plutôt nos nouvelles valeurs.

4.2 ... En intégrant de nouvelles valeurs.

L'essentiel de la description du code permet de relever l'essentiel des valeurs déjà importantes que nous pouvons mesurer ou manipuler, cependant, un ajout se voit nécessaire pour des raisons de praticité. En effet, remarquons que la manipulation dynamique du système soulève une problématique qu'il est important de prendre en compte dès le début de la mise en place de la recherche : Le risque de centralisation. La définition de nos MAS décrit fort bien une nécessité d'absence de centralisation du système.

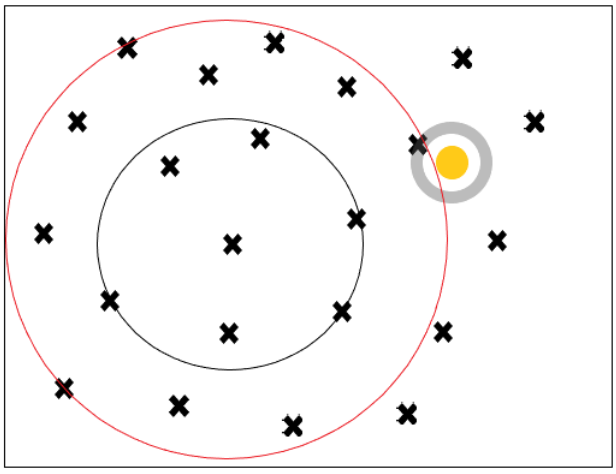

Figure 4: Schéma d'une frame d'exécution du code.

Observons le cercle noir, nous le décrirons comme k, cette zone dépendant directement du nombre de voisins qu'il est légitime pour notre agent central de considérer. Observons ensuite que, modifier dynamiquement k sans limite conduirait potentiellement à k devenant égal au nombre total d'agents, centralisant l'information, plus aucune communication ou manque de connaissances, l'agent saurait tout. Il est donc nécessaire, nous le préciserons si c'est fait, de mettre en place une limite arbitraire k_max, similaire à ce cercle rouge qui interdirait toute communication avec des agents effectivement trop lointains.

Une autre valeur se montre digne d'intérêt : le coût de fonctionnement des agents. Nous avons déjà décrit deux types d'agents, les agents rapides et lents, mais il est évident que ceux-ci ne posssèdent pas un coût d'exploitation et de fabrication similaire, nous pourrions nous contenter des plus performants. Voyons alors un schéma lorsque nous définissons une valeur arbitraire à chaque agent, par exemple, un agent rapide ayant le coût de deux agents lents, appliquant la formule suivante :

$$\frac{Score}{2*nf+nl}$$

Avec nf le nombre d'agents rapides et nl le nombre d'agents lents.

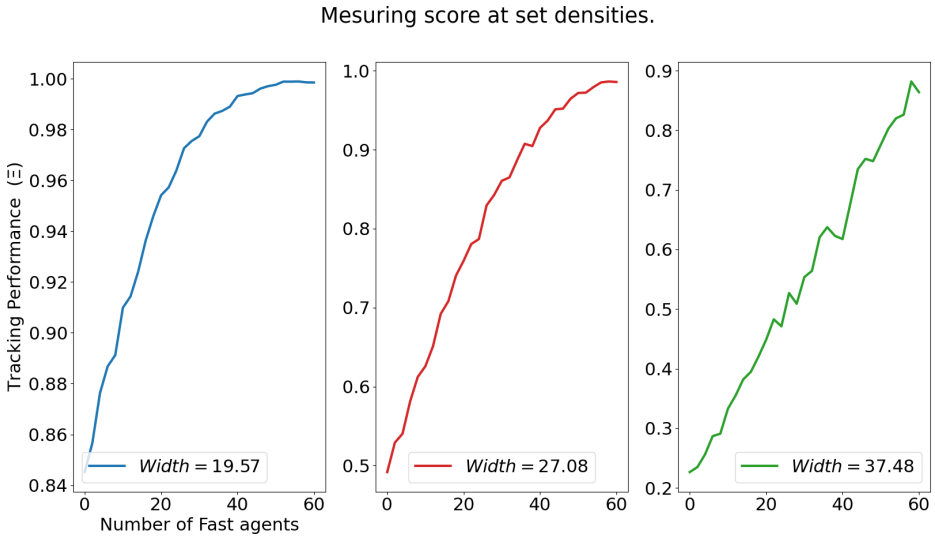

Figure 5

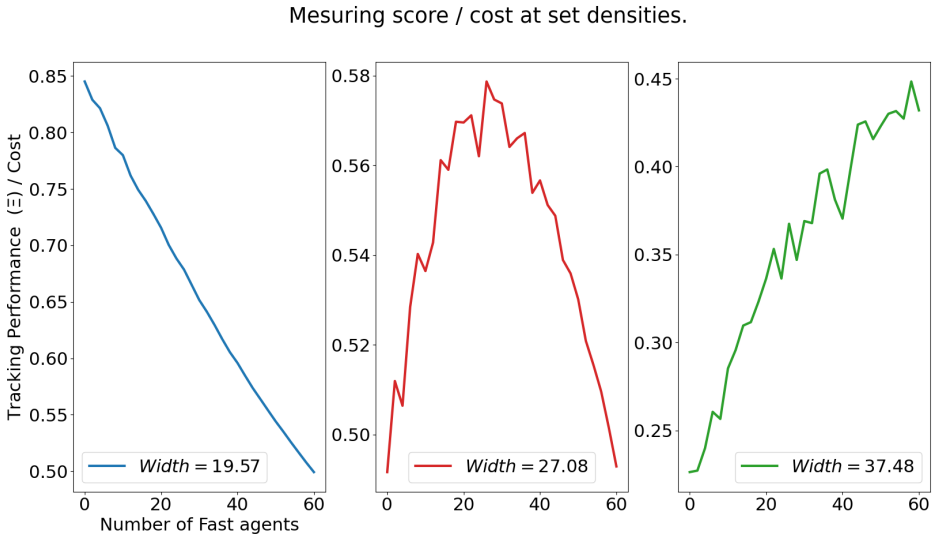

Figure 6

Nous observons clairement sur ce graphe une limite confirmant l'idée du besoin de limiter le coût : Si les agents rapides ont des capacités intéressantes que nous soulèverons plus tard, il est très clair qu'il faut prendre en compte l'intrinsèque limite de leur coût, révélant que pour un nombre d'agents ici fixé, augmenter la proportion d'agents rapides va toujours augmenter le score. Cependant, en s'intéressant à la corrélation entre le score et le coût, il existe un très clair point à partir duquel exploiter des agents rapides à haute densité n'est pas rentable, le ratio score / coût se réduisant, et exploiter des agents rapides à basse densité propose cependant un gain notable, même en subissant l'augmentation du coût. Obtenir une balance entre gain de score et coût total pour l'obtenir devient alors une considération digne d'intérêt.

5 Recherche et Résultats.

5.1 Notre démarche de recherche.

Dans le cadre de ce projet, une importante part de celui-ci fut la compréhension et la prise en main du fonctionnement de ces MAS. En effet, la collection de résultats de simulation pour comprendre l'impact de la modification de différentes parties du code fut cruciale, et dans la suite de cette section qui recoupera thématiquement les explorations de solutions et l'interprétation de résultats, nous nous pencherons aussi de fait sur le raisonnement conduisant à ces modifications. Certains résultats, les premières tentatives de modification de notre système, ne seront que rapidement mentionnés en raison de leur faible apport à la recherche que nous menons ici, mais leur utilité dans ma compréhension personnelle du fonctionnement de ce projet ainsi que dans le choix des directions prises plus tard aura été une part importante, autant en temps qu'en gain d'informations, de mon travail au long de ce semestre.

Mentionnons de plus dans ce préambule que les résultats qui vont suivre n'ont pas vocation à révolutionner le fonctionnement des MAS mais suivent l'objectif de ce travail qui est d'identifier des leviers intéressants pour le contrôle de ces essaims et l'implémentation de stratégies les exploitant pour observer leur effet sur notre score. De fait, ces expérimentations sont essentiellement séparées les unes des autres et sont appliquées individuellement à notre système pour observer leur effet propre.

Finalement, relevons pour des raisons de clarté que les expérimentations sont ici faites dans un environnement avec un aléatoire seeded pour la reproductibilité des résultats, de fait, lorsque nous mentionnerons tout résultat "Classique", il sera défini comme le résultat obtenu lors de l'exécution du code original avec un set de paramètres fixé, opposé à des résultats implémentant nos diverses stratégies.

5.2 La base de nos modifications : La valeur de k.

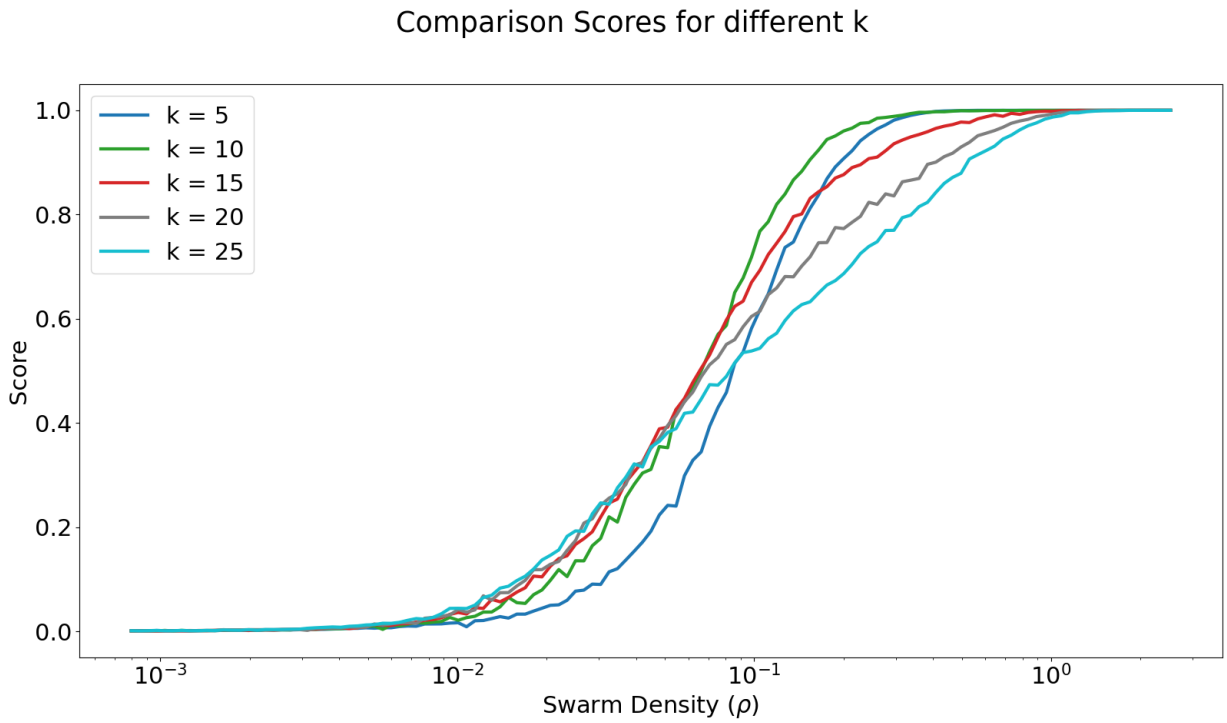

Figure 7

Nous observons ici un élément déjà couvert dans l'article de Bouffanais et al. (2023), qui nous intéressera particulièrement dans les sous-sections qui vont suivre. En effet, nous le remarquerons dans les derniers résultats soulignant une certaine difficulté à obtenir des résultats concrets en appliquant des stratégies diverses, le nombre de voisins k est un élément particulièrement efficace à manipuler pour optimiser le score de nos MAS. Reconnaissons ici une claire supériorité à basse densité des MAS avec un k élevé, contre inversement une telle supériorité à haute densité pour les MAS avec un k plus faible. En plus de reconnaitre une certaine limite à ces modifications de k, que ce soit pour k = 5 ou 25 qui ont un gain négligeable ou même inexistant comparé à leurs plus proches voisins que sont 10 et 15, nous reconnaissons sans peine l'outline d'un graphe optimisé : Obtenir un k optimisé dynamiquement permettrait de garder un score le plus élevé possible pour toutes les densités.

La raison de ce comportement semble assez claire en observant les graphes : Augmenter le nombre de voisins permet, à basse densité, à un agent d'approcher plus aisément la cible et augmenter de manière générale l'exploitation de celle-ci, là ou au contraire, un k trop élevé à basse densité poussera les agents à une sur exploitation qui pourrait pécher dans le cas où celui-ci arrive à échapper à ses poursuivants et profite du manque d'exploitation général pour échapper temporairement à la surveillance de ses poursuivants, tous regroupés en un même endroit. Cela est compensé par la réduction de k, démontrant la viabilité de notre dilemme : Il faut pouvoir balancer exploration et exploitation, et k est pour cela, semble-t-il, un outil parfait.

5.3 Première étape dans la dynamicité : Changer la valeur de k selon les actions d'un agent.

Une des premières tentatives effectuées fut celle de la modification statique de la valeur de k en fonction de son mode, à savoir "tracking" ou "exploring". A partir d'une valeur de k fixée, nous décidons d'y ajouter ou soustraire une valeur statique de k, 5 dans nos exemples, et d'observer l'effet de cette modification sur le comportement de l'essaim lors de la disruption de ces deux phases de manière dynamique.

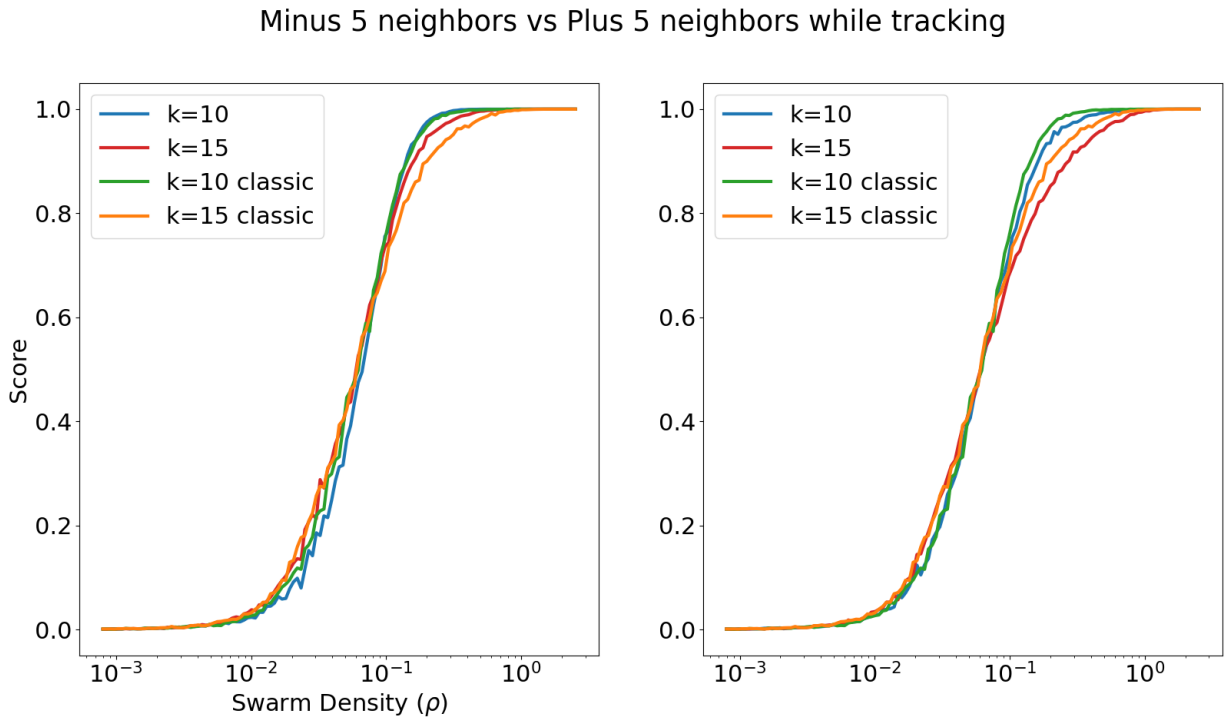

Figure 8

Ici nous réduisons à gauche, et augmentons à droite, le nombre de voisins k d'un agent lorsqu'il traque une cible. Cherchant à observer son comportement, nous relèverons l'intéressante observation que réduire, à gauche, le nombre de voisins lors du tracking permet, nous le voyons surtout avec la comparaison de k = 15 et sa contrepartie classique, d'augmenter notre performance de tracking à haute densité tout en maintenant une efficacité équivalente à basse densité, chose qui, en retirant 5 agents partout n'est pas le cas, puisque nous en viendrions à comparer ces deux graphes au k = 10 classique, moins bon à basse densité.

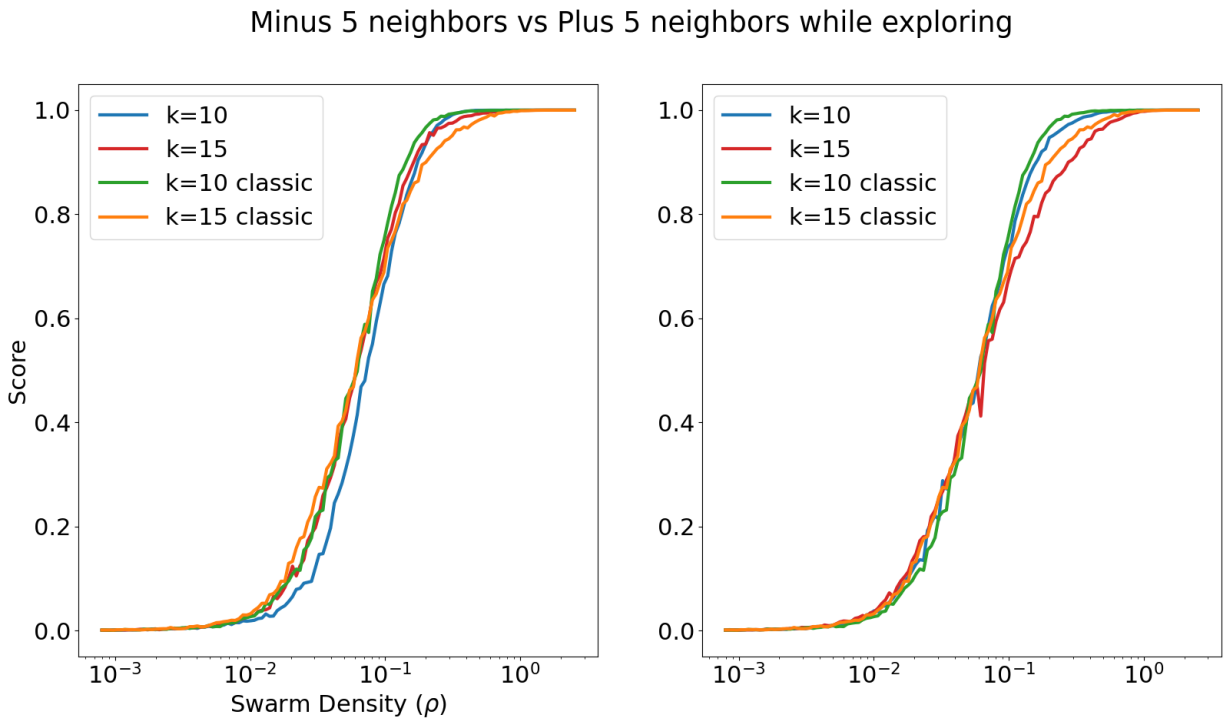

Figure 9

Sur ce second graphe où nous éditons k lors de l'exploration, reconnaissons un comportement approximativement similaire, avec une flagrante différence : S'intéresser à la façon dont agit le système à basse densité semble indiquer que notre graphe rouge, donc pour k = 15, celui-ci se confond avec k = 10, mais lui est inférieur à haute densité. En effet, nous n'avons aucun gain concret en modifiant notre comportement de la sorte. Cela démontre une faible mais notable différence entre la façon dont k impacte le système lorsqu'il est modifié dans une phase de tracking ou une phase d'exploration.

Relevons finalement, avec quelques mots, le léger gain de score à basse densité lorsque nous ajoutons, à droite, 5 voisins à k = 10 lors des situations d'exploration, au prix d'une baisse de ce score à haute densité. Cette observation ne s'étend d'ailleurs pas à la situation du tracking, révélant une nouvelle fois un caractère unique à la modification de k en fonction des phases où elle est appliquée.

Nous est ainsi dévoilée une importance de la valeur de k et de son évolution qui diffère en fonction des agissements d'un agent. Il faut cependant noter que nos modifications sont instantanées, et qu'une modification dynamique dépendant d'autres paramètres ainsi que des actions de l'agent pourrait ne pas être assez rapide pour réellement influencer le tracking ou l'exploration, suivant l'environnement et le nombre d'agents impliqués dans la simulation.

5.4 Optimiser k en connaissant ses valeurs.

Intéressons nous de nouveau à la figure 7. Celle-ci nous permet de déterminer comme déjà mentionné un optimum généralisé pour k. Il est donc possible, pour des exécutions fixées, de déterminer le k optimal à tout moment et le retourner pour une nouvelle exécution permettant d'obtenir le meilleur score possible.

Obtenir les valeurs optimales de k, décrites par les points rouges, permet d'entrainer un algorithme de Machine Learning, à savoir après comparaison de plusieurs d'entre eux, le MLP Classifier de Sklearn Pedregosa et al. (2011) (i.e. Un basique perceptron multicouches). Celui-ci nous permet de déterminer qu'il est théoriquement possible de proposer un entrainement sur les données d'assez de simulations pour approcher une prédiction efficace. Bien sûr, la quantité de données et le temps nécessaire pour les produire, ainsi que les possibles liens à ajouter pour peut-être justifier l'utilisation de modèles plus complexes est difficile à quantifier. Notons que la performance claire du classifier est ici causée par l'utilisation d'un ensemble de données très faibles (une unique simulation) car nous cherchons ici à produire une baseline. Dans une situation réelle avec la possibilité de produire un très grand nombre de données, il est plus que probable que d'autres algorithmes, par exemple Kernel Ridge, Lasso ou une régression polynomiale puissent toutes produire un meilleur résultat qu'un classifier, qui plus est un perceptron.

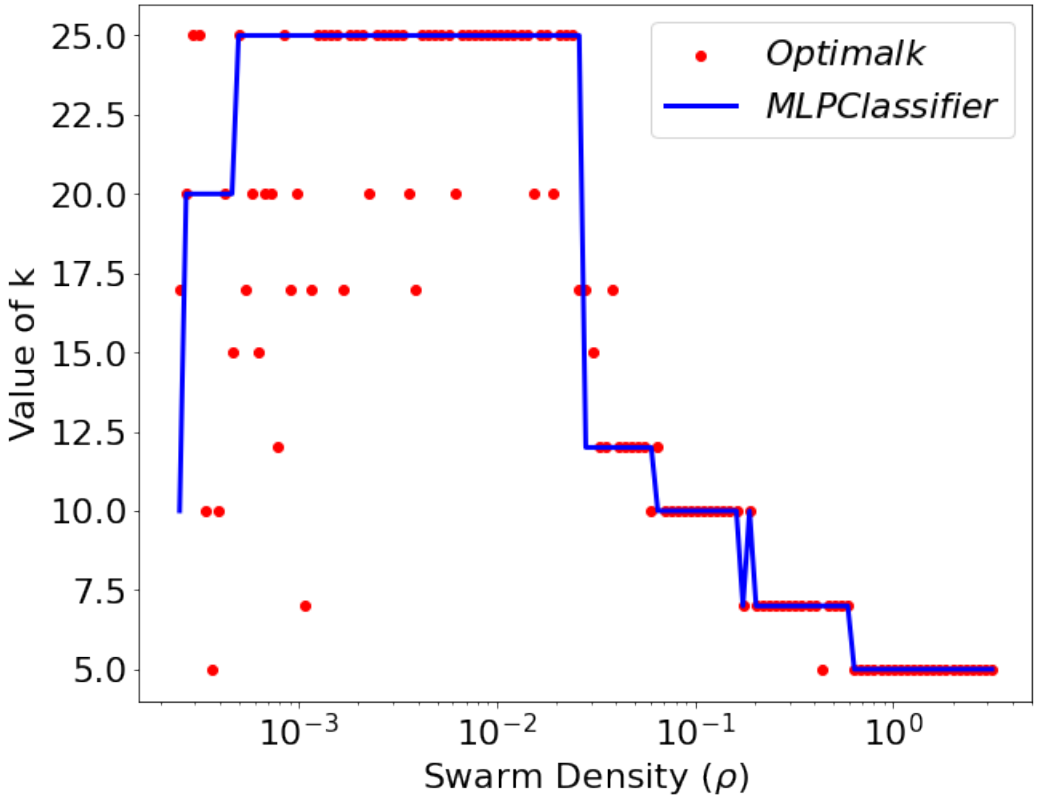

Figure 10

Remarquons sur ce graphe une certaine instabilité à basse densité, le nombre k de voisins possibles variant grandement là où l'optimum devrait être un k grand. Il semblerait qu'à ce point, le score dépende essentiellement du hasard, les hautes valeurs de k plus représentées pour l'entrainement semblant indiquer une légère influence de k tout de même. Par la suite, remarquons l'existence d'un semblant d'escalier des valeurs optimales après une importante chute qui semble indiquer que les valeurs de k situées entre 20 et 10 n'ont le temps de briller que sur un écart de densité particulièrement faible.

Ce résultat permet ainsi l'utilisation dans les prochains graphes d'un résultat optimisant l'exploration et l'exploitation au regard de k. Observons dans le graphe suivant la capacité de notre collection des k optimums, si donnée à une simulation, à optimiser notre k à chaque pas de densité.

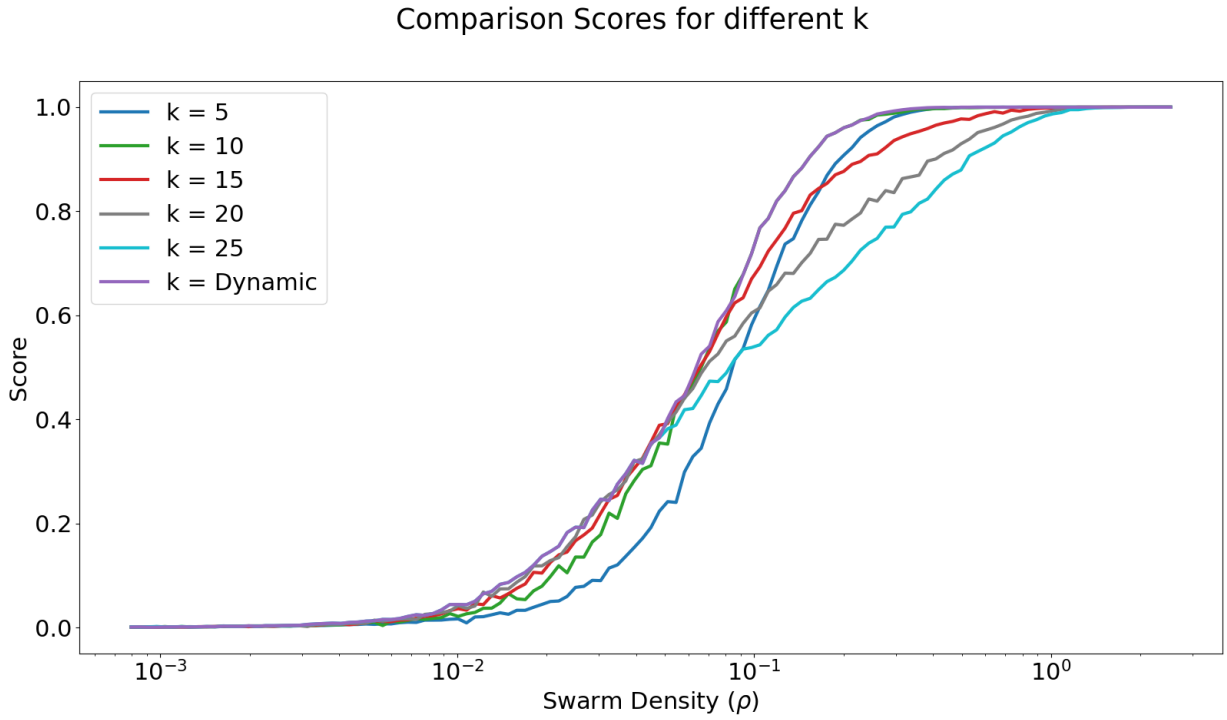

Figure 11

Notons cependant qu'ici, k est encore statique lors des 100 000 itérations dans une densité donnée. En effet, si ce résultat nous servira de baseline et de preuve de l'optimisation possible de k, la modification dynamique de k durant les itérations d'une densité précise reste une partie importante de la recherche à effectuer, la plus importante pourrions nous dire, puisqu'elle s'intéresse au comportement de nos agents.

5.5 Optimiser k selon nos observations.

Du résultat précédent nous avons tiré une solution optimale. Quel est alors le ratio d'exploration optimal pour une telle solution ?

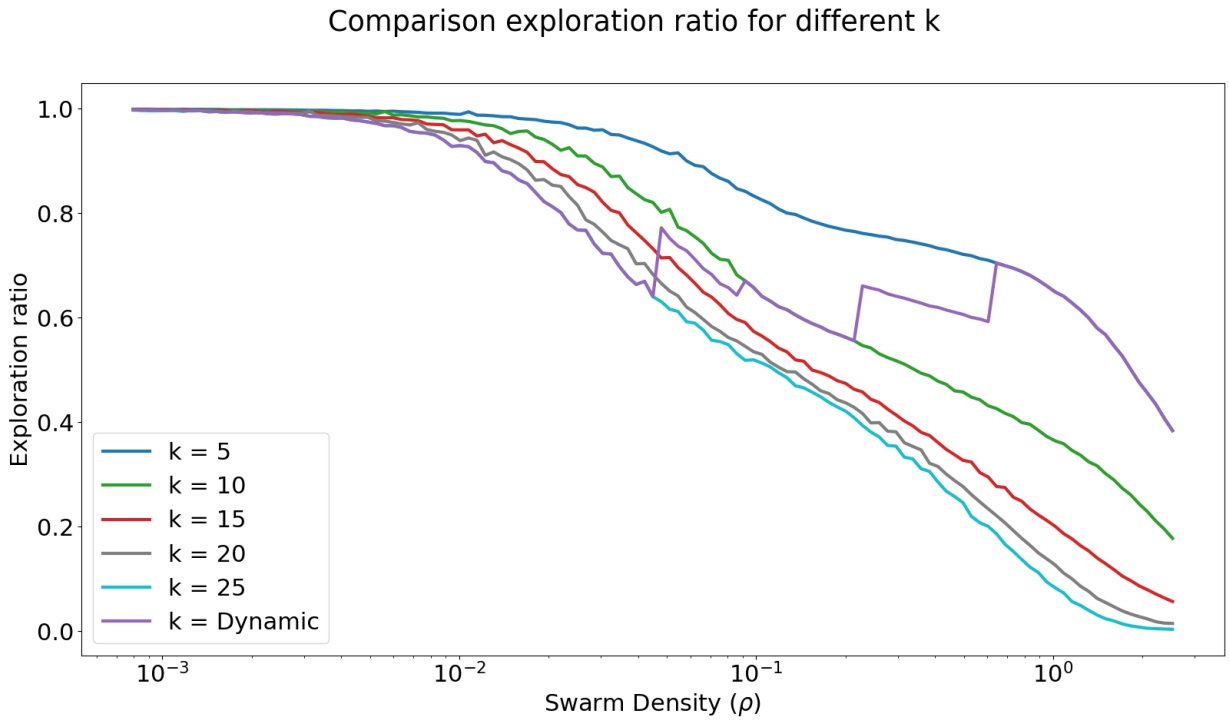

Figure 12

Cette figure nous donne la réponse : L'exploration doit être réduite au possible à basse densité. En effet, le problème principal dans une telle situation est le manque d'exploitation d'une cible ayant trop de possibilités de fuir. Alors que la densité augmente, le ratio d'exploration optimal n'est plus le plus bas, mais bien un ratio qui gravite autour de 0.60 à 0.70, suivant les valeurs de k = 12, 10, 7 puis finalement 5. A haute densité nous maximisons effectivement l'exploration, permettant de parer à toute éventualité d'une cible capable de s'échapper. Si parfaitement expliquer le comportement de marches que nous avons déjà vu plus haut semble difficile, s'ouvre l'exploitation de cette valeur d'exploration pour une balance dynamique de nos valeurs de k.

Observons par la suite une exploitation de la valeur du ratio local d'exploration pour un agent, lui dictant l'évolution de k qui augmentera ou baissera s'il est respectivement plus grand ou plus petit que cette valeur seuil de 0.70 de ratio d'exploration, définie pour nous de manière locale comme : $\frac{exploring\_neigbors}{k}$ avec k au choix notre k maximum décrit plus tôt (à savoir 25) où le nombre de voisins k au temps t de notre agent.

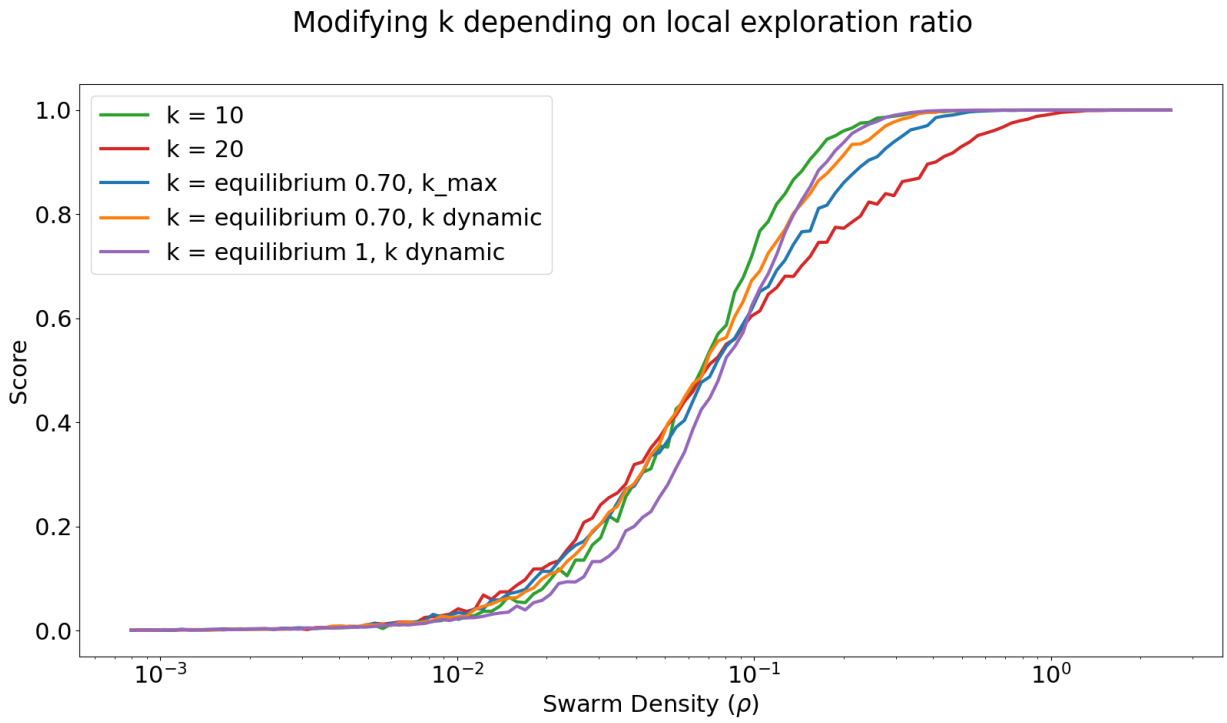

Figure 13

Le premier résultat sautant aux yeux avec l'équilibre entre la tentative avec k_max contre celle avec un k dynamique est la très courte supériorité de notre k_max très vite ratrappée quand la densité augmente : En effet, une analyse locale de l'environnement est ici plus intéressante, comme très souvent, qu'une analyse prenant en compte trop de paramètres. Finalement, une tentative qui sert plutôt de preuve avec une valeur seuil de 1 (équivalente à fixer k à 5) nous est simplement utile pour souligner l'incapacité de notre modification à efficacement s'adapter aux situations des extrêmes où la dynamicité ne peut suivre le comportement attendu. Remarquons cependant, par comparaison avec les versions classiques avec k = 10 et k = 20, la capacité d'un tel algorithme à effectivement conserver des résultats supérieurs à ces deux versions dans leurs densités où ils sont le plus faibles, et conserver une certaine proximité de score avec ceux-ci dans la situation où ils sont le plus forts : Notre dynamicité est effectivement fonctionnelle et propose un résultat satisfaisant.

Il est intéressant de mentionner rapidement l'effet d'une telle modification sur le ratio d'exploration général en s'intéressant au graphe y correspondant.

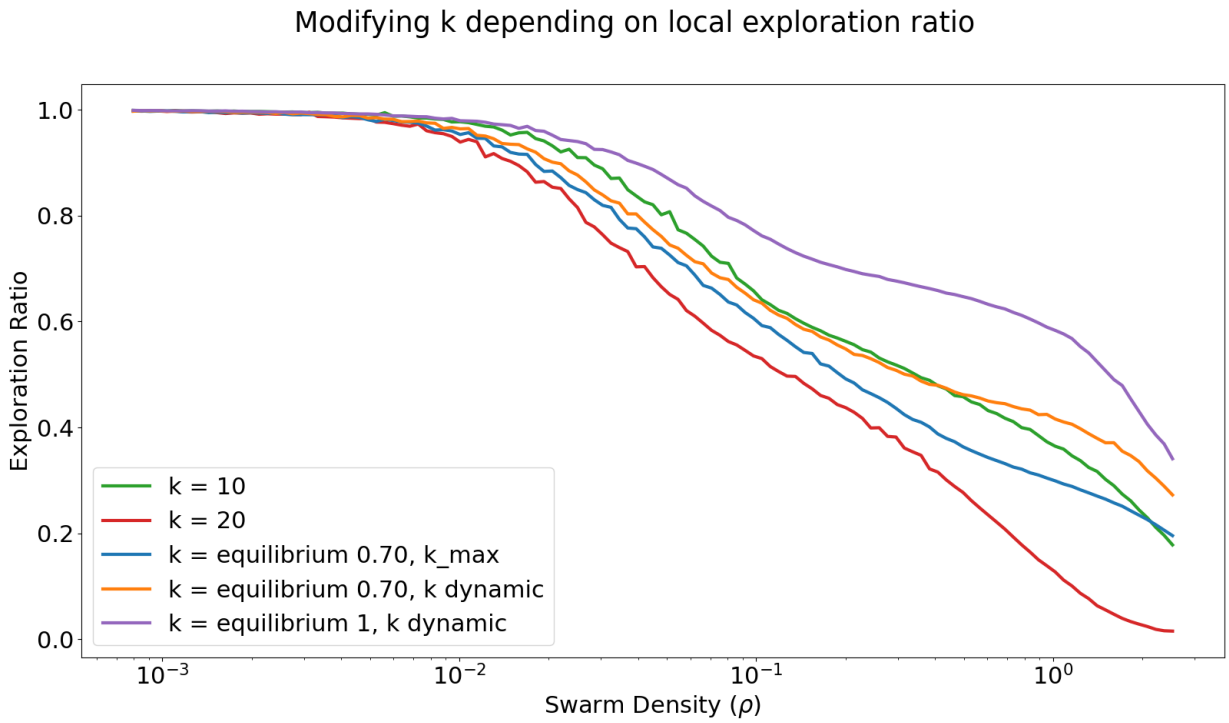

Figure 14

En effet, remarquons que nos modifications tendent à aller à contrecourant des simulations classiques, se stabilisant voire remontant en approchant la haute densité, dépassant le ratio d'exploration pour k = 10 : Le ratio d'exploration est ici maintenu par notre modification dynamique de k, ne permettant pas d'atteindre la chute visiblement nécessaire à l'optimisation du score à haute densité, que l'on voit pour les graphes verts et violets. Une seconde mention est nécessaire pour souligner que l'explication de ce ratio d'exploration est nécessaire mais non suffisante : En effet, le comportement du graphe violet indique que même avec un ratio d'exploration toujours supérieur, il est capable d'être meilleur que le graphe orange à haute densité.

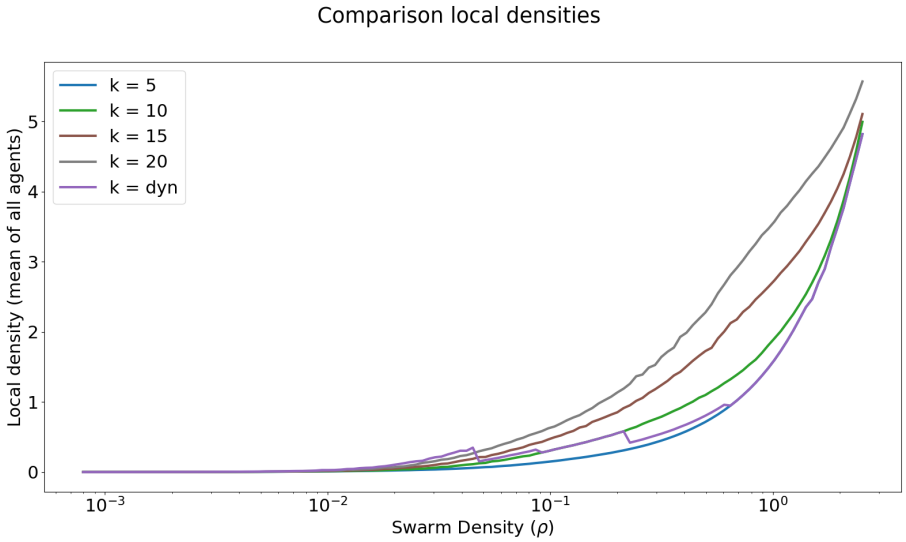

Figure 15

Finalement relevons la capacité de notre représentation à souligner l'effet de la dynamicité de k sur la densité. En effet, nous voyons clairement sur ce graphe qui décrit l'évolution de la densité locale d'une manière similaire à celle de l'évolution du ratio d'exploration. L'optimiser de la même manière est possible, intéressons nous ici à la claire frontière que l'on trouve vers 0.35 de densité sur l'essentiel du graphe précédent.

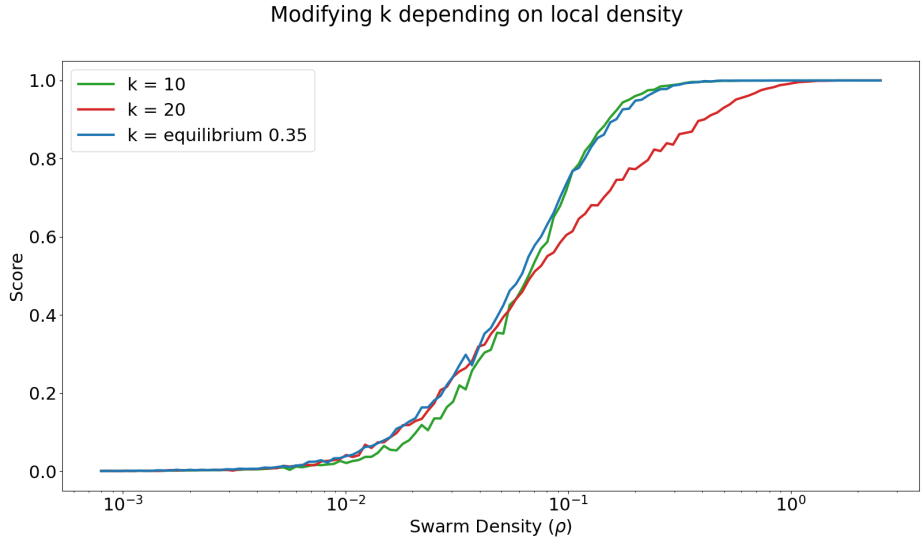

Figure 16

Remarquons une fois encore une capacité d'adaptation de notre k dynamique, particulièrement qualitative ici, pour suivre le score à basse densité du k = 20, tout en dépassant très largement celui-ci à haute densité et approchant le k = 10 dans sa qualité. Vient alors le questionnement de la spécificité d'une telle valeur pour notre simulation, et une difficulté à pouvoir la prédire ou la faire évoluer de manière logique en toute circonstance.

5.6 Modifier le comportement du tracking : Hétérogénéité.

Dans les prochaines sections, éloignons nous de la modification de k et intéressons nous à d'autres comportements de notre MAS. Le premier est la qualité du tracking dans un cas d'hétérogénéité, soulevant la question du rôle d'un agent en fonction de ses qualités intrinsèques, à savoir dans notre cas la vitesse.

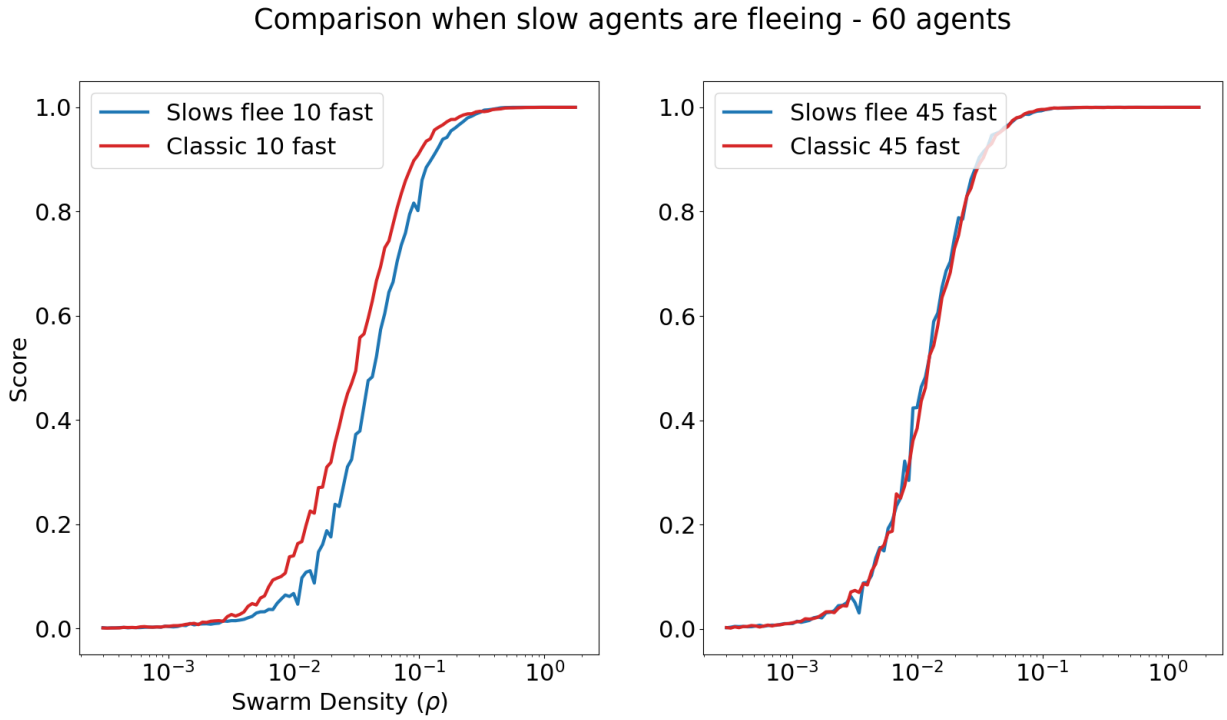

Figure 17

Remarquons dans ces deux graphes une claire différence du rôle que les agents lents doivent adopter en fonction de la quantité d'agents rapides, si le changement se fait surprenamment tard, trop pour qu'il soit réellement exploitable, il est clair que les agents lents ont, dans une situation où ils sont en sous-nombre, une capacité réduite à participer efficacement à l'exploitation, et le système dans son ensemble bénéficie alors de leur participation à l'exploration, laissant le rôle d'exploitation à des agents plus rapides capables de maintenir leur poursuite pendant une plus grande période de temps.

5.7 Modifier le comportement du tracking : Optimiser la balance.

L'optimisation du tracking des agents est plus complexe. En effet, l'idée serait de pouvoir éliminer une partie des agents effectuant du tracking, les agents en théorie moins performants, de sorte à les pousser à l'exploration plutôt qu'à une participation peu efficace à l'exploitation. Dans la continuité directe de la section précédente, cela peut-être l'idée d'éliminer les agents les plus lents, ceux qui sont les plus proches d'un reset de leur mémoire, ou ceux étant géographiquement les plus éloignés de la cible, mais participant au tracking de celle-ci. Dès le dépassement d'un seuil de traqueurs, l'un d'entre eux est éliminé et retourné à l'exploration.

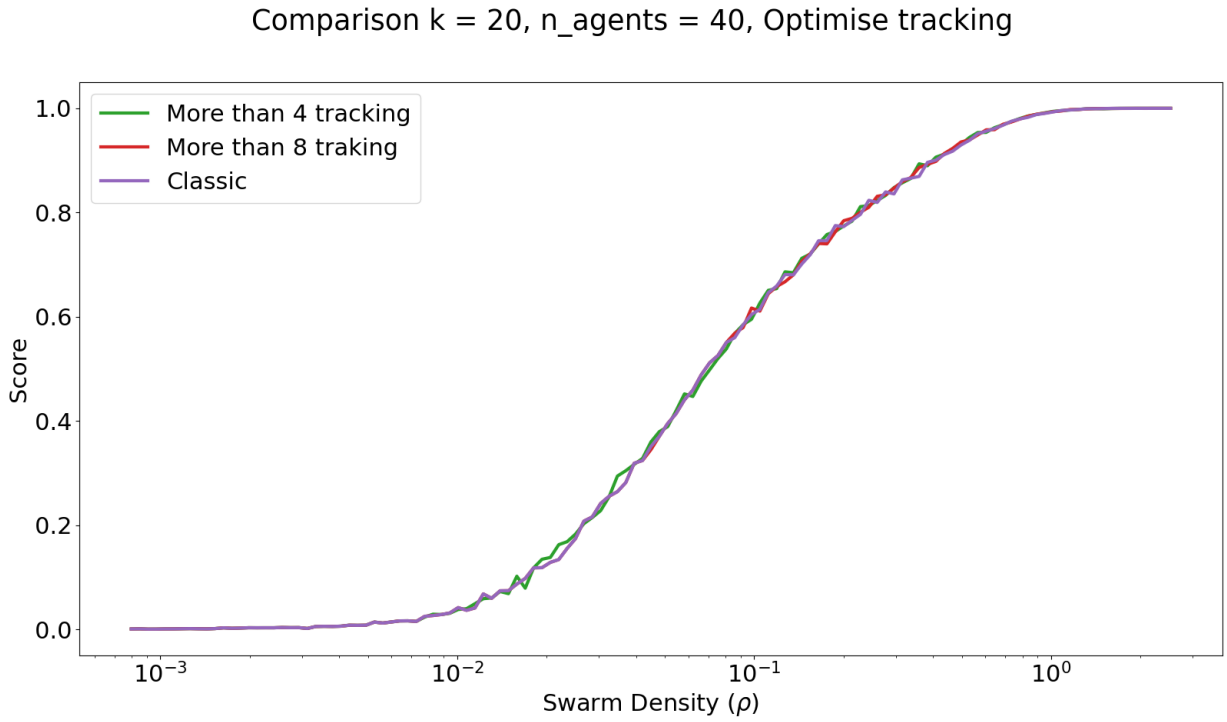

Figure 18

Cependant si un résultat légèrement satisfaisant est ici visible, ou sur ce graphe les agents les plus proches du reset mémoire (effectué s'ils sont dans une situation de tracking depuis trop longtemps) sont éliminés, le gain est particulièrement faible et variable. Un seuil trop bas pour l'élimination expose au risque de ne pas permettre l'utilisation d'une stratégie de swarming, et un seuil trop haut, jamais atteint, rend la stratégie inexploitable. Bien que digne d'être mentionné, ce résultat nécessite certainement l'exploitation d'autres stratégies en plus pour être réellement efficace.

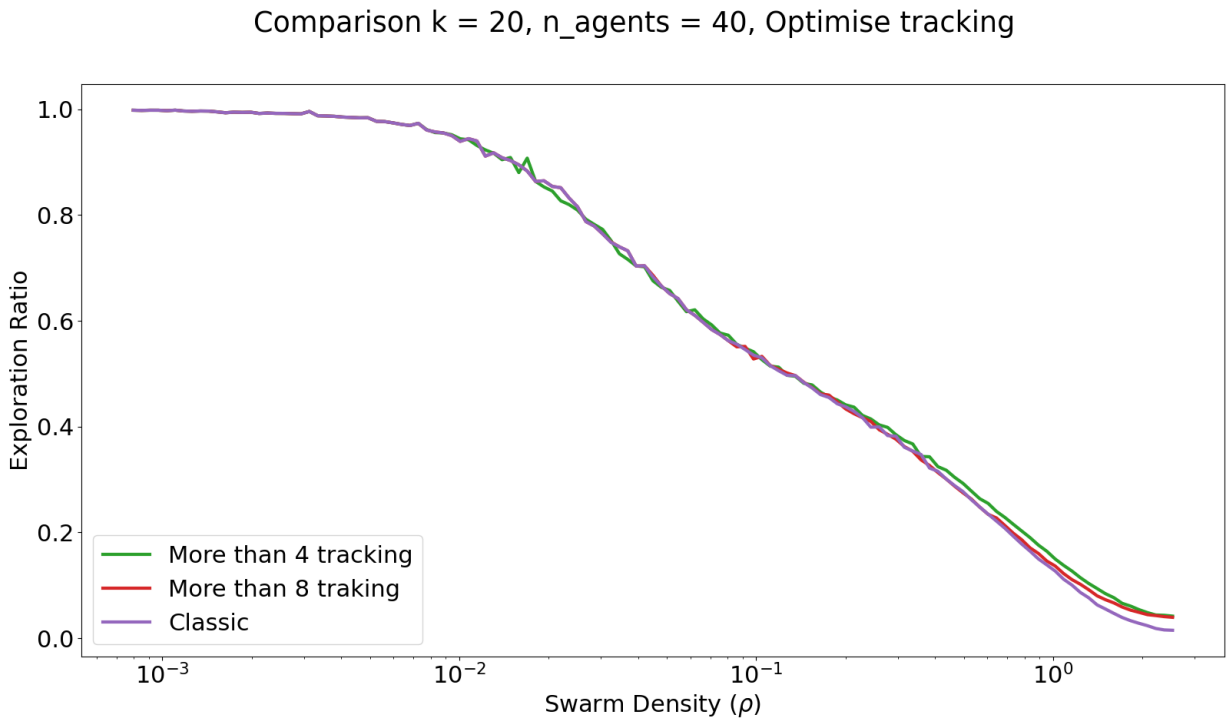

Figure 19

Est lisible dans la figure du ratio d'exploration correspondant à notre simulation un intérêt dans sa réduction à basse densité, et son augmentation à haute densité, rappelant les résultats précédents mais indiquant clairement l'incapacité de telles stratégies à réellement influencer l'état du système. Les autres stratégies mises en place ne seront ainsi pas générées et montrées ici, mais contenaient entre autres l'élimination de communication avec les voisins dans le cas de l'approche d'une zone à trop haute densité ou lors de la sortie d'une phase de tracking pour empêcher la reprise de celui-ci. Auront aussi été tentées la stratégie d'ignorer temporairement les voisins en cas de réduction de la vitesse en dessous d'un seuil indiquant que l'agent stagnait à une même position, ou encore celle de s'éloigner uniquement de l'agent le plus proche dans le cas d'une densité locale trop haute.

Si ces résultats portent des observations occasionnellement dignes d'intérêts, elles sont essentiellement couvertes par le reste de nos analyses ici et, implémentées de manière individuelle sans offrir à nos agents des outils plus concrets pour exploiter ces stratégies. Les gains de score sont négligeables à coté des autres sous-sections. De manière générale, l'observation de ce genre de résultats tend à décrire l'exploitation de stratégies modifiant le comportement d'agents dans des situations précises sans impacter des leviers intégrés à leur comportement déjà établi comme incapables de proposer une augmentation notable du score. Le changement d'algorithme des agents devrait certainement être effectué durant la création de l'essaim, et non pas après.

5.8 Observer le fonctionnement de la répulsion entre agents.

Une modification plus notable est cependant celle de la répulsion, et celle-ci est très notable car elle n'influe pas directement sur k et pourrait donc s'ajouter à sa modification sans causer de conflits. De la formule originale :

Algorithm 1 Alorithme de gestion de la répulsion

Paramètres: repel_vel (Vitesse de répulsion de l'agent), alpha_r (Variable fixée), d (Variable fixée), agent (notre agent)

1: **for** each neighbor of the agent (k) **do**
2: $vector = agent.position - neighbor.position$
3: $distance = norm(vector)$
4: $repel\_vel \mathrel{+}= (\frac{alpha\_r}{distance})^d * \frac{vector}{distance} * \frac{neighbor.density}{agent.density}$
5: **end for**

Nous en dérivons une amélioration dépendant de la densité locale en y ajoutant la dernière partie de notre quatrième ligne : Ainsi, tout agent va, s'il est dans une zone à haute densité d'agents, repousser de manière plus intense le reste des agents dans le but de limiter le groupement d'agents qui signifierait de fait la présence de zones à faible couverture où la cible pourrait se cacher longuement.

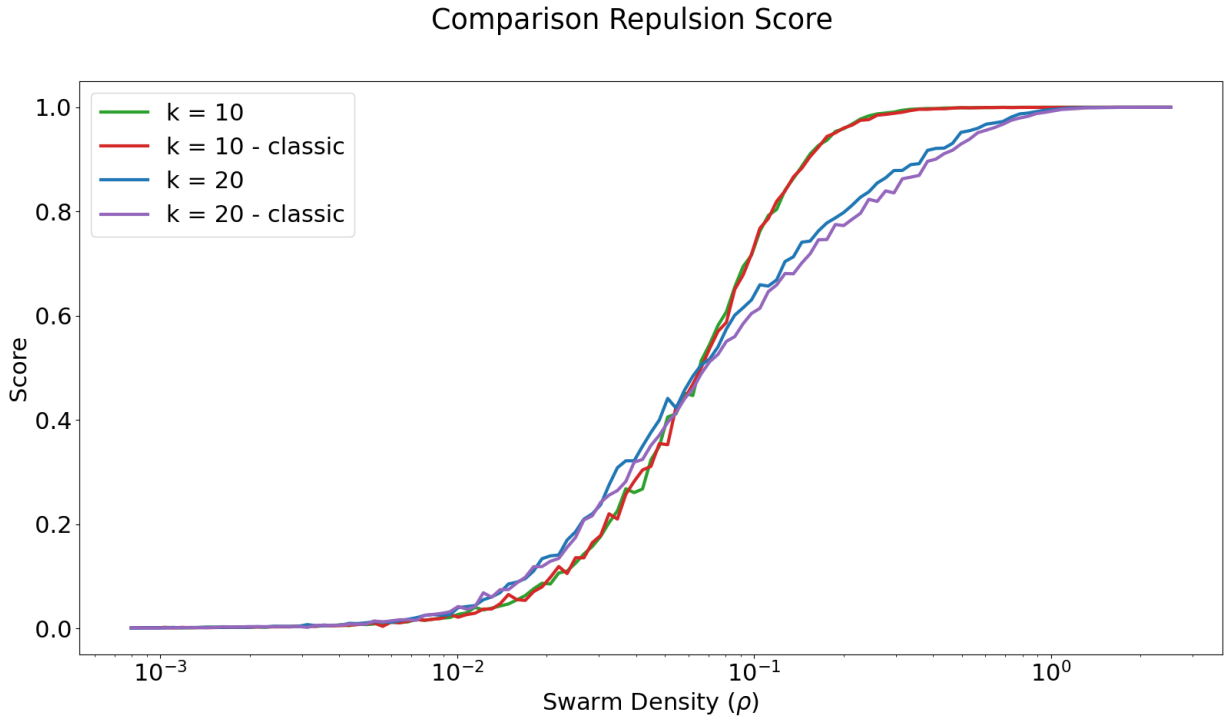

Figure 20

Une amélioration claire de nos résultats est visible pour un nombre élevé de voisins, indiquant la capacité de notre modification à réduire la densité locale produite par la valeur élevée de k en les poussant à s'éloigner les uns des autres malgré une plus grande quantité de communication, là où pour un faible nombre d'agents, on reconnait un gain particulièrement minime, puisque le défaut du groupement de haute densité est moins présent, et que la répulsion de chaque agent dépend de moins de voisins.

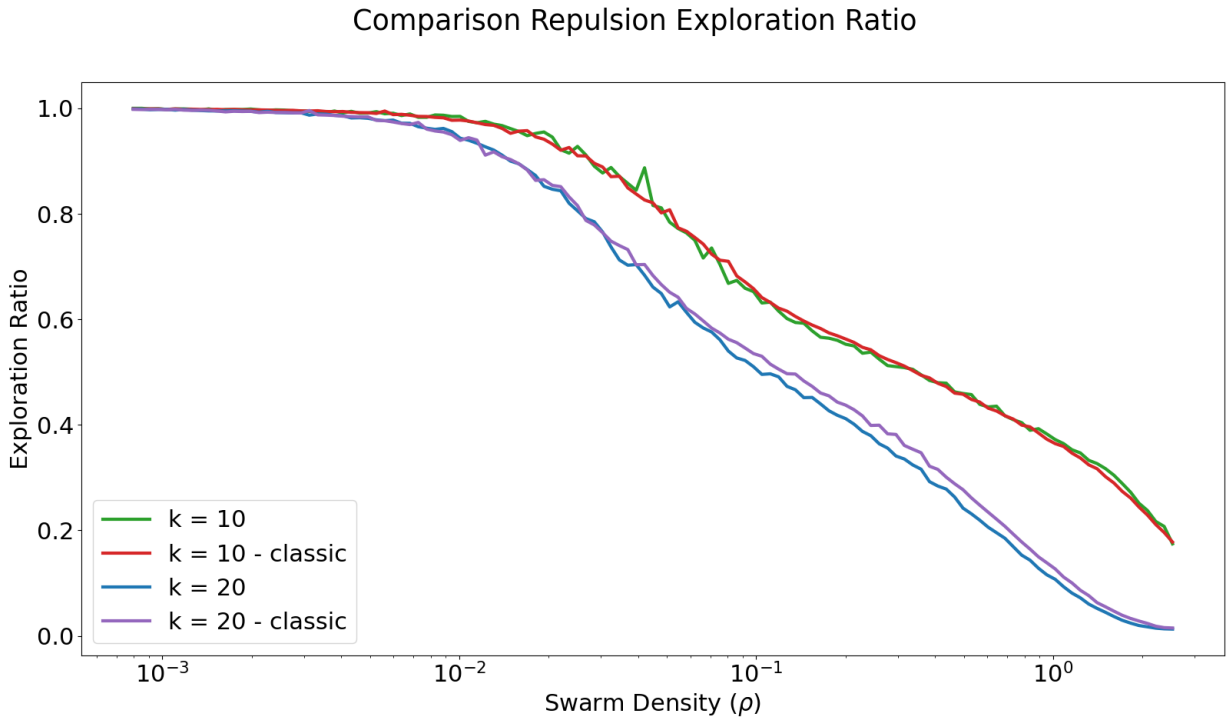

Figure 21

Une importante observation de cette notable amélioration, essentiellement pour k grand, est la réduction généralisée du ratio d'exploration, là où nous l'avons vu, son augmentation devrait permettre l'augmentation de notre score. Une nouvelle considération intéressante est alors soulevée ici : Plus que la balance entre quantité d'exploration et d'exploitation, influe aussi sur le score une largement moins mesurable qualité d'exploration qu'il est possible, avec une telle modification par exemple, de contrôler. C'est essentiellement, ici, une réflexion concernant le design des essaims plutôt que son optimisation.

5.9 Observer l'impact du coût sur le score.

Finalement, intéressons nous de nouveau au coût, dans les deux graphes qui suivent, nous lancerons des simulations avec un coût fixé à un agent rapide valant deux agents lents, puis trois, de sorte à commencer à observer l'effet de ce ratio, et préciser notre compréhension de l'hétérogénéité. Notons toutefois que nous ferons évoluer k, le réduisant au fur et à mesure de la réduction du nombre total d'agents pour empêcher les agents d'avoir accès à l'ensemble des informations.

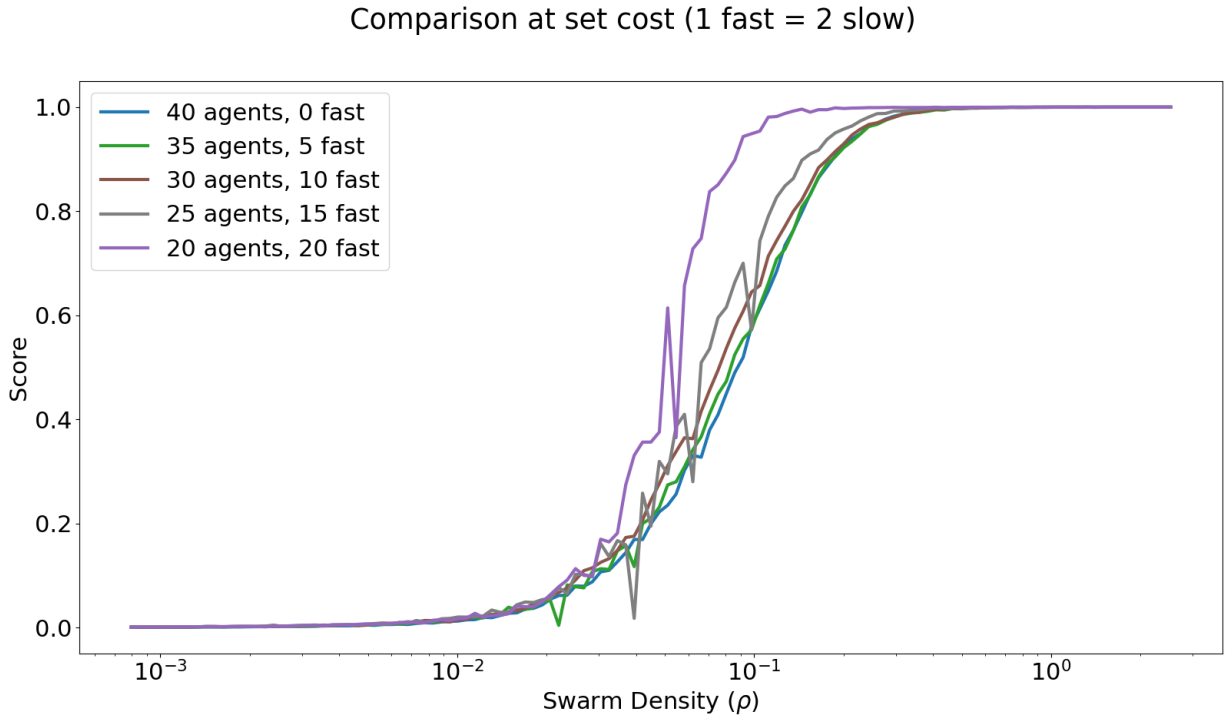

Figure 22

Le premier graphe, celui où un agent rapide vaut deux fois un agent lent, démontre que le ratio n'est pas suffisant pour analyser clairement notre hétérogénéité, démontrant clairement qu'un ensemble d'agents rapides à un aussi faible coût est plus intéressant. Notons tout de même que l'apparition de pics pour les simulations avec le plus faible nombre d'agents semble indiquer qu'une trop faible quantité d'agents, même plus performants, expose au risque de ne pas exploiter ou explorer correctement en raison du hasard des déplacements, et ainsi obtenir un score particulièrement mauvais. C'est ainsi une problématique qu'il est nécessaire de prendre en compte.

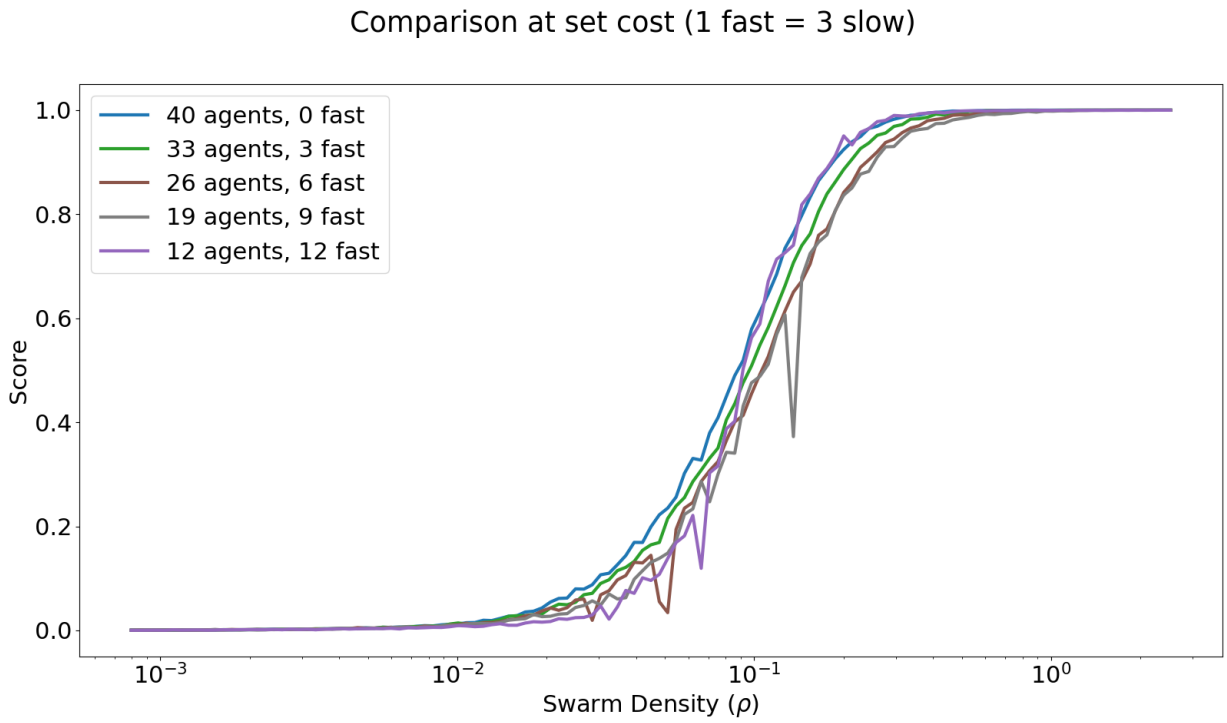

Figure 23

Le second graphe nous expose de nouveau à ces pics, mais, la valeur des agents rapides étant plus équilibrée, nous observons une claire différence : Le mix d'agents lents et rapides est une solution moins efficace que les solutions aux extrêmes, que ce soit en nombre d'agents lents ou rapides. Une seconde observation semble indiquer une meilleure performance de manière générale des agents plus lents et plus nombreux dans le cadre d'une densité faible (i.e une zone particulièrement large) démontrant ce que l'on peut certainement considérer comme une meilleure capacité à retrouver efficacement une cible en raison du nombre très élevé d'observateurs, chose dont n'est pas capable un système avec peu d'agents.

Notons que ces observations restent particulièrement limitées à cet égard et ne servent que d'introduction à une possible recherche plus approfondie. En effet, la façon dont fonctionne notre simulation fait qu'il n'est pas particulièrement aisé de définir un coût réaliste, et que cela relève entre autres de considérations plus complexes qui touchent au design de l'essaim en lui même et qui ne sont pas le focus de ce dossier. Il est en effet possible qu'un agent rapide nécessite un coût plus de 4 ou 5 fois plus élevé, et de fait ne démontre alors aucun intérêt dans une situation réelle.

6 Conclusion

6.1 Résumé des stratégies

Dans le cadre de ce travail, nous aurons ainsi pu obtenir une vue générale des stratégies qu'il est possible d'adopter dans le cadre d'implémentation de systèmes multi-agents pour optimiser le score de tracking de ceux-ci. En s'intéressant au nombre de voisins de chaque agent et manipulant dynamiquement cette valeur, nous aurons pu confirmer qu'il existe une valeur optimale de k pour toute taille de zone de recherche. Cette valeur optimale de k peut être contrôlée sur l'ensemble du système ou adaptée aux actions de l'agent au moment t, et peut être contrôlée en observant le ratio d'exploration local, ou encore la densité locale autour d'un agent.

D'autres stratégies peuvent être implémentées avec moins de succès, la manipulation du comportement d'un agent dans des cas précis n'étant que très rarement la plus efficace des tentatives. Cependant, s'intéresser à la formule de la répulsion semble être une autre façon d'améliorer notre score, optimisant la qualité de notre exploration plutôt que de manipuler la quantité d'exploration et d'exploitation.

Finalement, nous aurons pu voir l'effet du coût sur nos systèmes, révélant une nouvelle façon de penser l'hétérogénéité, démontrant qu'il serait aussi théoriquement possible, en s'intéressant aux systèmes multi-agents par ce prisme, d'optimiser la composition d'un essaim en fonction de la densité ou de la taille de l'environnement dans lequel il évolue.

6.2 Perspectives d'évolution

La valeur de k est, nous l'avons vu, un élément clef à l'optimisation de notre MAS. Cependant, vient une question de réalisme qui n'est pas suivie par notre code : Le nombre de voisins k ne devrait-il pas, dans une situation réelle, plutôt dépendre de la distance par rapport aux autres agents en considérant une distance maximale d'émission de leurs informations pour chacun d'entre eux ? Un agent pourrait-il, pour chaque valeur importante, que ce soit la densité locale, la position de la cible, le mode des autres agents et de multiples autres potentielles valeurs, ajuster son k pour exploiter chacune d'entre elles dans une mesure différente, peut-être même en l'adaptant à la situation de tracking et d'exploration, de sorte à optimiser au maximum notre balance ?

A cela s'ajoute l'idée du coût, si nous l'avons exploitée pour l'idée des agents plus lents ou rapides, se pose aussi la question pour leur qualité et distance de communication. Ce coût est une direction de recherche particulièrement vaste. Nous avons déjà pu témoigner du meilleur fonctionnement à coût fixe d'une composition d'agents par rapport à une autre, ainsi qu'observer la qualité des agents rapides à produire un gain de score plus notable lorsque la densité du système est élevée que quand elle est faible. Viennent ainsi la question de son optimisation et son extension pour prendre en compte des valeurs plus diverses qu'un simple ratio de valeur associé à deux types d'agents différents.

6.3 Mot Personnel

Ce travail fut une intéressante plongée dans le monde de la recherche, si le fait de partir d'une simple base de recherches et de code aura nécessité un certain temps d'adaptation, le fait de comprendre, interpréter et générer des résultats déjà existants aura été un bon tremplin pour par la suite s'atteler à la réflexion sur des stratégies plus novatrices. Le travail de recherche reste une pratique exigeante, l'interprétation et la réflexion pour la mise en place de stratégies diverses largement plus compliquée que je ne l'aurais cru au premier abord, mais qui en vaut la peine.

Merci à M.Bouffanais d'avoir proposé ce travail de Bachelor et pour l'accompagnement tout au long du semestre. Merci aussi à Julien Philippot pour pour son accueil, ses conseils et remarques lors de nos rencontres. Merci aussi à Hian Lee Kwa pour la production du code sur lequel j'aurai basé ce travail, ainsi que son retour sur mes quelques remarques.

6.4 Contenu du projet

En suivant ce lien, vous pourrez trouver le contenu du code utilisé lors de la production des résultats de ce rapport. Le code n'a pas originellement été conçu pour être partagé, de fait, en cas de manque de clarté ou de mauvais fonctionnement de certaines parties de celui-ci, n'hésitez pas à me contacter par mail.

References

O Berger-Tal, J Nathan, E Meron, and D Saltz. The exploration-exploitation dilemma: A multidisciplinary framework. 2014. doi:https://doi.org/10.1371/journal.pone.0095693.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. URL https://doi.org/10.1137/141000671.

R Bouffanais, Kit J Leong, and HL Kwa. Balancing collective exploration and exploitation in multi-agent and multi-robot systems: A review. 2022. doi:https://doi.org/10.3389/frobt.2021.771520.

R Bouffanais, Kit J Leong, and HL Kwa. Effect of swarm density on collective tracking performance. 2023. doi:https://doi.org/10.1007/s11721-023-00225-4.

Michael Bowling. Convergence and no-regret in multiagent learning. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004. URL https://proceedings.neurips.cc/paper_files/paper/2004/file/88fee0421317424e4469f33a48f50cb0-Paper.pdf.

S. Carmel, D.; Markovitch. Exploration strategies for model-based learning in multi-agent systems: Exploration strategies. 1999. doi:https://doi.org/10.1023/A:1010007108196.

Jonathan D Cohen, Samuel M McClure, and Angela J Yu. Should i stay or should i go? how the human brain manages the trade-off between exploitation and exploration. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 362(1481):933–942, 2007. doi:10.1098/rstb.2007.2098. URL https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2007.2098.

Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6: 28573–28593, 2018. doi:10.1109/ACCESS.2018.2831228.

R. A. Johnstone, S. R. X. Dall, Luc–Alain Giraldeau, Thomas J. Valone, and Jennifer J. Templeton. Potential disadvantages of using socially acquired information. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 357(1427):1559–1566, 2002. doi:10.1098/rstb.2002.1065. URL https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2002.1065.

V. Julian, V.; Botti. Multi-agent systems. 2019. doi:https://doi.org/10.3390/app9071402.

James G. March. Exploration and exploitation in organizational learning. *Organization Science*, 2(1):71–87, 1991. doi:10.1287/orsc.2.1.71. URL https://doi.org/10.1287/orsc.2.1.71.

Tal Avgar Oded Berger-Tal. The glass is half-full: Overestimating the quality of a novel environment is advantageous. 2012. doi:https://doi.org/10.1371/journal.pone.0034578.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Hamed Rezaee and Farzaneh Abdollahi. Average consensus over high-order multiagent systems. *IEEE Transactions on Automatic Control*, 60(11):3047–3052, 2015. doi:10.1109/TAC.2015.2408576.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

Dayong Ye, Minjie Zhang, and Athanasios V. Vasilakos. A survey of self-organization mechanisms in multiagent systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(3): 441–461, 2017. doi:10.1109/TSMC.2015.2504350.

An-Min Zou, Krishna Dev Kumar, and Zeng-Guang Hou. Distributed consensus control for multiagent systems using terminal sliding mode and chebyshev neural networks. *International Journal of Robust and Nonlinear Control*, 23(3):334–357, 2013. doi:https://doi.org/10.1002/rnc.1829. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.1829`.