M1 Informatique - Projet ANDROIDE

Rapport

Année : 2020/2021

Robotique en essaim et apprentissage: influence du périmètre d'apprentissage et de la récompense

Cédric Cornede

Manel Khenifra

Damien Marillet

Encadrant:
Nicolas Bredeche

Contents

1 Problématique

En intelligence artificielle la notion d'apprentissage consiste pour un agent autonome tel qu'un robot à apprendre les meilleures décisions à prendre à partir de multiples expériences de façon à optimiser une récompense quantitative au cours du temps. Un agent va chercher dans le cadre de l'apprentissage à adopter le meilleur comportement c'est à dire à maximiser la somme des récompenses au cours du temps. C'est donc cette notion de récompense plus communément appelée fitness ou fonction objectif qui va influer grandement sur le comportement des agents, nous allons chercher à trouver la meilleure fonction objectif dans le cadre d'une tâche donnée à réaliser par plusieurs agents.

Dans le cadre de notre projet, nous chercherons à étudier différentes méthodes de récompenses pour optimiser l'apprentissage dans un cadre de robotique en essaim.
Nous nous inspirerons pour notre implémentation de l'expérience de E. Ferrante[1] dans laquelle des robots doivent apprendre à se coordonner pour ramener des items d'une certaine zone à une autre une zone de dépôt prédéfinie.

Nous utiliserons des algorithmes évolutionnaires et une approche clonale pour permettre un apprentissage en garantissant que chaque évolution voit tous les agents partager un seul et même génôme.
En revanche ces agents initialisés aléatoirement seront récompensés avec des méthodes de fitness différentes ce qui aura un impact important pour le processus de sélection et donc l'apprentissage en général.
Nous comparerons ces différentes méthodes ainsi que l'impact du principe de pré-évolution pour optimiser l'apprentissage de nos agents dans le cadre de notre expérience.

2 Implémentation et expérience

Dans le but de mettre en place des éléments de réponses à nos différentes questions nous nous sommes basés sur l'article de E. Ferrante[1] mettant en place des équipes simulées de robots qu'ils ont ensuite fait artificiellement évoluer pour atteindre des performances maximales dans une tâche de recherche de nourriture.

2.1 Roborobo

Pour mettre en place nos différentes expériences nous avons utilisé Roborobo qui est un simulateur de robot pour des expériences de robotique collective à grande échelle. Ce simulateur nous a donc permis de modéliser un environnement et des agents pour ensuite réaliser différentes tâches telles qu'aller et chercher des

objets pour ensuite les ramener au nid.

La structure de Roborobo ainsi que de notre projet se décompose en 3 grandes classes primordiales :

- La classe Controller qui représente un agent et dans laquelle nous gérons tout ce qui lui est propre. Nous pouvons ainsi définir le comportement de nos agents et leurs prochaines actions à effectuer dans le simulateur avec la méthode step() qui est appelée à chaque pas de temps de la simulation. Dans cette classe nous influons également sur les caractéristiques des agents tel que leurs vitesses ou leurs orientations définies par les 2 attributs suivants : translation et rotation.

- La classe WorldObserver qui permet de gérer l'environnement dans lequel nos agents vont évoluer où nous nous occupons plus précisément de placer les différents éléments physiques extérieurs de notre setup comme la nourriture devant être ramenée au nid ou encore les murs délimitant notre zone d'expérimentation, nous y intégrons également les différentes délimitations pour nos différentes zones. C'est également dans cette classe que l'environnement est remis à un état initial à chaque fin de simulation et que nous replaçons agents et objets dans l'environnement.

- La classe Objects dans laquelle différents types d'objets ainsi que leur différentes caractéristiques peuvent être crées. On y gère notamment la collision avec les différents agents présents dans l'environnement. C'est dans cette classe que nous avons crée notre type d'objet Feuille() correspondant à la nourriture que doivent ramener les agents au nid.

2.2 Terrain et objets

Nous avons donc simuler le terrain dans lequel les fourmis graviraient un arbre pour y récolter des feuilles, voici une visualisation de notre terrain sur Roborobo :

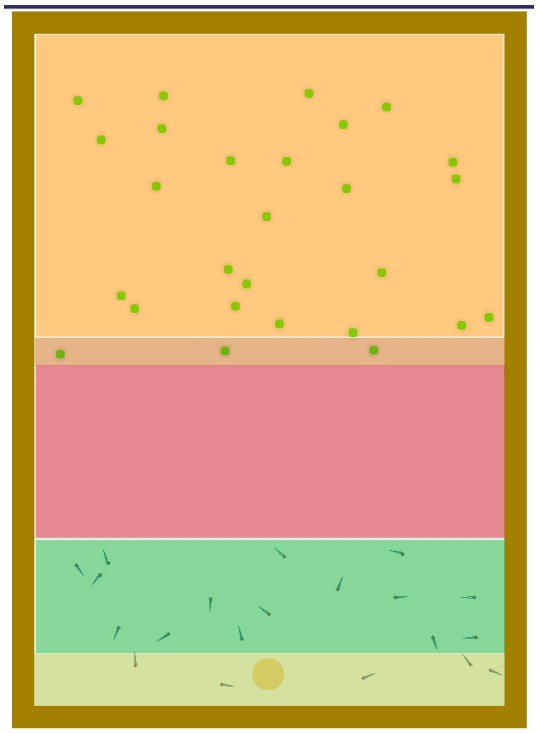


Figure 1: Capture d'écran d'une simulation

Notre terrain est composé de trois zones :

- La zone de collecte d'objets (zone orange) dans laquelle les agents peuvent aller récupérer les objets, ces derniers réapparaissent après un certain temps lorsqu'ils sont déposés au nid afin de simuler l'abondance du feuillage dans l'arbre une fois que l'objet est déposé au nid un booléen passant à True enclenche la décrémentation d'un compteur à chaque pas de temps dans la simulation de l'objet et une fois ce compteur à zéro l'objet est replacé à une position aléatoire dans la zone de collecte.
- La zone de la pente (zone rouge) que les agents doivent gravir afin de parvenir à la zone de collecte, ils ralentissent ici car ils sont soumis à la gravité dans les deux sens de circulation, en effet cette zone étant censée représenter la montée d'un arbre il est logique que nos agents soient ralentis dans les deux sens de circulation car une fourmis voulant descendre d'un arbre ne va pas s'y jeter. Lors de nos simulations nous modifions/ralentissons donc la vitesse de nos agents en fonction de leurs vitesses

actuelles car même soumis à la gravité il paraît incohérent que chaque fourmis se déplace exactement à la même vitesse. Un agent lachant un objet en haut de la pente (zone marron clair) verra l'objet réapparaitre à la même coordonnée en abscisse mais en bas de la pente en ordonnée.

- La zone de dépôt représentant le nid (zone vert clair), c'est ici que les agents déposent les objets collectés qui vont être comptés par notre simulateur. Un agent étant présent dans cette zone aura la possiblité de lacher son objet et de voir le score et/ou sa fitness augmenter en conséquence.

2.3 Agents et réseau de neurones

Nos agents agissent et réagissent à leur environnement grâce à un réseau de neurones à une couche cachée.

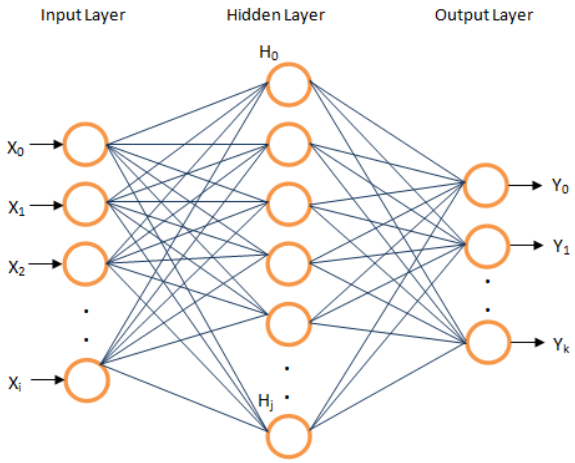


Figure 2: Exemple de réseau de neurones à une couche cachée

Ces réseaux possèdent 12 valeurs en entrées: les 8 premières correspondent aux valeurs de distances des 8 senseurs disposés autour d'eux et qui leur permettent de détecter des collisions avec des murs, des agents ou des objets.

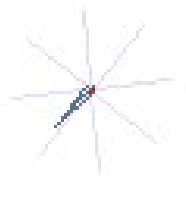


Figure 3: Agent

3 autres valeurs leurs permettent aussi de connaitre leur environnement.
Un booléen pour savoir s'ils ont observés un objet (i.e. une feuille), cela nous permet de différencier les comportements où une collision est faite avec un agent/un mur et avec une feuille: idéalement un agent devrait se diriger vers une feuille et au contraire éviter les obstacles.

Un autre booléen permet de savoir si notre agent possède actuellement un objet ou non. On espère idéalement qu'il ne cherche par exemple plus à perdre du temps à se diriger vers des feuilles lorsqu'il en transporte déjà une.
Enfin les dernières valeurs sont celles de la position de l'agent, qui lui permet de se représenter au moins de manière abstraite les différentes zones de l'environnement, et celle pour finir du biais, nécessaire au bon fonctionnement du réseau de neurones.

Nous avons fait le choix arbitraire d'avoir 14 neurones dans notre couche de neurones cachés.

De plus les réseaux possèdent 4 sorties:

- La valeur de translation, comprise entre -1 et 1, qui détermine la vitesse de déplacement de l'agent
- La valeur de rotation, comprise entre -1 et 1, qui lui permet de pivoter sur lui-même
- Un booléen wantDrop codant le dépôt ou non d'objet, lorsque l'agent a un objet
- Un booléen wantTake codant si l'agent décide de récupérer un objet lorsqu'il en aura l'occasion

2.4 Sélection et évolution

Nous utilisons pour effectuer l'apprentissage de nos agents un algorithme évolutionniste de type ES-(mu,lambda) qui repose sur le principe de la sélection naturelle: ici les mu agents ayants été le plus récompensés (i.e. leur comportement leur a attribué une meilleure valeur de fitness) sont sélectionnés pour pouvoir générer

une nouvelle génération d'individus à partir de leurs génomes, auquels on vient appliquer en guise de mutation un bruit gaussien.

Il est important de noter que pour éviter de rester bloquer dans des maximums locaux nous ne gardons pas les meilleurs parents tels quels d'une génération à une autre, pour éviter un processus élitiste.
Cependant nous gardons en mémoire le génome ayant obtenu le meilleur score dans un HallOfFame. Celui-ci nous a par exemple été utile par la suite pour démarrer nos simulations depuis une pré-évolution plutôt que depuis des poids aléatoires initialement.

Nous sommes également dans un paradigme d'approche clonale, c'est-à-dire qu'en chaque simulation de notre programme tous les agents possèdent le même génome. Les scores et fitness individuels obtenues par chaque agent sont ensuite sommées ensemble pour attribuer une valeur de performance au génome.
Enfin pour palier au caractère aléatoire de certains comportements, et différences à l'initialisation nous faisons la moyenne sur plusieurs runs de chaque génome pour lui attribuer une note plus représentative. Dans notre implémentation nous effectuons exactement 3 répétitions par génome.

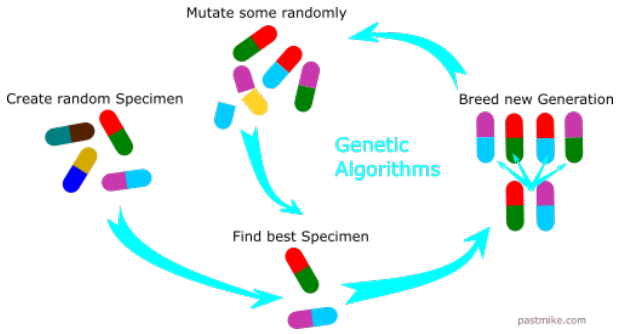


Figure 4: Principe des algorithmes d'apprentissage évolutionnistes

Voici à quoi correspondent les différents termes utilisés dans notre cas :

- Une génération : Groupe constitué de plusieurs génomes.
- Un génome/Spécimen : Les poids d'un agent/robot fonctionnant avec un réseau de neurones.
- Sélection : On utilise la méthode de sélection appelée "sélection μ - λ "(mu-lambda)" comme expliqué précédemment.
- Fitness : Fonction permettant d'évaluer un génome et avec lesquelles nous allons jouer et observer les différences induites par leurs modifications car

la fitness permet de définir un comportement que l'on souhaite obtenir chez nos agents.

- Mutation : La mutation permet de modifier légèrement un génome, dans notre cas elle correspond à appliquer un bruit gaussien sur notre génome.

3 Comparaison de techniques d'optimisation évolutionnaire

3.1 Score et fitness

Nos études de performance se feront à l'aide de deux mesures; celles de la fitness et du score des génomes. La fitness qui est une mesure dont les variations en fonction du comportement d'agents vont permettre de pénaliser ou de récompenser nos génomes pour effectuer un apprentissage correct.

Cependant deux méthodes de récompense ne peuvent pas être comparées avec leurs valeurs de fitness directement. Les ordres de grandeurs et l'aspect général peuvent être totalement différent ce qui rendrait absurde une quelconque comparaison. En revanche la visualisation de ces valeurs va tout de même nous permettre de voir qualitativement si une fitness va permettre un apprentissage correct ou non (arrive-t'on à optimiser cette fitness ?).

Pour les comparaisons entre deux fitness différentes en revanche on ne peut comparer qu'avec le score: ici, cela correspond au nombre de feuilles ramenées depuis la zone de récupération des objets jusqu'au nid. Ainsi on peut comparer les performances des fitness entre elles selon un critère objectif et qui témoigne du bon comportement de nos agents.

3.2 Méthodologie

Toutes nos fitness ont été évaluées sous les mêmes conditions et avec les mêmes paramètres :

- Notre population est de taille 20.
- On évalue une fitness sur 150 générations.
- Chaque génome est testé sur 3000 itérations.
- Chaques génome est évalué trois fois et on fait une moyenne de ses résultats pour le juger.
- Pour chaque fitness nous avons 5 réplicas, ce qui va nous permettre de tracer des boxplots pour chaque génération et ainsi identifier les valeurs extrêmes et comprendre la répartition des observations sur chaque génération pour nos fitness.

- $\mu = 5$, dans notre algorithme évolutionniste on sélectionne donc les 5 meilleurs parents.
- Dans notre algorithme évolutionniste on applique un bruit gaussien avec $\sigma = 0.01$

Les graphes avec les boxplots qui vont être utilisés pour évaluer les performances de nos agents et l'efficacité des méthodes de fitness ont été obtenus selon la méthodologie suivante :

Pour chaque fitness implémentée nous effectuons 5 simulations avec les paramètres cités précédemment (ces paramètres restent les mêmes dans chaque étude), et au cours de ces simulations nous conservons en mémoire la meilleure valeur de fitness et de score obtenue pour chaque génération.

Nous traçons ensuite les boxplots toutes les 10 générations avec ces 5 meilleures valeurs de fitness ou de score pour obtenir nos graphes.

3.3 Comparaisons de fitness pour les performances d'apprentissage

Lors de nos premières expériences nous avons tout d'abord modéliser un setup dans lequel les agents ne pouvaient déposer des objets n'importe où, un agent possédant un objet ne pouvait donc que le garder indéfinimment ou le déposer au nid. Avec ce premier setup nous avons pu comparer deux fitness d'apprentissage :

- $Fitness = Score$: La fitness basique qui correspond à récompenser l'agent lorsqu'il dépose un objet au nid.
- $Fitness = Distance\ Euclidienne$: Une fitness de distance euclidienne qui récompense l'agent qui a un objet plus il est proche du nid avec mais il n'est cependant pas récompensé pour déposer un objet au nid. Après nos 5 réplicas on obtient la figure ci dessous :

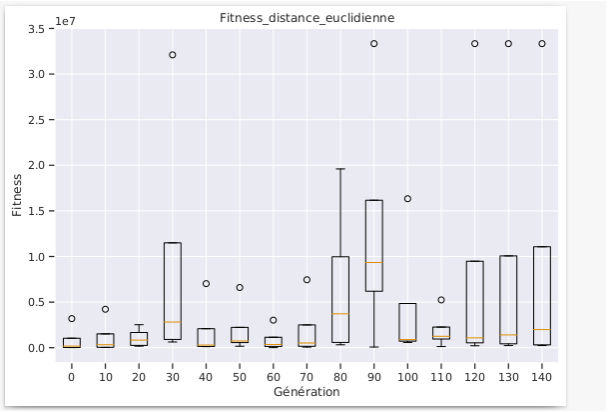
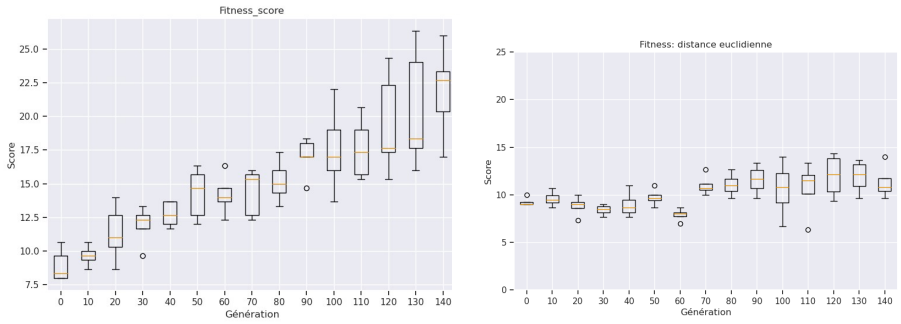


Figure 5: Performance de la fitness

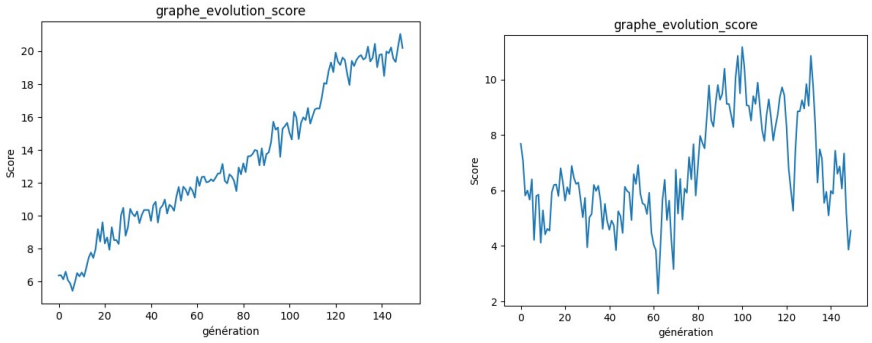
On obtient donc les scores respectifs en 5 réplicas dans les figures suivantes:



(a) $Fitness = Score$ (b) $Fitness = Distance\ Euclidienne$

Figure 6: Comparaison des scores du premier setup

Sur ces expériences on peut donc constater que la fitness de base correpondant au score est finalement assez efficace par rapport à une fitness de direction pure comme la distance euclidienne. On constate en effet qu'avec la distance euclidienne pure la quantité d'objets ramenés au nid reste inférieure à celle avec la distance euclidienne. Au vu de l'évolution de nos figures on peut supposer facilement que cet écart va se creuser de plus en plus, on constate en effet une progression constante du score de la $Fitness = Score$ contrairement à la $Fitness = Distance\ Euclidienne$ qui stagne dans les mêmes plages de valeurs on se doute alors qu'avec plus de générations un écart encore plus flagrant serait observable. Cette différence se voit également avec les courbes d'évolution de leur score où l'on voit clairement la différence d'évolution:



(a) $Fitness = Score$ (b) $Fitness = Distance\ Euclidienne$

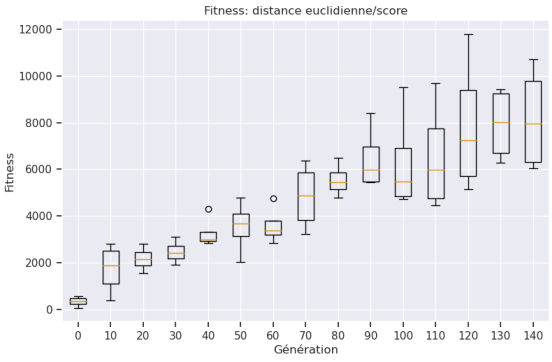
Figure 7: Graphe d'évolution des scores

Avec ces résultats on constate alors que l'utilisation d'une fitness d'apprentissage récompensant un déplacement dans une direction donnée se révèle finalement bien moins efficace que la fitness de base récompensant le dépôt d'objets au nid qui elle finalement s'avère assez efficace.

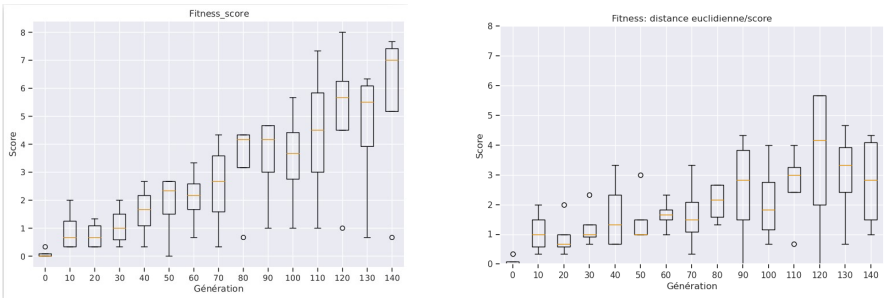
Après avoir implémenté un setup dans lequelle les agents peuvent lacher des objets n'importe où sur le terrain et également en haut de la pente afin que les objets réapparaissent en bas nous avons décidé de comparer une nouvelle fois la $Fitness = Score$ avec une nouvelle fitness :

- $Fitness = Distance/Score$:

  La fitness combinant nos deux précédentes, un agent est récompensé par sa distance euclidienne au nid avec un objet tant qu'il n'a pas laché cet objet au nid et s'il lache cet objet au nid il sera alors récompensé pour avoir déposé un objet dans la zone du nid (il n'est pas récompensé par la distance euclidienne et le lacher au nid mais bien par l'une ou l'autre). On obtient alors la figure ci dessous:



On obtient également les scores respectifs des deux fitness avec ce nouveau setup :



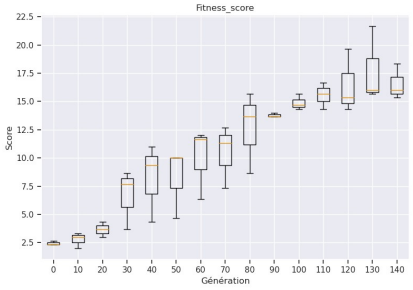
(a) $Fitness = Score$ (b) $Fitness = Distance/Score$

Figure 8: Comparaison des scores

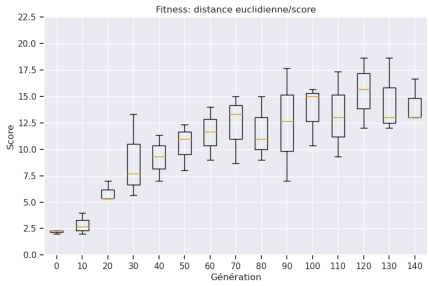
On s'apercoit maintenant qu'il y a une différence avec ce nouveau setup, en effet moins d'objets sont ramenés au nid par $Fitness = Score$ et l'on s'apercoit que $Fitness = Score$ et $Fitness = Distance/Score$ sont assez similaires, on peut même constater que la médiane de $Fitness = Score$ est largement supérieure sur les dernières générations ce qui est assez étonnant car dans le principe $Fitness = Distance/Score$ est censée être une amélioration de cette dernière. Plusieurs hypothèses nous sont donc venus en tête concernant ces résulats :

- Avec le nouveau setup beaucoup de possibilités s'offrent aux agents, ce qui change par rapport au dernier, en effet avec une plus grande complexité de possiblités par rapport à avant il y a encore trop peu de générations pour laisser pleinement ces deux fitness se développer et se départager l'une de l'autre, il en nécessiterait donc beaucoup plus comparé au dernier setup.
- Jusqu'à maintenant pour l'initialisation de nos génomes nous utilisions une initialisation avec un génome déjà performant tiré d'une simulation qui a pour fitness d'apprentissage la distance euclidienne d'un agent à la zone des feuilles, hors ce bon génome était tiré d'une run avec notre premier setup où les agents ne pouvaient déposer les objets n'importe où, ce qui fait une grosse différence avec maintenant et cela n'est peut être plus adapté.

Nous avons alors décidé de recomparer nos fitness cette fois avec une nouvelle initialisation qui aiderait encore plus nos agents car nous ne pouvions pas nous permettre de faire beaucoup plus de générations:



(a) $Fitness = Score$



(b) $Fitness = Distance/Score$

Figure 9: Comparaison des scores

Le fait d'avoir utilisé un meilleur génome à l'initisalisation a amélioré notre apprentissage, en effet les scores ont bien augmenté par rapport aux résultats précédents après un pré-apprentissage plus performant. On s'apercoit ensuite qu'il est toujours difficile de différencier les deux fitness qui sont assez similaires bien que la $Fitness = Score$ soit légèrement meilleure. Avec les formes de nos boxplot on pourrait supposer que la complexification d'un fitness comme par exemple $Fitness = Distance/Score$ qui est une version plus détaillée et améliorée de $Fitness = Score$ mettra plus de temps à se démarquer sur peu de générations par rapport à une fitness plus basique. La seule récompense pour un agent avec $Fitness = Score$ correspond à déposer un objet au nid ce qui est le seul moyen de faire progresser sa fitness alors qu'avec $Fitness = Distance/Score$ un agent peut également faire progresser sa fitness en se baladant avec un objet et étant récompensé avec la distance euclidienne et ne voit peut être pas tout de suite l'avantage de la récompense à déposer un objet au nid et mettra donc plus de temps à obtenir des résultats.

On va comparer nos précédents résultats à une nouvelle fitness :

- $fitness(x) = \sum_{o}^{carriedObjects} max_t(1.0 - dist_t(o, nid))$

  Avec cette nouvelle fitness chaque agent est récompensé pour chaque objet qu'il a transporté, à chaque objet transporté l'agent va être récompensé soit par 1 si l'objet a été déposé au nid et si ce n'est pas le cas il va être récompensé par un moins la distance euclidienne au nid de la position la plus proche du nid à laquelle l'objet a été déposé. Si un objet est transporté par plusieurs agents alors chaque agent ayant participé au transport de cet objet sera récompensé de la même valeur.
  On obtient les figures ci-dessous:

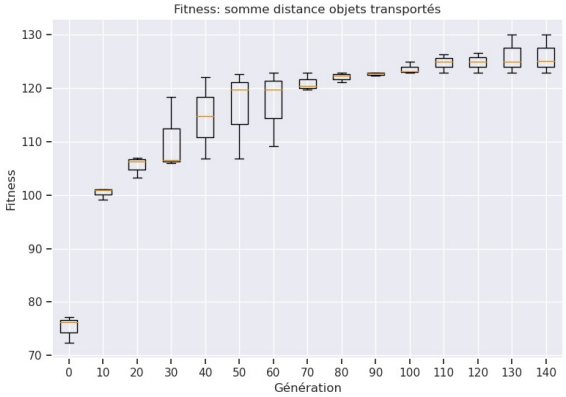


Figure 10: Performance de la fitness

Figure 11: Score

On observe que cette méthode de récompense permet un apprentissage sur les individus correct dans le sens où la courbe de fitness s'optimise bien ce qui témoigne d'un bon processus de sélection mais que ce comportement seul semble ne pas permettre d'obtenir un nombre d'items ramenés au nid très important, contrairement aux résultats espérés.
Le score étant plus haut au début de l'étude peut s'expliquer par la meilleure performance de notre pré-évolution qui est le point de départ de nos agents ici. On constate donc que cette fitness semble bien moins efficace que les deux autres comparées précédemment cependant on pourrait à nouveau également supposer que cette fitness étant plus complexe pousse nos agents à adopter un comportement nécessitant beaucoup plus de générations pour s'avérer efficace car nécessitant un bon partage d'objet intelligent entre les agents pour se révéler avantageuse ce qui peut mettre beaucoup plus de temps à se développer par

rapport aux autres comportements induit par nos deux fitness précédentes.

3.4 Principe de pré-évolution et ses conséquences

Un apprentissage performant peut demander énormement d'évalutations/ générations. Les agents peuvent notamment prendre du temps à dépasser certains seuils comportementaux importants; par exemple apprendre à se diriger vers la zone de récupération des items.

Il peut alors être intéressant d'effectuer une pré-évolution; on va orienter le comportement de nos agents en les récompensant pour un comportement qui ne leur permettront pas nécessairement d'augmenter leur score final tel quel, mais leur permettront de dépasser un cap comportemental, comme une première étape vers un comportement optimal.

Le génome que nous avons sélectionné comme point de départ dans le cadre pré-évolutif est un génome obtenu avec comme fitness $fit(x) = \frac{1}{dist_{ZdF}(x,y)}$ l'inverse de la distance euclidienne à la zone de récupération des feuilles. L'intuition derrière ce choix étant d'avoir une fitness continue qui permettrait à nos agents de savoir se déplacer jusqu'aux objets, comme point de départ des autres études. Ce premier apprentissage a été effectué en 45 générations, et avec une population de taille similaire aux autres études.

Par exemple ci-dessous, nous pouvons comparer les performances obtenues avec et sans pré-évolution, avec la mesure de score, dans le cas où les agents étaient sélectionnés avec comme fitness le nombre d'items ramenés au nid, sans plus d'indications.
Cette fitness est particulièrement appropriée pour l'étude de pré-évolution car les agents ne sont récompensés que tardivement dans leur processus comportemental; tout le temps où ils sont censés apprendre à se déplacer correctement et récupérer les objets dans la zone de récupération des feuilles aucune sélection ne permet de guider l'apprentissage au cours des générations.

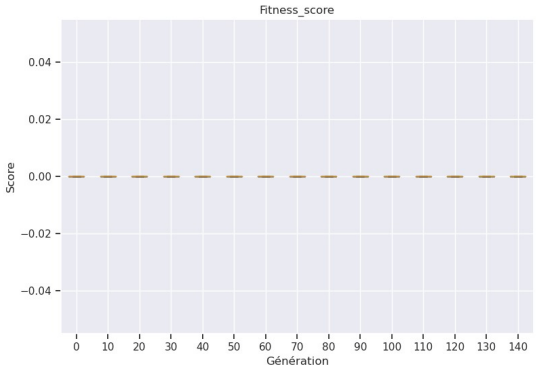


Figure 12: Etude sans pré-évolution

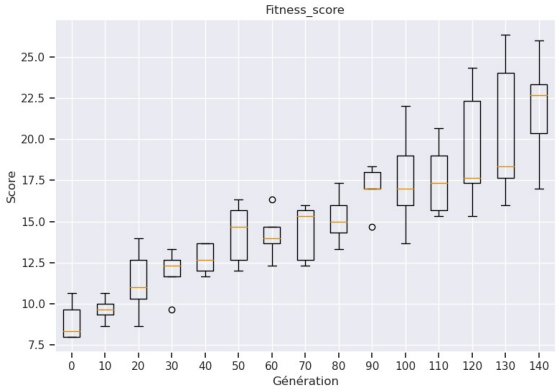


Figure 13: Etude avec pré-évolution

Il faut garder en tête que bien que dans les 2 études les tailles de populations et le nombre de générations semblent être les mêmes, les agents des études pré-évolutives avaient déjà effectué 45 générations lorsque nous débutons nos nouveaux runs.
Néanmoins si l'on regarde le graphe de l'étude sans pré-évolution on voit que même au bout de 45 générations et bien au-dela nos agents n'ont pas su ramené d'objets jusqu'au nid; il leur a été impossible de développer le comportement désiré.
Le comportement à développer avant même d'obtenir la 1ère récompense et donc avant même de pouvoir effectuer une sélection sur les individus qui aient

un sens semble être trop complexe pour réussir à être obtenu avec des poids aléatoires, même sur un nombre énorme de simulations (ici 150).

Tandis que pour l'étude avec pré-évolution on peut voir dès les premières générations certains agents arrivent à ramener des objets au nid grâce au comportement qu'ils avaient déjà développés, ce qui permet alors une sélection et donc un apprentissage efficace comme le témoigne l'allure de la courbe ci-dessus.

Une chose intéressante à noter est qu'une fois cette pré-évolution effectuée, la méthode de récompense $Fitness = Score$ est une fitness très efficace, on pourrait supposer que c'est le cas parce qu'elle récompense de la manière la plus directe l'augmentation du score contrairement aux autres qui vont chercher à optimiser d'autres aspects parallèlement, mais cependant sans pré-évolution la même fitness donne les pires résultats avec des valeurs de fitness nulles sur 150 génération (et donc aucun apprentissage).

Ce qui nous fait observer que certaines fitness peuvent être très efficaces, mais seulement à partir d'un certain point dans le développement de nos agents.

Ici le fait d'apprendre à se déplacer correctement jusqu'aux objets, d'en récupérer un, puis d'en ramener un au nid et un comportment beaucoup trop complexe pour être obtenu purement aléatoirement. Mais une fois ce cap passé la méthode $Fitness = Score$ est un outil très efficace d'apprentissage malgré sa simplicité. Plus que d'autres fitness qui récompensent des comportements autre que l'objectif premier de rapporter des objets au nid (par exemple juste réduire distance au nid mais sans obligation de déposer les objets).

4 Conclusion

Dans le cadre de ce projet nous avons pu comparer les différences de résultats obtenues dans la réalisation d'une tâche par différentes récompenses, pour effectuer ces comparaisons nous avons pu nous aider du principe de pré-évolution qui s'est avéré très utile permettant d'obtenir des résultats concrets qu'il nous était impossible d'obtenir dû à la longueur de nos expériences sur un gros nombre de générations. A partir de nos nombreuses comparaisons nous avons pu constater que dans le cadre de notre expérience la fitness la plus basique consistant à récompenser un agent après qu'il ait déposé un objet au nid s'est montrée finalement plus efficace que ce que nous aurions pu imaginer, elle n'a finalement que rarement été égalée par les autres fitness que nous avons pu tester. A travers les tests et en partant de différentes initialisations on peut également supposer que cette performance vient de la simplicité de cette fitness qui n'est récompensée que par une seule action et n'induit pas de comportement particulièrement fastidieux à mettre en place et donc que cette fitness serait une des

meilleures sur le court terme pour obtenir des résultats rapidement en peu de générations.

Il aurait été pertinent dans l'optique de déterminer la meilleure fitness de bien évidemment réaliser nos différents tests sur bien plus de générations et de constater sur un ordre de grandeur tel que 3000 générations si la $Fitness = Score$ aurait gardé une même efficacité tout du long. En constatant l'efficacité de cette fitness et les effets de la pré-évolution il aurait pu être intéressant de tester une fitness qui change au cours des générations une fois que la première fitness a été bien appris, changer la récompense pour le reste des générations avec par exemple $Fitness = Score$ dont nous avons pu constaté l'efficacité après avoir passer un certain cap d'apprentissage avec la pré-évolution.

L'approfondissement de la relation entre la complexité de la fonction de récompense et du temps d'obtention de résultats serait également pertinent à étudier, une récompense induisant un comportement coopératif comme nous avons pu le voir avec la fitness récompensant plusieurs agents ayant transportés un objet serait finalement moins avantageuse qu'une fitness plus simple centrée sur l'agent lui-même uniquement car moins difficile à se mettre en place.

5 Bibliographie

Norme utilisée : IEEE

[1] E. Ferrante, A. E. Turgut, E. Duéñez-Guzmán, M. Dorigo et T. Wenseleers, Evolution of Self-Organized Task Specialization in Robot Swarms , PLOS Computational Biology, t.11, n8, p. 1–21, août 2015.doi:10.1371/journal.pcbi.1004273.adresse:https://doi.org/10.1371/journal.pcbi.1004273.

[2] N. Bredèche, J. Montanier, B. Weel et E. Haasdijk, Roborobo! a Fast Robot Simulator for Swarm and Collective Robotics , CoRR, t. abs/1304.2888, 2013. arXiv : 1304.2888. adresse: http://arxiv.org/abs/1304.2888.

[3] Van Essche, Steven Van, Eliseo Ferrante, Ali Emre Turgut, Rinde Van Lon, Tom Holvoet, et Tom Wenseleers. Environmental factors promoting the evolution of recruitment strategies in swarms of foraging robots . In Swarm 2015. Kyoto, Japan, 2015. https://hal.archives-ouvertes.fr/hal-01405907.

[4] E. Haasdijk, N. Bredeche, et A. E. Eiben, Combining Environment-Driven Adaptation and Task-Driven Optimisation in Evolutionary Robotics , PLOS ONE, vol. 9, no 6, p. e98466, juin 2014, doi: 10.1371/journal.pone.0098466.

[5] N. Bredeche, J.-M. Montanier, W. Liu, et A. F. T. Winfield, Environment-driven distributed evolutionary adaptation in a population of autonomous robotic

agents , Math. Comput. Model. Dyn. Syst., vol. 18, no 1, p. 101-129, févr. 2012, doi: 10.1080/13873954.2011.601425.

[6] X. Deng et C. H. Papadimitriou, On the Complexity of Cooperative Solution Concepts ,Mathematics of Operations Research, t. 19, no 2, p. 257–266,1994. doi:10.1287/moor.19.2.257.eprint:https://doi.org/10.1287/ moor.19.2.257.adresse: https://doi.org/10.1287/moor.19.2.257.

[7] P. Stone, G. A. Kaminka, S. Kraus et J. S. Rosenschein, Ad Hoc Autonomous Agent Teams : Collaboration without Pre-Coordination , in Proceedings of the Twenty-Fourth Conference on Artifcial Intelligence, juil. 2010.

[8] P. MacAlpine et P. Stone, Evaluating Ad Hoc Teamwork Performance in Drop-In Player Challenges , in AAMAS Multiagent Interaction without Prior Coordination (MIPC) Workshop, Sao Paulo, Brazil, mai 2017.adresse: http://www.cs.utexas.edu/users/ailab/?macalpine:mipc17.

[9] L. Bayındır, A review of swarm robotics tasks , Neurocomputing, vol. 172, p. 292-321, janv. 2016, doi: 10.1016/j.neucom.2015.05.116.

[10] S.Liemhetcharat et M.Veloso, Weighted synergy graphs for role assignment in ad hoc heterogeneous robot teams , oct. 2012, p. 5247–5254, isbn : 978-1-4673-1737-5. doi : 10. 1109/IROS.2012.6386027

[11] F. Rosenblatt, The Perceptron : A Probabilistic Model for Information Storage and Organization in The Brain , Psychological Review, p. 65–386, 1958.

[12]E. Teruel, R. Aragues, et G. Lopez-Nicolas, A Practical Method to Cover Evenly a Dynamic Region With a Swarm , IEEE Robot. Autom. Lett., vol. 6, n 2, p. 1359-1366, avr. 2021, doi: 10.1109/LRA.2021.3057568.

[13]S. Mukherjee et T. L. Vu, On Distributed Model-Free Reinforcement Learning Control With Stability Guarantee , IEEE Control Syst. Lett., vol. 5, n 5, p. 1615-1620, nov. 2021, doi: 10.1109/LCSYS.2020.3041218.