

Modélisation des fourmis *Pachycondyla apicalis* appliquée à l'optimisation numérique

G. Venturini* M. Slimane* N. Monmarché* D. Fresneau**

Rapport Interne No. 194, E3i, septembre 1997.

*Laboratoire d'informatique,
Université de Tours,
64, Avenue Jean Portalis,
37200 Tours, France.
venturini,slimane@univ-tours.fr
Tel: 02 47 36 14 14, Fax: 02 47 36 14 22

**Laboratoire d'Éthologie
Expérimentale et Comparée
Université Paris 13,
Avenue Jean-Baptiste Clément,
93430 Villetaneuse, France.
fresneau@leec.univ-paris13.fr
Tel: 01 49 40 32 61, Fax: 01 49 40 39 75

Résumé

Nous présentons dans ce rapport une modélisation du comportement de fourragement d'une population de fourmis primitives (*Pachycondyla apicalis*) et son application au problème général d'optimisation numérique. Ces fourmis sont caractérisées par une stratégie de recherche de proie relativement simple où les individus chassent en solitaires et tentent de couvrir uniformément un espace donné autour de leur nid par des recherches locales sur des sites de chasse. Le nid peut être déménagé périodiquement. Cela correspond donc en optimisation à un algorithme effectuant plusieurs recherches aléatoires en parallèle et localisées uniformément dans un sous-espace centré en un point et de taille donnée. Le déplacement du nid des fourmis correspond à un opérateur de réinitialisation dans les recherches parallèles où le point central est déplacé. Nous testons ce modèle simple, appelé API, sur un ensemble de fonctions classiques que nous cherchons à minimiser et nous montrons notamment que cet algorithme obtient des résultats comparables à un algorithme génétique classique et plus robustes qu'une ascension locale aléatoire.

Mots clés : Optimisation, Modélisation , Fourmis *Pachycondyla apicalis*.

Table des matières

1	Aperçu de la stratégie de fourragement de <i>Pachycondyla apicalis</i>	4
2	Modélisation algorithmique	6
2.1	Description de l'algorithme	6
2.2	Comparaison analytique avec d'autres méthodes connexes	7
3	Résultats	9
3.1	Principes des tests réalisés	9
3.2	Premiers résultats d'API	11
3.3	Etude comparative	11
4	Conclusion	13

Introduction

Les algorithmes modélisant des populations de fourmis sont apparus et se sont multipliés depuis peu en vie artificielle. On peut compter maintenant des applications de ces algorithmes dans différents domaines comme la robotique, la classification d'objets, l'optimisation combinatoire ou numérique.

En robotique, les colonies de fourmis servent de modèle pour construire des sociétés d'agents qui n'ont pas de connaissances globales sur leur environnement et la société à laquelle ils appartiennent (Goss et Deneubourg 1991) (Drogoul et al. 1992) (Corbara et al. 1993). Les problèmes à résoudre sont du type construction de la fourmilière, exploitation de ressources alimentaires, ou colonisation de territoires sur d'autres colonies. La fourmi ne connaît en général pas le but global à atteindre mais sait effectuer quelques tâches simples.

Les travaux de (Deneubourg et al. 1991) traitent du problème de tri d'objets : les fourmis se déplacent dans un environnement à deux dimensions et ne sont capables que de prendre ou déposer des objets. Elles n'ont pas de capacités de communication inter individus ni de connaissances de leur espace de déplacement. Pour la classification d'objets, dans (Lumer et Faieta 1994) les capacités des fourmis pour agréger des tas d'objets sont exploitées et les fourmis peuvent mesurer la similarité entre les objets.

En optimisation combinatoire, l'aspect discret de ce genre de problème se prête bien à la modélisation par des colonies de fourmis (Kuntz et Snyers 1994). Dans (Colomi et al. 1992) le problème du voyageur de commerce est abordé. Les fourmis sont alors des agents qui prennent des états discrets et dont la séquence fournit la combinaison recherchée. Les phéromones sont utilisées pour marquer les transitions entre les états et attirer ainsi d'autres fourmis.

Enfin, un algorithme à base de fourmis a été conçu pour résoudre un problème d'optimisation numérique avec contraintes (Bilchev et Parmee 1996). Les fourmis, dans ce cas, partagent un ensemble de directions jugées intéressantes pour leur recherche. Afin de prendre en compte la continuité du domaine exploré, les fourmis font des pas qui se réduisent à chaque itération de l'algorithme. Les récompenses données aux fourmis permettent de les orienter vers des solutions réalisables du problème.

Nous nous intéressons dans ce rapport à une modélisation de la stratégie de fourrage-ment d'une espèce de fourmis ponérines *Pachycondyla apicalis* (Fresneau 1985) (Goss et al. 1991) (Deneubourg et al. 1987) (Fresneau 1994), et à son application à un problème d'optimisation numérique. L'intérêt de ces fourmis pour l'optimisation vient du fait qu'elles utilisent des principes relativement simples à la fois d'un point de vue global et local pour rechercher leurs proies. A partir de leur nid, elles couvrent globalement une surface donnée en la partitionnant uniformément en sites de chasse individuels. Pour une fourmi donnée, on observe une stratégie d'exploration aléatoire des sites sensible au succès rencontré. Ces principes peuvent être repris dans un problème analogue qu'est la recherche d'un minimum global d'une fonction f de R^l dans R .

La suite de ce rapport est organisée comme suit : la section 1 décrit de manière succincte les principes utilisés dans la recherche de proies de *Pachycondyla apicalis*. La section 2 donne une modélisation possible de ces principes, appelée API, à la fois du point de vue mathématique de l'optimisation numérique et du point de vue algorithmique de l'informatique. La section 3 décrit les tests expérimentaux qui ont été réalisés sur 8 fonctions classiques ainsi que des comparaisons avec une recherche locale aléatoire et un algorithme génétique. La section 4 résume l'algorithme API, analyse ses faiblesses et ses atouts et présente les perspectives possibles.

1 Aperçu de la stratégie de fourragement de *Pachycondyla apicalis*

Les fourmis considérées ont été étudiées au Mexique dans la forêt tropicale près de la frontière du Guatemala (Fresneau 1985) (Fresneau 1994). Le nombre de fourmis dans une colonie de l'espèce considérée peut aller de quelques dizaines à une centaine de fourmis environ. On peut noter que les fourmis d'un nid ne participent pas toutes à la recherche de proies à un instant donné, et l'on peut estimer le nombre de fourmis allant chasser à quelques dizaines. Dans la suite, nous ne considérerons que ces fourmis dont l'effectif relativement faible permet une modélisation algorithmique réalisable en pratique.

On peut caractériser la stratégie globale de recherche de proies de la manière suivante. Les fourmis se répartissent des sites de chasse de manière relativement uniforme autour du nid. Ces sites sont situés à une distance maximum moyenne d'environ une dizaine de mètres autour du nid. Ils ont chacun un rayon de 2.5 m environ et sont répartis dans toutes les directions autour du nid. Ils couvrent donc par une mosaïque de petites régions une surface assez grande centrée sur le nid. On observe également périodiquement des changements d'emplacement du nid. Ces changements peuvent être expliqués soit par l'étroitesse ou le délabrement du nid actuel, soit par l'appauvrissement en proies. Le déménagement du nid est un processus assez complexe qui met en jeu des fourmis spécialisées dans la recherche d'un nouvel emplacement ainsi que le recrutement de fourmis pour faire effectivement déménager toute la colonie. Il permet cependant aux fourmis d'explorer un nouveau terrain.

D'un point de vue local, la stratégie d'une fourmi fourrageuse se traduit de la manière suivante. Une fourmi choisit initialement un site de chasse aléatoirement. Après un premier succès, elle mémorise le site de capture de sa proie. Ensuite, une fourmi a tendance à retourner au dernier site de chasse fructueux et à la piste qui y mène. Elle suit cette piste sans utiliser de traces chimiques sur le sol, mais plutôt en se servant de repères visuels. L'exploration suivant une capture d'une proie reprend généralement à partir de cet endroit. Lorsqu'un site de chasse s'appauvrit et ne reçoit plus le renforcement que représente la capture d'une proie, la fourmi a tendance à explorer d'autres directions alors qu'auparavant elle restait très attachée au site fructueux. En particulier, elle peut se déplacer à nouveau sur un ancien site de chasse, ce qui met en évidence une capacité de mémorisation de plusieurs sites. Enfin, on peut préciser que lorsqu'une proie est capturée,

la fourmi la ramène au nid en ligne droite.

Les interactions entre fourmis sont assez limitées en ce qui concerne le fourragement. Comme on l'a vu dans le paragraphe précédent, les fourmis utilisent des repères visuels plutôt que des traces de phéromones pour se localiser. Ainsi, les interactions entre fourmis ne peuvent pas avoir lieu par le tracé d'un chemin qui serait suivi par de nombreuses fourmis comme cela existe dans d'autres espèces. Cet aspect simplifie également la modélisation algorithmique des fourmis.

2 Modélisation algorithmique

2.1 Description de l'algorithme

Nous allons donc considérer une population de n fourmis fourrageuses de l'espèce décrite dans la section précédente en les modélisant du point de vue de l'optimisation numérique de la manière suivante :

- l'espace dans lequel vont se déplacer les fourmis et où se trouve le nid N est R^l , et la fonction f à minimiser va de R^l dans R .
- les fourmis vont initialement partir du nid et se construire initialement chacune p sites de chasse en mémoire de manière aléatoire avec une répartition des sites uniforme, centrée autour du nid, et à une distance maximum de celui-ci.
- chaque fourmi va rechercher localement des proies sur ses sites de chasse (voir figure 1). Initialement, lorsque l'intérêt des sites est inconnu, la fourmi sélectionne aléatoirement un de ses p sites de chasse et s'y déplace. Elle effectue un mouvement aléatoire autour de ce site et obtient une proie si cette exploration locale a permis de trouver une valeur plus faible pour f qu'au site lui-même. C'est ainsi que l'on modélise la notion de proie. Chaque fois qu'un site $s \in R^l$ donne lieu à une exploration locale fructueuse en un point $s + \delta, \delta \in R^l$, la fourmi mémorise ce succès et change les coordonnées de s en mémoire : $s \leftarrow s + \delta$. La fourmi considérée retournera précisément en s lors de sa prochaine sortie, et non vers un autre site mémorisé. Si l'exploration locale d'un site s n'est pas fructueuse, alors la fourmi choisira aléatoirement lors de sa prochaine sortie un site parmi ses p sites en mémoire. Lorsqu'un site s a été exploré successivement plus de Pat_{locale} fois sans apporter de proie, celui-ci est abandonné définitivement et remplacé en créant un nouveau site en partant du nid. Pat_{locale} représente un paramètre de "patience" locale.
- le déplacement du point central N qui sert de nid est effectué toutes les T_N itérations de l'algorithme. À chaque itération de l'algorithme principal, les n fourmis sont simulées en parallèle. Le déplacement du nid a donc lieu tous les $n \times T_N$ déplacements individuels. Ici, par déplacement, on entend sortie du nid pour aller chercher une proie ou pour créer un nouveau site de chasse. À chaque déplacement du nid, les fourmis perdent la trace qui mènent à leurs sites de chasses : les sites sont donc "réinitialisés" dans la mémoire des fourmis. Le nouvel emplacement du nid est le meilleur point trouvé par les fourmis depuis le dernier emplacement du nid.

Certains points peuvent être décrits plus précisément. Nous considérons que l'optimisation porte sur un sous-espace de R^l délimité par l intervalles de la forme $[b_i, B_i]$ ($i \in [1..l]$). Dans la suite, un déplacement aléatoire d'amplitude maximum A à partir d'un point $P = (x_1, \dots, x_l)$ vers un nouveau point $P' = (x'_1, \dots, x'_l)$ est réalisé de la manière suivante :

$$\forall i \in [1..l], x'_i = x_i + U[-0.5, +0.5] \times A \times (B_i - b_i)$$

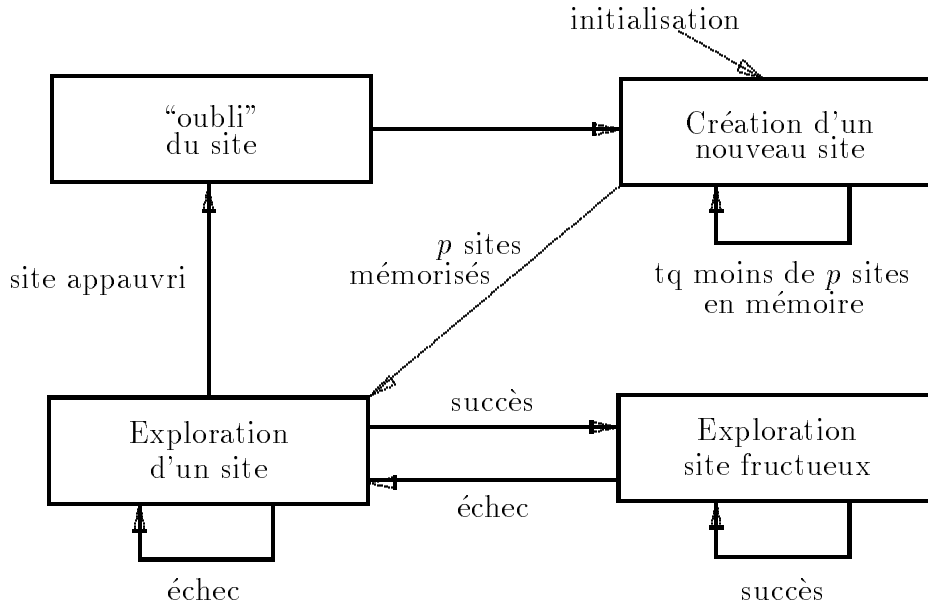


Figure 1: Automate représentant le comportement local d'une fourmi. Le retour au nid, implicite après chaque sortie (exploration ou création) n'est pas représenté ici.

où $U[-0.5, +0.5]$ désigne une loi uniforme à valeurs dans $[-0.5, +0.5]$.

La création d'un nouveau site en mémoire s'effectue en déplaçant la fourmi une fois depuis le nid avec un mouvement aléatoire d'amplitude maximum A_{site} . Lorsqu'une fourmi cherche une proie sur un site, elle effectue un déplacement aléatoire d'amplitude A_{locale} autour du site courant.

L'algorithme global qui simule les fourmis en parallèle ainsi que les déplacements du nid est donc le suivant :

1. Choix initial aléatoire d'un emplacement du nid N dans R^l ,
2. Simuler une sortie du nid pour chacune des n fourmis en utilisant l'automate de la figure 1,
3. Changer l'emplacement du nid toutes les T_N itérations,
4. Aller en 2 ou Stop.

2.2 Comparaison analytique avec d'autres méthodes connexes

Du point de vue des algorithmes évolutionnaires, on peut dire qu'API n'utilise pas du tout la notion de croisement, ni même la notion de sélection centralisée des individus. Il s'éloigne donc des algorithmes génétiques classiques. Par rapport aux stratégies d'évolution, le mécanisme de déplacement d'un point central est sans doute ce qui distingue le plus API.

Le “delta coding” est une technique d’optimisation génétique introduite par (Whitley et al. 1991) qui a un point commun important avec API. Le delta-coding utilise en effet la notion de point central à partir duquel sont effectuées les explorations selon un algorithme génétique. De manière cyclique, le delta-coding change ce point central suivant les explorations effectuées précédemment. Ces explorations sont effectués dans un voisinage du point central avec un pas donné. Cet algorithme a plutôt été utilisé dans des espaces binaires.

Les algorithmes à base de fourmis que l’on peut trouver mettent en général en avant l’utilisation des phéromones (Colorni et al. 1992) (Bilchev et Parmee 1996) qui ont certes l’avantage d’introduire un mécanisme de communication entre les agents mais cela augmente en général la taille des données à manipuler.

3 Résultats

3.1 Principes des tests réalisés

	Fonction	Intervalle de x_i
f_1	$x_1^2 + x_2^2 + x_3^2$	$[-5.12, 5.11]$
f_2	$100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$[-2.048, 2.047]$
f_3	$50 + \sum_{i=1}^5 (x_i^2 - 10 \cos(2\pi x_i))$	$[-5.12, 5.11]$
f_4	$1 + \sum_{i=1}^2 \frac{x_i^2}{4000} - \prod_{i=1}^2 \cos\left(\frac{x_i}{\sqrt{i}}\right)$	$[-512, 511]$
f_5	$1 + \sum_{i=1}^5 \frac{x_i^2}{4000} - \prod_{i=1}^5 \cos\left(\frac{x_i}{\sqrt{i}}\right)$	$[-512, 511]$
f_6	$0.5 + \frac{\sin^2(\sqrt{x_1^2 + x_2^2}) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2}$	$[-100, 100]$
f_7	$(x_1^2 + x_2^2)^{0.25} (1 + \sin^2 50(x_1^2 + x_2^2)^{0.1})$	$[-100, 100]$
f_8	$ x_1 + 2x_2 + 3x_3 + 4x_4 $	$[-100, 100]$

Table 1: Les 8 fonctions utilisées comme problème de tests. Les fonctions f_1 à f_7 ont été utilisées notamment pour l'évaluation des algorithmes génétiques dans différents travaux référencés dans (Whitley et al. 1995).

Les tests ont été réalisés sur un ensemble de 8 fonctions classiques (Whitley et al. 1995) représentées dans la table 1. Si l'on note $API(p,n)$ un algorithme API à n fourmis et p sites de chasse par fourmi, nous présentons ici les tests de $API(1,1)$, $API(1,10)$, $API(2,10)$ et $API(1,20)$. Pour chacun de ces algorithmes, nous avons fait varier les autres paramètres de la manière suivante : $A_{locale} = 1\%, 2.5\%, 5\%, 10\%, 20\%, 30\%, 40\%$, $Pat_{locale} = 5, 10, 15, 20, 25$, $A_{site} = 25\%, 50\%$ $T_N = 50, 100, 150$. Cela représente donc en tout $7 \times 5 \times 2 \times 3 = 210$ jeux de paramètres différents.

En plus d'API, nous utilisons à titre de comparaison d'autres algorithmes comme une ascension locale aléatoire ("Random Hill Climbing", RHC dans la suite) et un algorithme génétique (AG dans la suite). RHC génère initialement un point p_1 aléatoirement dans R^l . Ensuite RHC explore aléatoirement un point p_2 dans un voisinage de p_1 fixé par une amplitude maximum A_{RHC} . p_2 remplace p_1 si $f(p_2) < f(p_1)$ et ainsi de suite. Les valeurs de A_{RHC} sont $1\%, 2.5\%, 5\%, 10\%, 20\%, 30\%, 40\%, 50\%$, ceci afin de ne pas défavoriser RHC devant API en lui donnant a priori des amplitudes de déplacements qui ne seraient pas similaires.

AG utilise une représentation réelle des points de R^l et une sélection par tournoi binaire avec remplacement des parents. Les opérateurs génétiques utilisés sont le croisement 1X sur des vecteurs réels (probabilité p_{cross}) et la mutation qui ajoute un bruit uniforme d'amplitude A_{AG} avec une probabilité p_{mut} à chaque gène réel. La taille de la population est notée $|pop|$ et est fixée à 100 dans la suite. De même, le nombre de générations est fixé à 100. Les paramètres utilisés dans les tests pour AG sont les suivants :

$A_{AG} = 1\%, 2.5\%, 5\%, 10\%, 20\%$, $p_{cross} = 0, 0.5, 0.8$ et $p_{mut} = 0, 0.01, 0.02, 0.05, 0.1, 0.2$. Cela représente donc 75 jeux de paramètres différents.

Ensuite, chaque algorithme peut utiliser exactement 10000 évaluations de la fonction à minimiser. Pour chaque algorithme et pour chaque jeu de paramètres, on mesure le minimum atteint et combien d'évaluations de la fonction ont été nécessaires pour l'atteindre. Tous les résultats sont donnés en moyenne sur 10 essais. Cela permet d'obtenir les meilleures performances quel que soit le jeu de paramètres de l'algorithme considéré et pour chacune des fonctions. Bien sur, d'une fonction à l'autre ce jeu de paramètres est différent pour un même algorithme. Les résultats indiqueront donc des performances absolues, mais généralement impossible à obtenir en pratique avec un unique jeu de paramètres, puisque les algorithmes considérés ne peuvent modifier automatiquement leurs paramètres de recherche.

Ensuite, on peut déterminer le jeu de paramètres moyen de l'algorithme en effectuant une moyenne des 8 jeux de paramètres j_1, \dots, j_8 ayant donné respectivement les meilleurs résultats pour chacune des fonctions f_1, \dots, f_8 . Cela permet de comparer les algorithmes sur une base méthodologique commune en évitant les problèmes de mauvais réglages des paramètres par la méthode essai/erreur.

3.2 Premiers résultats d'API

Algo.	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
API(1,1)	0.00002 6074	0.00002 6086	2.01727 4921	0.01378 3827	0.28552 4483	0.00701 3513	0.31662 4570	0.00066 4022
API(1,10)	0.00002 6400	0.00002 5731	0.89551 6974	0.01357 5887	0.30366 5464	0.00640 6703	0.33984 5619	0.00084 5173
API(2,10)	0.00007 6716	0.00004 4591	1.11423 7937	0.01739 2875	0.35533 6625	0.00642 6100	0.34770 5965	0.00080 4699
API(1,20)	0.00002 7493	0.00004 6176	1.17703 8363	0.01652 6277	0.33693 5002	0.00700 7458	0.34936 6355	0.00080 5435

(a)

Algo.	A_{locale}	Pat_{locale}	A_{site}	T_N
API(1,1)	4.4%	23	25%	119
API(1,10)	3.6%	22	25%	100
API(2,10)	4.4%	17	25%	131
API(1,20)	2.7%	20	28%	106

(b)

Table 2: En (a) sont représentés les meilleurs résultats obtenus sur les 8 fonctions tests par différents algorithmes API parmi 210 jeux de paramètres et en moyenne pour 10 essais. En (b) figure la moyenne des paramètres pour ces meilleures performances.

La table 2 donne les meilleurs résultats obtenus pour les algorithmes API ainsi que le jeu de paramètres moyen pour chaque algorithme. On peut remarquer que ces résultats sont relativement uniformes. Il n'y a pas d'algorithme qui se dégage distinctement des autres. En particulier, il semble que l'utilisation de plusieurs sites de chasse par fourmis (API(2,10)) ne donne ni d'avantages ni de désavantages par rapport à un algorithme à un seul site de chasses (API(1,10)). Il faudrait peut-être pour cela donner une stratégie de sélection du site à explorer parmi les p sites en mémoire qui soit plus élitiste et qui tienne plus compte du renforcement moyen associé à chaque site, un peu à la manière d'un algorithme génétique lors de la sélection des individus.

De même, le nombre de fourmis ne semble pas affecter dans l'ensemble les performances absolues de l'algorithme. Ce point n'est pas surprenant puisque la coopération entre les fourmis pour la recherche de proie est inexistante dans la version actuelle d'API. Si nous avons utilisé une version parallèle d'API, un nombre important de fourmis aurait certainement été avantageux.

3.3 Etude comparative

Les performances absolues obtenues par RHC et AG sont représentées dans les tables 3 et 4. On peut constater, par rapport à la table 2(a), que AG semble se démarquer en

Fonction	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
Min.	0.00001	0.00000	6.63545	0.00210	0.26513	0.00867	0.46802	0.00072
Temps	4867	5807	6133	6312	6234	5661	5245	5995
A_{RHC}	1%	1%	20%	1%	2.5%	10%	20%	1%

Table 3: Meilleures performances obtenues sur les 8 fonctions tests par une méthode d’ascension locale aléatoire. On a indiqué les valeurs du paramètre A_{RHC} pour chacune des ces performances. La moyenne de ce paramètre vaut 7.1%.

Fonction	minimum	temps	A_{AG}	p_{cross}	p_{mut}
f_1	0.00001	8742	1%	0.8	0.10
f_2	0.00050	7264	10%	0.0	0.20
f_3	0.05041	8811	20%	0.5	0.05
f_4	0.00366	8046	2.5%	0.0	0.20
f_5	0.04517	9207	5%	0.8	0.05
f_6	0.00727	5381	10%	0.5	0.20
f_7	0.08235	7520	1%	0.8	0.20
f_8	0.00027	5344	1%	0.8	0.20
moy			6.3%	0.525	0.15

Table 4: Résultats obtenus par l’algorithme génétique avec le meilleur jeu de paramètre associé ainsi que la moyenne des ces meilleurs jeux.

apportant les meilleures performances pour les fonctions f_3 , f_5 , f_7 et f_8 . API est le meilleur pour f_6 , et RHC se distingue avec f_1 , f_2 et f_4 . Cela signifie que si l’on savait adapter de manière optimale les paramètres d’AG, il serait un peu plus performant que les autres méthodes. Néanmoins, cela n’est pas possible en pratique.

La table 5 donne donc des résultats plus comparables en utilisant le jeu de paramètres moyen. On constate alors que par exemple RHC est une méthode relativement peu robuste car elle obtient des performances désastreuses sur f_3 et f_7 , qui sont des fonctions très multimodales. RHC ne dépasse légèrement les autres algorithmes qu’uniquement sur la fonction f_2 . AG obtient également des performances très médiocres sur f_2 mais dépasse les autres sur les fonctions f_1 , f_4 , f_5 et f_7 . API(2,10) obtient les meilleures performances sur les fonctions f_3 , f_6 et f_8 . Il dépasse AG sur 4 fonctions et obtient des temps de découverte du minimum souvent plus faibles qu’AG. Il est notamment beaucoup plus robuste que RHC.

Algo.	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
RHC 7.1 %	0.00050 5274	0.00002 4067	45.95351 2277	0.01185 5482	0.33535 5081	0.00637 5724	5.08074 5199	0.00385 3738
AG	0.00001 8289	0.015110 6011	2.70585 7574	0.00770 7216	0.21655 5723	0.008747 6931	0.18526 6769	0.00509 5291
API(1,1)	0.00019 4651	0.00008 6081	5.05514 4997	0.02482 4520	0.33255 4559	0.00847 5651	0.42448 5008	0.00291 4648
API(1,10)	0.00015 4516	0.00008 4903	2.75149 6652	0.02809 5710	0.32349 6106	0.00885 4922	0.37618 5685	0.00260 4216
API(2,10)	0.00015 5206	0.00011 5100	2.64872 6896	0.02626 5518	0.45277 5791	0.00865 7260	0.40149 5666	0.00201 4541
API(1,20)	0.00011 5609	0.00010 5902	3.12689 8681	0.05749 5849	0.41893 5863	0.00968 6912	0.34565 7302	0.00273 3565

Table 5: Performances globales obtenues sur les 8 fonctions et en moyenne sur 10 essais avec un jeu de paramètres fixés pour chaque algorithme d’après les expériences précédentes.

4 Conclusion

Nous avons présenté dans ce rapport un nouvel algorithme d’optimisation s’inspirant d’une population réelle de fourmis. Ces fourmis ont une stratégie efficace pour rechercher de la nourriture dans leur environnement. L’algorithme API qui en résulte simule de la manière la plus simple et la plus fidèle possible une population de fourmis artificielles. Ces fourmis partent d’un point central, leur nid, et se construisent un ensemble de sites de chasse qui couvrent initialement uniformément une zone délimitée centrée sur le nid. Les principes locaux de la recherche de proie des fourmis artificielles sont par exemple une recherche locale sur les sites utilisant un pas local d’amplitude faible, une descente vers les minima locaux et une élimination des sites non fructueux qui donne lieu à la création de sites avec un pas plus élevé en partant du nid. D’un point de vue plus global, le point central est régulièrement déplacé ce qui réinitialise la recherche.

Les résultats présentés ont montré que API était plutôt robuste par rapport notamment à une recherche aléatoire. En ce qui concerne la comparaison avec un algorithme génétique, les résultats obtenus montrent que les deux méthodes sont compétitives chacune sur 4 fonctions parmi les 8. Ces tests permettent donc d’envisager une application sur des problèmes de plus grande échelle. En particulier, des tests doivent être menés sur des espaces de dimension plus importante, et nous envisageons notamment dans ce cadre d’appliquer API à l’apprentissage de chaînes de Markov cachées. Les principes même d’API permettent également de traiter des problèmes d’optimisation plus complexes, comme l’optimisation de fonctions non-stationnaires. En effet, API n’utilise pas d’informations mémorisées de manière permanente ce qui serait pénalisant pour des fonctions dépendant du temps.

Références

- Bilchev G. and Parmee I. (1996) Constrained Optimisation with an Ant Colony Search Model. Proceedings of ACEDC'96. PEDC, University of Plymouth, UK. pp. 145-151.
- Colorni A., Dorigo M. and Maniezzo V. (1992) An investigation of some properties of an "Ant algorithm". Parallel Problem Solving from Nature, 2 R. Männer and B. Manderick (Editors) Elsevier Science Publishers B.V. 1992. pp. 509-520.
- Corbara B., Drogoul A., Fresneau D., Lalande S. (1993), Simulating the sociogenesis process in ant colonies with MANTA. In : Self-organization and life. From the simple rules to global complexity. MIT press, pp 12-24.
- Deneubourg, J.L., Goss, S., Pasteels, J.M., Fresneau, D., Lachaud, J.P., (1987), Self-organization mechanisms in ant societies (II) : learning in foraging and division of labor. In : From Individual Characteristics to Collective Organisation : the example of Social Insects. J.L. Deneubourg and J.M. Pasteels eds., Experientia suppl., 54, 197-217.
- Deneubourg J.-L., S. Goss, N. Franks, A. Sendova-Franks, C. Detrain and L. Chretien (1991). "The dynamic of collective sorting robot-like ants and ant-like robots", in Proc. of the 1st Conf. on Sim. of Adaptive Behavior, J.-A. Meyer and S. Wilson (Eds), MIT-Press.
- Drogoul A., Ferber J., Corbara B., Fresneau D. (1992), A Behavioural simulation model for the study of emergent social structures. In : Toward a practice of autonomous systems, F.J. Varela et P. Bourguine (eds), pp 161-170.
- Fresneau D. (1985), Individual foraging and path fidelity in a ponerine ant, *Insectes Sociaux, Paris*, 1985, Volume 32, n 2, pp 109-116.
- Fresneau D. (1994), Biologie et comportement social d'une fourmi ponérine néotropicale (*Pachycondyla apicalis*, Thèse d'Etat, Université de Paris XIII, Laboratoire d'Ethologie Expérimentale et Comparée, 1994.
- Goss and Deneubourg J.-L. (1991), Harvesting by a group of robots, Proceedings of the First ECAL Conference, 1991. F. Varela and P. Bourguine (Eds), pp. 195-204, MIT-Press.
- Goss S., Fresneau D., Deneubourg J.L., Lachaud J.P. and Valenzuela-Gonzalez, J., (1989). - Individual foraging in the ant *Pachycondyla apicali*. *Oecologia*, 80, pp 65-69.
- Kuntz P. and Snyers D. Emergent Colonization and Graph Partitioning. Proceedings of the Third International Conference on Simulation of Adaptive Behavior, 1994. pp 494-500, MIT-Press.
- Lumer E.D. and Faieta B.(1994) Diversity and Adaptation in Populations of Clustering Ants. Proceedings of the Third International Conference on Simulation of Adaptive Behavior, 1994. pp. 501-508.

- Whitley D., Mathias K. and Fitzhorn P. (1991), Delta coding: an iterative search strategy for genetic algorithms, Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, R.K. Belew and L.B. Booker (Eds), Morgan Kaufmann, pp 77-84.
- Whitley D., Mathias K., Rana S. and Dzubera J. (1995), Building better test functions, Proceedings of the Sixth International Conference on Genetic Algorithms, Eshelman L.J. (ed), Morgan Kaufmann publishers, pp. 239-246.