Optimisation Dichotomique Adaptative : une nouvelle méthode pour le calibrage de modèles à base d'agents.

Benoît CALVEZ
benoit.calvez@ibisc.univ-evry.fr

Guillaume HUTZLER
guillaume.hutzler@ibisc.univ-evry.fr

IBISC
Université d'Évry-Val d'Essonne
CNRS FRE 3190
523, place des Terrasses de l'Agora 91000 Évry – FRANCE

Résumé

Dans ce papier, nous proposons une nouvelle approche pour l'exploration de l'espace d'un modèle à base d'agents : l'optimisation dichotomique adaptative. Un modèle à base d'agents est généralement caractérisé par un grand nombre de paramètres, une partie d'entre eux ne peut pas être choisi étant donnée les connaissances sur le système réel. Le but de ce travail est de fournir des outils pour la calibration d'un tel modèle, ce qui consiste à trouver le jeu de paramètres optimal pour un certain critère. Notre approche consiste en la partition de l'espace de paramètres (l'intervalle variation de chaque variable est divisé en un nombre fini d'intervalle) et sur l'exploration en parallèle de nombreux paramètres par les agents du modèle. La navigation dans l'espace de paramètres est fait en groupant ou divisant de façon adaptative certains paramètres, selon un algorithme s'inspirant des algorithmes de colonie de fourmis.

Mots-clés : *Agent, Simulation à base d'agents, calibrage de paramètres, optimisation par colonie de fourmis*

Abstract

In this paper, we propose a new approach for the exploration of the parameter space of agent-based models : Adaptative Dichotomic Optimization. Agent-based models are generally characterized by a great number of parameters, a lot of which cannot be evaluated with the current knowledge about the real system. The aim of the work is to provide tools for the calibration of these models, which consists in finding the optimal set of parameters for a given criterion. Our approach is based on the partition of the parameter space (the interval of variation of each variable is divided into a finite number of intervals) and on a parallel exploration of the various parameters by the agents of the model. The navigation in the parameter space is done by grouping or dividing adaptively some of the intervals, according to an algorithm which is adapted from Ant Colony Systems.

Keywords: *Agent, Agent-based simulation, parameter calibration, ant colony optimization*

1 Introduction

La simulation à base d'agents s'intéresse à la modélisation et à la simulation de systèmes complexes. L'objectif est de reproduire la dynamique d'un système réel en modélisant les entités composant le système par des agents, dont le comportement et les interactions sont spécifiés, et qui « vivent » dans l'environnement simulé. Une première validation de tels modèles est obtenue en comparant la dynamique résultante, quand le modèle est simulé, avec celle du système réel (mesurée grâce à des données expérimentales).

Un des aspects cruciaux dans le processus de modélisation est lié au calibrage du modèle. En effet, ce genre de modèle est généralement caractérisé par un grand nombre de paramètres qui déterminent la dynamique globale du système. L'espace de recherche est alors gigantesque. De plus, le comportement de ces systèmes complexes est souvent chaotique : d'un côté, de petits changements d'un seul paramètre conduisent parfois à une modification radicale de la dynamique du système entier ; d'un autre côté, certains phénomènes émergents n'aparaissent que dans des conditions très spécifiques. L'espace des solutions peut alors être très petit. La conséquence est que le calibrage de tels modèles peut rapidement devenir long et fastidieux en l'absence de stratégie précise, automatique et systématique pour explorer l'espace des paramètres.

Le développement d'une telle stratégie est précisément l'objectif de ce papier, qui fait suite à plusieurs travaux sur le sujet (comme le montre la section suivante). Ces recherches précédentes apparaissent cependant comme étant trop limi-

tées, soit parce qu'elles permettent seulement une exploration limitée de l'espace des paramètres, soit parce qu'elles requièrent un nombre exagérément grand de simulations pour converger. La principale différence entre l'optimisation classique et le calibrage de modèles à base d'agents est que l'évaluation d'un simple jeu de paramètres requiert au minimum une exécution complète d'une simulation, ce qui nécessite un temps non négligeable. Si le modèle est stochastique, il requiert même plusieurs exécutions pour un même jeu de paramètres, afin d'avoir une confiance correcte dans l'évaluation. Le plus important est alors de réduire au maximum le nombre de simulations si nous voulons disposer d'une méthode utilisable.

L'idée principale de l'approche que nous proposons est d'utiliser l'aspect concurrentiel présent dans les modèles à base d'agents : généralement, un modèle à base d'agents est souvent composé d'un grand nombre d'agents identiques. Nous proposons d'utiliser ce fait pour explorer en parallèle l'espace de recherche. Dans cette approche, que nous avons appelé *Optimisation Dichotomique Adaptative* (ODA), chaque agent individuel est instancié avec des valeurs de paramètres qui lui sont propres. Ces valeurs de paramètres sont choisies en fonction du découpage en intervalles de chaque paramètre. Les différents agents du modèle sont ainsi initialisés avec des paramétrages très différents d'un agent à l'autre. L'idée de base est que la « performance » du modèle résultant sera d'autant meilleur statistiquement q'il inclut une large proportion d'agents avec de « bons » paramètres. L'intervalle dans lequel les valeurs des paramètres ont été choisies pour les agents d'un modèle donnée peut alors être récompensé en fonction de la performance du modèle. L'exploration de l'espace de paramètres sera alors redéfinie de façon adaptative en fusionnant les intervalles adjacents qui ont reçu de faibles récompenses, et en divisant en deux intervalles ceux qui ont reçu des récompenses fortes.

La section suivante décrit différents travaux antérieurs sur le sujet et discute de leurs forces et faiblesses respectives. Nous présentons ensuite en détail notre approche dans la section 3. Nous présentons et discutons quelques résultats dans les sections 4 et 5 avant de conclure en section 6.

2 Travaux antérieurs

Différentes méthodes ont déjà été proposées pour explorer automatiquement l'espace des paramètres de modèles discrets. Dans la plateforme `NetLogo` par exemple, l'outil « BehaviorSpace » [17] permet d'explorer de façon automatique et systématique l'espace des paramètres. Cet espace est un produit cartésien des valeurs que peut prendre chaque paramètre : il est possible de fixer les valeurs des paramètres ou de les limiter à un sous-ensemble de valeurs possibles. Cependant quand un modèle a de nombreux paramètres, dont certains peuvent prendre un grand nombre de valeurs (comme par exemple, un paramètre à valeurs réelles), l'espace de paramètres est alors gigantesque et l'exploration systématique devient impossible. Le nombre de valeurs testées pour chaque paramètre sera alors très faible et l'exploration correspondante limitée à une petite partie de l'espace des paramètres.

D'autres méthodes ont été proposées, qui explorent différentiellement l'espace des paramètres entier, en se focalisant sur les zones les plus intéressantes. C'est le cas de la méthode développée par Brueckner et Parunak [3]. Ces auteurs utilisent une infrastructure de balayage de paramètres (« parameter sweep infrastructure »), qui est similaire à l'outil « BehaviorSpace » de `NetLogo`. Cependant, pour éviter une exploration systématique, ils utilisent des agents chercheurs et introduisent la notion de *fitness* (ou fonction objectif), qui sert à caractériser la qualité d'un modèle. Le but d'un agent chercheur est de voyager dans l'espace de paramètres à la recherche de *fitness* plus hautes. Partant d'un point donné de l'espace de paramètres, les agents chercheurs ont deux choix : bouger ou simuler. Chaque agent choisit selon sa confiance dans l'estimation de la *fitness* (proportionnelle aux nombres de simulations à ce point) et la valeur de la *fitness*. S'il choisit de bouger, il se dirige vers les régions voisines de plus hautes *fitness*. Un désavantage de cette méthode est que les agents chercheurs peuvent se diriger vers des maximums locaux de *fitness*.

Une autre méthode consiste à ajouter de la connaissance à un modèle à base d'agents comme dans le cas du calibrage en boîte blanche [10, 11]. Le principe est d'utiliser cette connaissance sur le modèle à base d'agents pour améliorer le processus de calibrage. Dans ce papier, cette connaissance est utilisée afin de réduire l'espace de paramètres en divisant le mo-

dèle en sous-modèles, ce qui peut être réalisé en utilisant différentes méthodes (*General Model Decomposition, Functional Decomposition, ...*). Chaque sous-modèle est ensuite calibré, puis l'ensemble des sous-modèles est fusionné pour reformer le modèle. Les opérations de division et de fusion sont des étapes difficiles de la méthode. L'opération de division, d'un côté, nécessite l'ajout de connaissances sur le modèles, qui ne sont pas nécessairement disponibles. L'opération de fusion, d'un autre côté, doit fusionner les sous-modèles calibrés dans un modèle de plus haut niveau, ce qui n'est pas automatique si l'on veut assurer que le modèle de plus haut niveau soit lui-même calibré.

Sallans et ses coauteurs [14] présentent un travail sur un modèle avec de nombreux paramètres. Certains de ces paramètres sont choisis à la main en faisant des simulations. Les autres paramètres sont choisis par une adaptation de la méthode de Monte-Carlo par chaînes de Markov pour faire une marche aléatoire dirigée à travers l'espace des paramètres, couplée avec un algorithme de Metropolis. Cette méthode fonctionne bien sur un espace continu, mais sera difficilement utilisable quand l'espace des paramètres est chaotique et quand il y a de la stochasticité dans la simulation.

Une dernière approche consiste à considérer le problème de développement et de validation d'un modèle à base d'agent comme un problème d'optimisation. La validation peut être alors reformulée comme l'identification d'un jeu de paramètres qui optimise une fonction. Une fonction d'optimisation serait par exemple la distance entre le modèle artificiel que nous simulons et le système réel. [4, 5, 13, 12] proposent l'utilisation d'un algorithme génétique avec différentes variantes.

L'utilisation de ces méthodes nécessite le calcul d'une fonction de *fitness*. Le problème, comme nous l'avons souligné dans l'introduction, est que le calcul de cette *fitness* pour un seul modèle requiert potentiellement de nombreuses simulations, chacune d'elle peuvent durer plusieurs heures. Sans un cluster de calcul puissant, la méthode est alors difficile à appliquer.

3 Vision globale de l'*ODA*

Une première idée est d'utiliser le fait que les simulations à base d'agents contiennent de nombreux agents. Cette particularité va être utilisée pour explorer l'espace de paramètres : au lieu de considérer que tous les agents doivent être paramétrés de manière identique, nous proposons que pour une simulation donnée, les différents agents de même type vont être initialisés individuellement avec des jeux de paramètres différents. Au cours d'une simulation, plutôt que d'évaluer un jeu de valeurs de paramètres de façon individuelle, nous pouvons alors évaluer différents jeux de paramètres en parallèle dans un seul modèle au sein d'une même simulation. L'idée sous-jacente est que le résultat d'une simulation sera d'autant meilleur que la simulation compte de « bons » agents, c'est-à-dire des agents ayant un bon paramétrage. D'une certaine manière, l'idée est similaire au principe des algorithmes de colonie de fourmis[9].

Une deuxième idée est d'explorer l'espace de paramètres en fonction de l'intérêt potentiel des différentes régions de l'espace. Tirant inspiration de la recherche dichotomique, nous considérons qu'un espace de paramètres de dimension n (n paramètres indépendants) est initialement divisé en hypercubes de dimensions n. D'un point de vue pratique, l'ensemble de définition de chaque paramètre est initialement divisé en un nombre fixé d'intervalles identiques. Pour chaque paramètre individuel, nous pouvons alors diviser ou fusionner chaque intervalle en fonction des récompenses qu'il a reçues. Si un intervalle a reçu de fortes récompenses, cela montre que cet intervalle est une zone intéressante de l'espace des paramètres. L'intervalle correspondant peut alors être divisé en deux sous-intervalles afin d'affiner l'évaluation sur cette zone. A l'inverse, si deux intervalles adjacents reçoivent peu de récompenses, cela montre que ces intervalles correspondent à une zone de l'espace de paramètres avec un faible intérêt. Ces intervalles peuvent être alors fusionnés afin d'arrêter de perdre du temps dans l'exploration de cette zone.

Lors de la création d'un agent, la valeur de chaque paramètre est choisie de façon aléatoire parmi les intervalles composant l'ensemble de définition du paramètre. Après qu'un modèle ait été évalué (c'est-à-dire en exécutant une ou plusieurs simulations en fonction de la stochasticité et en calculant la *fitness* du modèle), les intervalles dans lequels les paramètres des agents ont été choisis sont récompensés. Pour chaque paramètre, et pour chaque intervalle, la récompense est proportionnelle à la *fitness* globale du modèle et au nombre d'agents qui ont des valeurs de paramètres comprises dans l'intervalle.

Le processus global de l'optimisation dichoto-

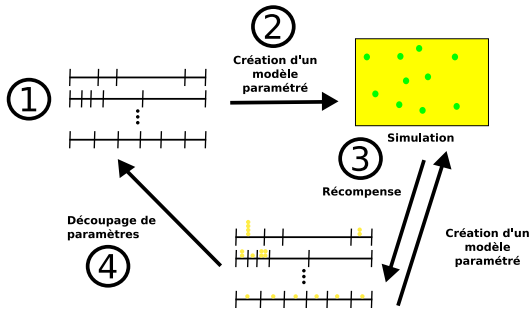

FIG. 1 – Schéma résumant la méthode *ODA*

mique adaptative se déroule de la façon suivante (voir la figure 1) : à partir des intervalles (schéma ①), un modèle est créé (schéma ②). Puis le modèle est simulé et sa *fitness* est calculée. Les intervalles correspondants sont ensuite récompensés (schéma ③). Cette étape est itérée plusieurs fois (cela peut aussi être distribué automatiquement sur plusieurs ordinateurs), jusqu'à que les différents intervalles reçoivent suffisamment de récompenses pour que l'évaluation devienne significative. Pour chacun des paramètres, et indépendamment des autres, le meilleur intervalle (celui ayant les récompenses les plus élevées en moyenne) est choisi, puis est divisé en deux (schéma ④). L'ensemble du processus est itéré jusqu'à sa stabilisation.

Dans cette méthode, nous avons fait l'hypothèse que les différents paramètres sont indépendants les uns des autres, ce qui revient à dire que l'application de l'algorithme *ODA* s'effectue indépendamment sur chacun des paramètres. Dans ce cas, notre espace de paramètres est divisé en n paramètres divisés en m_i intervalles (la taille de l'espace de recherche est alors environ $n \times \mathbb{R}$). En réalité, l'espace de paramètres est un espace de dimensions n (la taille de l'espace de recherche est environ $\mathbb{R}^n$.). Sans cette hypothèse, dans la phase de sélection, le nombre d'intervalles augmente de $2^n - 1$, ce qui pourrait causer des difficultés sérieuses. Avec cette hypothèse, le nombre d'intervalles augmente seulement de n.

Cette hypothèse se justifie par le fait que nous exécutons un grand nombre de simulations avec de nombreux agents, ce qui implique que l'espace de paramètres est globalement couvert. Si nous considérons un paramètre donné, il a été évalué alors que les autres paramètres adoptaient un grand nombre de configurations différentes. Si un intervalle a reçu de fortes récompenses moyennes, cela signifie qu'il conduit à des résultats de simulation intéressants, quelles que soient les valeurs des autres paramètres. Cette hypothèse a été confirmée expérimentalement : nous avons effectué un grand nombre de tests avec différents modèles, et il apparaît que cela reste correct pour toutes les configurations testées. La figure 2 montre la distribution des valeurs des paramètre pour les 100 premières simulation dans le modèle à base d'agents (voir la sous-section 4.1 page 7). Malgré le peu d'agents (25 agents) dans le modèle, la distribution couvre globalement tout l'espace. Il reste naturellement possible de tomber sur des cas défavorables dans lesquels seule une combinaison particulière des valeurs de plusieurs paramètres produirait des résultats de simulation intéressants mais le cas ne semble pas si fréquent que cela, en tout cas pour l'échantillon de modèles testé (modèles de la bibliothèque `NetLogo`). On peut par ailleurs envisager, en cas de problème sur un modèle spécifique, d'affaiblir l'hypothèse en considérant non plus des paramètres individuels mais des groupes de paramètres indépendants.

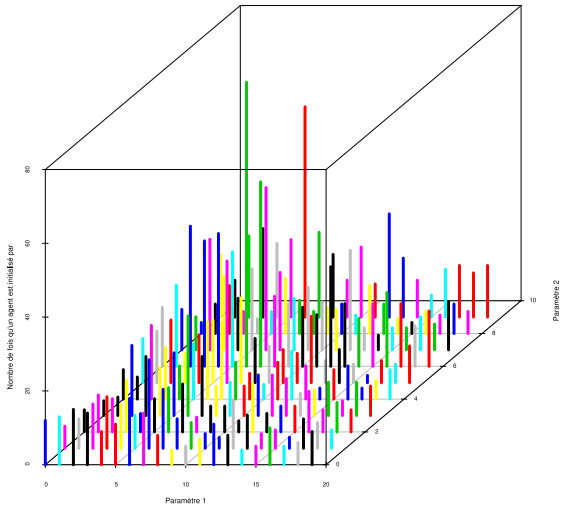

FIG. 2 – Distribution des valeurs de paramètres pour les 100 premières simulations.

3.1 Méthode

Nous avons développé différentes variantes de la méthode ODA. Dans ce papier, nous ne présentons que celle qui produit les meilleurs résultats. Nous présenterons d'abord l'optimisation par colonie de fourmis avant d'exposer notre méthode dans le détail, et les résultats correspondants.

Colonie de fourmis. Les algorithmes colonie de fourmis [1, 6, 7] (ACO) peuvent être considérés comme faisant partie de l'intelligence col-

lective (« *Swarm Intelligence* ») [2], c'est-à-dire la création de systèmes multi-agents intelligents en prenant inspiration de comportements naturels collectifs intelligents chez les animaux sociaux, et plus particulièrement chez les insectes.

Le principe est inspiré par la création de chemins de phéromones, comme dans le modèle du fourragement que nous avons utilisé comme exemple (voir la sous-section 4.1 page 7). Nous nous sommes plus particulièrement inspirés des *Ant systems* [8]. Pour illustrer cet algorithme, nous allons utiliser l'exemple du voyageur de commerce : c'est un graphe complet non-orienté où les nœuds et les arcs représentent respectivement les villes et les distances entre les villes. Les fourmis sont positionnées initialement sur un nœud choisi aléatoirement dans le graphe. Celles-ci explorent ensuite le graphe selon des règles probabilistes qui indiquent, lorsque la fourmi est sur le nœud i, la probabilité de continuer le parcours vers le nœud j :

$$p_{ij}^k = \begin{cases} \frac{(\tau_{ij})^\alpha \times (\eta_{ij})^\beta}{\sum_{l \in J_i^k} (\tau_{ij})^\alpha \times (\eta_{ij})^\beta} & \text{if } j \in J_i^k \\ 0 & \text{if } j \notin J_i^k \end{cases}$$

où τ_{ij} est la quantité de phéromones, η_{ij} est la visibilité ($= \frac{1}{d_{ij}}$ où d_{ij} est la distance entre les villes i et j), et J_i^k est une mémoire des villes déjà visitées.

Après la fin d'un tour, chaque fourmi k dépose une quantité $\Delta\tau_{ij}^k$ de phéromones sur l'arc (i, j) :

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L^k} & \text{si } (i,j) \in T^k \\ 0 & \text{if } (i,j) \notin T^k \end{cases}$$

où T^k est le parcours effectué par la fourmi k à l'itération t, L^k sa longueur, et Q un paramètre.

La quantité de phéromones est alors mise à jour dans le but de simuler l'évaporation, en utilisant la formule :

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \Delta\tau_{ij}$$

où $\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k$ (m est le nombre de fourmis).

Nous pouvons généraliser la méthode avec la méta-heuristique d'optimisation par colonie de fourmis [6]. Le schéma général de la méta-heuristique d'optimisation par colonie de fourmis est :

---

tant que conditions de terminaison non atteintes **faire**
  OrdonnanceActivités
    ConstructionSolutionParFourmis()
    MiseAJourPheromones()
    ActionsDémons()
  fin OrdonnanceActivités
fin tant que

Notre espace de paramètres étant un espace continu, nous nous sommes par ailleurs intéressés à l'algorithme proposé par Socha [15], plus particulièrement adapté à ce type d'espaces.

La méthode en détail. Prenant inspiration des ACO, nous avons fait un parallèle entre les phéromones dans les modèles de ACO et les récompenses données à un intervalle dans l'approche ODA. Plus précisément, c'est la récompense moyenne qui est choisie comme phéromone.

De plus, nous ajoutons une information heuristique et une confiance sur cette information heuristique. Pour chaque intervalle, nous calculons l'information heuristique en trois étapes. Dans la première étape, nous créons un modèle où tous les agents sont initialisés avec le même jeu de paramètres, qui est choisi de la façon suivante :

– pour l'intervalle spécifique, dont nous voulons estimer l'heuristique, nous choisissons la valeur médiane de l'intervalle.
– pour tous les autres paramètres, nous choisissons la valeur médiane du paramètre.

La figure 3 résume le principe.

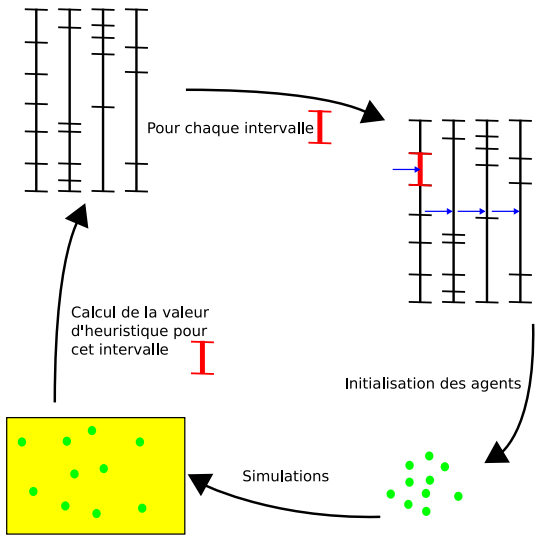

FIG. 3 – Calcul de l'heuristique.

Dans une seconde étape, nous simulons le modèle plusieurs fois (à cause de la stochasticité). Dans la dernière étape, nous calculons l'information heuristique : la valeur de l'heuristique est la valeur moyenne des *fitness* pour les différents exécutions de la simulation. Durant la phase de fusion, la confiance est la moyenne des deux confiances des deux intervalles. Durant une étape de division, la confiance est divisée en deux. Nous calculons de nouveau l'heuristique quand la confiance est plus basse qu'une valeur donnée.

La phase de choix des valeurs de paramètres est modifiée pour distinguer deux étapes. D'abord nous choisissons un intervalle j du paramètre i avec une probabilité proportionnelle à

$$[\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}$$

où τ est la quantité de phéromones de cet intervalle (c'est-à-dire la récompense moyenne), η est la valeur de l'heuristique, et α et β sont des paramètres qui contrôlent l'importance relative des phéromones par rapport à l'heuristique. Nous choisissons ensuite une valeur dans l'intervalle avec une probabilité uniforme.

Les phases de division et de fusion sont aussi modifiées. Un intervalle j du paramètre i est divisé si $\tau_{ij} > \overline{\tau_i} + 2 \times \sigma_i$, où σ_i est l'écart-type. Un intervalle j du paramètre i est fusionné si $\tau_{ij} < \overline{\tau_i} - \sigma_i$.

Pour résumer, nous pouvons adopter le schéma de l'ACO :

- `ConstructionSolutionParFourmis()` : cette étape correspond à la phase de création de la simulation avec le choix des valeurs des paramètres ;
- `MiseAJourPheromones()` : cette étape correspond au calcul de la simulation et à la récompense des intervalles ;
- `ActionsDémons()` : cette étape correspond à la mise à jour de l'heuristique.

Nous pouvons résumer notre méthode par la figure 4.

Chaque fourmi « représente » un agent. Chaque fourmi voyage à travers les paramètres pour choisir un intervalle par paramètre. Le choix de ces intervalles dépend de la quantité de phéromones et de l'heuristique. À la fin du parcours des fourmis, nous obtenons les différents intervalles de paramètres nécessaires pour créer l'agent.

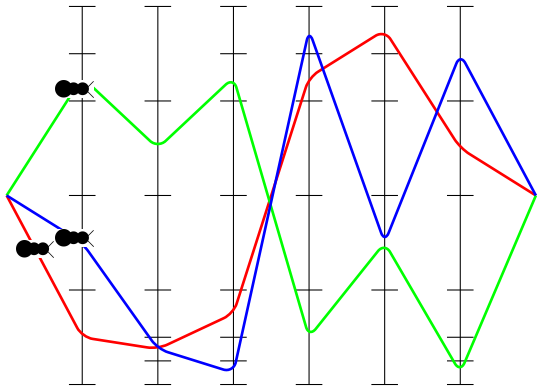

FIG. 4 – Schéma résumé de l'approche ODA inspiré par ACO

3.2 Solutions

Après que l'algorithme ait convergé, il est nécessaire de calculer une solution candidate et d'évaluer la *fitness* de cette solution. En effet, ce que l'algorithme calcule est une discrétisation adaptative de l'espace des paramètres. Mais puisque la discrétisation est achevée indépendamment sur chaque axe, cela ne mène pas nécessairement vers une solution globale unique. Nous devons donc recomposer une telle solution globale et évaluer sa *fitness*. Il y a plusieurs façons possibles de calculer cette solution finale.

Une première solution est, à chaque phase de sélection de l'algorithme, d'identifier pour chaque paramètre, les intervalles qui reçoivent le plus de récompenses. La solution globale sera composée par tous ces intervalles. La *fitness* correspondante sera obtenue par exécution d'une nouvelle simulation (en fait plusieurs exécutions à cause de la stochasticité) avec tous les agents initialisés avec ces paramètres.

Une autre possibilité pour calculer la solution globale est de considérer que l'intervalle solution pour chaque paramètre est l'intervalle de longueur la plus petite parmi tous les intervalles du paramètre. En effet, cet intervalle est celui qui a reçu le plus de récompenses dans les itérations précédentes de l'algorithme et donc qui a été le plus divisé. En fait, il y a au moins deux tels intervalles puisque durant la phase de sélection, l'intervalle qui reçoit le plus de récompenses est divisé en deux intervalles de même taille. Dans le cas d'intervalles adjacents de taille minimale, nous prenons alors l'union des deux intervalles. Ensuite, s'il reste plusieurs intervalles de même taille, l'un de ces intervalles est choisi de façon aléatoire. La *fitness* globale est calculé comme précédemment.

Après différents tests, cette dernière possibilité a été adoptée pour calculer la solution car elle est moins variable que la première. En effet, le choix des intervalles solutions ne dépend pas seulement des récompenses de la dernière itération de l'algorithme mais aussi des récompense des précédentes itérations. Dans le reste du papier, tous les résultats sont calculés avec cette dernière version.

4 Résultats

4.1 Modèle

Nous avons testé notre méthode sur différents modèles. Dans ce papier, nous allons nous focaliser sur un modèle simple pour illustrer et tester la méthode. À cette fin, nous avons décidé d'utiliser le modèle bien connu du fourragement par des fourmis, fourni par la plate-forme d'environnement de simulation `NetLogo` [17, 16]. La figure 5 montre une illustration de ce modèle. Dans ce modèle très simple, les agents sont des fourmis, et leur but est de rapporter la nourriture jusqu'à la fourmilière : celle-ci est localisée au centre de l'environnement, et les unités de nourriture sont dispersées en trois zones situées à la périphérie. Initialement les agents fourmis quittent la fourmilière et font une recherche aléatoire de nourriture. Quand un agent trouve de la nourriture, il la rapporte à la fourmilière en sécrétant des phéromones sur son passage. Quand d'autres agents fourmis perçoivent les phénomènes, ils remontent le gradient correspondant jusqu'à la source de nourriture. Ce comportement renforce la présence de phéromones et finalement produit des chemins chimiques, ainsi que des files d'individus entre la fourmilière et les sources de nourriture, ce qui correspond à ce que l'on peut observer en conditions naturelles.

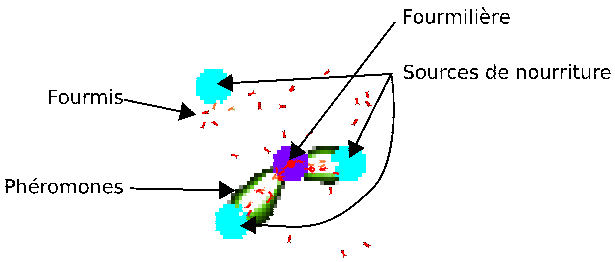

FIG. 5 – Exemple de simulation à base d'agents : le fourragement par une colonie de fourmis.

Ce modèle se caractérise par différents paramètres. Deux d'entre eux sont globaux et contrôlent la façon dont les phéromones diffusent dans l'environnement :

– `diffusion-rate` : ce paramètre caractérise la diffusion des phéromones dans l'environnement.
– `evaporation-rate` : ce paramètre caractérise l'évaporation des phéromones.

Nous avons aussi conçu différentes variantes de ce modèle pour tester nos méthodes, en ajoutant notamment de nouveaux paramètres, qui caractérisent par exemple la vitesse des agents fourmis. Les principales modifications sont l'ajout de paramètres locaux aux agents (ces paramètres sont déjà présents dans le modèle original mais sont fixés et non modifiables, et ils ne sont donc pas considérées comme des paramètres) :

– `speed` : ce paramètre caractérise la vitesse d'un agent. Il varie entre 0 et 20 unités de distance par pas de simulation.
– `patch_ahead` : ce paramètre caractérise la distance de perception des phéromones par l'agent. Il varie entre 0 et 10 unités de distance.
– `angle_vision` : ce paramètre caractérise l'angle de vision. Il varie entre 0°et 360°.
– `drop_size` : ce paramètre caractérise la quantité initiale de phéromones que l'agent dispose lors de son retour à la fourmilière suite à la découverte de nourriture. Il varie entre 0 et 200.

4.2 Résultats

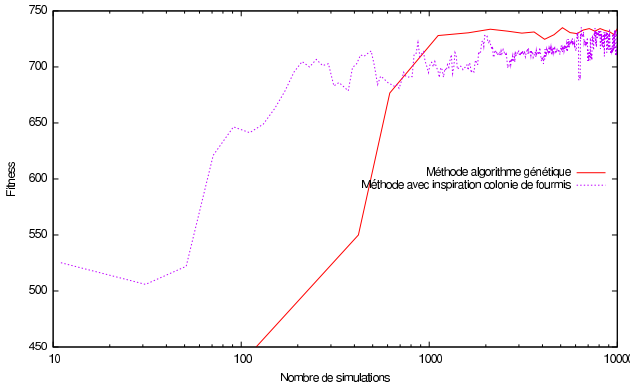

FIG. 6 – Évolution de la *fitness* en fonction du nombre d'exécutions de la simulation

Dans notre exemple, le modèle compte 25 fourmis, et nous simulons le modèle sur 500 pas. Les paramètres globaux restent constants. Nous essayons d'optimiser les quatre paramètres locaux, définis précédemment. La *fitness* est la quantité de nourriture rapportée à la fourmilière

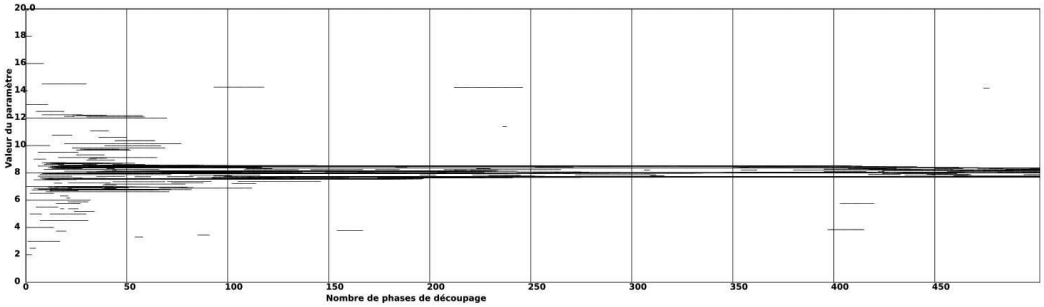

FIG. 7 – Évolution des divisions du paramètre « speed »

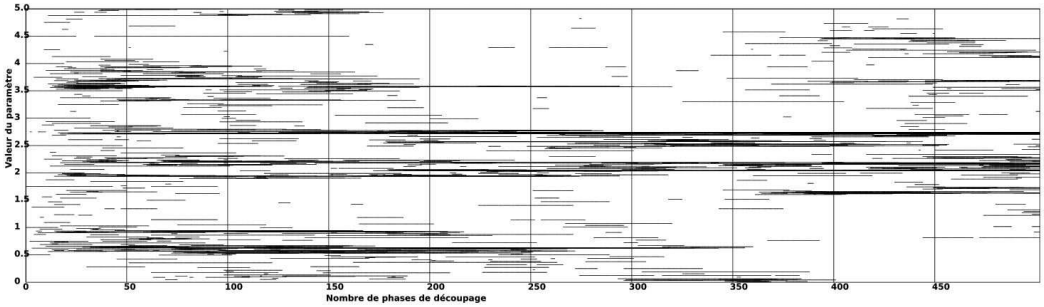

FIG. 8 – Évolution des divisions d'un paramètre artificiel

TAB. 1 – Valeurs des paramètres optimisés

	Méthode algorithme génétique	Méthode colonie de fourmis
speed	7.35	7.69
patch_ahead	9.76	9.84
angle_vision	193.76	195.18
drop_size	109.89	144.99

durant la simulation. Pour comparer notre méthode avec des travaux précédents, nous avons aussi exécuté une méthode à base d'algorithme génétique (décrit dans [4]). Nous stoppons la méthode après $10\,000$ simulations. La figure 6 montre l'évolution de la *fitness* selon le nombre d'exécutions de la simulation.

Un premier résultat intéressant est qu'avec peu d'exécutions de la simulation, la version colonie de fourmis de la méthode ODA donne de meilleurs résultats que la méthode à base d'algorithme génétique. Dans cet exemple, avec seulement 300 exécutions de simulations, nous obtenons déjà une bonne approximation de la solution.

En outre, nous obtenons plus d'informations avec la méthode ODA qu'avec la méthode à base d'algorithme génétique. La figure 7 présente l'évolution des divisions du premier paramètre par notre méthode. Au départ de l'algorithme, le paramètre est divisé en dix intervalles de même taille, divisions qui évoluent ensuite lors de l'exécution de la méthode. La division de l'intervalle converge rapidement vers une zone précise de l'espace de paramètres. Ceci illustre un des intérêts de cette méthode : nous démarrons l'exploration en considérant le paramètre en entier avant de se focaliser sur les valeurs les plus intéressantes, tout en conservant la possibilité d'évaluer les valeurs a priori moins intéressantes. Au lieu de fournir seulement une valeur optimale, la méthode fournit ainsi une espèce de cartographie de l'espace de paramètres. La figure 8 montre l'évolution des divisions d'un paramètre test (qui n'est pas utilisé par le modèle, et dont les valeurs n'ont donc aucune influence sur la *fitness*) dans notre méthode ODA. Dans ce cas, nous n'avons pas de convergence des divisions mais plutôt des divisions répandues sur la largeur du paramètre de façon régulière.

Nous pouvons aussi étudier les paramètres solutions obtenus par les deux méthodes. La table 1 montre la valeur optimale des paramètres calculée par les deux méthodes. Quelque soit la méthode, les solutions ont les même caractéristiques : les agents se déplacent avec une vitesse élevée (environ $7,5$), avec une vision panora-

mique vers l'avant (environ $190°$d'ouverture), et une grande distanc de perception. La quantité initiale de phéromones est très haute mais sans maximiser la quantité. Pour résumer, le meilleur agent est celui qui perçoit très bien et qui se déplace rapidement, ce qui apparaît conforme à l'intuition.

5 Discussion

Nous avons présenté un algorithme général et différentes variantes pour le calibrage de modèles à base d'agents, et nous avons montré, sur un exemple jouet (le fourragement par une colonie de fourmis), que la méthode obtenait de bonnes performances par rapport à des méthodes d'optimisation plus classiques. Les premiers résultats montrent ainsi que la méthode ODA a une convergence rapide vers une solution qui est presque aussi bonne que celle trouvée par l'algorithme génétique. De plus, ces méthodes sont facilement distribuables sur différentes machines afin d'accélérer le processus de calibrage. Un reproche qui pourrait être adressé à la méthode ODA est qu'elle fonctionne bien sur les paramètres propres agents, mais qu'elle a des résultats en retrait avec des paramètres globaux (variables propres à l'environnement par exemple, s'appliquant de manière uniforme sur l'ensemble des agents). Cependant, ceci ne semble pas trop problématique car les paramètres des agents sont généralement les seuls paramètres qui nécessitent d'être calibrés, alors que les paramètres globaux peuvent être estimés à partir de données réelles. Par ailleurs, on cherchera généralement à faire varier de manière indépendante le modèle comportemental des agents et les contraintes environnementales auxquelles ils sont soumis.

Comparé aux algorithmes génétiques, l'approche ODA a cependant deux grands avantages. Le premier avantage est que la convergence est très rapide avec peu de simulations, alors qu'un grand nombre de simulations est nécessaire pour démarrer l'optimisation avec l'algorithme génétique. Nous pouvons alors faire une première exploration de l'espace de paramètres avec seulement 100 ou 200 simulations, ce qui est potentiellement intéressant pour le modélisateur durant la phase de prototypage. Le deuxième avantage est qu'avec la méthode ODA, l'espace de paramètres peut, à tout moment de l'algorithme, être interprété comme une sorte de cartographie de l'espace de paramètres : de grands intervalles correspondent à des régions a priori sans intérêt ; des intervalles denses correspondent au contraire à des régions à fort intérêt. Des techniques de visualisation spécifiques restent à développer afin de tirer parti de ces caractéristiques.

6 Conclusion et perspectives

Dans ce papier, nous avons présenté un nouvel algorithme pour le calibrage automatique de modèle à base d'agents, avec différentes versions et améliorations. Nous avons aussi présenté des résultats préliminaire sur un test de performance avec quatre paramètres continus indépendants à calibrer. Ces résultats suggèrent que la méthode est intéressante plus spécialement quand l'exécution de simulations coûte cher, à cause de sa rapide convergence. L'argument est que les modèles à base d'agents présentent des caractéristiques qui rendent leur calibrage difficile, principalement à cause du temps requis pour exécuter des simulations, à comparer aux fonctions de *fitness* habituellement utilisées en algorithme génétique. Une autre caractéristique des modèles à base d'agents est que ce sont des modèles concurrents, propriété que nous avons exploitée afin de proposer un algorithme efficace, inspiré de la recherche dichotomique et de systèmes de colonies de fourmis. De plus, nous soulignons que la méthode permet d'obtenir une cartographie exhaustive de l'espace de paramètres, ce qui est un réel avantage par rapport aux algorithmes génétiques.

De nouvelles investigations sont encore nécessaires pour bien comprendre le rôle du nombre d'agents dans le processus d'optimisation. Nous étudierons aussi la signification des patterns de divisions pour les paramètres et développerons des outils de visualisation afin d'analyser ces patterns. Si un paramètre montre une zone très dense de divisions en petits intervalles, le paramètre est probablement un paramètre important dans la dynamique de la simulation. Au contraire, si les divisions sont relativement homogènes, ce paramètre peut être un paramètre non essentiel pour le modèle. Finalement, nous chercherons la possibilité de mélanger différentes approches, comme par exemple l'Optimisation Dichotomique Adaptative et les algorithmes génétiques, afin de combiner les forces et faiblesses de ces différentes méthodes.

De plus, il est nécessaire d'élargir le panel de modèles sur lesquels la méthode est évaluée, avec des espaces de paramètres plus larges. Nous somme actuellement en train d'utiliser

cette méthode pour l'étude de modèles réels, dans lesquels se posent de vraies questions de modélisation, avec des données expérimentales, des hypothèses, et des théories sous-jacentes. Dans ce cadre, l'un des enjeux est de savoir dans quelle mesure notre approche de calibrage automatique peut ou non proposer des débuts de réponses et suggérer des pistes de réflexion à des questions formulées par le modélisateur.

Références

[1] Christian Blum. Ant colony optimization : Introduction and recent trends. *Physics of Life Reviews*, 2(4) :353–373, December 2005.

[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence From Natural to Artificial Systems*. Oxford University Press, 1999.

[3] Sven A. Brueckner and H. Van Dyke Parunak. Resource-aware exploration of the emergent dynamics of simulated systems. In *AAMAS '03 : Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 781–788, New York, NY, USA, 2003. ACM Press.

[4] B. Calvez and G. Hutzler. Automatic tuning of agent-based models using genetic algorithms. In Luis Antunes and Jaime Simao Sichman, editors, *Proceedings of the 6th International Workshop on Multi-Agent Based Simulation (MABS'05)*, volume 3891 of *lnai*, pages 39–50. Luis Antunes, Jaime Simao Sichman, 2005.

[5] B. Calvez and G. Hutzler. Exploration de l'espace de paramètres d'un modèle à base d'agents. In Emmanuel Guéré, editor, *7èmes rencontres nationales des jeunes chercheurs en intelligence artificielle (RJCIA 2005)*, pages 99–112. Presse Universitaires de Grenoble (PUG), 2005.

[6] Marco Dorigo and Gianni Di Caro. *The Ant Colony Optimization Meta-Heuristic*, chapter New Ideas in Optimization, pages 11–32. McGraw-Hill, 1999.

[7] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2) :137–172, 1999.

[8] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B : Cybernetics*, 26(1) :29–41, 1996.

[9] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Positive feedback as a search strategy. *Technical Report No. 91-016, Politecnico di Milano, Italy, 1991*, 1999.

[10] Manuel Fehler, Franziska Klügl, and Frank Puppe. Techniques for analysis and calibration of multi-agent simulations. In Marie Pierre Gleizes, Andrea Omicini, and Franco Zambonelli, editors, *ESAW*, volume 3451 of *Lecture Notes in Computer Science*, pages 305–321. Springer, 2004.

[11] Manuel Fehler, Franziska Klügl, and Frank Puppe. Approaches for resolving the dilemma between model structure refinement and parameter calibration in agent-based simulations. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 120–122. ACM, 2006.

[12] Giuseppe Narzisi, Venkatesh Mysore, and Bud Bud Mishra. Multi-objective evolutionary optimization of agent-based models : An application to emergency response planning. In B. Kovalerchuk, editor, *The IASTED International Conference on Computational Intelligence (CI 2006)*, 2006.

[13] Alex Rogers and Peter von Tessin. Multi-objective calibration for agent-based models. In *5th Worksho on Agent-Based Simulation*, 2004.

[14] Brian Sallans, Alexander Pfister, Alexandros Karatzoglou, and Georg Dorffner. Simulation and validation of an integrated markets model. *J. Artificial Societies and Social Simulation*, 6(4), 2003.

[15] Krzysztof Socha and Marco Dorigo. Ant colony optimization for continuous domains. In *EURO XXI in Iceland*, 2006.

[16] Uri Wilensky. Netlogo ants model, 1998. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[17] Uri Wilensky. Netlogo, 1999. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL.