Exploration de l'espace de paramètres d'un modèle à base d'agents

Benoît Calvez, Guillaume Hutzler

Université d'Evry-Val d'Essonne/CNRS
LaMI, UMR 8042
523, Place des Terrasses 91000 Evry, France
bcalvez@lami.univ-evry.fr et hutzler@lami.univ-evry.fr

Résumé :

Quand on développe un système multi-agent (SMA) ou des modèles dans le contexte de la simulation à base d'agents, la mise au point du modèle constitue un pas important dans le processus de conception. En effet les modèles à base d'agents sont généralement caractérisés par de nombreux paramètres qui déterminent la dynamique globale du système. De plus, des changements même faibles sur certains paramètres (parfois un seul) peuvent mener quelquefois à une modification radicale de la dynamique du système dans son ensemble. Le développement d'un modèle à base d'agents et le choix de ses paramètres peuvent alors devenir longs et fastidieux en l'absence de stratégie précise, automatique, systématique pour explorer cet espace de paramètres.

C'est au développement d'une telle stratégie que nous travaillons en suggérant l'utilisation des algorithmes génétiques. L'idée est de capturer dans la fonction de fitness le but du processus de conception (efficacité pour le SMA de réaliser une fonction donnée, réalisme pour les modèles à base d'agents...) et de faire évoluer automatiquement le modèle dans cette direction. Cependant l'utilisation des algorithmes génétiques dans le contexte de la simulation à base d'agents apporte des difficultés spécifiques que nous allons développer dans cette article, en expliquant des solutions possibles et en les illustrant sur un modèle simple et bien connu : le fourragement par une colonie de fourmis.

Mots-clés : Systèmes multi-agents et systèmes distribués, Modélisation & Simulation, Algorithmes génétiques

1 Introduction

La simulation à base d'agents (SBA) s'intéresse à la modélisation et la simulation de systèmes complexes. Son but est de reproduire la dynamique d'un système réel en modélisant les entités par des agents, dont le comportement et

les interactions ont été définis. Une première validation d'un tel modèle est obtenue en comparant la dynamique résultante du modèle simulé avec celui du système réel (mesurée grâce aux données expérimentales). De manière similaire, les systèmes multi-agents (SMA) sont conçus comme pour accomplir une fonction donnée de façon collective et décentralisée. Le système peut alors être validé que si la fonction est réalisée et si elle est efficace.

Dans les deux cas, un des aspects importants dans le processus de conception est lié à la mise au point des paramètres du modèle. En effet, la nature du modèle est généralement caractérisée par de nombreux paramètres qui déterminent la dynamique globale du système. L'espace de paramètres est alors gigantesque. De plus, le comportement de ces systèmes complexes est souvent chaotique : d'un côté de petits changements fait sur un seule paramètre peuvent mener quelquefois à une modification radicale de la dynamique du système entier : d'un autre côté, quelques phénomènes émergents se produisent seulement dans des conditions très spécifiques et ne se produiront pas si ces conditions ne sont pas rencontrées. L'espace de solution peut alors être très petit. Comme conséquence, le développement et le réglage des paramètres d'un modèle à base d'agents peut devenir longs et fastidieux si nous n'avons pas de stratégie précise, automatique et systématique pour explorer l'espace de paramètres.

L'approche que nous suggérons est de considérer le problème de développement et de validation des modèles SBA ou SMA comme un problème d'optimisation. La validation peut alors être reformulée comme l'identification d'un jeu de paramètres qui optimise une fonction. La fonction d'optimisation pour la SBA pourrait être la distance entre le modèle artificiel que nous simulons et le système réel. La fonction d'optimisation pour un SMA pourrait être l'efficacité dans la réalisation d'une fonction. Considérant la grande dimension du problème, des techniques d'optimisation telle que les algorithmes génétiques peuvent être alors utilisées pour explorer l'espace de paramètres et trouver le meilleur jeu de paramètres en respectant la fonction d'optimisation.

Cependant l'utilisation des algorithmes génétiques dans ce contexte n'est pas simple. La première raison tient dans le choix de la fonction de fitness : les systèmes ou les simulations à base d'agents sont dynamiques et souvent caractérisés par des phénomènes émergents et transitoires ce qui compliquent la mesure de la fonction de fitness. Qu'est ce qui doit être mesuré et quand sont des questions qui influencent fortement les caractéristiques du modèles obtenus par l'algorithme et aussi la qualité des résultats. Si la fonction de fitness n'est pas choisie avec soin, les modèles résultants seront optimisés pour cette fonction de fitness spécifique, qui peut ne pas correspondre aux buts initiaux du concepteur de modèle. La seconde raison est qu'aucun modèle mathématique ne permet d'anticiper la dynamique d'un modèle à base d'agents sans l'exécuter. Le calcul de la fonction de fitness requiert alors l'exécution du système ou de la simulation multi-agent (et même plusieurs exécutions pour prendre en compte la stochasticité du simulateur), ce qui implique un fort coût de calcul. C'est alors nécessaire de développer des stratégies pour accélérer la convergence de l'algorithme.

En section 2, nous présentons la problématique du réglage des paramètres

d'une simulation à base d'agents. Puis, en section 3, nous présentons la structure générale des algorithmes génétiques et montrons les difficultés qu'apporte l'application de ces techniques à la simulation multi-agents. En section 4, nous proposons un guide pour l'utilisation des algorithmes génétiques avec la simulation à base d'agents et montre comment çà s'applique dans l'exemple du fourragement par une colonie de fourmis avant de conclure en section 5.

2 Réglage des paramètres

2.1 Les paramètres des modèles à base d'agents

Dans le contexte de la simulation à base d'agent, un modèle et son simulateur correspondant incluent de nombreux paramètres. Ces paramètres sont de natures différentes. Quelques paramètres sont propres au simulateur : le pas de discrétisation pour la modélisation du temps et de l'espace par exemple peut être une caractéristique fixée du simulateur. Une des conséquences est que ces paramètres peuvent généralement ne pas être modifiables par l'utilisateur. Pour cette raison, nous n'incluons pas ce type de paramètre dans notre espace de recherche. Nous incluons seulement les paramètres qui sont spécifiques au modèle. Quelques-uns parmi eux peuvent être extraits de la connaissance du domaine (soit expérimentale ou soit théorique) et peuvent alors être associés à des valeurs fixes. D'autres paramètres doivent être gardés comme variable, pour différentes raisons; d'autre part, la connaissance peut ne pas être directement compatible avec le modèle. Dans ce cas, une approche simple peut être d'essayer quelques valeurs et de simuler le modèle pour voir comment il se comporte globalement. Ce que nous proposons est d'avoir une approche pour automatiser ce long et fastidieux processus.

2.2 Objectif

Selon la motivation du travail du concepteur, les critères utilisés pour explorer l'espace de paramètre seront en plus différents. La motivation peut être de modéliser et de simuler un système réel, mais cela peut être aussi étudié les modèles discrets qui peuvent produire un phénomène émergent donné. Finalement, la motivation peut être de proposer des modèles qui réalisent une meilleure performance dans la réalisation d'une fonction spécifique.

Dans le premier cas, nous voulons vérifier si le modèle artificiel simule correctement le comportement du système réel. La validation du modèle sera alors d'avoir un comportement identique (le près que possible) à la connaissance expérimentale. Le problème de recherche peut être vu comme la recherche du jeu de paramètres qui minimise la distance entre les données réelles et celle simulées.

Avoir un comportement similaire peut aussi signifier que des phénomènes émergents connus pour se passer dans la vie réelle peuvent être observés dans la simulation. Des lignes de fourmis émergentes, par exemple se produisent seulement si le chemin de phéromones a des propriétés spécifiques, comme nous le verrons

dans la section suivante. L'émergence de ce phénomène sera alors associée à des valeurs spécifiques de paramètres, et le problème de recherche consistera dans la recherche des différents intervalles de paramètres où un phénomène est observable. Dans quelques cas, choisir des valeurs légèrement différentes peut mener à des résultats complètement différents durant la simulation, ce qui complique l'exploration manuelle de l'espace de paramètres et justifie le développement de techniques automatiques.

2.3 Exemple

Nous présenterons la mise au point des paramètres d'un modèle à base d'agents avec l'exemple du fourragement de fourmis (nous utilisons la plateforme programmable multi-agents `NetLogo` (Tisue & Wilensky, 2004) avec son modèle « Ants »). Le principe est que chaque fourmi cherche de la nourriture et la rapporte au nid en sécrétant une substance chimique appelé phéromone sur le chemin du retour. Quand d'autres fourmis ressentent les phéromones, elles suivent le chemin de phéromones jusqu'à la source de nourriture, ce qui renforce la présence de phéromones et finalement produit des chemins entre le nid et les sources de nourriture. Nous pouvons voir des lignes de fourmis émergées, qui sont similaires à ce que l'on observe dans des conditions naturelles. Dans le modèle que nous utilisons, il y a un nid au centre de la zone de simulation, et trois sources de nourritures autour du nid.

Dans ce modèle, deux paramètres conditionnent la formation de chemins de phéromones. Le premier est le taux de diffusion des phénomènes qui correspond au fait qu'une proportion donnée de phéromones sera diffusée aux zones voisines au prochain pas de simulation. Ceci est utilisé pour simuler la diffusion des phéromones dans l'atmosphère. Le second paramètre est le taux d'évaporation des phéromones, ce qui correspond au fait qu'une proportion donnée de phéromones disparaîtra de la zone au prochain pas de simulation. C'est utilisé pour simuler l'évaporation des phéromones dans l'atmosphère.

Si nous changeons le second paramètre, nous pouvons avoir différents comportements de simulation. La table 1 montre la variation du taux d'évaporation.

	Modèle 1	Modèle 2	Modèle 3
Taux de diffusion	50	50	50
Taux d'évaporation	0	15	99

TAB. 1 – Modèles avec différents taux d'évaporation.

Modèle 1 Modèle 2 Modèle 3

FIG. 1 – Des résultats de simulation pour les trois modèles du tableau 1

La figure 1 montre les résultats durant la simulation. Nous obtenons deux dynamiques globales différentes pour le système. La première dynamique (simulation 1 et 3) est une recherche aléatoire de nourriture (l'une à cause qu'il y a trop de phéromones ou l'autre à cause qu'il n'y en a pas). La seconde dynamique (simulation 2) est la recherche de nourriture avec des files de fourmis.

Nous pouvons être intéressés plus précisément dans la dynamique des files de fourmis. Le tableau 2 montre trois modèles avec de petites modifications pour les deux paramètres. Nous pouvons voir dans la figure 2 que ces petites variations peuvent mener à différentes dynamiques : les différences tiennent dans la façon que les sources de nourritures sont exploitées. Dans le modèle 4, des sources de nourritures sont exploitées à tour de rôles alors que dans le modèle 6, elles sont exploitées toutes en même temps. Comme résultat, nous observons une, deux ou trois lignes de fourmis.

	Modèle 4	Modèle 5	Modèle 6
Taux de diffusion	40	50	60
Taux d'évaporation	15	15	20

TAB. 2 – Modèles avec des paramètres légèrement différents.

Modèle 4 Modèle 5 Modèle 6

FIG. 2 – Des résultats de simulation pour les trois modèles du tableau 2.

2.4 Travaux précédents

Différentes méthodes ont déjà été proposées pour explorer automatiquement l'espace de paramètres de modèles discrets. Dans la plateforme `NetLogo` par exemple, l'outil « BehaviorSpace » (Tisue & Wilensky, 2004) permet d'explorer automatiquement et systématiquement l'espace de paramètres. Cet espace est un produit cartésien de valeurs que chaque paramètres peuvent prendre, quelques-uns parmi eux étant choisis comme fixés, d'autres étant limités à un sous-ensemble de toutes les valeurs possibles. Cependant quand nous avons beaucoup de paramètres, quelques-uns parmi eux peuvent prendre un grand nombre de valeurs (paramètres de valeurs réels par exemple), l'espace de paramètres devient énorme et l'exploration systématique devient impossible.

D'autres méthodes ont été proposées, qui explorent différentiellement l'espace entier de paramètres, en se concentrant sur les zones les plus intéressantes. C'est les cas de la méthode développée par Brueckner et Parunak (Brueckner & Parunak, 2003). Ils utilisent une infrastructure de balayage des paramètres (« parameter sweep infrastructure »), qui est similaire à l'outil « BehaviorSpace ». Cependant pour éviter une exploration systématique, ils utilisent des agents

chercheurs et introduisent la notion de fitness. Le but d'un agent chercheur est de voyager dans l'espace de paramètres à la recherche de la plus haute fitness. Partant d'une zone de l'espace donnée dans l'espace de paramètres, des agents chercheurs ont deux choix : bouger ou simuler. Chaque agent choisit en fonction de la confiance qu'il a sur l'estimation de la fitness (proportionnelle au nombre de simulation à ce point de l'espace) et de la valeur de la fitness. S'il décide de bouger, il se dirige vers les régions voisines qui ont la plus hautes fitness. Un désavantage de cette méthode est que des agents chercheurs peuvent se diriger vers un maximum local de fitness.

3 Utilisation des algorithmes génétiques

Comme le réglage des paramètres d'un modèle est un problème fortement combinatoire, nous proposons d'utiliser des algorithmes génétiques qui apportent généralement de bons résultats sur des problèmes de ce genre.

3.1 Principe des algorithmes génétiques

Des algorithmes génétiques sont une famille de modèles informatiques inspirés par l'évolution. Ils permettent de résoudre de nombreuses classes de problèmes, plus spécialement des problèmes d'optimisation. Dans ce cadre, la solution potentielle d'un problème est encodée sur une structure linéaire de données, qui est appelé un chromosome. L'algorithme travaille sur un ensemble de plusieurs chromosomes qui est appelé une population. Des opérateurs sont appliqués à cette population.

La population des chromosomes est initialisée de façon aléatoire. Chaque chromosome est alors évalué en utilisant une fonction de fitness, qui mesure comment est la qualité de cette solution potentielle en fonction du problème initiale : cela revient à donner un score pour chaque chromosome.

Une sélection est faite parmi la population de chromosome : nous obtenons une nouvelle population nommée population parent. Des opérateurs de recombinaisons et mutations sont alors appliqués à cette population : nous obtenons une nouvelle population nommée une population intermédiaire. La recombinaison consiste dans l'échange de partie de génome (le génome est les données dans le chromosome) entre les deux chromosomes. Avec cette opération, nous obtenons deux nouveaux chromosomes. C'est l'opérateur le plus fréquent dans les algorithmes génétiques. Intuitivement le rôle de cette opération est de prendre la meilleur partie des chromosomes (c'est-à-dire des solutions potentielles) pour obtenir un meilleur chromosome. La mutation consiste dans le changement d'un gène (un chromosome est constitué par de plusieurs petites parties appelé gènes). Cette opération évite de converger prématurément vers une solution locale.

Les nouveaux chromosomes de la population intermédiaire sont évalués. Une nouvelle population est finalement créée à partie de la population initiale et la population intermédiaire, avant de recommencer un nouveau cycle dans l'algorithme génétique.

3.2 Choix de la fonction de fitness

Si nous considérons l'exploration de l'espace de paramètres comme un problème d'optimisation, nous devons définir de façon très attentionnée la fonction qui doit être maximisée par l'algorithme. Cette fonction de fitness est d'une importance fondamentale : les modèles, qui seront sélectionnés, sont ceux qui réalisent la meilleur performance selon cette fonction. Dans le contexte de la simulation à base d'agents, le choix de la fonction de fitness est problématique pour plusieurs raisons : d'abord, ce n'est pas le résultat d'un calcul mais de la dynamique d'un processus qui doit être évalué ; deuxièmement, des phénomènes émergents peuvent être difficile à caractériser de façon quantitative car ils sont souvent reliés à une interprétation subjective réalisée par un observateur humain.

3.2.1 Quantitative contre qualitative

Valider un modèle à base d'agents par l'évaluation de la distance entre la simulation et le système réelle peut être fait soit de façon quantitative ou soit de façon qualitative.

Dans le cas quantitatif, les données sont mesurées dans la simulation et comparées aux données mesurées dans des conditions similaires dans le système réel. La distance entre la simulation et le système réel est alors la distance euclidienne entre les deux vecteurs de données. Si nous essayons de sélectionner des modèles qui sont optimisés pour la réalisation d'une certaine fonction, la fonction de fitness peut être alors directement mesurée par la performance du système pour cette fonction. Dans la cas de fourragement par une colonie de fourmis, cela correspondrait à la quantité de nourriture ramenée au nid après une période donnée ou le temps nécessaire pour ramener toute la nourriture au nid.

Dans le cas qualitatif, ce qui est important est qu'un phénomène émergent donné soit présent dans la simulation. La difficulté est alors de traduire ces observations dans une mesure quantitative (la fonction de fitness). Dans l'exemple de fourragement par une colonie de fourmis, nous connaissons à partir d'observations de fourmis réelles qu'elles s'organisent dynamiquement le long d'une ligne entre le nid et des sources de nourritures à cause de la création d'un chemin de phéromones entre le nid et les sources de nourritures. La fonction de fitness pourrait alors être conçue comme pour récompenser des modèles dans lesquels le chemin de phéromones est formé complètement entre le nid et les sources de nourriture. Dans quelques cas, la caractérisation de tels phénomènes émergents peut ne pas être aussi simple puisqu'elle peut être le résultat d'une interprétation subjective réalisée par l'observateur, qui peut ne pas être facilement quantifiable.

3.2.2 Un processus dynamique

Dans un problème d'optimisation classique, la fonction de fitness correspond au résultat d'un calcul. Par conséquent, la question à quel moment fait-on la mesure n'a pas de sens : la mesure est faite quand la calcul est fini. Au contraire,

les simulations à base d'agents sont des processus dynamiques qui évoluent au fil du temps et généralement ne finissent jamais.

Nous pouvons clairement voir dans les exemples donnés dans la section précédente que l'évaluation de la fonction fitness doit être généralement faite à un instant donné. Le choix de cet instant n'est pas neutre et peut influencer grandement les performances de l'algorithme génétique et le modèle résultant. Si nous essayons par exemple de sélectionner des modèles qui montrent un comportement donné qui est transitoire, l'évaluation de la fonction de fitness sur un seul pas de simulation peut ne pas être sensible à ce comportement, mais il faudra répéter la mesure à différents pas de simulation pour que la fonction de fitness puisse prendre en compte ce phénomène.

3.3 Calcul de la fonction de la fonction de fitness

3.3.1 Temps

Comme aucun modèle mathématique ne peut anticiper la dynamique d'un modèle à base d'agents sans l'exécuter, le calcul de la fonction de fitness requiert une, voire même plusieurs simulations. Ce qui signifie que le temps requis pour calculer la fonction de fitness serait significatif (ce qui n'est pas généralement le cas pour des problèmes d'optimisation classique). Le temps global de calcul pour réaliser un algorithme génétique est $(n \times N) \times T_f$ où n est le nombre de chromosomes, N est le nombre de générations et T_f est le temps pour calculer la fonction de fitness. Si T_f est de 10 minutes, n est de 20 chromosomes et N est de 100 générations, alors le temps global pour réaliser l'algorithme génétique est presque de deux semaines. Nous devons par conséquent trouver des méthodes pour réduire soit le nombre de chromosomes, soit le temps pour converger vers un optimum ou soit le temps pour calculer la fonction de fitness. Nous avons principalement étudié la dernière possibilité à travers le calcul distribué et l'approximation de la fonction de fitness(Jin, 2003).

Comme les différents modèles sont indépendants des uns des autres, l'évaluation de leur fitness est aussi indépendante. Donc chaque évaluation de leur fitness (c'est-à-dire chaque simulation à base d'agents) peut être faite sur différents ordinateurs. Nous pouvons alors avoir plusieurs ordinateurs pour simuler des modèles et utiliser un algorithme génétique parallèle de type maitre-esclave (Cantú-Paz & Goldberg, 2000), qui améliore les performances comparées à un algorithme génétique classique éxécuté sur un seul ordinateur.

L'approximation de la fitness revient à approximer le résultat de la simulation par un modèle mathématique, tel qu'un réseau de neurones ou un polynôme par exemple. Nous essayons cette approche en entraînant un réseau de neurones avec des données de test. Après la phase d'apprentissage, nous utilisons ça pour calculer la fitness, avec une approche de contrôle à base de générations, dans lequel la population entière de η générations est évaluée avec la fonction de fitness réelle toutes les λ générations (Jin *et al.*, 2002). Cependant les résultats ne sont pas aussi bons et cette approche a été temporairement abandonnée.

3.3.2 Stochasticité

Les résultats de deux simulations à bases d'agents fondées sur exactement le même modèle peuvent comporter généralement des légères différences dues à la stochasticité du modèle et du simulateur. Une simulation n'est pas assez pour évaluer la fonction de fitness : elle peut seulement être considérée comme une estimation du résultat pour la fitness.

Nous avons étudié la stochasticité du modèle « Ants » et du simulateur `NetLogo` . Nous avons choisi trois fonctions de fitness différentes (expliquées dans les trois exemples de la section 4) et nous avons évalué, en fonction du nombre de simulations, le taux d'erreur dans l'estimation de la fitness comparé à la fitness « réelle » (estimée avec 100 simulations). Le résultat est montré dans le tableau 3.

	Fitness 1	Fitness 2	Fitness 3
Nombre de Simulations	Taux d'erreur	Taux d'erreur	Taux d'erreur
1	$\simeq$34.57%	$\simeq$10.92%	$\simeq$13.28%
5	$\simeq$14.74%	$\simeq$4.85%	$\simeq$6.34%
10	$\simeq$10.3%	$\simeq$3.38%	$\simeq$4.39%
15	$\simeq$8.3%	$\simeq$2.85%	$\simeq$3.58%
20	$\simeq$7.26%	$\simeq$2.39%	$\simeq$3.15%
σ^*	0.41	0.14	0.17

TAB. 3 – Exemple de stochasticité

Nous voyons que nous pouvons obtenir 5% d'erreur sur l'estimation de la fitness en simulant le modèle plus cinq ou dix fois. Mais la stochasticité est plus ou moins importante en fonction de la fitness. La fitness 1, par exemple, est beaucoup plus sensible que la fitness 2.

Dans un tel environnement bruité, une première solution est d'augmenter la taille de la population (Beyer, 2000; Goldberg *et al.*, 1992). Multiplier le nombre de modèles simulés réduit les effets de la stochasticité. Une seconde solution est de simuler chaque modèle plusieurs fois pour améliorer l'évaluation de la fonction de fitness ; ces deux solutions augmentent grandement le nombre de simulations, d'où le temps, de l'algorithme génétique.

Une autre solution est d'utiliser la même technique qu'avec l'approximation de la fonction de fitness. Une solution pour le problème de stochasticité est alors d'estimer la fitness de chaque modèle avec une simulation, et à chaque n générations de l'algorithme génétique (n à choisir selon la stochasticité du modèle et la qualité désirée de l'estimation), pour estimer la fitness de chaque modèle avec x simulations.

Nous utilisons un algorithme génétique élitiste (Beker & Hadany, 2002) c'est-à-dire nous gardons les meilleurs chromosomes durant l'algorithme, ce qui permet d'améliorer continuellement la solution. Notre implémentation de l'algorithme génétique remplace seulement 25% de la population à chaque génération. Toutes les trois générations, nous estimons la fitness des modèles avec plus de simulations.

4 Structure générale et application

Jusqu'à maintenant, nous avons identifié plusieurs difficultés particulières aux systèmes à base d'agents. Nous proposons maintenant une structure générale ou des grandes lignes pour l'application des algorithmes génétiques dans ce contexte très spécifique et montrons avec de nombreux exemples comment cela peut être utilisés.

1. déterminer le but de l'étude ;
2. élaborer un modèle multi-agent ;
3. choisir les paramètres du modèle à faire évoluer par l'algorithme génétique ;
4. choisir la fonction de fitness : qu'est-ce que nous voulons optimiser ? ; quand et comment devons nous évaluer la fonction ? ;
5. étudier la stochasticité du modèle ; simuler le modèle plusieurs fois et étudier les résultats (calculer l'écart-type) ; déterminer la procédure pour l'exploration en conséquence ;
6. étudier le temps de calcul de la simulation ; si la simulation requiert beaucoup de temps, utiliser le calcul distribué ; si la simulation requiert trop de temps, utiliser une approximation de la fitness ;
7. choisir le nombre de chromosomes dans la population ;
8. éxécuter l'algorithme génétique.

Nous détaillerons maintenant comment ceci peut être appliqué pour l'exemple de fourragement par une colonie de fourmis. Dans les trois exemples, nous utilisons le modèle qui a été présenté dans la section précédente, avec 10 fourmis. La principale différence entre eux est liée à la fonction de fitness.

4.1 Exemple 1

1. notre but est d'optimiser le comportement de fourragement ;
2. le modèle est le modèle « ants » de `NetLogo` ;
3. les paramètres que nous faisons évoluer sont le taux de diffusion et le taux d'évaporation ;
4. la fonction de fitness est la quantité de nourriture ramenée au nid entre le centième et le deux centième pas de simulation ;
5. le résultat de l'étude de la stochasticité est montré dans le tableau 3 pour la fitness 1 ; pour évaluer la fonction de fitness, nous utilisons une simulation ; toutes les 3 générations nous utilisons 10 simulations pour évaluer la fonction de fitness de façon plus précise ;
6. l'évaluation de la fonction de fitness (c'est-à-dire une simulation) requiert environ 15 secondes ; nous utilisons seulement un seul ordinateur ; une nuit de calcul est assez pour calculer 100 générations ;
7. nous prenons 20 chromosomes ;
8. nous exécutons l'algorithme génétique pour 100 générations.

La figure 3 montre les résultats. À la fin des 100 premiers pas de simulation, les lignes de fourmis sont construites. Durant les 100 pas de simulations suivants,

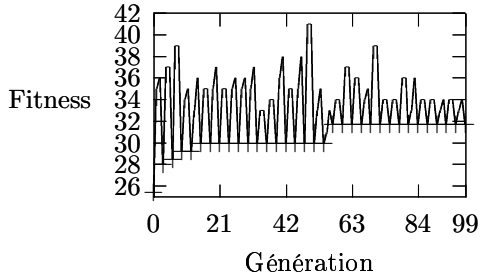

FIG. 3 – Résultat de l'algorithme génétique pour l'algorithme génétique

les fourmis exploitent les sources de nourritures en utilisant les chemins de phéromones. Les meilleurs modèles sont ceux où les trois sources de nourritures sont exploitées en même temps. Cependant le taux d'évaporation est plutôt faible. D'où le résultat est quand il n'y a plus de nourriture dans une source, le chemin de phéromones reste dans l'environnement et les fourmis prennent du temps pour exploiter une autre source de nourriture.

4.2 Exemple 2

Les différences avec l'exemple 1 sont :

4. la fonction de fitness est le temps pour ramener toute la nourriture au nid ;
5. le résultat de l'étude de la stochasticité est montré dans le tableau 3 pour la fitness 2 ;
6. l'évaluation de la fonction de fitness requiert environ 20 secondes pour un bon modèle et jusqu'à quelques minutes pour un très mauvais.

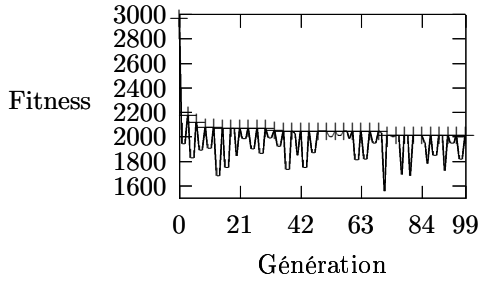

FIG. 4 – Résultat de l'algorithme génétique pour l'exemple 2

La figure 4 montre les résultats. Comme le précédent exemple, les meilleurs modèles sont ceux où les trois sources de nourriture sont exploitées en même temps. Mais le taux d'évaporation dans cet exemple est plus fort que dans l'exemple précédent. Cela améliore le comportement dynamique des fourmis : quand une source de nourriture devient épuisée, les fourmis stoppent rapidement d'aller vers cette source et cherchent une autre sources.

4.3 Exemple 3 - Fitness qualitative

Les différences avec l'exemple 3 sont :

1. Notre but est d'obtenir des modèles dans lesquels des lignes de fourmis peuvent émerger ;
4. La fitness est le nombre de lignes de fourmis ; Pour simplifier, nous déterminons le nombre de chemins de phéromones continus entre le nid et la zone de nourriture ; le nombre de chemin est évalué tous les 10 pas de simulations et la fitness est la somme de ces valeurs durant 400 pas de simulation.
5. Les résultats de l'étude de la stochasticité sont montrés dans le tableau 3 pour la fitness 3 ;
6. L'évaluation de la fonction de fitness requiert environ 30 secondes ; nous utilisons seulement un ordinateur durant une nuit.

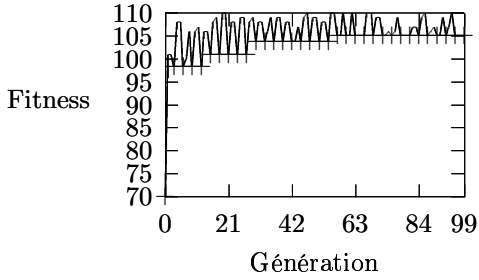

FIG. 5 – Résultat de l'algorithme génétique pour l'exemple 3

La figure 5 montre les résultats. Les meilleurs modèles exploitent encore les trois sources de nourriture en même temps. À la différence des deux précédents exemples, les taux d'évaporation et de diffusion sont très faibles. Cette faiblesse permet de concentrer les phéromones sur de petites zones de l'environnement sans couvrir tout l'environnement, mais aussi cela réduit la flexibilité du comportement de la colonie de fourmis.

4.4 Discussion

Comme nous pouvons l'avoir déjà vu avec l'étude de la stochasticité, nous obtenons des résultats très différents en fonction du choix de la fonction de fitness. Les modèles sont fortement optimisés pour une fonction de fitness spécifique et ne peuvent pas réaliser une performance d'aussi bonne qualité avec une autre fonction de fitness. L'optimisation crée une perte de flexibilité dans la dynamique des modèles à base d'agents. Une solution possible serait d'utiliser plusieurs conditions initiales différentes pour évaluer la fonction de fitness. Nous devons alors imaginer dans notre exemple de varier la position et la distance des sources de nourritures ou d'ajouter de nouvelles sources dynamiquement pour sélectionner des modèles qui montrent de hautes capacités d'adaptation. Cependant ceci augmenterait le temps nécessaire pour exécuter l'algorithme.

L'optimisation par l'algorithme génétique dépend aussi des contraintes imposées aux agents dans le modèle. Si un modèle a beaucoup de contraintes (peu de ressources par exemple), il est nécessaire d'optimiser le fonctionnement global. Au contraire, si les ressources sont abondantes, la pression sur le modèle pour s'adapter et optimiser son fonctionnement deviendra plus faible. Donc, l'utilisation de notre approche sera principalement bénéfique quand des contraintes sur le modèle sont fortes. Dans le cas du fourragement par une colonie de fourmis, si la fitness est le temps pour ramener toute la nourriture au nid, une importante ressource correspond au nombre de fourmis. Moins il y a de fourmis, meilleurs seront les organisations qu'elles auront besoin de créer pour fourrager de manière efficace. Au contraire, si les fourmis sont très nombreuses, le modèle sera bien performant quel qu'en soit le signalement par les phéromones. Dans certain cas, c'est alors peut-être utile de renforcer les contraintes sur le modèle pour obtenir de meilleures améliorations.

5 Conclusion

Ce papier présente une méthode, basée sur des algorithmes génétiques, pour explorer automatiquement l'espace de paramètres de modèles à base d'agents. Nous avons expliqué les difficultés spécifiques liées à l'utilisation de cette approche avec des modèles à base d'agents : le choix de la fonction de fitness, la stochasticité, le coût de calcul. Nous avons montré aussi quelques solutions possibles. Finalement nous avons suggéré une structure générale pour aider à utiliser les algorithmes génétiques dans ce contexte.

Aussi nous avons appliqué la méthode à quelques exemples simples : le fourragement par une colonie de fourmis avec différentes fitness (à la fois quantitative et qualitative). Nous avons obtenu des modèles qui sont optimisés selon la fonction de fitness donnée, qui est choisie en relation avec des buts spécifiques du concepteur.

La prochaine étape est d'appliquer la méthode à un exemple plus complexe. Nous commençons un travail pour la simulation de la glycolyse et la phosphotransferase chez Escherichia Coli : dans ce travail, nous nous sommes intéressés au test de l'hypothèse d'hyperstructures(Amar *et al.*, 2002) : les hyperstructures sont des complexes dynamiques de molécules qui amélioreraient le comportement d'une cellule. Nous essayons de déterminer sous quelles conditions ceci peut être vrai et comment ces hyperstructures peuvent fonctionner.

Pour explorer cet exemple complexe, nous aurons besoin de développer des stratégies additionnelles pour réduire l'espace de paramètres (par l'introduction de couplages entre des paramètres) , pour accélérer l'évaluation de la fonction de fitness (par le développement de méthodes d'approximation) et d'accélérer la convergence de l'algorithme génétique (par l'utilisation d'algorithme génétique interactif (Takagi, 2001)). Finalement, un autre important travail est d'explorer l'effet de la variation dynamique des conditions de simulation afin de produire des modèles plus polyvalents.

Références

Amar P., Bernot G. & Norris V. (2002). Modelling and Simulation of Large Assemblies of Proteins. *Proceedings of the Dieppe spring school on Modelling and simulation of biological processes in the context of genomics*, p. 36–42.

Beker T. & Hadany L. (2002). Noise and elitism in evolutionary computation. In *Soft Computing Systems - Design, Management and Applications*, p. 193–203.

Beyer H.-G. (2000). Evolutionary Algorithms in Noisy Environments : Theoretical Issues and Guidelines for Practice. *Computer methods in applied mechanics and engineering*, **186**.

Brueckner S. & Parunak H. V. D. (2003). Resource-Aware Exploration of the Emergent Dynamics of Simulated Systems. *AAMAS 2003*, p. 781–788.

Cantú-Paz E. & Goldberg D. E. (2000). Efficient parallel genetic algorithms : theory and practice. *Computer Methods in Applied Mechanics and Engineering*, **186**.

Goldberg D. E., Deb K. & Clark J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex System*, **6**.

Jin Y. (2003). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing Journal*.

Jin Y., Olhofer M. & Sendhoff B. (2002). A Framework for Evolutionary Optimization with Approximate Fitness Functions. *IEEE Transactions on Evolutionary Computation*, **6**(5), 481–494.

Takagi H. (2001). Interactive evolutionary computation : fusion of the capabilities of EC optimization and human evaluation. In *Proceedings of the IEEE*, volume 89, p. 1275–1296 : IEEE Press.

Tisue S. & Wilensky U. (2004). Netlogo : Design and Implementation of a Multi-Agent Modeling Environment. *Proceedings of Agent 2004*.