THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PARIS 12-VAL DE MARNE
UFR de Sciences et Technologie

par

Walid TFAILI

pour obtenir le grade de

DOCTEUR EN SCIENCES

Spécialité : **SCIENCES DE L'INGÉNIEUR**

Option : Optimisation

Équipe d'accueil : Laboratoire Images, Signaux et Systèmes Intelligents (LiSSi, E.A. 3956)

Ecole Doctorale : École Doctorale Sciences et Ingénierie : Matériaux - Modélisation - Environnement (SIMME)

Sujet de la thèse:

Conception d'un algorithme de colonie de fourmis pour l'optimisation continue dynamique

soutenue le 13 décembre 2007 devant la commission d'examen composée de :

M. Mhand HIFI	Université de Picardie	**Président**
M. Marc SEVAUX	Université de Bretagne Sud	**Rapporteur**
M. Mohamed SLIMANE	Université de Tours	**Rapporteur**
M. Yskandar HAMAM	ESIEE (Noisy-le-Grand)	**Examinateur**
M. Ioan Cristian TRELEA	INA-Grignon	**Examinateur**
M. Patrick SIARRY	Université de Paris 12	**Directeur de thèse**

Table des matières

Remerciements

Un nouveau chapitre de ma vie se ferme et un nouveau s'ouvre avec la rédaction de ce manuscrit. Un chapitre qui, par moments, fut difficile, avec des hauts et des bas. Ce travail n'aurait pu aboutir sans l'aide de plusieurs personnes, que je ne remercierai jamais assez. Mes remerciements vont en tout premier lieu à mon directeur de thèse :

M. Patrick SIARRY, Professeur à l'Université de Paris 12 Val de Marne, à qui j'exprime ma profonde gratitude pour son encadrement sans faille, sa disponibilité et sa rigueur lors de la direction de ce travail de thèse.

Mes remerciements vont aussi à M. Johann DRÉO, chercheur chez Thalès et ex-doctorant au LiSSi, pour son aide, ses conseils et ses encouragements. Je remercie aussi tous mes collègues au LiSSi et à l'Université de Paris 12 Val de Marne.

Je remercie très cordialement mes deux rapporteurs, M. Marc SEVAUX, Professeur à l'Université de Bretagne Sud et M. Mohamed SLIMANE, Professeur à l'Université Polytechnique de Tours, qui ont marqué leur intérêt pour mon travail en acceptant de l'examiner en détail, malgré leur nombreuses occupations.

Je remercie également M. Yskandar HAMAM, Professeur à l'ESIEE (Noisy-le-Grand), M. Mhand HIFI, Professeur à l'Université de Picardie et M. Ioan Cristian TRELEA, Maître de Conférences à l'INA-Grignon, pour le grand honneur qu'ils me font en participant à mon jury.

Je remercie mes parents Mounir et Naziha, mes soeurs Hala, Sana et Hana, mon frère Ali, qui m'ont soutenu tout au long de mes études et dont je serai indéfiniment redevable.

Je tiens aussi à remercier tous ceux qui ont, de près ou de loin, aidé à rendre ce travail possible, que ce soit par des idées ou par des encouragements.

A vous tous, je dis merci.

Résumé

Les métaheuristiques constituent une famille d'algorithmes inspirés de la nature. Ces algorithmes sont particulièrement utiles pour résoudre des problèmes où les algorithmes d'optimisation classiques sont incapables de produire des résultats satisfaisants. Parmi ces méthodes, les algorithmes de colonies de fourmis constituent une classe de métaheuristiques proposée pour l'optimisation difficile. Ces algorithmes s'inspirent des comportements des fourmis observés dans la nature ; notamment l'utilisation de la communication indirecte (à travers l'environnement), les traces de phéromone que les fourmis laissent sur leur passage, et la construction itérative d'une solution globale qui se base sur une sorte d'intelligence collective. Les algorithmes de colonies de fourmis représentent une grande flexibilité aux changements de l'environnement, ce qui les rend aptes à l'optimisation des fonctions changeantes et plus particulièrement à l'optimisation continue dynamique. Dans cette thèse, nous proposons une plate-forme de test de métaheuristiques, *oMetah*, basée sur une architecture générique de "patron" de conception, pour faciliter et rendre plus cohérents les tests et les comparaisons entre différents algorithmes d'optimisation. La performance de l'algorithme DHCIAC, un algorithme qui utilise deux canaux de communication entre fourmis et un simplexe pour la recherche locale, a été testée sur des fonctions de test continues dynamiques. Et un nouvel algorithme de colonies de fourmis, CANDO, basé sur des agents avec des charges électrostatiques, a été conçu pour permettre un équilibre entre l'exploration et l'exploitation et une adaptation continue dans le temps, nécessaire dans le cas de l'optimisation dynamique.

Mots-clés : Optimisation difficile, optimisation continue dynamique, métaheuristiques, algorithmes de colonies de fourmis, auto-organisation, plate-forme de test.

Abstract

Metaheuristics are a family of optimization algorithms inspired by nature. These algorithms are particularly suitable to solve problems when traditional optimization algorithms are unable to produce satisfactory results. Among these methods, the ant colony algorithms represent a class of metaheuristics for hard optimization problems. These algorithms were inspired by the behavior of ants observed in nature ; more particularly, the use of indirect communication (through environment), the pheromone tracks that ants leave on their passage, and the iterative construction of a global solution, which is based on a kind of collective intelligence. The ant

colony algorithms show the advantage of flexibility when environment changes, what makes them suitable to the optimization of changing functions and more specifically to dynamic continuous optimization. In this thesis we propose a metaheuristic test platform *oMetah*, based on a generic design pattern architecture, to facilitate and make more coherent the tests and the comparisons between various optimization algorithms. The performance of DHCIAC algorithm, an algorithm which makes use of two communication channels between ants and a simplex method for local search, was tested on dynamic continuous test functions. And a new ant colony algorithm, CANDO, based on agents with electrostatic charges, was designed to allow a balance between exploration and exploitation and a continuous adaptation over time, necessary in the case of dynamic optimization.

Key-words : Hard optimization, continuous dynamic optimization, metaheuristics, ant colony algorithms, auto-organization, test platform.

Introduction générale

Les problèmes d'optimisation occupent actuellement une place importante dans la communauté scientifique. Les problèmes peuvent être combinatoires (discrets) ou à variables continues, avec un seul ou plusieurs objectifs (optimisation multiobjectif), statiques ou dynamiques. Cette liste n'est pas exhaustive et un problème peut être à la fois continu et dynamique, etc.
Un problème d'optimisation est défini par un ensemble de variables, une fonction objectif (fonction de coût) et un ensemble de contraintes. L'espace d'état, appelé aussi domaine de recherche, est l'ensemble des domaines de définition des différentes variables du problème. Il est en général fini, puisque les méthodes opèrent dans des espaces bornés. Et, pour des raisons pratiques et de temps de calcul, l'espace de recherche doit être fini. Cette dernière limitation ne pose pas de problème, puisqu'en général le décideur précise exactement le domaine de définition de chaque variable. Enfin, même dans le cas des problèmes à variables continues, une certaine granularité est définie. La fonction objectif définit le but à atteindre, on cherche à minimiser ou à maximiser celle-ci. L'ensemble des contraintes est en général un ensemble d'égalités ou d'inégalités que les variables de l'espace d'état doivent satisfaire. Ces contraintes limitent l'espace de recherche. Les méthodes d'optimisation recherchent un point ou un ensemble de points dans l'espace de recherche qui satisfont l'ensemble des contraintes, et qui maximisent ou minimisent la fonction objectif.

Cette thèse a été préparée au sein de l'université de Paris 12, dans le *Laboratoire Images, Signaux et Systèmes Intelligents* (LiSSi, E.A. 3956). Ce laboratoire est orienté principalement vers l'imagerie et le traitement du signal en génie biologique et médical. Ce travail a été financé en troisième année par un demi poste d'attaché temporaire d'enseignement et de recherche (ATER), partagé entre le LiSSi et l'IUT de Créteil-Vitry. Cette thèse a été encadrée par P. Siarry, directeur du LiSSi, professeur et responsable de l'équipe "optimisation", activité transversale au LiSSi. Cette équipe est notamment spécialisée dans les métaheuristiques et dans

leurs applications en génie biologique et médical. Les travaux de recherche dans l'équipe ont ainsi concerné l'adaptation au cas continu des algorithmes de colonies de fourmis, du recuit simulé, des algorithmes évolutionnaires, de la recherche tabou, l'optimisation multiobjectif.

L'optimisation dynamique s'efforce de minimiser ou maximiser un indice de performance qui varie en fonction du temps. En pratique, l'optimisation dynamique peut être appliquée par exemple pour déterminer de bonnes manoeuvres dans le domaine aéronautique, le contrôle de robots, etc.

Dans le cadre de l'équipe optimisation, une thèse soutenue en décembre 2004 s'est attachée à l'élaboration d'un nouvel algorithme de colonie de fourmis, pour l'optimisation en variables continues (thèse de Johann Dréo). L'algorithme conçu a été appliqué avec succès au recalage des images d'angiographie rétinienne. Une autre thèse, qui se termine bientôt, vise plus particulièrement à améliorer la segmentation des images au moyen des métaheuristiques (thèse de Amir Nakib). Enfin, une thèse en cours porte sur le développement d'un algorithme d'optimisation par "essaim particulaire", qui s'inspire de la dynamique de populations se déplaçant en foules compactes (vols groupés d'oiseaux, bancs de poissons) (thèse de Yann Cooren).

Les principaux apports de ce travail de thèse sont :

- la participation à la conception d'une plate-forme de test de métaheuristiques,
- l'amélioration et le test de l'algorithme DHCIAC,
- la conception d'une nouvelle méthode de colonies de fourmis, CANDO, adaptée aux environnements continus dynamiques, qui utilise une diversification continue dans le temps et, plus précisément, exploite des charges électrostatiques.

Le plan du présent manuscrit est le suivant :

Dans le **premier chapitre**, nous allons définir le problème "d'optimisation difficile", puis nous allons exposer les deux grandes catégories d'algorithmes d'optimisation : les méthodes déterministes, comme les méthodes de gradient et le simplexe de Nelder & Mead ; puis les métaheuristiques, comme les algorithmes de colonie de fourmis, les essaims particulaires, la recherche tabou, le recuit simulé, les algorithmes génétiques et les algorithmes à estimation de distribution. Et nous décrivons l'inspiration biologique, et plus précisément l'auto-organisation et l'intelligence collective observées dans la nature, et quelques exemples qui donnent des idées pour la conception algorithmique.

Dans le **deuxième chapitre**, nous présentons l'état de l'art des méthodes d'optimisation dynamiques, puis nous nous intéressons plus particulièrement à l'optimisation dynamique continue.

Dans le **troisième chapitre**, nous montrons que les métaheuristiques à population peuvent être vues comme des algorithmes manipulant un échantillonnage probabiliste d'une distribution de probabilité, représentant la fonction objectif d'un problème d'optimisation. Et nous présentons une nouvelle plate-forme de test, *oMetah*, ainsi que l'approche ALS, qui peut être employée pour mettre en application les métaheuristiques, tout en facilitant leur conception et leur analyse.

Dans le **quatrième chapitre**, nous présentons les résultats de test de l'algorithme DHCIAC, un algorithme de colonies de fourmis basé sur la communication directe entre fourmis, et qui utilise un simplexe pour la recherche locale.

Dans le **cinquième chapitre**, notre nouvelle méthode est présentée et testée sur une batterie de fonctions de test dynamiques.

Nous concluons, et présentons les perspectives de ce travail dans le **dernier chapitre**.

CHAPITRE 1

ETAT DE L'ART SUR DES MÉTHODES D'OPTIMISATION DÉTERMINISTES ET SUR QUELQUES MÉTAHEURISTIQUES LES PLUS CONNUES

1.1 Problème "d'optimisation difficile"

Les problèmes d'optimisation difficile représentent une classe de problèmes pour lesquels les méthodes directes ou déterministes n'arrivent pas à trouver une solution satisfaisante. Nous pouvons citer, à titre d'exemple, parmi ces problèmes, le problème du voyageur de commerce.

En théorie de la complexité, un problème est formalisé de la manière suivante : un ensemble de données en entrée, et une question sur ces données (pouvant demander éventuellement un calcul). La théorie de la complexité ne traite que des problèmes de décision binaire, c'est-à-dire posant une question dont la réponse est oui ou non. Cependant on étend la notion de complexité aux problèmes d'optimisation. En effet, il est facile de transformer un problème d'optimisation en problème de décision. Si, par exemple, on cherche à optimiser une valeur n, on traite le problème de décision qui consiste à comparer n à un certain k. En traitant plusieurs valeurs de k, on peut déterminer une valeur optimale. On confondra souvent un problème d'optimisation et son problème de décision associé.

On considère que l'ensemble des instances d'un problème est l'ensemble des données que peut accepter ce problème en entrée, par exemple l'ensemble des permutations de n entiers à trier, pour un algorithme de tri.

1.2 Algorithmes d'optimisation

Comme son nom l'indique, l'optimisation cherche à trouver une solution optimale à un problème donné. En optimisation de forme, par exemple, nous pouvons chercher la forme optimale d'une structure métallique qui subit des forces externes, la forme optimale d'une chaise, la distribution de différents containers au port pour avoir le plus de place possible, le chemin optimal (le plus court) pour visiter N villes sans passer plusieurs fois par la même (le problème du "voyageur de commerce").

1.2.1 Méthodes déterministes

Dans la littérature, nous trouvons de nombreuses méthodes d'optimisation déterministes. Nous pouvons les diviser en deux grandes catégories : les méthodes locales, qui cherchent à converger vers l'optimum le plus proche de la solution courante, en utilisant des méthodes d'exploration dans son voisinage. La méthode de recherche locale la plus célèbre est la descente de gradient. Et les méthodes globales, qui cherchent à faire converger la solution vers l'optimum global de la fonction. En général, ces méthodes procèdent de la façon suivante : avant de commencer le processus de recherche, il faut réaliser un échantillonnage de l'espace de recherche. L'efficacité de ces méthodes dépend de l'adaptation entre la taille de l'échantillonnage et la forme de la fonction objectif.

Méthodes de gradient

La méthode de descente de gradient est la méthode la plus ancienne parmi les méthodes d'optimisation. Cette méthode suppose que nous avons accès aux dérivées partielles de la fonction objectif, ce qui n'est pas le cas en général, pour les problèmes d'optimisation difficile. Cette famille de méthodes procède de la façon suivante.
On initialise un point de départ, $\overrightarrow{x_k}$, et on calcule le gradient $\nabla f(x_k)$. Le gradient indique la direction de la plus grande augmentation de la fonction f ; on se déplaçe dans le sens opposé (s'il s'agit de minimiser la fonction objectif), en utilisant un coefficient λ_k. Le nouveau point obtenu sera alors :

$$x_{k+1} = x_k - \lambda_k \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|} \qquad \lambda_k > 0$$

Une suite de points $x_0, x_1, \ldots, x_n$ est alors obtenue, et nous nous approchons de l'optimum. λ_k définit le pas de déplacement à chaque itération. Si le pas de déplacement est fixe, la méthode s'appelle à pas déterminé.

L'inconvénient de ces méthodes est que la convergence est ralentie pour certains types de fonctions : les déplacements successifs sont orthogonaux, donc l'algorithme va être piégé si les vallées (s'il s'agit d'une minimisation) sont étroites. Dans le cas des fonctions non convexes, la méthode risque de converger vers un optimum local dépendant du point de départ choisi. Dans des régions plates, ou raides, la convergence sera fortement ralentie.

Le simplexe de Nelder & Mead

Les premières méthodes "directes" (i.e. sans gradient) ont été présentées au début des années 1950, et d'autres méthodes directes ont continué à être proposées jusqu'à la fin des années 1960. Elles sont basées sur l'intuition géométrique, plutôt que sur des preuves mathématiques.
L'algorithme de simplexe traditionnel (voir algorithme 1.1) a été introduit par *Nelder & Mead* dans les années 1960. Le simplexe de Nelder & Mead peut être défini de la façon suivante. Soit f la fonction objectif que l'on cherche à optimiser (dans ce cas, trouver le maximum). Soit un simplexe S défini par $N+1$ sommets, dans un espace de dimension N. Soit $\{x_j\}_{j=1,\ldots,n+1}$, ces derniers sommets, tels que $f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{N+1})$, donc x_1 est le plus mauvais point et x_{N+1} est le meilleur point. x_{N+1} est modifié à chaque itération, selon l'équation suivante : $x_{(\mu)} = (1+\mu)x^* - \mu x_{N+1}$, où x^* désigne le centroïde de $x_1, x_2, \ldots, x_N$, $x^* = \frac{1}{N}\sum_{i=1}^{N} x_i$. Les opérations de base (voir figure 1.1) sont les suivantes :

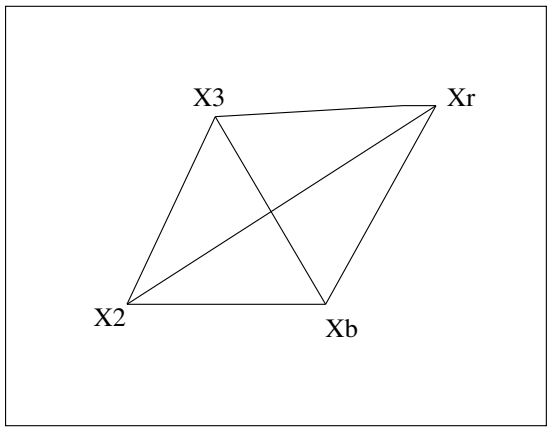
- Réflexion $\mu = 1,\ x = 2x^* - x_{N+1}$
- Expansion $\mu = 2,\ x = 3x^* - 2x_{N+1}$
- Contraction externe $\mu = 0.5,\ x = \frac{3}{2}x^* - \frac{x_{N+1}}{2}$
- Contraction interne $\mu = -0.5,\ x = \frac{x^*}{2} + \frac{x_{N+1}}{2}$

Une autre étape importante est le rétrécissement :

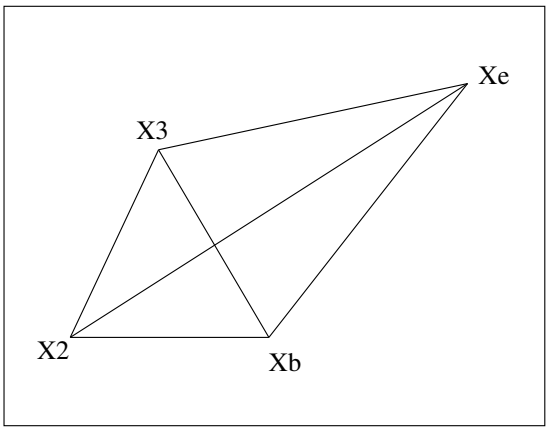
$$\text{pour } i = \{2, \ldots, n\}, \quad x_i = x_1 - (x_i - x_1)/2$$

La méthode du simplexe de Nelder-Mead classique comporte les étapes suivantes :

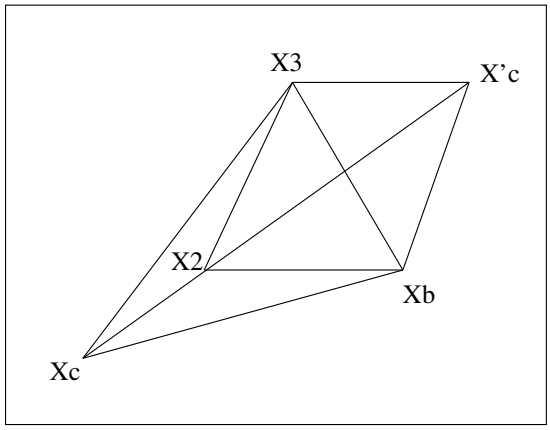
- Si $f(x_1) \leq f(x_R) \leq f(x_N)$, réflexion : $x_{N+1} = x_R$
- Si $f(x_E) < f(x_R) < f(x_1)$, expansion : $x_{N+1} = x_E$
- Si $f(x_R) < f(x_1)$ et $f(x_E) \geq f(x_R)$, réflexion : $x_{N+1} = x_R$
- Si $f(x_N) \leq f(x_R) < f(x_{N+1})$ et $f(x_{CE}) \leq f(x_R)$, contraction externe : $x_{N+1} = x_{CE}$
- Si $f(x_R) \geq f(x_{N+1})$ et $f(x_{CI}) < f(x_{N+1})$, contraction interne : $x_{N+1} = x_{CI}$
- Sinon rétrécissement : pour $i = \{2, \ldots, n\},\ x_i = x_1 - (x_i - x_1)/2$



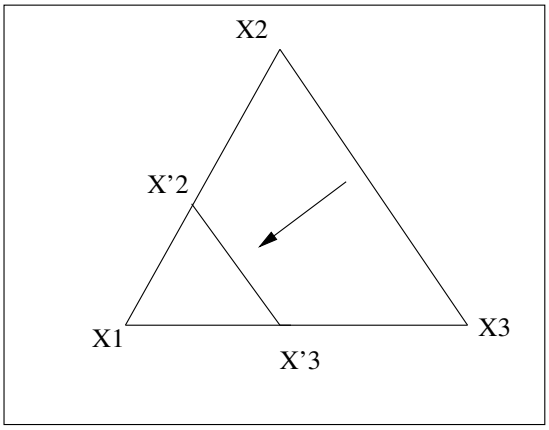
(a) Réflexion



(b) Expansion



(c) Contraction (interne ou externe)



(d) Rétrécissement

FIG. 1.1 – Exemples de modifications du simplexe de Nelder-Mead sur un problème à deux dimensions : réflexion, expansion, contraction externe/interne, rétrécissement. x_b est le sommet associé à la valeur minimale de la fonction objectif.

Algorithme 1.1 La méthode du simplexe de Nelder-Mead

Initialiser un simplexe et calculer les valeurs de la fonction sur ces noeuds ;

Répéter les itérations $t, t \geq 0$ jusqu'au critère d'arrêt ;

Ordonner les noeuds $x_0^t, x_1^t, \ldots, x_n^t$ conformément à :

$$f(x_0^t) \leq f(x_1^t) \leq \ldots \leq f(x_n^t)$$

Calculer le centre de gravité :
$\bar{x}^t = \frac{1}{n} \cdot \sum_{i=0}^{n-1} x_i^t$

Réflexion, calculer le point de réflexion à partir de :
$x_r^t = \bar{x}^t + \rho(\bar{x}^t - \bar{x}_n^t)$
Si $f(x_0^t) \leq f(x_r^t) < f(x_{n-1}^t)$ alors $x_n^t \leftarrow x_r^t$, itération suivante.

Expansion, si $f(x_r^t) < f(x_0^t)$ calculer le point d'expansion :
$x_e^t = \bar{x}^t + \chi(x_r^t - \bar{x}^t)$
Si $f(x_e^t) < f(x_r^t)$ alors $x_n^t \leftarrow x_e^t$, itération suivante ;
Sinon $x_n^t \leftarrow x_r^t$, itération suivante.

Contraction
Extérieure
Si $f(x_{n-1}^t) \leq f(x_r^t) < f(x_n^t)$, effectuer une contraction extérieure :
$x_{oc}^t = \bar{x}^t + \gamma(x_r^t - \bar{x}^t)$
Si $f(x_{oc}^t) \leq f(x_r^t)$ alors $x_n^t \leftarrow x_{oc}^t$, itération suivante ;
Sinon aller à l'étape *Raccourcir*.
Intérieure
Si $f(x_{n-1}^t) \leq f(x_r^t) \geq f(x_n^t)$, effectuer une contraction intérieure :
$x_{ic}^t = \bar{x}^t + \gamma(x_r^t - \bar{x}^t)$
Si $f(x_{ic}^t) \leq f(x_r^t)$ alors $x_n^t \leftarrow x_{ic}^t$, itération suivante ;
Sinon aller à l'étape *Raccourcir*.

Raccourcir le simplexe autour de x_0^t :
$x_i^t = \hat{x}_i^t = x_i^t + \frac{1}{2}(x_0^t - x_i^t), i = 1, \ldots, n$

Fin

1.2.2 Métaheuristiques

Les premières métaheuristiques datent des années 1980, et bien qu'elles soient d'origine discrète, on peut les adapter à des problèmes continus. Elles sont utilisées généralement quand les méthodes classiques ont échoué, et sont d'une efficacité non garantie. Le terme métaheuristique est utilisé par opposition aux heuristiques particulières pour un problème donné. Les métaheuristiques peuvent être utilisées pour plusieurs types de problèmes, tandis qu'une heuristique est adaptée à un problème donné. Les métaheuristiques ont également comme caractéristiques communes leur caractère stochastique, ainsi que leur inspiration, une analogie avec d'autres domaines de recherche (la biologie, la physique, etc.). Les métaheuristiques ne sont pas des méthodes figées ; il n'y a pas de relation d'ordre quant à l'efficacité d'un algorithme ou d'un autre, cela dépend plutôt des paramètres utilisés, de l'application elle-même ou du problème. Pour plus d'information, on peut consulter [41].

Inspiration biologique

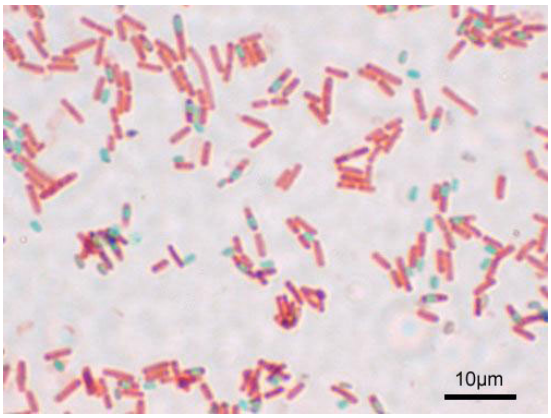
À la fin des années 80, une nouvelle voie d'exploration est apparue en intelligence artificielle. Il s'agit de l'étude et de l'utilisation des phénomènes observés dans la nature (voir figure 1.2). En effet, ce sont des systèmes composés d'agents très simples, qui peuvent produire des constructions complexes et des solutions à des problèmes non triviaux (tri, parcours optimaux, répartition de tâches, . . .).
Les informaticiens ont repris les principes d'auto-organisation et d'"émergence" présents dans ces sociétés, pour définir ce que l'on nomme l'intelligence collective.
Nous pouvons dire que l'intelligence collective caractérise un système où le travail collectif des entités (non complexes) interagissant entre elles fait "émerger" un comportement complexe global.

La méthode de recherche par colonie de fourmis basée sur le dépôt et l'évaporation de pistes de phéromone, et la méthode d'essaims particulaires ne sont que deux méthodes parmi d'autres qui s'inspirent de la biologie. Nous trouvons ainsi, dans la nature, plusieurs exemples qui donnent des idées pour la conception algorithmique. À titre d'exemple, nous pouvons citer les vols groupés d'oiseaux. Ces comportements se trouvent aussi chez des organismes plus simples unicellulaires, comme les bactéries. De nouvelles recherches ont établi que les bactéries utilisent des molécules pour communiquer entre elles. Elles utilisent un réseau composé de liens de cellule à cellule. Elles sont capables de détecter des changements dans leur environnement, s'associer avec des bactéries d'une même espèce, faire des alliances bénéfiques avec d'autres

(a)

(b)

(c)



(d)

FIG. 1.2 – Auto-organisation dans les systèmes biologiques : (a) Une colonie de fourmis qui ramène de la nourriture vers le niz (b) un essaim d'abeilles (c) un vol groupé d'oiseaux (d) une formation de bactéries [158].

espèces, et prendre l'avantage sur des espèces concurrentes. Ce type de stratégie collective est également attribué aux fourmis, aux abeilles et . . . aux humains. Les avantages liés à l'utilisation d'une telle approche sont :

- la production d'une performance collective supérieure à celle des individus ;
- une plus grande adaptation et flexibilité aux environnements réels (en général dynamiques) ;
- la fiabilité du système dans son ensemble (la perte d'un agent ne met pas en cause le processus général) ;
- le faible coût des "unités".

Par contre, un certain nombre de problèmes surgissent :

- difficulté pour anticiper la résolution d'un problème par une intelligence "émergente" ;
- problèmes de formulation, de compréhension et de preuve de la résolution ;
- nécessité d'un grand nombre d'agents (notion de masse critique), donc risques de conflits ;
- risques de comportements oscillatoires ou bloquants ;
- pas de coopérations locales intentionnelles, c'est-à-dire de comportements volontairement coopératifs dédiés ou exigés par les individus.

Auto-organisation

C'est un mécanisme connu par les biologistes [22], qui stabilise les systèmes biologiques et évite les fluctuations indésirables. Un exemple est la régulation du niveau de la température du corps. Si le corps passe par des fluctuations de température, alors une non-conformité entre la température actuelle du corps et la température de référence du corps est détectée. Ce phénomène est détecté par des capteurs sensibles dans le cerveau.

Un autre exemple bien connu est la régulation du niveau de sucre dans le sang, c'est un système qui opère très lentement chez les personnes normales, mais fonctionne mal chez les personnes atteintes de diabète. Le niveau du sucre est réglé par un mécanisme d'asservissement négatif, qui s'effectue par l'introduction d'insuline. Une augmentation du niveau du glucose dans le sang, après un repas sucré, déclenche rapidement la sécrétion d'insuline par le pancréas, ce qui a des effets physiologiques, incluant la rétroaction, contre l'augmentation du niveau de sucre dans le sang. Dans ce cas, le système procède de façon à empêcher les grandes fluctuations du niveau de glucose dans le sang.

Dans les deux exemples, l'individu reçoit et envoit des informations qui déclenchent un asservissement négatif. Un système, qui subit de petites perturbations, déclenche une réponse oppo-

sée pour contrebalancer la perturbation. Cette réponse opposée agit normalement sur le système tout entier. Dans le premier exemple, une baisse de température déclenche des réponses, qui augmentent la température du corps. Dans le deuxième exemple, une augmentation du niveau du glucose dans le sang déclenche une réponse, qui fait diminuer le niveau. Dans d'autres cas, la réaction joue un rôle continu, sans nécessité de signaux d'augmentation ou de baisse pour la déclencher.

Un exemple célèbre de l'auto-organisation est l'évaporation de traces de phéromone chez certains insectes sociaux, cette évaporation agit d'une façon globale et continue dans le temps. Sans ce processus, l'utilisation de phéromone par la colonie serait sans utilité, puisque tout le système se mettrait à renforcer le même chemin de plus en plus, ce qui n'est pas adapté pour des environnements variables. Un autre exemple est observé dans les traces de pas dans les sentiers, où les conditions physiques ou climatiques comme l'érosion jouent le rôle d'asservissement négatif. En résultat, nous obtenons une mémoire dynamique et robuste (une sorte de plan global) partagée par tous les individus, permettant de trouver la bonne solution ou le bon chemin, par exemple le chemin entre deux villages. Ce qui est frappant dans cet exemple, c'est que l'ensemble de la population arrive au bout du compte à trouver un chemin commun sans communication directe. Le signal de renforcement positif se fait d'une façon discontinue dans le temps, et, à chaque nouveau changement, la population est redistribuée localement pour reconnecter le chemin global. Le plan du système cognitif en entier est variable au fil du temps, s'adapte intelligemment, et s'auto-organise en réponse à des environnements non connus ou variables, sans plan ou stratégie globale. C'est comme si l'environnement coopérait directement à l'aide d'un plan global et fictif, pour former un plan complètement nouveau.

Le chemin ne sera jamais réorganisé pour explorer de nouvelles régions dans l'espace de recherche, sans l'apparition d'obstacles ou sans que des changements brusques arrivent dans l'environnement, et sans que des individus qui perdent leur chemin arrivent à trouver d'autres solutions. Mais il faut dire aussi que le chemin n'aura jamais été créé, sans la présence locale de certains individus pour renforcer les traces. Donc, sans le processus de rétroaction négative, le système se stabilise sur une configuration donnée, et un chemin marqué ne changera jamais, les traces seront renforcées de plus en plus, sans se déplacer au fil du temps. Mais le système a besoin de composantes (des individus) pour assurer le renforcement des solutions précédentes, c'est l'*exploitation* de l'espace de recherche ; et de composantes qui évoluent, volontairement ou pas, d'une façon aléatoire, ces dernières serviront au processus d'*exploration* de l'espace de recherche. En général, les composantes d'un système social peuvent assurer les deux processus (exploration / exploitation).

L'augmentation explosive de la population humaine est un exemple de l'effet de la rétroaction positive. Des termes comme auto-avancement, amplification, facilitation et auto-catalyse sont utilisés pour désigner la rétroaction positive. Un autre exemple est le processus d'échantillonnage ou d'agrégation d'individus. Certaines espèces d'oiseaux vivent en colonies. Le fait de vivre en colonie a des bénéfices, comme une meilleure détection des prédateurs, ou encore la facilité de recherche des sources de nourriture. Dans ce cas, le mécanisme est l'imitation : des oiseaux d'une même espèce, qui cherchent à nicher, sont attirés par des sites où d'autres oiseaux de cette espèce nichent déjà, le comportement est le suivant "nicher à côté du lieu où l'autre a niché". Chez les insectes sociaux, la rétroaction positive est illustrée par le renforcement des traces de phéromone, ce qui permet à la colonie toute entière d'exploiter des solutions anciennes ou récentes. En général, comme dans les exemples précédents, la rétroaction positive est imposée implicitement au système, et localement à travers les unités qui la constituent. Les bancs de poissons suivent le comportement "je te suis où tu vas". Chez les humains, les bâillements et les rires contagieux sont des exemples de rétroaction positive, concernant la forme, "je fais ce que tu fais". Le fait de voir quelqu'un bailler ou même vouloir bailler peut déclencher chez nous un bâillement.

Cependant, il y a un risque si la rétroaction positive agit toute seule, sans la présence d'une rétroaction négative. Cette dernière a un rôle important, puisqu'elle maintient sous contrôle l'"effet boule de neige", ce qui permet d'avoir des paramètres adaptés. Le fait que la rétroaction positive par nature s'amplifie peut engendrer un potentiel destructif ou explosif dans le système où elle opère. Le comportement est donc plus compliqué qu'il n'y paraît. Par exemple, pour un banc de poissons, le vrai comportement est le suivant "je vais où les autres vont, sauf s'il n'y a plus de place". Dans ce dernier cas, les rétroactions positive et négative peuvent être implémentées dans des règles comportementales.

Il est intéressant de savoir comment les individus ont accès à l'information, ou la communiquent, puisque l'organisation émerge de plusieurs interactions. Il y a deux formes importantes de communication : soit les informations sont recueillies des voisins directement (communication directe), soit à partir du travail en cours d'exécution, c'est la *stigmergie*.

Algorithme de colonie de fourmis

D'une manière simplifiée, les fourmis commencent par se déplacer au hasard. Puis, lorsqu'elles ont trouvé de la nourriture, elles retournent vers leur colonie, en marquant leur chemin à l'aide de phéromone [34]. Si d'autres fourmis rencontrent ce chemin, il y a de fortes chances

qu'elles arrêtent leurs déplacements aléatoires et qu'elles rejoignent le chemin marqué, en renforçant le marquage à leur retour, s'il mène bien vers de la nourriture.

Dans le même temps, le chemin le plus court sera davantage parcouru, et donc plus renforcé et plus attractif, on parle de rétroaction positive (*positive feedback*). En considérant que la phéromone s'évapore, les chemins les moins renforcés finissent par disparaître, ce qui amène toutes les fourmis à suivre ce chemin le plus court.

L'algorithme de colonies de fourmis a été à l'origine principalement utilisé pour produire des solutions quasi-optimales au problème du voyageur de commerce, puis, plus généralement, aux problèmes d'optimisation combinatoire. On observe, depuis ses débuts, que son emploi se généralise à plusieurs domaines, depuis l'optimisation continue jusqu'à la classification, ou encore le traitement d'image [74], [125], [129], [109], [103], [69], [92] et [75].

Les variantes combinatoires apportent un avantage, par rapport aux autres métaheuristiques, dans le cas où le graphe étudié peut changer dynamiquement au cours de l'exécution : la colonie de fourmis s'adaptera en temps réel aux changements. Ceci est par exemple intéressant pour le routage d'un réseau.

Le problème du voyageur de commerce TSP Les fourmis partent initialement de la source et parcourent les différents chemins possibles (figure 5.1) jusqu'à la fin. Les quantités de phéromone sont initialement égales. Sur leur chemin de retour, les fourmis déposent de la phéromone sur les différents chemins. Les valeurs des quantités de phéromone sont renouvelées selon l'équation :

$$\tau_i = \tau_i + \frac{Q}{l_i}$$

avec Q une constante et l_i la longueur du chemin parcouru. Donc la nouvelle valeur de la quantité de phéromone est inversement proportionnelle à la longueur du chemin.
Un aller-retour d'une seule fourmi constitue une construction complète de la solution. Le processus est réitéré, et les fourmis repartent à nouveau de la source. Une fourmi donnée va choisir entre les chemins selon la probabilité :

$$P_{e_i} = \frac{\tau_i}{\sum_{i=1}^{n} \tau_i}$$

Dans notre exemple (figure 1.3) les probabilités seront les suivantes :

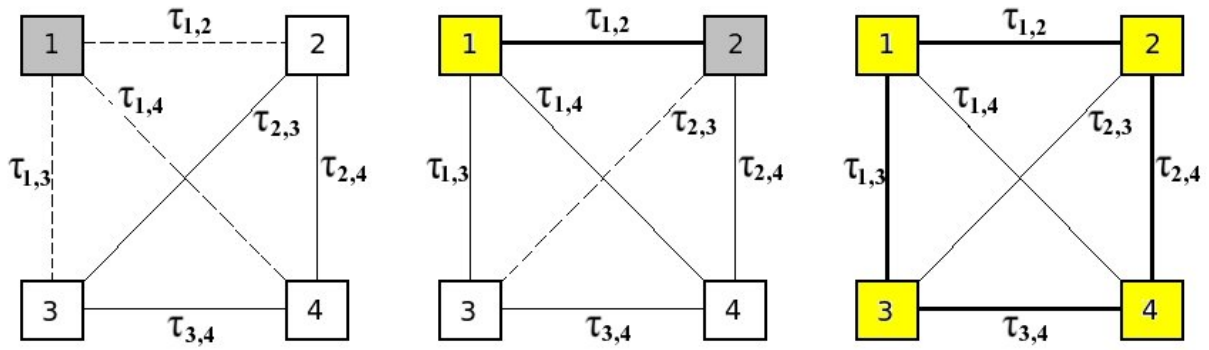
$$P_{e_1} = \frac{\tau_1}{\tau_1 + \tau_2 + \tau_3} \qquad P_{e_2} = \frac{\tau_2}{\tau_1 + \tau_2 + \tau_3} \qquad P_{e_3} = \frac{\tau_3}{\tau_1 + \tau_2 + \tau_3}$$

Donc, si nous supposons que $l_1 < l_2 < l_3$, alors $\tau_1 > \tau_2 > \tau_3$. Par conséquent, $P_{e_1} > P_{e_2} > P_{e_3}$. Le chemin choisi sera alors e_1.

D'autres changements affectent la phéromone, l'évaporation. Ce dernier changement a pour objectif d'empêcher une augmentation continue d'une seule solution, et de réduire l'importance d'autres solutions :

$$\tau_i = \tau_i.(1 - \rho) \qquad \text{avec} \quad \rho \in [0, 1]$$

ρ est le paramètre qui va régler la quantité de phéromone déposée, donc c'est le paramètre qui va influencer la vitesse de convergence de l'algorithme vers une solution donnée.



(a) Le début. (b) Le choix d'un chemin. (c) Un chemin complet.

FIG. 1.3 – Un exemple représentant la construction d'une solution dans le cas d'un problème discret, pour un problème de TSP avec quatres villes, N=4.

La méthode de colonies de fourmis a été initialement dédiée aux problèmes discrets (i.e. le cas où les valeurs possibles des variables d'un problème sont en nombre fini). Dans ce cas, nous pouvons définir un ensemble de composantes de la solution ; la solution optimale d'un problème donné sera alors une série ordonnée de ces composantes. Et un problème donné possède un ensemble de solutions. Pour fixer les idées, nous prenons l'exemple du problème du voyageur de commerce TSP. Le TSP peut être représenté par un graphe complètement connecté (un graphe où tous les noeuds sont inter-connectés). Les villes sont représentées par N_i noeuds avec $i = \{1, ..., n\}$ et n désigne le nombre total des noeuds. Un trajet entre deux noeuds (exemple : N_i et N_j) est désigné par e_{ij}. Par conséquent, une solution à un problème est un ensemble ordonné de $n - 1$ composantes $\{e_{ij}\}$ avec $i, j = \{1, ..., n\}$. Une construction d'une solution dans le cas du TSP consiste à partir d'une solution vide, à choisir une ville de départ et à ajouter les villes l'une après l'autre, puis à réitérer le processus jusqu'à ce que le nombre total de composantes ajoutées soit égal au nombre total de villes n. Dans le cas discret, les solutions ne sont pas

connues à l'avance, donc la valeur de la quantité de phéromone ne peut plus être attribuée à une solution complète, mais plutôt aux composantes de la solution.

Essaims particulaires

L'optimisation par essaims particulaires (OEP) est une métaheuristique d'optimisation, inventée par Russel Eberhart (ingénieur en électricité) et James Kennedy (socio-psychologue) en 1995. Cet algorithme s'inspire à l'origine du monde du vivant [16]. Il s'appuie notamment sur un modèle développé par le biologiste Craig Reynolds à la fin des années 1980, permettant de simuler le déplacement d'un groupe d'oiseaux. Une autre source d'inspiration, revendiquée par les auteurs, est la socio-psychologie.

Cette méthode d'optimisation se base sur la collaboration des individus entre eux. Elle a d'ailleurs des similarités avec les algorithmes de colonies de fourmis, qui s'appuient eux aussi sur le concept d'auto-organisation. Cette idée veut qu'un groupe d'individus individuellement peu intelligents puisse posséder une organisation globale complexe.

Ainsi, grâce à des règles de déplacement très simples (dans l'espace des solutions), les particules peuvent converger progressivement vers un minimum local. Cette métaheuristique semble cependant mieux fonctionner pour des espaces en variables continues.

Au départ de l'algorithme, chaque particule est donc positionnée, aléatoirement ou non, dans l'espace de recherche du problème. Chaque itération fait bouger chaque particule, en fonction de trois composantes :

1. sa vitesse actuelle V_k,
2. sa meilleure solution P_i,
3. la meilleure solution obtenue dans son "voisinage" P_g.

Cela donne l'équation de mouvement suivante :

$$V_{k+1} = a.V_k + b_1(P_i - X_k) + b_2(P_g - X_k)$$

$$X_{k+1} = X_k + V_{k+1}$$

avec :

b_1 tiré aléatoirement dans $[0, \psi 1]$

b_2 tiré aléatoirement dans $[0, \psi 2]$

Le pseudo-code pour la version la plus générale de la version continue de l'algorithme est présenté dans l'algorithme 1.2.

Algorithme 1.2 Optimisation par essaim particulaire (en variables continues)

n = **nombre** d'individus
D = **dimensions** du problème
Tant que critère d'arrêt :
    Pour $i = 1$ à n :
        Si $F(\vec{x}_i) > F(\vec{p}_i)$ alors :
            Pour $d = 1, \ldots, D$:
                pid = kid *// pid est donc le meilleur individu trouvé*
            fin d
        fin si
        $g = i$
        Pour j = index des voisins :
            Si $F(\vec{p}_j) > F(\vec{p}_g)$ alors :
                $g = j$ *// g est le meilleur individu du voisinage*
            fin si
        fin j
        Pour $d = 1, \ldots, D$:
            $v_{id}(t) = v_{id}(t-1) + \varphi_1(P_{id} - x_{id}(t-1)) + \varphi_2(P_{gd} - x_{id}(t-1))$
            $v_{id} \in (-V_{max'}, +V_{max})$
            $v_{id}(t) = x_{id}(t-1) + v_{id}(t)$
        fin d
    fin i
fin

Recherche tabou

L'idée de la recherche tabou est la suivante : à partir d'une position donnée, on explore le voisinage et on choisit la position dans ce voisinage qui minimise la fonction objectif. Il est essentiel de noter que cette opération peut conduire à augmenter la valeur de la fonction : c'est le cas lorsque tous les points du voisinage ont une valeur plus élevée. C'est à partir de ce mécanisme que l'on échappe aux minima locaux.

Le risque cependant est qu'à l'étape suivante, on retombe dans le minimum local auquel on vient d'échapper. C'est pourquoi, il faut que l'heuristique ait de la mémoire : le mécanisme consiste à interdire, d'où le nom de *tabou*, de revenir sur les dernières positions explorées.

Les positions déjà explorées sont conservées dans une file FIFO *First In First Out* (appelée souvent liste tabou) d'une taille donnée, qui est un paramètre ajustable de l'heuristique. Cette

file doit conserver des positions complètes, ce qui, dans certains types de problèmes, peut nécessiter l'archivage d'une grande quantité d'informations. Cette difficulté d'archivage peut être contournée en ne gardant en mémoire que les mouvements précédents.

Recuit simulé

L'algorithme du Recuit Simulé permet de résoudre le problème de minimum local. En effet, contrairement à la méthode du Gradient, un nouveau trajet de coût supérieur à celui du trajet courant ne sera pas forcément rejetté, son acceptation sera déterminée aléatoirement en tenant compte de la différence entre les coûts ainsi que d'un autre facteur appelé 'température'. Le recuit simulé (algorithme 1.3) s'appuie sur l'algorithme de Metropolis décrit en algorithme 1.4, qui permet de décrire l'évolution d'un système en thermodynamique. Par analogie avec le processus physique, la fonction à minimiser deviendra l'énergie E du système. On introduit également un paramètre : la température T du système.

Algorithme 1.3 La méthode du recuit simulé

$s := s_0$
$e := E(s)$
$k := 0$
tant que $k < k_{max}$ et $e > e_{max}$
    s_n := voisin(s)
    e_n := E(sn)
    Si $e_n < e$ ou aléatoire() < P($e_n - e$, temp(k/k_{max})) alors
        $s := s_n$; $e := e_n$
    $k := k + 1$
Retourner s

Algorithme 1.4 L'algorithme de Metropolis

Choisir $s \in \mathcal{S}$
Choisir $\lambda \in E$ où E.
Calcul de la variation d'énergie ΔE lors du passage de la valeur en s de x_s^n à λ.
si $\Delta E \leq 0$ le changement est accepté : $x_s^n = \lambda$
sinon, le changement est accepté avec la probabilité
    $p = \exp(\frac{-\Delta E}{T})$.

La solution initiale $s0$ peut être prise au hasard dans l'espace des solutions possibles s. À cette solution correspond une énergie initiale $E = E_0$. Une température initiale $T = T_0$ élevée est également choisie.

À chaque itération de l'algorithme, une modification élémentaire de la solution est effectuée. Cette modification entraîne une variation ΔE de l'énergie du système. Si cette variation est négative (c'est-à-dire qu'elle fait baisser l'énergie du système), elle est acceptée. Sinon, elle est acceptée avec une probabilité $e^{-\frac{\Delta E}{T}}$.
On itère ensuite selon ce procédé, en gardant la température constante.

Lorsque le système atteint un équilibre thermodynamique (au bout d'un certain nombre de changements), on diminue la température du système. On parle alors de paliers de température. Si la température atteint un seuil assez bas ou que le système devient figé, l'algorithme est stoppé.

La température joue un rôle important. À haute température, on est libre de se déplacer dans l'espace des solutions ($e^{-\frac{\Delta E}{T}}$ est proche de 1) en choisissant des solutions ne minimisant pas forcément l'énergie du système. À température intermédiaire, les modifications baissant l'énergie du système sont choisies en priorité, mais d'autres peuvent encore être acceptées, empêchant ainsi l'algorithme de tomber dans un minimum local.

Le pseudo code de l'algorithme 1.3 met en oeuvre le recuit simulé tel que décrit plus haut, en commençant à l'état s_0 et continuant jusqu'à un maximum de k_{max} étapes ou jusqu'à ce qu'un état ayant pour énergie e_{max}, ou moins, soit trouvé. L'appel *voisin* (s) engendre un état aléatoire voisin d'un état s. L'appel *aléatoire* renvoit une valeur aléatoire dans l'intervalle [0, 1]. L'appel *temp*(r) renvoit la température à utiliser, selon la fraction r du temps total déjà dépensée.

Algorithme génétique

Les algorithmes génétiques (voir algorithme 1.5) sont sans doute la classe la plus représentative et la plus populaire parmi les algorithmes évolutionnaires. Ils sont basés sur les travaux de Holland [73] et de Goldberg [57], et ils ont été utilisés au début avec des représentations binaires, où les opérateurs (voir tableau 1.1) de croisement et de mutation jouent un rôle majeur. Un algorithme génétique est un modèle d'apprentissage machine, qui est inspiré du processus de l'évolution observé dans la nature. Pour ce faire, un système artificiel contenant une population d'individus représentée par des chromosomes est créée. Des processus observés dans la nature montrent que différents individus sont en compétition pour l'accès aux ressources dans leur environnement. Des individus sont meilleurs que d'autres. Ces derniers individus ont plus

Terme	signification algorithmique
Individu	une solution à un problème donné.
Correspondance	c'est une fonction de correspondance de la qualité de la fonction objectif trouvée. En biologie, elle caractérise la capacité d'un individu à transmettre ses caractéristiques génétiques à la génération suivante.
Phénotype	la solution actuelle
Génotype	une représentation de la solution sur laquelle sont appliqués les opérateurs de croisement et de mutation
Gène	une partie du génotype
Allèle	la valeur d'un gène
Mutation	un opérateur aléatoire, qui reçoit en entrée une solution, et donne en sortie une nouvelle solution
Croisement	un opérateur qui croise les informations de deux individus, pour donner en sortie un ou deux nouveaux individus
Parent	l'individu utilisé pour produire de nouveaux individus (après une mutation et/ou un croisement)
Enfant	l'individu engendré par la mutation et/ou le croisement
Population	un ensemble de solutions
Génération	une itération de l'algorithme génétique

TAB. 1.1 – Les termes biologiques utilisés dans les algorithmes génétiques :

de chances de survivre et par conséquent de transmettre leurs gènes à la génération suivante. Dans la nature, on observe que l'encodage des informations génétiques (génome), qui résulte d'une reproduction non sexuelle, produit des enfants génétiquement proches de leurs parents. Par contre, une reproduction sexuelle permet la création d'enfants radicalement différents de leurs parents, tout en ayant les mêmes caractéristiques communes à l'espèce.

D'une façon simplifiée, le mécanisme au niveau moléculaire est le suivant : deux chromosomes se réunissent, et un échange d'information génétique est effectué. Ce dernier processus est appelé le croisement, puisque les composantes génétiques sont croisées d'un chromosome à l'autre. L'opération de croisement se passe dans un environnement dans lequel le choix d'un partenaire ou l'opération de sélection est une fonction de correspondance, par exemple on choisit le plus compétitif dans son environnement.

Quelques algorithmes génétiques utilisent une fonction de sélection simple (par exemple probabiliste) : dans ce cas, le croisement et la reproduction non sexuelle (où les composantes génétiques restent intactes) sont négligés. D'autres méthodes d'implantation choisissent un modèle où quelques individus sont sélectionnés au hasard dans des sous-groupes et les plus aptes sont conservés. Ce type de sélection est observé dans la nature.

Algorithme 1.5 L'algorithme génétique classique

Début
t=0
Initialiser P(t)
Évaluer la population P(t)
Tant que condition de terminaison non satisfaite faire
Début
t = t+1
Sélectionner P(t) à partir de P(t-1)
Mutations
Croisements
Évaluations
Fin
Fin

Les deux processus qui contribuent le plus à l'évolution sont le croisement et la correspondance basée sur la sélection / reproduction. Il y a eu des preuves mathématiques qui montrent que le processus de reproduction basé sur la correspondance est en fait sous-optimal.
La mutation joue aussi un rôle dans le processus de l'évolution, mais l'importance de son rôle reste un sujet de débat (certains la considèrent comme un opérateur de second plan, tandis que d'autres lui attribuent un rôle dominant dans le processus de l'évolution). Nous ne pouvons pas dire que les algorithmes génétiques (AGs) (une simulation du processus génétique) ne sont pas une recherche aléatoire. Les AGs utilisent un processus stochastique, mais les résultats obtenus sont meilleurs que des résultats aléatoires.
Les AGs sont utilisés dans de nombreux domaines d'application. Par exemple, les problèmes d'optimisation multidimensionnelle ; dans ce type de problème, les chaînes de caractères du chromosome sont utilisées pour coder les différents paramètres à optimiser. En pratique, le modèle génétique peut être implémenté en utilisant une matrice de bits pour représenter les chromosomes. Les simples opérations de manipulation de bits permettent d'implémenter le croisement, la mutation et d'autres opérations. Même si certains chercheurs ont étudié les AG avec des chaînes de caractères ou avec d'autres structures variables, la plupart des recherches se focalisent sur les chaînes de caractères de longueur fixe.
Quand un algorithme génétique est implémenté (voir algorithme 1.5), il est fait en général de la façon suivante : (1) Évaluer la valeur de la fonction objectif de tous les individus de la population. (2) Créer une nouvelle population se basant sur les opérations de croisement, évaluations

de correspondance et mutation, sur les individus que nous venons d'évaluer dans la première étape. (3) Eliminer l'ancienne population et itérer en utilisant la nouvelle.
Une seule itération de cette boucle est appelée *génération*. Il faut noter qu'il n'y a pas de bases théoriques pour ce modèle d'implémentation. En fait, ce processus en entier, ou bien le modèle, n'est pas observé dans la nature, mais il semble être un modèle d'implémentation convenable. La première génération (génération 0) de ce processus opère sur une population d'individus engendrés aléatoirement. Et, à partir de cette dernière génération, les opérateurs génétiques opèrent en parallèle, avec l'évaluation de la fonction pour améliorer la population [130], [133].

Estimation de distribution

Les algorithmes à estimation de distribution (EDA) [2] utilisent des techniques d'apprentissage pour résoudre des problèmes d'optimisation, en essayant d'"apprendre" au fur et à mesure les régions les plus prometteuses de l'espace de recherche. Plus particulièrement, un modèle probabiliste est utilisé pour engendrer des solutions candidates, l'apprentissage est utilisé pour adapter le modèle probabiliste et pour explorer des régions de l'espace de plus en plus prometteuses.

Chaque composante (variable) dans un individu est appelée *gène*. Ces dernières composantes (gènes) sont parfois indépendantes les unes des autres, et d'autres fois corrélées. Mais des communications et des échanges d'informations sont effectués entre individus à l'aide des opérateurs de sélection et de combinaison. Ce type d'échange permet de combiner des solutions (individus) partielles pour engendrer des solutions de haute qualité (Holland 1975 ; Goldberg 1989). Le comportement des AGs dépend du choix des opérateurs de sélection, croisement, mutation, des probabilités de croisement et de mutation, la taille de la population, la vitesse de reproduction d'une génération, le nombre de générations, etc. Mais le problème d'interaction entre les variables est rarement considéré. Ce qui a pour conséquence de produire des enfants de moindre qualité à partir de la combinaison fixe des deux parents, d'où des convergences vers des optimums locaux.
Pour éviter cela, le processus de combinaison des deux parents est remplacé par une génération de nouvelles solutions, selon une distribution de probabilité sur toutes les solutions prometteuses de la génération précédente. Cette nouvelle approche est appelée *algorithme à estimation de distribution* (EDA). Un pseudo-code de l'approche EDA est donné dans l'algorithme 1.6. Les EDAs ont été introduits dans le domaine évolutionnaire par Mühlenbein et Paab (1996).
Dans les EDAs le problème d'interaction entre les variables d'individus est pris en considé-

ration. Dans les algorithmes évolutionnaires, les interactions sont considérées implicitement, tandis que, dans les EDAs, ces interactions sont explicitement exprimées via des distributions de probabilités associées à des individus, dont les variables ont été sélectionnées à chaque génération.

Algorithme 1.6 Un algorithme à estimation de distribution

Étape 1 : $D_0 \leftarrow$ Engendrer aléatoirement M individus (la population initiale).
Étape 2 : $D_{l-1}^{se} \leftarrow$ Répéter les étapes 3-5 pour $l = 1, 2, \ldots$ jusqu'à ce que les critères d'arrêt soient atteints.
Étape 3 : Sélectionner N individus à partir de D_{l-1} selon les méthodes de sélection.
Étape 4 : $pl(x) = p(x|D_{l-1}^{se})$ Estimer la distribution de probabilité d'un individu sélectionné.
Étape 5 : $D_l \leftarrow$ Échantillonner M individus (la nouvelle population) selon $pl(x)$.

La probabilité de distribution est calculée à partir d'une base de données contenant des individus sélectionnés de la génération précédente. Les méthodes de sélection utilisées pour les AGs peuvent être également utilisées pour les EDAs.
L'échantillonnage de cette dernière distribution engendre des enfants (nouveaux individus). Ni les opérateurs de mutation, ni les opérateurs de croisement, n'ont été appliqués sur les EDAs. Il faut noter que l'estimation de la distribution de probabilité associée à la base de données contenant les individus n'est pas une tâche facile.

1.3 Conclusion

Nous avons présenté dans ce chapitre l'état de l'art sur des méthodes d'optimisation déterministes et sur quelques métaheuristiques les plus connues. Dans un premier temps, nous avons défini les problèmes "d'optimisation difficile", puis nous avons exposé les deux grandes catégories d'algorithmes d'optimisation : les méthodes déterministes, comme les méthodes de gradient et le simplexe de Nelder & Mead ; ainsi que les métaheuristiques, comme les algorithmes de colonie de fourmis, les essaims particulaires, la recherche tabou, le recuit simulé, les algorithmes génétiques et les algorithmes à estimation de distribution. Et nous avons exposé l'inspiration biologique et plus précisément l'auto-organisation et l'intelligence collective observées dans la nature, et quelques exemples qui donnent des idées pour la conception algorithmique.

CHAPITRE 2

ETAT DE L'ART DES ALGORITHMES POUR L'OPTIMISATION DYNAMIQUE

2.1 Définition

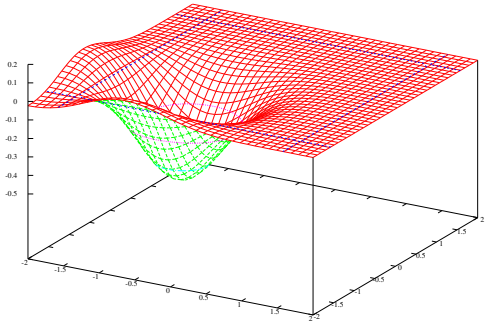
L'optimisation dynamique est un domaine nouveau, qui consiste à optimiser des fonctions objectifs qui changent au cours du temps. Parmi les méthodes connues figurent DHCIAC (recherche globale et locale), Monte-Carlo, le Simplexe dynamique, PSO dans le cas dynamique. L'optimisation dans le cas dynamique continu est un domaine de recherche récent. Dans la littérature, on le rencontre sous des appellations diverses : "optimisation dynamique" [111], [162], [20], [46], [138] ,"optimisation temps réel", "optimisation non stationnaire" [10] ou encore "optimisation des environnements changeants" [59].

D'une façon très générale, nous pouvons dire qu'une fonction statique est indépendante du paramètre temps : $f(\vec{x})$, où $\vec{x} = (x_1, x_2, \ldots, x_n)$, n est la dimension. En revanche, une fonction objectif dynamique $f(\vec{x}, T(t))$ dépend du temps. Un exemple d'une telle fonction est présenté en figure 2.1. Il en résulte que l'optimum global de la fonction que l'on cherche à optimiser peut évoluer avec le temps. La problématique de l'optimisation des fonctions dynamiques pose de nouveaux défis.

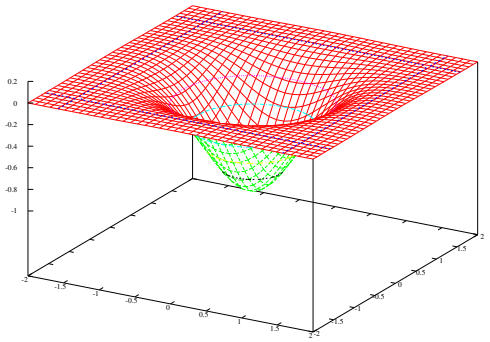
Les trois "étapes" nécessaires pour l'optimisation dans le cas dynamique sont [11] :

- trouver l'optimum le plus rapidement possible ;
- détecter un changement éventuel dans la fonction objectif ;
- suivre l'optimum.

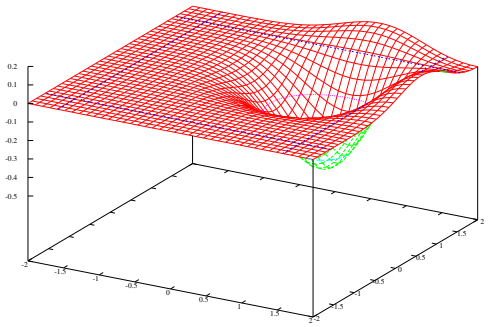
Les critères de performance d'une méthode ne peuvent plus être les mêmes dans le cas dynamique, puisque la cible recherchée (l'optimum) change.



(a) temps t_1



(b) temps $t_2 > t_1$



(c) temps $t_3 > t_2$

FIG. 2.1 – Un exemple simple de fonction dynamique.

La figure 2.1 est un exemple simple d'une fonction dynamique (Easom) : l'optimum se déplace au cours du temps. Le changement pourrait affecter d'autres paramètres, comme la valeur de la fonction objectif, la phase, etc. . . .

Parmi les applications réelles possibles de l'optimisation dynamique, nous pouvons citer le routage dans les réseaux de télécommunication, ou encore la gestion du trafic aérien, etc.

2.2 Mesure de performance

La mesure de performance d'un algorithme est beaucoup plus difficile dans le cas dynamique que dans le cas statique.
Malgré l'intérêt croissant concernant les métaheuristiques dans le cas dynamique, il n'y a pas de vision uniforme de la mesure de performance pour ces problèmes. La plupart des articles traitant du cas dynamique ne décrivent pas le problème testé ou la méthode de mesure de performance.
Dans la littérature, nous rencontrons plusieurs méthodes de mesure de la performance. Dans [97] les méthodes sont classifiées de la façon suivante :

- une mesure de la différence entre la valeur trouvée (le meilleur individu de la population) et la vraie valeur de l'optimum, juste après que le changement ait eu lieu dans la fonction objectif ;
- une mesure de la moyenne de la distance euclidienne à l'optimum à chaque génération ;
- la moyenne des meilleures générations, sur plusieurs lancements d'un algorithme sur un problème précis ;
- ou encore la différence entre la meilleure et la pire génération, dans un petit intervalle de générations récentes.

Nous allons utiliser les mesures suivantes pour les mesures de performance dans les sections suivantes :

$$\text{erreur moyenne sur la valeur} = moyenne\{\mid ROV_{t_j} - BFOV_{t_j} \mid\}$$

$$\text{erreur moyenne sur la position} = moyenne\{\mid ROP_{t_j} - BFOP_{t_j} \mid\}$$

où ROV_{t_j} et $BFOV_{t_j}$ sont respectivement la vraie valeur de l'optimum et la meilleure valeur trouvée de l'optimum à l'instant t_j, et ROP_{t_j} et $BFOP_{t_j}$ sont respectivement la vraie position de l'optimum et la meilleure position trouvée de l'optimum à l'instant t_j (avec $j \in \{0, \ldots, n\}$). Pour tenir compte de la variation des résultats entre les différents lancements de l'algorithme, nous avons moyenné les résultats sur 100 lancements.

2.3 Principales méthodes rencontrées dans la littérature

2.3.1 Le simplexe dynamique

L'algorithme simplexe dynamique [161] est largement inspiré du simplexe de *Nelder & Mead*. Un simplexe S_N est un polytope convexe avec $N+1$ sommets $\{x_j\}_{j=1}^{N+1}$ dans un espace à N dimensions dans $\mathbb{R}^N$. Le simplexe de *Nelder & Mead* est un algorithme itératif, qui commence à partir d'un simplexe initial, puis teste si la fonction objectif a une meilleure valeur sur un sommet voisin. Une itération k commence par ordonnancer et étiqueter les sommets du simplexe, de telle manière que :

$$f_1^{(k)} \geq f_2^{(k)} \geq \ldots \geq f_{N+1}^{(k)}$$

où $f_j^{(k)} = f_j(x_j^{(k)})$ est la valeur de la fonction objectif en $x_j^{(k)}$.

L'algorithme est décrit dans l'algorithme 2.1.

2.3.2 Algorithmes génétiques dynamiques

Les algorithmes génétiques décrits en section 1.2.2 ont été largement appliqués aux environnements dynamiques, dans le cas discret.

Dans [20], J. Branke classifie les méthodes dynamiques de la littérature de la façon suivante :

- Les méthodes réactives, qui réagissent au changement (ex : par le déclenchement de la diversité). Le principe général de ces méthodes est d'effectuer une action extérieure lors d'une détection d'un changement. Le but de cette action externe est d'augmenter la diversité. En général, les méthodes basées sur des populations perdent leur diversité lorsque la solution converge vers l'optimum, ce qui est handicapant si l'optimum change. Donc, en augmentant à nouveau la diversité, le processus de recherche est relancé.
- Les méthodes de maintien de la diversification. Ces méthodes essaient de maintenir la diversité de la population en espérant que, lorsque la fonction objectif est modifiée, la répartition des individus dans l'espace de recherche permettra de retrouver rapidement le nouvel optimum.
- Les méthodes qui gardent en mémoire les anciens optimums. Ces méthodes enregistrent l'évolution des différentes positions de l'optimum pour les réutiliser plus tard. Ces méthodes sont efficaces seulement dans le cas périodique.

Algorithme 2.1 La méthode du simplexe dynamique

1. **Le simplexe de départ** à l'itération k est $S_0 = \{x_j\}_{j=1}^{N+1}$ et les valeurs de la fonction objectif sur ses sommets valent $\{f_j\}_{j=1}^{N+1}$,

2. **Trier les sommets** du simplexe afin que $f_1 \geq f_2 \geq \ldots \geq f_{N+1}$,

3. **Réflexions successives**. Une réflexion du plus mauvais point du simplexe S_0, i.e., x_1, est effectuée $x_{N+2} = \frac{2}{N} \sum_{j=2}^{N+1} x_j = x_1$
   et on évalue $f_{N+2} = f(x_{N+2})$.
   Formant ainsi un nouveau simplexe $S_1 = \{x_j\}_{j=2}^{N+2}$.
   Réfléchir le premier (le plus mauvais) point de ce nouveau simplexe S_1, i.e., x_2, $x_{N+3} = \frac{2}{N} \sum_{j=3}^{N+1} x_j = x_2$
   et obtenir l'évaluation $f_{N+3} = f(x_{N+3})$. Nous obtenons ainsi le second nouveau simplexe $S_2 = \{x_j\}_{j=3}^{N+3}$.
   Continuer à réfléchir le premier point du simplexe nouvellement formé.
   Après que la réflexion se soit répétée M fois, nous obtenons
   $S_M = \{x_j\}_{j=M+1}^{N+M+1}$
   et une série de nouveaux simplexes $\{S_p\}_{p=1}^{M}$.

4. **Choisir** le simplexe de départ pour l'itération $k+1$.
   Calculer la valeur moyenne de la fonction à partir des sommets $\{S_p\}_{p=1}^{M}$
   $\bar{f}_{S_p} = \frac{1}{N+1} \sum_{j=p+1}^{N+p+1} f_j,\ p = 1, 2, \ldots, M.$
   Sélectionner le simplexe S_q satisfaisant
   $\bar{f}_{S_q} = \min \left\{ \bar{f}_{S_p} \right\}_{p=1}^{M}$

5. **Re-mesurer** la valeur au point x_{N+1}.

6. **Si** le nombre maximum d'évaluations est atteint, stopper,
   Sinon initialiser S_q comme nouveau simplexe de départ, et retourner en 2.

– Les méthodes qui utilisent un ensemble de plusieurs populations réparties sur les différents optimums. En utilisant plusieurs sous-groupes répartis sur les différents optimums, la probabilité de retrouver l'optimum global est renforcée.

La plupart des études concernant les problèmes dynamiques utilisent des algorithmes génétiques. H. Yu et al. teste dans [164] un algorithme génétique à taille variable sur des problèmes d'ordonnancement de tâches dynamiques ; la méthode proposée est flexible lorsqu'un changement arrive dans le système.

A. M. L. Liekens et al. étudient dans [86], un modèle stochastique à population finie. Les matrices stochastiques de transition sont combinées dans une seule matrice de Markov.

L'influence d'un algorithme génétique simple (sans mutation) à population infinie est étudiée dans [116] ; de plus, l'article montre que, pour un problème à changement périodique, la population converge vers une espèce généralisée. En se basant sur cette dernière observation, l'auteur calcule le seuil d'erreur et le pourcentage de mutation optimaux.

H. G. Cobb compare dans [26] l'AG classique à une méthode co-évolutionnaire (la notion de coopération / compétition inspirée de la nature). La méthode s'inspire de la façon dont le RNA (*Ribo-Nucleic acid*) est utilisé dans les organismes. Une stratégie adaptative de contrôle par mutation est testée pour maintenir une moyenne globale dans le temps sur la meilleure génération. La fonction testée est uni-modale et son optimum varie sinusoïdalement.

Dans [61], J. J. Grefenstette et C. L. Ramsey utilisent une technique d'apprentissage continue dans un environnement dynamique ; chaque agent teste d'une façon continue de nouvelles stratégies. On re-initialise dynamiquement la base de données utilisées par les agents, selon les résultats. Quand l'environnement change, le processus d'apprentissage est réinitialisé. Le modèle est testé dynamiquement. J. J. Grefenstette stipule dans [59] que la nature distribuée de l'AG le rend particulièrement convenable pour les problèmes dynamiques, mais la tendance à converger rapidement réduit leur capacité à détecter des régions qui deviennent soudainement intéressantes (optimales) après un changement de l'environnement. Alors la méthode employée permet de maintenir une diversification dans la population pour suivre l'optimum.

J. J. Grefenstette et H. G. Cobb évaluent dans [27] trois stratégies de maintien de la diversité dans la population. La première utilise l'AG classique avec un opérateur de mutation à niveau constant. La deuxième exploite un opérateur spécial, appelé *Random immigrants*, qui consiste à remplacer une partie de la population par des individus tirés au hasard à chaque génération. Le troisième opérateur, appelé hyper mutation, consiste à augmenter le niveau de mutation à chaque fois que le niveau de performance baisse. Les trois stratégies sont comparées sur des environnements où l'optimum change linéairement, périodiquement et aléatoirement.

Dans [112], la méthode citée dans [61] est utilisée et complétée par de nouvelles stratégies.
T. Sasaki et M. Tokoro étudient dans [122] la relation entre l'apprentissage et l'évolution sur un modèle de réseau de neurones où les noeuds capables d'apprendre évoluent au moyen d'un AG.
Dans [101], l'idée principale de T. Nanayakkara et al. est de faire durer davantage l'évolution en contrôlant la mutation. Les tests portent sur les environnements statiques, ainsi que dynamiques.
C. Bierwirth et al. [12] s'intéressent aux stratégies d'optimisation adaptative pour l'ordonnancement de tâches dans les chaînes de fabrication. L'article décrit brièvement l'application des algorithmes génétiques dans ce type de problème.
D. C. Mattfeld et C. Bierwirth [89] montrent expérimentalement la supériorité des AGs sur les méthodes de priorité utilisées pour les problèmes d'ordonnancement.
Dans [106], l'opérateur de sélection utilisé par G. Ochoa et al. est non aléatoire. L'article conclut que différentes valeurs de mutation peuvent changer radicalement les résultats et qu'une relation existe entre le taux de mutation choisi et l'opérateur de sélection. Nous rappelons que la mutation sert à maintenir la diversification dans la population.
C. Bierwirth et D. C. Mattfeld [13] proposent un modèle général pour l'ordonnancement de tâches dans des environnements statiques, dynamiques et non déterministes. Le modèle est testé sur un algorithme génétique classique.
R. Tinós et A. de Carvalho [144] associent à chaque individu une probabilité de mutation indépendante. Ces dernières probabilités sont changées au cours de l'exécution ; s'il s'avère que le changement augmente la performance, la probabilité de mutation est augmentée. Par conséquent, la recherche est concentrée dans des régions où le taux de mutation des individus est élevé.
Dans [55], une méthode pour augmenter l'adaptabilité dans les AG est proposée par A. Gaspar et P. Collard.
C. N. Bendtsen et T. Krink proposent dans [10] un algorithme génétique à base de mémoire, et le comparent avec l'AG classique. L'idée est de mémoriser d'anciennes bonnes solutions, et de les utiliser plus tard, ce qui n'est faisable qu'avec une très forte hyper mutation.
Dans [118], R. Salomon et P. Eggenberger définissent l'adaptation comme étant le processus de suivi de l'optimum dans un environnement dynamique, tandis que l'optimisation s'occupe du cas statique. Et l'article conclut qu'une diversification suffisante dans la population, avec un opérateur de sélection convenable, est suffisante pour l'adaptation (la mutation n'est pas utilisée).
K. Trojanowski, Z. Michalewicz et J. Xiao appliquent, dans [147], les AGs aux stratégies de

planification et de navigation d'un robot dans un environnement changeant.

Dans [120], une comparaison entre les modèles Darwinien et Lamarckien est faite par T. Sasaki et M. Tokoro. La principale différence entre ces modèles est que Lamarck considère qu'il y a un lien d'adaptation directe entre le niveau individuel et le niveau de la population, tandis que Darwin considère que ces deux niveaux sont complètement séparés. Les résultats concluent que le modèle Darwinien est plus stable et performant dans des environnements variables, et que les agents du modèle Darwinien sont plus adaptables.

K. Weicker et N. Weicker [155] comparent deux types d'auto-adaptation. La première est classique, tandis que la deuxième utilise une matrice de covariance. La fonction testée est une fonction de test classique en rotation. La première est trouvée plus performante.

S. M. Garrett et J. H. Walker essaient d'unifier dans [53] les méthodes évolutionnaires et non évolutionnaires dans des environnements dynamiques. L'idée est d'utiliser une méthode non évolutionnaire pour maintenir un niveau élevé de mutation non destructive.

S. Lin, E. D. Goodman et W. F. Punch [87] proposent une méthode d'ordonnancement dynamique basé sur les AGs.

Dans [160], une méthode appelée SBGA (*Shifting Balance Genetic Algorithm*) est étudiée en détail sur des environnements dynamiques par M. Wineberg et F. Oppacher. La SBGA est meilleure que l'AG classique en statique, mais l'écart de performance s'accroît en dynamique. La SBGA force un sous-groupe de la population à explorer d'autres espaces de recherche que ceux explorés par la population principale.

Dans [108], F. Oppacher et M. Wineberg ont fait des expérimentations sur une SBGA modifiée pour empêcher une convergence prématurée. J. J. Grefenstette [60] explore une alternative à la mutation classique, où tous les individus subissent le même niveau de mutation, à savoir une mutation contrôlée génétiquement.

W. Cedeño et V. R. Vemuri [24] réussissent à maintenir un sous-groupe de la population sur chaque sommet, par la méthode MNC-GA (*Multi-Niche Crowding Genetic Algorithm*). Cette dernière méthode est présentée comme un bon compromis entre l'exploitation et l'exploration.

A. Ghosh, S. Tsutsui et H. Tanaka introduisent dans [56] la notion de l'âge des individus. L'algorithme 2.2 décrit la méthode. La méthode est testée sur des fonctions dynamiques.

Dans [4], la spécificité est l'utilisation d'un opérateur de macromutation, une forme spéciale de la mutation pour maintenir la diversité.

P. D. Stroud montre dans [131] comment le filtre de Kalman peut donner de bons résultats dans un espace de recherche bruité ou changeant.

A. Simões et E. Costa [126] proposent un AG avec un opérateur génétique d'inspiration bio-

Algorithme 2.2 Algorithme génétique avec la notion de l'âge des individus

Engendrer aléatoirement les individus et leur âge
Calculer la valeur de la fonction objectif
Sélectionner quelques individus selon la valeur de performance
Appliquer les opérateurs de mutation et de croisement, et mettre l'âge de tous les nouveaux individus à zéro
Éliminer les individus les moins performants
Augmenter l'âge de un
Arrêter si les critères d'arrêt sont satisfaits
Sinon revenir à l'étape **Engendrer**

logique, pour remplacer l'opérateur de croisement classique. Ce dernier permet de maintenir la variété dans la population. La méthode est capable de réagir rapidement à des changements, sans avoir recours à l'opérateur de mutation. K. Trojanowski et Z. Michalewicz considèrent dans [146] deux techniques : la première est à mémoire et la deuxième est *Random immigrant*. Les deux méthodes sont étudiées et comparées.

Dans [84], K. Krishnakumar montre que le *Micro-Genetic Algorithm* est plus performant que le SGA (*Simple Genetic Algorithm*) pour résoudre les problèmes multimodaux et dynamiques.

Dans [99], l'hyper mutation orientée est revisitée et testée par R. W. Morrison et K. A. De Jong.

T. Hussain, D. Montana et G. Vidaver abordent dans [77] le problème de stationnement de véhicules en dynamique avec un AG.

C. W. Wilke [159] teste un AG sur un problème NK simple à variation lente pour essayer de comprendre l'évolution d'un algorithme génétique en dynamique.

J. E. Smith et F. Vavak [127] comparent plusieurs stratégies de sélection et de remplacement avec des approches élitistes et avec des hypermutations. Les méthodes élitistes sont trouvées plus performantes que les autres approches.

R. K. Ursem [149] propose un algorithme à multi-population, où les sous-populations sont distribuées pour détecter les optimums locaux ainsi que globaux. Une méthode de détection de vallées est utilisée pour guider la distribution des sous-populations. La méthode est trouvée particulièrement performante pour l'optimisation multi-modale.

J. Sarma et K. De Jong [119] effectuent des tests empiriques avec des stratégies décentralisées de diversification, pour confirmer l'hypothèse de l'utilité de la diversification en optimisation dynamique.

K. Yamasaki [162] stipule que la façon de sélectionner les agents est cruciale quand la vitesse de changement de l'environnement est plus rapide que la vitesse d'adaptation de l'algorithme.

La méthode proposée s'appelle "*Dynamic Pareto Optimum GA*". Un certain nombre d'agents choisis selon la méthode "*Pareto-optimum*" conservent les anciens optimums trouvés.
H. C. Andersen fait dans [3] une étude des algorithmes génétiques, et la relation entre la spécification et le suivi des optimums dans les fonctions dynamiques.
Dans [21] une classification des travaux concernant les environnements dynamiques est faite par J. Branke.
R. K. Ursem, B. Filipič et T. Krink appliquent dans [150] les algorithmes génétiques sur le problème *Greenhouse Control*. L. Schönemann montre dans [123] que même des petits changements de paramétrages peuvent apporter des résultats importants. Une mesure de performance est proposée pour pouvoir comparer avec d'autres approches.
C. N. Bendsten introduit dans sa thèse [9] une mémoire explicite, et teste sur des environnements où les changements sont cycliques ou répétitifs. La mémoire s'auto-adapte en déplaçant graduellement les solutions candidates mémorisées vers la solution des meilleurs individus. La méthode est comparée avec un algorithme génétique classique, et quatre algorithmes de colonies de fourmis.

2.3.3 La méthode de Monte-Carlo

On appelle méthode de Monte-Carlo toute méthode visant à calculer une valeur numérique, et utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Le nom de ces méthodes fait allusion aux jeux de hasard pratiqués à Monte-Carlo. La méthode de Monte-Carlo est un algorithme de recherche aléatoire qui cherche l'optimum d'une fonction en engendrant une suite aléatoire de nombres en fonction d'une loi uniforme. La méthode se présente en général comme indiqué dans l'algorithme 2.3.

Algorithme 2.3 Méthode de Monte-Carlo

1. **Engendrer** un vecteur initial $\overrightarrow{x}$ dans l'espace de recherche, ce dernier est la solution courante.
2. **Engendrer** un vecteur $\overrightarrow{x}'$
3. **Si** $\overrightarrow{x}'$ est meilleur que $\overrightarrow{x}$, $\overrightarrow{x}'$ devient la solution courante.
4. **Si** le critère d'arrêt est satisfait, alors fin.
5. **Sinon** retour à 2.

La méthode de Monte-Carlo fournit des solutions approximatives à une variété de problèmes mathématiques [70].

Dans les années 1970, le développement de la théorie de la complexité a fourni des preuves rationnelles pour l'utilisation de la méthode de Monte-Carlo. Cette théorie de la complexité a identifié une classe de problèmes pour laquelle le temps de calcul augmente au mieux d'une façon exponentielle en fonction du nombre de variables N.
La question était alors : la méthode de Monte-Carlo peut-elle approcher une solution à ce type de problème avec une précision statistique donnée, moyennant un temps de calcul polynomial ? En 1985, Karp a prouvé cette dernière propriété pour la fiabilité de l'estimation de la méthode de Monte-Carlo sur un réseau dont les terminaux tombent en panne d'une façon aléatoire. En 1989, Dyer a utilisé la méthode de Monte-Carlo pour estimer le volume d'un corps convexe dans un espace à M dimensions.

2.3.4 Les essaims particulaires en dynamique

D. Parrott et X. Li présentent dans [110], un algorithme d'essaims particulaires pour l'optimisation multimodale. La technique utilisée consiste à définir un rayon R où la meilleure valeur locale trouvée est attribuée à la population appartenant au cercle de rayon R. Cette technique permet à une sous-population de converger vers un optimum local. Ce qui a pour effet de former un ensemble de sous-populations réparties sur les optimums locaux (les optimums locaux peuvent changer au cours du temps pour devenir globaux). Un paramètre spécifique est défini pour limiter le nombre de particules dans les sous-populations. Les particules qui sortent d'une sous-population sont réinjectées aléatoirement dans l'espace de recherche.
T. M. Blackwell utilise dans [14] une technique pour résoudre des problèmes dynamiques, qui consiste à garder la diversité en attribuant aux particules des charges de répulsion en analogie avec les charges électrostatiques. Ainsi des essaims de particules chargées se forment autour des particules neutres (non chargées). L'article propose une mesure de diversité pour prévenir la faisabilité de détection des optimums changeants. La méthode est détaillée et testée dans [15].

Une variante avec des sous-populations

La méthode *Particle Swarm Optimization* (PSO), des essaims particulaires classique est décrite en section 1.2.2. La variante décrite dans cette section (voir [110] pour plus de détails) est un modèle dédié aux environnements dynamiques continus contenant plusieurs optimums. Le principe est d'utiliser des sous-groupes de populations en parallèle, et un mécanisme pour encourager le suivi de plusieurs optimums, en empêchant les encombrements sur un optimum trouvé.

Les sous-groupes sont centrés sur la meilleure position $pbest$ de la particule, dans une région locale définie par un rayon r (le rayon du sous groupe). L'initialisation aléatoire des particules d'un sous-groupe donné est faite dans une sphère de centre $pbest$ et de rayon r. Une particule candidate x est acceptée dans un sous-groupe si : $d(x, s) \leq r$, en désignant par d la distance entre la particule et le sous-groupe s :

$$d(x, s) = \sqrt{\sum_{i=1}^{n} (x_i - s_i)^2}$$

Si une particule est candidate à deux sous-groupes, elle choisira celui possédant la meilleure valeur.

Pour empêcher que toutes les particules convergent vers un seul optimum, un paramètre p_{max} est défini, tel que seulement les meilleures p_{max} particules soient choisies pour former un sous-groupe. Les autres particules sont redistribuées avec des positions aléatoires dans l'espace de recherche.

CPSO pour des environnements dynamiques

Une variante *Charged Particule Swarm Optimization* (CPSO) a été adaptée pour le cas dynamique dans [14] et [15]. Le principe de cette méthode consiste à associer à chaque particule une charge. La force de répulsion entre les particules chargées permet de garder une diversité dans la population et a pour conséquence de répartir les particules autour des noyaux de particules neutres.

L'essaim est un ensemble $S = \{x_i, v_i\}$ de particules ou individus, x_i est la position et v_i la vitesse, avec $i = \{1 \dots N\}$. Chaque vecteur est de dimension d, $j = \{1, \dots, d\}$. La position des particules est renouvelée en ajoutant une accélération à la vitesse courante. Et la nouvelle position est obtenue en ajoutant la nouvelle vitesse à la position courante des particules.

Le pseudo-code de la méthode est donné dans l'algorithme 2.4.

On pose : $a_i = \sum_{k \neq i} a_{ik}$.

$$a_{ik} = \frac{Q_i.Q_k}{|x_i - x_k|^3}(x_i - x_k), \text{ si } r_c \leq |x_i - x_k| \leq r_p$$

$$a_{ik} = \frac{Q_i.Q_k}{r_c^2}\frac{x_i - x_k}{|x_i - x_k|}, \text{ si } |x_i - x_k| < r_c$$

Algorithme 2.4 CPSO pour des environnements dynamiques

Initialiser $S = \{x_i, v_i\}$ dans un cube $[-X, X]^d$
$g, t = 0$

Répéter sur les N individus de la population
    $p_i = x_i$
    Si $f(p_i) < f(p_g)$ alors $g = i$
Fin

Répéter
    $t++$
    Répéter sur les N individus de la population
        Calculer a_i
    Répéter sur les d dimensions
            $v_{ij} = \chi(v_{ij} + \xi_1\phi_1(p_{ij} - x_{ij}) + \xi_1\phi_2(p_{gj} - x_{ij}))$
            $v_{ij} = v_{ij} + a_{ij}$
            $x_{ij} = x_{ij} + v_{ij}$
        Fin
        Si $f(x_i) < f(p_i)$ alors $p_i = x_i$
        Si $f(p_i) < f(p_g)$ alors $g = i$
    Fin

Fin si critère d'arrêt

Et

$$a_{ik} = 0, \text{ si } r_p < |x_i - x_k|$$

i, k sont les indices des particules, r_c et r_p sont des rayons de perception des particules. χ est une constante qui assure la convergence. ξ_1 et ξ_2 sont des nombres aléatoires dans l'intervalle $[0, 1]$.

2.3.5 Colonie de fourmis en optimisation dynamique

M. Guntsch et M. Middendorf testent dans [63] deux problèmes discrets combinatoires et dynamiques, le TSP (le voyageur de commerce) dynamique, et le QAP (affectation quadratique) dynamique sur un algorithme de colonies de fourmis modifié. L'idée principale est un transfert direct de l'ensemble des solutions trouvées à une itération à l'itération suivante, puis le calcul de la quantité de phéromone nécessaire pour la prochaine itération. Dans les algorithmes de colonies de fourmis traditionnels, c'est la phéromone qui est transférée directement d'une itération à une autre.

Puis les auteurs proposent dans [64] une modification de la façon dont la phéromone change, de manière à permettre de garder plus longtemps une trace des bonnes solutions jusqu'à un certain temps, puis éliminer explicitement leur influence de la matrice de phéromone. La méthode a été testée sur un TSP dynamique.

Dans [62], M. Guntsch et M. Middendorf proposent trois stratégies, la première approche permettant une ré-initialisation locale à une même valeur de la matrice de phéromone lorsqu'un changement est détecté. La deuxième approche consiste à calculer la valeur de la matrice selon la distance entre les villes (cette méthode est appliquée à un problème de voyageur de commerce dynamique, où une ville peut être enlevée ou ajoutée). La troisième approche utilise la valeur de la phéromone à chaque ville.

Les trois précédentes stratégies sont modifiées dans [65], en introduisant une notion élitiste : les meilleures fourmis sont seules autorisées à changer la phéromone à chaque itération, et lorsqu'un changement est détecté, les anciennes bonnes solutions ne sont pas oubliées, mais modifiées au mieux, pour qu'elles puissent devenir des nouvelles solutions raisonnables.

D. Merkle et M. Middendorf étudient dans [91] la dynamique de ACO ("*Ant Colony Optimization*"), puis proposent un modèle déterministe, basé sur un moyennage du comportement prévu des fourmis. Leur travaux mettent en évidence comment le comportement des fourmis est influencé par les caractéristiques de la matrice de phéromone, ce qui explique le comportement dynamique complexe. Les différents tests sont effectués sur un problème de permutation. Mais

les auteurs ne traitent pas des problèmes dynamiques.
Puis, dans [90], une nouvelle méthode de calcul de phéromone (la phéromone relative) est proposée, et comparée avec le modèle standard. Les nouvelles équations proposées sont :

$$p_{ij} = \frac{\tau^*_{ij} \cdot \eta_{ij}}{\sum_{h \in S} \tau^*_{ih} \cdot \eta_{ih}} \text{ où } \quad \tau^*_{ij} = \{\frac{\sum_{k=1}^{i} \tau_{kj}}{\sum_{k=1}^{n} \tau_{kj}}\}^{\gamma}$$

R. Schoonderwoerd et al. [124] appliquent l'algorithme de colonie de fourmis à un modèle simulé de réseau de télécommunication ; le réseau comporte une population d'agents mobiles (les fourmis), qui se déplacent d'un noeud à un autre du réseau, selon la quantité de phéromone déposée précédemment. La phéromone est déposée au fur et mesure de leur déplacement, selon la distance entre le noeud de départ et celui de la destination. Le modèle est comparé avec la méthode de routage par le chemin le plus court (*fixed shortest-path routes*).
AntNet est introduite dans [23] par G. D. Caro et M. Dorigo, c'est une nouvelle approche d'apprentissage adaptative dans le routage des tableaux de communication. Les agents parcourent le réseau et collectent les informations, la communication entre agents est indirecte et asynchrone. Le modèle est comparé à six méthodes de routage de la littérature.

2.3.6 Systèmes immunitaires en optimisation dynamique

Dans [54], une étude a été faite par A. Gaspar sur les systèmes immunitaires. Et deux modèles sont présentés : le SAIS (*Simple Artificial Immune System*) et le YaSAIS (*Yet Another Simple Artificial Immune System*) ; les deux méthodes sont appliquées sur un problème dynamique cyclique ou périodique. Les algorithmes immunitaires sont inspirés du système immunitaire chez les êtres vivants. Le SAIS utilise des vecteurs binaires pour la structure des antigènes et des récepteurs des cellules B. L'optimum représente l'antigène et la population de cellules B évolue pour le détecter. Le modèle comporte deux types de réponses immunitaires face à une variation de l'environnement : la première est réactive et la deuxième utilise la mémoire. Le modèle démarre avec une population de cellules B choisie aléatoirement. Puis il applique à chaque génération trois types d'opérateurs : l'évaluation, qui est faite selon une fonction de distance entre l'antigène (l'optimum) et les cellules B de la population, la sélection de clone, qui consiste à appliquer, sur les meilleurs individus choisis dans la phase précédente, des opérateurs semblables aux opérateurs de mutation utilisés par les algorithmes génétiques ; le dernier opérateur est le recrutement, où les individus clonés vont remplacer des individus de l'ancienne population. La méthode est une tentative d'imiter le système immunitaire, qui réagit face à de

nouvelles maladies inconnues, ou face à des maladies rencontrées dans le passé (mémoire). YaSais diffère principalement par l'utilisation d'une sous-population de cellules B (individus) dédiée à la préservation des anciens optimums.

2.3.7 Autres travaux

K. Weicker et N. Weicker [156] proposent un générateur de fonctions de test dynamiques basé sur la rotation des coordonnées, et sur la visibilité partielle. Un modèle mathématique pour l'environnement dynamique est proposé dans [154], une méthode de routage dynamique est proposée dans [151], et les réseaux de neurones sont appliqués au cas dynamique dans [105] et [121].

R. W. Morrison et K. A. De Jong [98] et Shengxiang Yang [163] proposent des générateurs de problèmes de test dynamiques.

K. Verbeeck et A. Nowé étudient dans [152] les colonies de fourmis avec une approche multi-agent. Cette approche est proposée pour comprendre les colonies de fourmis. Elle consiste à utiliser l'apprentissage machine avec des *Markovian Decision Problems* (MDP).

Dans [88], V. Maniezzo et A. Colorni appliquent les colonies de fourmis au problème d'affectation quadratique.

Dans [96], R. Montemanni et al. appliquent les colonies de fourmis au problème de distribution de fréquences dans un réseau de télécommunication sans fil. L'objectif de leur travaux est de minimiser la bande de fréquence requise pour garantir un niveau de qualité de réception dans le réseau.

T. Stützle et M. Dorigo présentent dans [132] une preuve de convergence pour une classe des algorithmes de colonies de fourmis.

E. M. T. Hendrix, P.M. Ortigosa et I. Garcìa présentent dans [71] la méthode *Controlled Random Search* (CRS) et des résultats sur la vitesse de convergence de ce type d'algorithmes.

Dans [137], J. Teng et Y. Liu appliquent les colonies de fourmis au problème d'allocation d'interrupteurs dans un réseau électrique.

2.4 Jeux de fonctions de test dynamiques

Pour effectuer des comparaisons, il est nécessaire de garder le même ensemble de problèmes à traiter d'une variante à l'autre, pour mieux comparer les résultats. D'où le jeu d'essais défini ici (voir tableaux 2.1 et 2.2), qui comprend plusieurs cas de figures, depuis la fonction simple

avec un minimum unique jusqu'à celle ayant un nombre élevé de minima locaux de valeurs proches. Notons qu'il s'agit ici uniquement de fonctions continues dynamiques. Ces fonctions ont été élaborées par Johann Dréo dans le cadre de sa thèse de doctorat [36].

Soit $f(\vec{x}, T(t))$ la forme générale d'une fonction objectif dynamique. On distingue alors deux principaux composants des problèmes d'optimisation dynamique : le premier concerne un changement dans la structure de la fonction objectif elle-même f, et le second concerne l'évolution de la fonction, appelé fonction de temps $T(t)$, où t est le temps écoulé. A titre d'illustration, on prend l'exemple de deux fonctions dynamiques basées sur des fonctions classiques, *Easom* et *B2* respectivement :

$$f(\vec{x}, T(t)) = -\cos(x_1) \cdot \cos(x_2) \cdot e^{-(x_1 - T(t))^2 - (x_2 - T(t))^2}$$

La position de l'optimum est la seule valeur qui change, comme le montre la figure 2.1, et :

$$f(\vec{x}, T(t)) = (x_1 - T(t))^2 + 2 \cdot (x_2 - T(t))^2 + 0.7$$

$$-0.3 \cdot \cos(3\pi - (x_1 - T(t))) - 0.4 \cdot \cos(4.\pi - (x_2 - T(t)))$$

où la fonction toute entière change.

Les différentes variantes de fonctions dynamiques que l'on rencontre le plus souvent sont les suivantes :

- variation de la fonction objectif et de l'optimum ;
- variation de l'optimum tout seul ;
- variation de la fonction objectif et non pas de l'optimum.

Nous pouvons observer aussi d'autres variations plus complexes :

- variation du nombre de dimensions ;
- variation des contraintes ;
- variation de la phase (rotation autour d'un axe) ;
- changement complet (par exemple, passage de convexe à concave).

La fonction du temps $T(t)$ peut être linéaire, périodique (par exemple $T(t) = 2 \cdot \cos(t)$) ou non linéaire.

Pour pouvoir tester notre algorithme, J. Dréo a élaboré un ensemble de fonctions de test pour l'optimisation dynamique. Huit fonctions dynamiques de test sont construites à partir de fonctions classiques, modifiées par ajout de $T(t)$:

AbPoP	basée sur la fonction *Morrison* : toute la structure de f, sauf la position, varie périodiquement.
AbVP	basée sur la fonction *Morrison* : toute la structure, sauf la valeur, varie périodiquement.
ADL	basée sur la fonction *Rosenbrock* : toutes les dimensions varient linéairement.
APhL	basée sur la fonction *MartinGady* : la phase varie linéairement.
APoL	basée sur la fonction *B2* : la position varie linéairement.
AVP	basée sur la fonction *Shekel* : la valeur varie périodiquement.
OPoL	basée sur la fonction *Easom* : la position de l'optimum varie linéairement.
OVP	basée sur la fonction *Morrison* : la valeur de l'optimum varie périodiquement.

TAB. 2.1 – Les abréviations des fonctions de test et leur signification.

Fonction ID	**Equation**
AbPoP	$min. \sum_{i=1}^{5} \left(\left(-H_i + R_i . \left((x_1 - P_{i,1}(T(t)))^2 + (x_2 - P_{i,2})^2 \right) \right) \right)$
AbVP	$min. \sum_{i=1}^{5} \left(\left(-H_i(T(t)) + R_i . \left((x_1 - P_{i,1})^2 + (x_2 - P_{i,2})^2 \right) \right) \right)$
ADL	$\sum_{i=1}^{T(t)} (100.(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2)$
APhL	$(X_1 - X_2)^2 + (\frac{X_1 + X_2 - 10}{3})^2$ avec $X_1 = x_1 - T(t)$ et $X_2 = x_2 - T(t)$
APoL	$(x_1 - T(t))^2 + 2 \cdot (x_2 - T(t))^2 + 0.7$ $-0.3 \cdot \cos(3\pi - (x_1 - T(t))) - 0.4 \cdot \cos(4.\pi - (x_2 - T(t)))$
AVP	$T(t) + \sum_{i=1}^{n} \frac{1}{(x - a_i)^T . (x - a_i) + c_i}$
OVP	$min. \sum_{i=1}^{5} \left(\left(-H_i + R_i . \left((x_1 - P_{i,1})^2 + (x_2 - P_{i,2})^2 \right) \right) \right)$

TAB. 2.2 – Les fonctions de test et leur expression mathématique.

Les premières lettres signifient (**A** : Toute, **Ab** : Toute sauf, **Po** : Position, **V** : valeur, **Ph** : Phase). La dernière lettre (**P** : périodique, **L** : linéaire).

H_i, R_i et c_i sont des vecteurs de dimensions 5, 5, 10 respectivement ; et $P_{i,j}$, a_i sont des matrices de dimensions (4×2), (4×10) respectivement.

2.5 Conclusion

Nous avons présenté dans ce chapitre l'état de l'art des méthodes d'optimisation dynamiques comme le simplexe dynamique, les algorithmes génétiques dynamiques, la méthode de Monte-Carlo, les essaims particulaires en dynamique, les algorithmes de colonie de fourmis en optimisation dynamique ainsi que les systèmes immunitaires en optimisation dynamique et quelques autres travaux, puis nous nous sommes intéressés plus particulièrement à l'optimisation dyna-

mique continue. Le domaine de recherche a été défini en section 2.1, ainsi que les mesures de performance qui sont propres aux environnements dynamiques. Un jeu de fonctions de test élaboré au LiSSi a été proposé en section 2.4. Nous allons maintenant décrire la plate-forme de test des métaheuristiques, élaborée dans l'équipe.

CHAPITRE 3

CONTRIBUTION À L'ÉLABORATION DE LA PLATE-FORME *oMetah* DE TEST DES MÉTAHEURISTIQUES

3.1 Introduction

Les problèmes d'optimisation apparaissent dans divers domaines, comme les problèmes d'identification, l'apprentissage supervisé dans les réseaux de neurones, les problèmes de chemin les plus courts, *etc.* Les métaheuristiques [43] sont une famille d'algorithmes d'optimisation stochastiques, souvent appliquées aux problèmes combinatoires complexes pour lesquels on ne connaît pas de méthodes directes plus efficace. Elles ont l'avantage d'être des méthodes génériques, qui n'exigent pas de réglage complexe pour chaque problème et peuvent être considérées comme des "boîtes noires". Nous rappelons que les algorithmes d'optimisation cherchent un *point* dans un *espace de recherche*, qui optimise (*i.e.*, minimise ou maximise) la *fonction objectif* (appelée aussi la fonction de coût). Les métaheuristiques peuvent être divisées en deux catégories :

1. Les algorithmes qui traitent un *point unique*, qui évoluera vers la solution voulue.
2. Les algorithmes qui traitent une *population*, *i.e.*, un ensemble fini de points, et qui calculent une nouvelle population à chaque itération.

Une observation importante est que la population de la deuxième catégorie est un échantillonnage de la fonction objectif. Bien que ces deux catégories d'algorithmes sont indissociables (un algorithme peut appartenir à deux classes, selon notre point de vue). Dans la suite, les algorithmes à population sont les seuls considérés comme étant des *métaheuristiques*.

Afin d'analyser les métaheuristiques, plusieurs formalisations ont été proposées [94], [95], [28], suivant le concept de la programmation à mémoire adaptative [136], qui vise à unifier les métaheuristiques à population, en mettant en valeur des processus communs. En s'appuyant sur notre observation précédente, nous présentons un nouveau modèle qui considère la population comme un échantillonnage de probabilité, et illustrons notre approche avec un logiciel développé pour ce nouveau modèle, *Open Metaheuristics* [37, 38, 29]. Ce dernier fournit un cadre commun à toutes les métaheuristiques, et ainsi facilite leur implantation et leur comparaison. Ce qui diffère des plates-formes générales [153] comme *HotFrame* [49], *Templar* [80] ou *HSF* [35], où les métaheuristiques sont considérées seulement comme des algorithmes sans structure spécifique, et des cadres spécifiques, tels qu'*Evolving-Objects* [81], où seulement une structure spécifique d'une métaheuristique est considérée. Nous présentons les concepts fondamentaux dans la section 3.2, puis nous présentons notre modèle de recherche par apprentissage adaptatif dans la section 3.3. La section 3.4 se focalise sur la structure du logiciel *Open Metaheuristics* avec plusieurs exemples. Et nous concluons en section 3.5.

3.2 Concepts fondamentaux

La plupart des métaheuristiques partagent des propriétés communes. La propriété principale est qu'elles manipulent un échantillonnage de la fonction objectif, en utilisant des processus communs. Le cadre de la programmation à mémoire adaptative vise à accentuer ces processus.

En principe, l'échantillonnage de probabilité devrait sélectionner les meilleures solutions qui ont une probabilité plus élevée. Cependant, dans un problème d'optimisation, le but n'est pas de trouver un échantillonnage de la fonction objectif, mais de trouver la fonction de distribution de l'optimum. Ainsi, l'échantillonnage doit se concentrer sur les espaces d'intérêt, tout en convergeant graduellement vers l'optimum, au moyen d'algorithmes dédiés. Du point de vue de l'échantillonnage, cette convergence est effectuée par une baisse progressive de la dispersion dans ces secteurs.

3.2.1 Programmation à mémoire adaptive (PMA)

La programmation à mémoire adaptative (PMA) décrite dans l'algorithme 3.1 est un cadre général aux métaheuristiques [136]. Elle souligne les notions de *mémoire*, *intensification*, et *diversification*. Dans la littérature des algorithmes évolutionnaires, ces deux dernières notions

sont souvent remplacées par les mots *exploitation* et *exploration*, qui ont une signification semblable.

Algorithme 3.1 Programmation à mémoire adaptative.

1. La mémorisation d'un ensemble de solutions, ou d'une structure de données contenant les caractéristiques des solutions produites par la recherche.
2. La construction d'une solution provisoire basée sur les données mémorisées.
3. L'amélioration de la solution par un algorithme de recherche locale.
4. La mémorisation de la nouvelle solution, ou de la structure de données associée.

Nous allons détailler brièvement chaque élément de la PMA :

– *La mémoire* est l'ensemble des informations rassemblées par l'algorithme sur la distribution de fonction objectif. Elle peut être simplement un ensemble de points, ou une structure plus complexe, comme des traces de phéromones pour les algorithmes de colonies de fourmis. La mémoire peut être définie comme étant globale (du point de vue du problème dans son ensemble) ou inter-individuelle (une solution parmi d'autres).
– *L'intensification* exploite l'information obtenue, afin d'améliorer les solutions courantes. C'est typiquement un algorithme de recherche locale (par exemple l'algorithme de Nelder & Mead [102] ou la recherche tabou).
– *La diversification* vise à rassembler de nouvelles informations, en explorant l'espace de recherche.

Les trois composants présentés ne sont pas toujours clairement distincts, et sont fortement interdépendants dans un algorithme. La méthode GRASP [114] est un exemple de métaheuristique qui correspond bien au modèle PMA.

3.2.2 Échantillonnage et PMA

Dans la majorité des métaheuristiques, l'échantillonnage de la fonction objectif est probabiliste (diversification). Idéalement, cet échantillonnage devrait être effectué par une approximation de la distribution des points, afin de localiser un centre d'intérêt, puis converger vers l'optimum (intensification).

La plupart des métaheuristiques n'ont aucune information *a priori* sur la distribution, ainsi elles apprennent *implicitement* par diversification et intensification, comme c'est le cas des algorithmes de colonies de fourmis, et des "métaheuristiques classiques". Au contraire, d'autres méthodes utilisent une approximation de la distribution, et sont appelées *explicites* (voir [6]).

Les algorithmes d'estimation de distribution (EDA) [2] sont un exemple des méthodes explicites. Nous distinguons également les méthodes *directes*, qui utilisent directement la fonction objectif, comme le recuit simulé.

3.2.3 Cadre général

On trouve plusieurs manières de structuration du domaine de l'échantillonnage de distribution dans la littérature. Par exemple, Monmarché et al. ont proposé la métaheuristique à recherche probabiliste *Probabilistic Search Metaheuristic* [94], [95] (PSM), basée sur la comparaison des algorithmes PBIL [5], [7], BSC [134], et le *ant system algorithm* [28]. Le principe général de la méthode PSM est présenté par l'algorithme 3.2. A noter la relation de cette approche avec les algorithmes à estimation de distribution. Mais l'approche PSM est limitée à l'utilisation d'un vecteur de probabilité, tout en spécifiant une règle de mise à jour pour ces vecteurs.

Algorithme 3.2 La description de la méthode *PSM*.

Initialiser le vecteur de probabilité $p_0(x)$

Jusqu'à atteindre le critère d'arrêt :

> **Construire** m individus $x_1^l, \ldots, x_m^l$ en utilisant $p_l(x)$
>
> **Évaluer** $f\left(x_1^l\right), \ldots, f\left(x_m^l\right)$
>
> **Reconstruire** le vecteur de probabilité $p_{l+1}(x)$ en prenant en considération $x_1^l, \ldots, x_m^l$ et $f\left(x_1^l\right), \ldots, f\left(x_m^l\right)$

Fin

Les EDA ont été présentés comme des algorithmes évolutionnaires, avec diversification explicite [100]. Ce sont les algorithmes les plus proches d'un cadre général. Les algorithmes *Iterated Density Evolutionary Algorithms* [19], [17], [18] (IDEA) sont une généralisation des algorithmes présentés dans l'algorithme 3.3.

IDEA utilise une diversification plus générale que PSM, et ne se limite pas au modèle du vecteur de probabilité, tout en spécifiant que la recherche de la meilleure distribution de probabilité constitue une partie complète de l'algorithme. La baisse de la diversification est effectuée en choisissant les meilleurs individus, aucune précision sur l'utilisation de différents principes d'intensification est donnée.

Algorithme 3.3 L'approche *IDEA*.

Initialiser les n points de la population P_0

Jusqu'à atteindre le critère d'arrêt :

Mémoriser les mauvais points θ

Rechercher une distribution appropriée $D_i(X)$ à partir de la population P_{i-1}

Construire une population O_i de m points selon $D_i(X)$, with $\forall O_i^j \in O_i : f\left(O_i^j\right) < f(\theta)$

Créer la population P_i à partir d'une partie de P_{i-1} et une partie de O_i

Évaluer P_i

Fin

3.3 Recherche par apprentissage adaptatif

Dans cette section, nous présentons la recherche par apprentissage adaptatif *Adaptive Learning Search* (ALS), une nouvelle plate-forme pour décrire la structure des métaheuristiques, basée sur l'approche PMA.

3.3.1 Processus principal

Au lieu de considérer seulement le procédé de mémorisation, nous proposons de considérer la phase de l'*apprentissage*. En effet, le concept de mémoire est tout à fait statique et passif ; dans une approche d'échantillonnage, il suggère que l'échantillon soit simplement stocké, et que la métaheuristique tient compte seulement de l'itération précédente, sans considérer le processus d'optimisation en entier. Nous soulignons le fait que les données apprises sont non seulement une entrée brute, mais fournissent de l'*information* sur la distribution.

De ce fait, nous proposons de considérer trois termes pour décrire les étapes principales dans une métaheuristique à population : l'apprentissage, la diversification et l'intensification, tout en suivant un échantillonnage explicite, implicite ou direct. Un algorithme ALS est ainsi organisé comme présenté dans l'algorithme 3.4.

3.3.2 Exemples

Nous présentons plusieurs métaheuristiques connues dans le cadre ALS.

Algorithme 3.4 Algorithme ALS.

Initialiser un échantillon ;

Itérer jusqu'à ce que le critère d'arrêt soit atteint :

Échantillonnage : explicite, implicite ou direct,

Apprentissage : l'algorithme extrait l'information à partir de l'échantillon,

Diversification : recherche de nouvelles solutions,

Intensification : améliore l'échantillon courant,

Remplacer l'échantillon précédent avec le nouveau.

Fin

Le recuit simulé

Le recuit simulé [83, 25] a été créé par analogie entre un processus physique (le recuit) et un problème d'optimisation. En tant que métaheuristique, il est basé sur des travaux simulant l'évolution d'un solide vers un état à énergie minimale [93], [68].

La description classique du recuit simulé le présente comme un algorithme probabiliste, où un point évolue dans l'espace de recherche. La méthode emploie l'algorithme de Metropolis, rappelé dans l'algorithme 3.5, qui utilise un processus markovien [1], [82]. Le recuit simulé, dans sa version classique ("homogène "), appelle cette méthode à chaque itération.

Algorithme 3.5 Échantillonnage par la méthode de Metropolis.

Initialiser un point de départ x_0 et une *température* T

De $i = 1$ à n :

Jusqu'à ce que x_i est accepté

Si $f(x_i) \leq f(x_{i-1})$: accepter x_i

Si $f(x_i) > f(x_{i-1})$: accepter x_i avec la probabilité $e^{-\frac{f(x_i)-f(x_{i-1})}{T}}$

Fin

Fin

Le recuit simulé peut être considéré comme un algorithme à population. En effet, l'algorithme de Metropolis emploie directement la fonction objectif en utilisant la distribution paramétrique de Boltzmann (avec un paramètre T). Par conséquent, un des paramètres essentiels est la diminution de la température, pour laquelle beaucoup de lois ont été proposées [145]. Il existe également quelques versions du recuit simulé portées davantage sur la manipulation d'une population de points [76, 157, 85, 78].

Ici, la méthode de Metropolis représente la diversification (couplée à l'apprentissage), alors que la diminution de la température contrôle le processus d'intensification. A noter que d'autres méthodes que celle de Metropolis peuvent être utilisées [30], [107].

L'algorithme 3.6 représente une synthèse du recuit simulé. L'étape d'apprentissage n'est pas présente dans les versions classiques, mais beaucoup de variantes ont essayé de lier la température à certaines caractéristiques de l'échantillonnage obtenu par la méthode de Metropolis [48], [104], [31]. En conclusion, le recuit simulé est principalement caractérisé par son échantillonnage direct de la fonction objectif.

Algorithme 3.6 Le modèle ALS pour le recuit simulé.

Échantillonnage : direct.
Apprentissage : relation entre l'ensemble des points et la température.
Diversification : échantillonnage de la fonction objectif utilisant la méthode de Metropolis.
Intensification : baisse de la température.

Algorithmes évolutionnaires

Les algorithmes évolutionnaires [47] sont inspirés du processus d'adaptation biologique des êtres vivants à leur environnement. L'analogie entre un problème d'optimisation et ce phénomène biologique a été formalisée par plusieurs approches [72], [50], [113], menant par exemple à la célèbre famille des algorithmes génétiques [58]. Le terme *population* convient parfaitement ; selon la métaphore, les populations successives s'appellent des *générations*. Une nouvelle génération est calculée en trois étapes, détaillées ci-dessous.

1. Sélection : améliore les capacités de reproduction des meilleurs individus adaptés.
2. Croisement : produit un ou deux nouveaux individus à partir de leurs deux parents, tout en recombinant leurs caractéristiques.
3. Mutation : modifie aléatoirement les caractéristiques d'un individu.

La troisième étape est clairement identifiable comme une étape de diversification, alors que la première représente l'intensification. Nous interprétons le croisement comme étant un apprentissage de l'information précédente (*i.e.* des ancêtres). Plusieurs méthodes [134], [67], [66], [8] ont été conçues pour les opérateurs de diversification, qui soulignent le processus implicite de l'échantillonnage de distribution.

Le modèle générique ALS de ce schéma est représenté dans l'algorithme 3.7.

Algorithme 3.7 Le modèle ALS pour les algorithmes évolutionnaires.

Échantillonnage : implicite.
Apprentissage : croisement.
Diversification : mutation.
Intensification : sélection.

Les algorithmes à estimation de distribution

Les algorithmes à estimation de distribution (EDA) ont été créés la première fois comme une alternative aux algorithmes évolutionnaires [100] : la différence principale est que les étapes de croisement et de mutation sont remplacées par une sélection aléatoire des individus en utilisant une estimation de distribution obtenue à partir des populations précédentes. Le processus général est représenté dans l'algorithme 3.8.

Algorithme 3.8 L'algorithme à estimation de distribution.

$D_0 \leftarrow$ Engendrer M individus aléatoirement.
$i = 0$
Tant que le critère d'arrêt n'est pas atteint :
  $i = i + 1$
  $D_{i-1}^{S_e} \leftarrow$ Choisir $N \leq M$ individus de D_{i-1} utilisant une méthode de sélection.
  $p_i(x) = p\left(x \mid D_{i-1}^{S_e}\right) \leftarrow$ Estimer la probabilité de distribution des individus choisis.
  $D_i \leftarrow$ Échantillonner M individus utilisant $p_i(x)$
Fin

La principale difficulté consiste à savoir comment estimer la distribution ; les algorithmes utilisés à cet effet sont basés sur une évaluation de la dépendance entre variables, et peuvent appartenir à trois catégories différentes :

1. Des modèles sans aucune dépendance : une distribution de probabilité est factorisée à partir des distributions indépendantes et univariantes, sur chaque dimension. Ce choix a le défaut de ne pas être réaliste en cas de problèmes d'optimisation difficiles, où la dépendance entre variables est souvent la règle.
2. Des modèles avec dépendance *bivariante* : une distribution de probabilité est factorisée à partir de distributions bivariantes. Dans ce cas, l'apprentissage de la distribution peut être étendu à la notion de *structure*.

3. Des modèles avec dépendances *multiples* : la factorisation de la distribution de probabilité est obtenue à partir de statistiques d'ordre *plus grand* que deux.

Pour les problèmes continus, le modèle de distribution est souvent basé sur une distribution normale.

Quelques variantes importantes ont été proposées, en utilisant par exemple le "*data clustering*" pour l'optimisation multimodale, ou les variantes parallèles pour des problèmes discrets (voir le [2]). Des théorèmes de convergence ont été également formulés, en particulier la modélisation par des chaînes de Markov, ou des systèmes dynamiques.

Les algorithmes EDA vus sous l'angle des ALS sont représentés dans l'algorithme 3.9.

Algorithme 3.9 Le modèle ALS pour les algorithmes à estimation de distribution.

Utilisation d'un échantillonnage *explicite*.
Apprentissage : extraction des paramètres de la distribution explicite ;
Diversification : échantillonnage de la distribution ;
Intensification : sélection.

3.3.3 Implémentation des métaheuristiques

Le concept d'ALS peut être employé comme un outil pour mettre en application plusieurs métaheuristiques dans un cadre commun. Cette approche laisse réutiliser le code commun et facilite l'exécution, en se concentrant sur les parties originales d'un algorithme. Ainsi nous avons choisi de séparer l'exécution du concept d'ALS de l'exécution des métaheuristiques elles mêmes.

Afin d'optimiser le logiciel, un langage orienté objet convient parfaitement. Quelques langages de modélisation, tels que UML [117], aident également à décrire les concepts manipulés. En outre, les patrons de conception *design patterns* fournissent des solutions génériques pour des problèmes communs dans la conception du logiciel, ils sont devenus populaires dans les années 90 [51], et sont maintenant généralement employés pour la plupart des projets mis en application avec des langages orientés objet, comme Java ou C.

Afin de mettre en application l'approche ALS, nous allons utiliser un modèle commun, à savoir le modèle *Template Method Pattern*. Cette méthode est un modèle comportemental tout à fait simple, elle consiste à définir le squelette d'un algorithme seulement, en fonction des opérations communes, tout en permettant à des sous-classes d'être définies pour des opérations particulières de l'algorithme. Elle peut être employée pour les classes de métaheuristiques (voir

la figure 3.1), qui partagent des attributs ou des méthodes de base (échantillon, sortie, *etc.*) mais elle appliquera aussi des méthodes communes (principalement la diversification, l'intensification et l'apprentissage), qui sont appelées par le processus principal. En plus, ce modèle permet de définir des opérations pour des étapes d'un algorithme spécifique à une métaheuristique donnée. L'avantage de ce modèle est qu'il force l'utilisation d'une interface commune pour toutes les métaheuristiques, ce qui facilite leur manipulation [52].

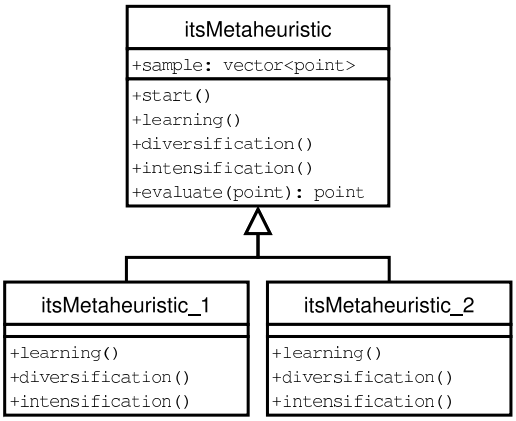


FIG. 3.1 – Diagramme UML du modèle *Template Pattern*, employé pour la mise en application d'une métaheuristique dans ALS.

La mise en application des métaheuristiques avec une approche ALS peut faciliter la compréhension et la comparaison de plusieurs algorithmes. En effet, comme les métaheuristiques sont structurées avec trois étapes principales et un échantillon, on peut comparer l'évolution de cet échantillon sur chacune de ces étapes, et non pas seulement par itération.

Un autre avantage de la mise en application de différentes métaheuristiques dans un cadre commun est de réduire les différences liées à l'exécution, ce qui facilite la comparaison de deux algorithmes.

3.4 Mise en application des métaheuristiques sur la plate-forme *Open Metaheuristics*

Nous avons mis en application (Johann Dréo, Jean-Philippe Aumasson, Walid Tfaili et Patrick Siarry), l'approche ALS dans une plate-forme appelée *Open Metaheuristics* [37] (oMetah), un projet sous licence LGPL (voir le diagramme UML complet de la plateforme oMetah en annexe, figure A.8).

3.4.1 Algorithmes, problèmes et couches de communication

Afin de séparer la partie métaheuristique de la partie problème, nous avons structuré le projet autour de trois composantes :

- métaheuristiques,
- problèmes,
- couches de communication.

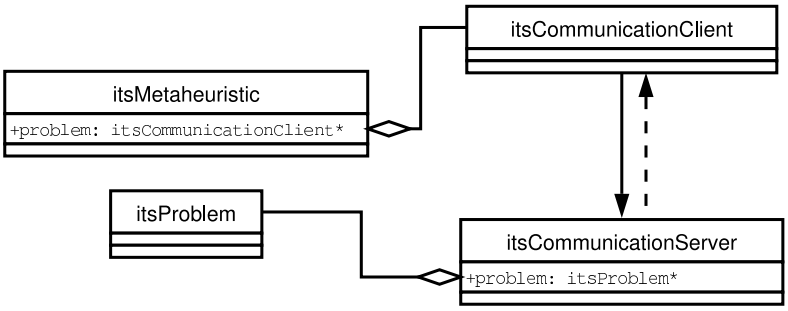


FIG. 3.2 – Diagramme UML représentant les relations entre les métaheuristiques et les problèmes dans *oMetah*.

Comme le montre la figure 3.2, les métaheuristiques et les problèmes communiquent par un module abstrait client/serveur. Ce qui permet de connecter facilement des problèmes et des algorithmes mis en application, et également des problèmes externes avec des métaheuristiques existantes. En effet, une telle approche peut aider à connecter des objets d'*oMetah* avec un autre logiciel. Par exemple, si un problème est disponible en ligne de commande comme logiciel binaire, on peut employer un protocole de transmission en utilisant un *fichier* temporaire où les résultats sont écrits, un protocole de communication peut être employé également. Une telle approche a été employée dans plusieurs plates-formes générales, mais avec des liens statiques, nous préférons le protocole client/serveur, qui est plus général et plus flexible.

Afin de manipuler des objets de différents types, nous employons une abstraction d'un ensemble d'objets (voir la figure 3.3). Grâce à cette classe et au modèle abstrait *abstract factory pattern*, il est possible de manipuler plusieurs objets partageant une interface commune. Cette classe peut être employée pour relier plusieurs métaheuristiques à plusieurs problèmes, par différents protocoles de communication.

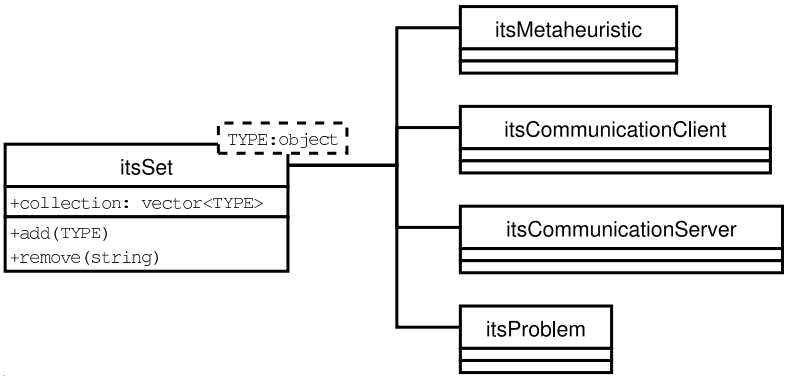


FIG. 3.3 – Diagramme UML de la classe Set.

3.4.2 La classe de base des métaheuristiques

Chaque nouvelle métaheuristique doit hériter d'une classe de base, qui représente des outils pour la construction et pour le contrôle de son interface. La fonction la plus importante dans cette classe de base est la fonction `start`, représentée dans l'algorithme 3.10, qui est employé pour déclencher le processus d'optimisation. Principalement, elle réitère les trois étapes, et produit l'échantillon, jusqu'à ce qu'un critère d'arrêt soit atteint. Des fonctions virtuelles sont fournies pour des processus additionnels, au début ou à la fin de l'optimisation.

Algorithme 3.10 Mise en application de la fonction *start* pour la classe de base métaheuristique.

```
void itsMetaheuristic : :start() {
 // une étape d'initialisation avant le démarrage de l'optimisation
   initialization();
 // tant que le critère d'arrêt n'est pas atteint
   while(!isStoppingCriteria() ) {
      learning(); // étape d'apprentissage
      outputSample();
      diversification(); // étape de diversification
      outputSample();
      intensification(); // étape d'intensification
      outputSample();
  // une itération de plus
      iterationsCurrent++;
   }
 // une étape d'arrêt, si nécessaire
   end();
}
```

3.4.3 Un exemple concret

La mise en application d'un algorithme de recherche dans *oMetah* est simple : définir les trois étapes (apprentissage, diversification et intensification) est suffisant. En effet, plusieurs algorithmes ont été mis en application :

- recherche aléatoire et exhaustive,
- la méthode de Nelder-Mead,
- les algorithmes génétiques,
- les algorithmes de colonies de fourmis,
- les algorithmes à estimation de distribution,
- le recuit simulé.

Nous allons présenter trois exemples de mises en application dans les sections suivantes.

Recherche aléatoire

Une illustration très simple de la mise en application dans *oMetah* est représentée dans l'algorithme 3.11, qui consiste en une recherche purement aléatoire, en utilisant une distribution uniforme. Cet exemple démontre la définition des trois étapes, la classe utilisée pour les points (vecteur de solution et valeur de solution) et certaines méthodes communes, telles que l'appel de la fonction objectif.

Algorithme 3.11 Mise en application d'une recherche aléatoire simple avec *oMetah*.

```
void itsRandom : :learning() { // Pas d'apprentissage. }

void itsRandom : :diversification() {
// initialise chaque point dans un hypercube
 for( unsigned int i=0; i < getSampleSize(); i++) {
  itsPoint p;
  p.setSolution(
   randomUniform(
    getProblem()->boundsMinima(),
    getProblem()->boundsMaxima()
   )
  );
  setSample[i]( evaluate(p) ); // appeler le problème
 }
}

void itsRandom : :intensification() { // Pas d'intensification. }
```

Estimation de distribution

Un autre exemple plus complexe, l'algorithme 3.12, met en application un EDA simple, en utilisant une fonction de densité de probabilité multi-normale (PDF) comme méthode d'apprentissage. Nous commentons les trois étapes ci-dessous :

1. l'étape d'apprentissage, extrait les paramètres (notamment le vecteur moyen et la matrice de variance-covariance) de la PDF multi-normale à partir de l'échantillon ;
2. l'étape de diversification, trouve un nouveau échantillon selon les paramètres ;
3. l'étape d'intensification : choisit les meilleurs points.

Algorithme 3.12 Mise en application d'un algorithme à estimation de distribution.

```
void itsEDA : :learning(){
 // le vecteur des moyennes
  this->parameterNormalMean = mean( this->getSample() );
 // la matrice de la variance-covariance
  this->parameterNormalVarCovar = varianceCovariance(
this->getSample() );
}
void itsEDA : :diversification() {
 // choisir selon une PDF multi-normale
  for( unsigned int i=0; i < getSampleSize(); i++) {
    itsPoint p;
  // choisir selon les paramètres
  vector<double> sol = randomNormalMulti( this->parameterNormalMean,
this->parameterNormalVarCovar);
  // une PDF uniforme est utilisée si on sort des bornes
    for( unsigned int i=0;
i < this->getProblem()->getDimension(); i++) {
      if( sol[i] < this->getProblem()->boundsMinima()[i]
     ||   sol[i] > this->getProblem()->boundsMaxima()[i] ) {
  sol = randomUniform( this->getProblem()->boundsMinima(),
this->getProblem()->boundsMaxima() );
      }
    }
    p.setSolution(sol);
    sample[i] = evaluate(p); // appeler le problème
  }
}
void itsEDA : :intensification(){
 // choisir les meilleurs points
 setSample( selectOnValues( getSample(), selectNumber ) );
}
```

Interface avec d'autres plates-formes

La structure de haut niveau d'*Open Metaheuristics* permet d'intégrer des algorithmes écrits dans d'autres plates-formes. Comme exemple, nous fournissons une interface à la plate-forme *Evolving-Objects* (EO) [81]. Cette plate-forme est consacrée aux algorithmes évolutionnaires, et emploie une approche basée sur des composants. L'algorithme 3.13 montre le squelette de l'interface, mis en application comme cadre général, à l'algorithme génétique de base (référencé comme SGA dans les EO). Les types d'attributs commençant par `eo` sont spécifiques à la plate-forme EO. Les deux méthodes `sampleToPop` et `popToSample` sont employées pour convertir l'échantillon d'*oMetah* en "population" de la plate-forme EO, et réciproquement.

3.4.4 Manipulation d'objets

Les objets (métaheuristiques, problèmes et couches de communication) dans *oMetah* peuvent être facilement manipulés par le modèle abstrait *abstract factory pattern*. L'*abstract factory pattern* appartient à une classe de modèles de conception dédiés à la création, et qui traitent la création d'objet dans un langage orienté objet. Il fournit un moyen facile de créer des objets partageant des propriétés communes – en fait ils vont avoir une classe père commune – mais on ne va pas spécifier explicitement quelles classes. D'une manière simple, on définira un *abstract factory*, comme instance d'une classe abstraite, qui sera employée pour créer des instances des classes dérivées, par l'intermédiaire de leurs propres *abstract factory* [52].

Par exemple, un *abstract factory* pour les problèmes permet de créer des instances de métaheuristiques spécifiques, par exemple, les ajouter à un ensemble spécifique, en employant une même interface pour toutes. On doit définir seulement une sous-classe de l'*abstract factory* métaheuristique pour chaque métaheuristique que nous programmons (voir la figure 3.4). En conséquence, nous n'avons pas l'obligation de spécifier le type d'objets voulus, il reste même inconnu, et le programmeur peut facilement le changer en modifiant la classe de création de la méthode *abstract factory*.

3.4.5 L'utilisation de la bibliothèque

L'accès aux résultats

Afin de faciliter l'étude du comportement des métaheuristiques, nous devons rassembler le plus d'informations possible. Un accès à l'état d'une métaheuristique à chaque itération est

Algorithme 3.13 Squelette d'une en-tête de l'interface de la plate-forme EO.

```
// oMetah utilise des paramètres réels
typedef eoReal<eoMinimizingFitness> EOType;
class itsEOInterface : public itsMetaheuristic {
// la structure de données d'EO
eoPop<EOType> * eoOffspring;
eoPop<EOType> * eoPopulation;
// les opérateurs d'EO et leurs paramètres
eoSelectOne<EOType> * eoSelect;  // sélection
eoQuadOp<EOType> * eoCross; // croisement
float eoCrossRate;
eoMonOp<EOType> * eoMutate;  // mutation
float eoMutateRate;
void learning() {
  sampleToPop();
 /* le croisement pour un ensemble fini de couple de points ... */
  popToSample();
}
void diversification() {
  sampleToPop();
 /* l'opérateur de mutation pour tous les points ... */
  popToSample();
}
void intensification() {
  sampleToPop();
 /* l'opérateur de remplacement ... */
 /* l'opérateur de sélection ... */
  popToSample();
  evaluate();
}
// Le constructeur est initialisé avec les opérateurs d'EO comme arguments
itsEOInterface( eoPop< EOType > * _pop,eoSelectOne<EOType> * _select,
eoQuadOp<EOType> * _cross, float _cross_rate, eoMonOp<EOType> *
_mutate, float _mutate_rate )
 : itsMetaheuristic(), eoPopulation( _pop ), eoSelect( _select ),
eoCross( _cross ), eoCrossRate( _cross_rate), eoMutate( _mutate ),
eoMutateRate( _mutate_rate )
{ }
// Convertir les individus eoPop de type EOType en individus de l'échantillon oMetah
void popToSample()  { /* ... */ }
// Convertir l'échantillon de oMetah en eoPop de type EOType
void sampleToPop()  { /* ... */ }
};
```

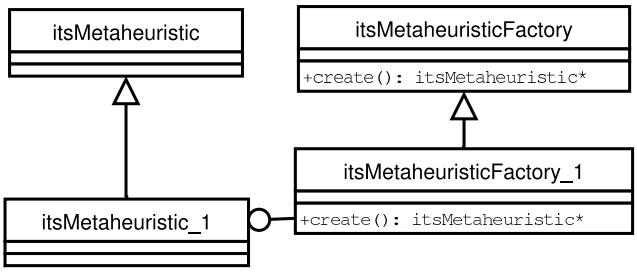


FIG. 3.4 – Le diagramme UML du modèle *abstract factory*, employé pour manipuler la mise en application des métaheuristiques.

ainsi crucial. Dans l'approche ALS, cet état est illustré par l'échantillon, modifié à chaque étape d'une itération.

Pour atteindre ce but, un format structuré de sortie est utile. Nous avons proposé un format de dossier XML (*eXtensible Markup Language*), recueillant toute les informations d'une session d'optimisation (voir la figure 3.5). L'avantage d'une telle approche est que l'on peut facilement extraire l'information à partir des dossiers. Et comme XML est extensible, on peut également ajouter une nouvelle représentation d'information. Par exemple, on peut ajouter une structure spécifique pour représenter une solution (comme un arbre ou une représentation plus complexe) ou ajouter les valeurs des paramètres des métaheuristiques à chaque itération (quand l'algorithme est dynamique, par exemple). Dans un document XML, cette addition non prévue ne dérangera pas la possibilité d'extraire d'autres informations.

Open Metaheuristics propose également un ensemble d'étiquettes pour inclure des informations sur l'algorithme et le problème employé pour des résultats obtenus. Ces informations, incluses dans l'en-tête du dossier XML, sont particulièrement utiles pour vérifier et reproduire des expériences (voir figure 3.5).

Automatisation des tests

Nous avons établi un ensemble d'outils (appelés *oMetahLab*) pour l'automatisation des tests. Ils permettent de tester les métaheuristiques, contrôler le stockage de données et analyser les résultats.

Cet ensemble d'outils est écrit dans le langage de programmation Python, et il est constitué de deux composants : le module `test` et le module `stats`. Le module de `test` vise à tester une métaheuristique donnée sur un problème donnée, plusieurs fois. Le rôle de `stats` est de lire les données XML, fournir les graphiques de sorties et les tableaux, et d'analyser les résultats dans un rapport. Un exemple de manuscrit automatisé pour la comparaison de deux

```
<optimization>
 <iteration id="0">
  <step class="start">
   <sample>
    <point><values>0.705915</values><solution>0.840188</solution><
/point>
    <point><values>0.155538</values><solution>0.394383</solution><
/point>
   </sample>
  </step>
  <evaluations>4</evaluations>
 </iteration>
 <iteration id="1">
  <step class="learning">
   <sample>
    <point><values>0.705915</values><solution>0.840188</solution><
/point>
    <point><values>0.155538</values><solution>0.394383</solution><
/point>
   </sample>
  </step>
  <step class="diversification">
   <sample>
    <point><values>0.0771588</values><solution>0.277775</solution><
/point>
   </sample>
  </step>
  <step class="intensification">
   <sample>
    <point><values>0.0348113</values><solution>0.186578</solution><
/point>
   </sample>
  </step>
  <evaluations>7</evaluations>
 </iteration>
 <optimum>
  <point><values>0.0348113</values><solution>0.186578</solution><
/point>
 </optimum>
</optimization>
```

FIG. 3.5 – Un exemple simple du format du fichier de sortie d'*oMetah* pour un problème d'optimisation. L'algorithme utilisé est un simple EDA, utilisant un échantillon de 2 points et optimisant le problème continu Sphère, dans un espace de dimension 2.

métaheuristiques est donné dans l'algorithme 3.14. Le module `stats` permet de choisir entre un ensemble de *plugins*, chacun manipule des résultats différents.

Algorithme 3.14 Exemple d'une session de test par *oMetahLab*. Ce code fait tourner 10 fois un EDA et un algorithme aléatoire, sur le problème *Rosenbrock* avec 2 variables, et fait sortir tous les graphiques disponibles dans un rapport en HTML.

```
import ometahtest
import ometahstats
path = './ometah'
# Nombre d'exécutions à effectuer
runs = 10
u = ometahtest.Test(path,'-s 10 -i 10 -e 100 -p Rosenbrock -d 2 -m
CEDA',runs)
u.start()
v = ometahtest.Test(path,'-s 10 -i 10 -e 100 -p Rosenbrock -d 2 -m
RA',runs)
v.start()
# calculer tous les graphiques et faire un rapport en format HTML
ometahstats.process(paths, 'all','html')
```

Sortie graphique

Le fait de connaitre l'état de l'échantillon permet de calculer la distribution des points à chaque itération ou étape, comme le montre la figure 3.6, ou les distributions de l'optimum, comme le montre la figure 3.7. En observant ces distributions, on peut les comparer rigoureusement par le biais de tests statistiques non paramétriques, comme le test de *Mann-Whitney-Wilcoxon*, recommandé par Taillard [135]. On peut également tracer l'optimum trouvé dans l'espace des solutions, comme le montre la figure 3.8.

Application

Open Metaheuristics a été utilisé avec succès pour résoudre un problème d'optimisation en imagerie biomédicale [39], [40] : des techniques d'optimisation pour le recalage d'images d'angiographie rétinienne (le recalage est un outil important pour résoudre beaucoup de problèmes d'analyse médicaux) ont été utilisées. Beaucoup de stratégies de minimisation classiques ont été appliquées au problème de recalage d'images, [115], [79], mais les métaheuristiques ont montré des résultats intéressants, particulièrement pour des problèmes difficiles de recalage d'images à haute résolution.

Le point central, dans cette application, était la capacité de mettre en application le problème d'une façon très simple, en utilisant le modèle *Template pattern*. En effet, la séparation entre les

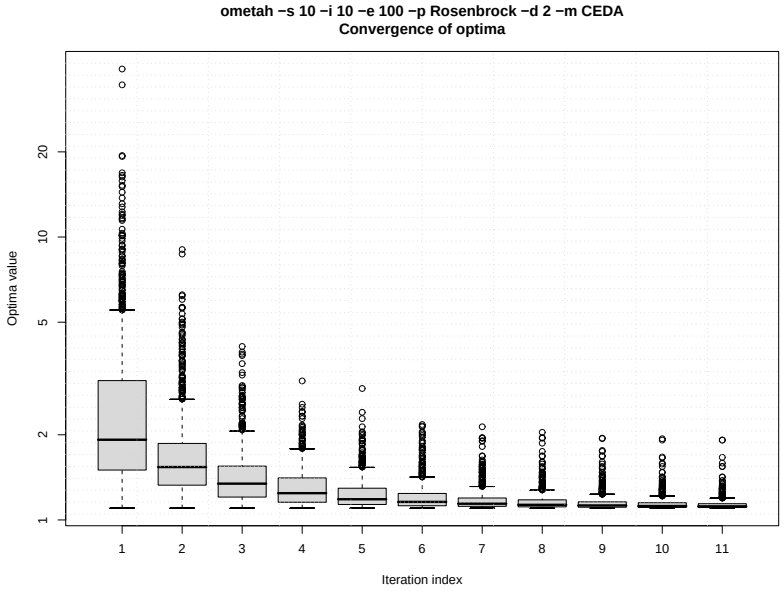


FIG. 3.6 – Distribution de l'échantillonnage sur l'espace des valeurs pour un algorithme à estimation de distribution [19] optimisant la fonction objectif *Rosenbrock* [32] à deux dimensions. Les distributions sont obtenues pour chaque itération, à l'étape d'intensification, après 50 itérations, avec un échantillon de 10 points, limité à 100 évaluations du problème.

métaheuristiques, la couche de communication et les problèmes permet de se concentrer sur la mise en application d'une classe simple, puisque tous les autres processus sont contrôlés par la plate-forme.

La partie imagerie de l'application a été faite avec la bibliothèque CImg [148]. L'algorithme 3.15 montre le squelette de l'en-tête du problème. La méthode principale est la fonction virtuelle `objectiveFunction`, qui prend un point comme argument, et renvoie le même point, avec sa valeur mise à jour. Les attributs obligatoires, fixant les intervalles du problème, sont à préciser par l'utilisateur.

Dès que le problème est mis en application, il est facile de l'optimiser avec les métaheuristiques disponibles, et de bénéficier du traitement de données fourni par la plate-forme. La figure 3.9 montre des exemples des graphiques produits par *oMetah*. Le code source est accessible en ligne, dans le module *registration* du *Open Metaheuristic project* [37], et les détails du problème sont présentés dans deux publications [39, 40].

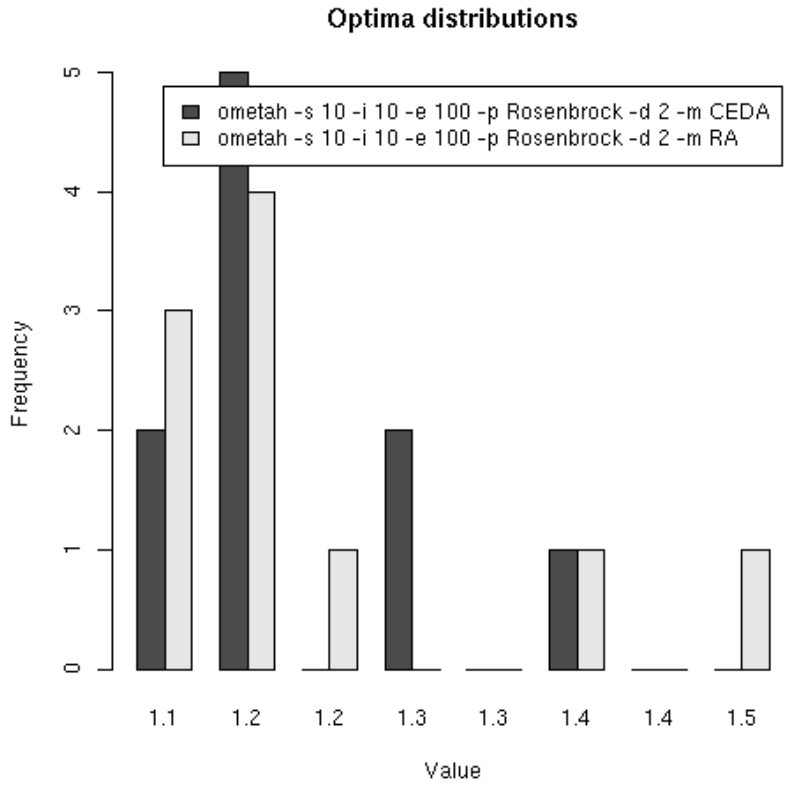


FIG. 3.7 – Distribution de l'échantillon sur l'espace des valeurs pour une évaluation de l'algorithme à estimation de distribution (CEDA) et d'une recherche aléatoire (RA) optimisant la fonction objectif *Rosenbrock* à deux dimensions. Les distributions sont obtenues pour chaque itération, à l'étape d'intensification, après 50 itérations, avec un échantillon de 10 points, limité à 100 évaluations du problème.

Algorithme 3.15 Le squelette d'un problème de recalage utilisant la bibliothèque d'imagerie CImg

```
// Cette classe hérite de la classe "base problem" d'oMetah
class itsRegistration : public itsProblem
{
public :
// La première image, de type CImg
CImg<unsigned char> img1;
// La deuxième image, à enregistrer
CImg<unsigned char> img2;
// La fonction objectif calcule les ressemblances
itsPoint objectiveFunction(itsPoint point);
// Le constructeur
itsRegistration();
}
```

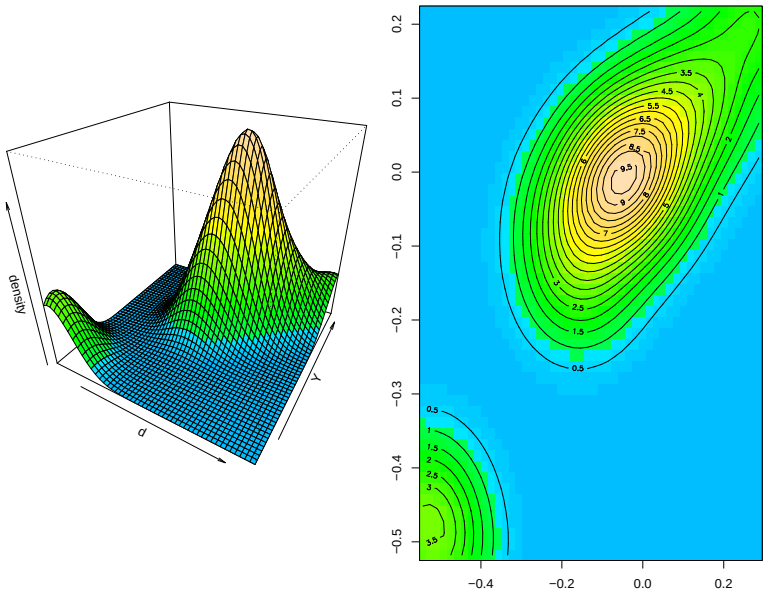


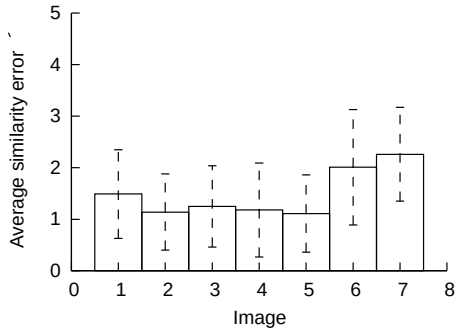
FIG. 3.8 – Distribution sur l'espace des solutions des optimums trouvés par une recherche aléatoire simple sur le problème *Rosenbrock* à deux dimensions. Les optimums sont obtenus après 50 itérations, avec un échantillon de 10 points, limité à 100 évaluations du problème.

3.5 Conclusion

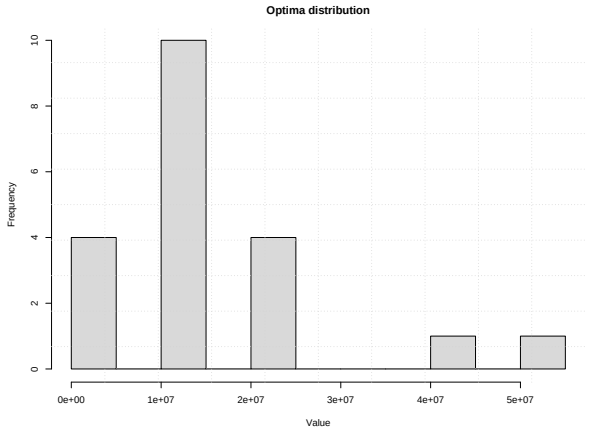
Nous avons démontré que les métaheuristiques à population peuvent être vues comme des algorithmes manipulant un échantillonnage probabiliste d'une distribution de probabilité, représentant la fonction objectif d'un problème d'optimisation. Ces algorithmes traitent l'échantillon itérativement grâce à trois processus : l'apprentissage, l'intensification et la diversification. Ces métaheuristiques sont considérées alors comme des algorithmes de recherche par apprentissage adaptatif.

L'approche d'ALS peut être employée pour mettre en application les métaheuristiques, tout en facilitant leur conception et leur analyse. En effet, mettre en application une métaheuristique avec l'approche ALS consiste à mettre en application les trois étapes principales. Et comme ceci simplifie l'exécution, les métaheuristiques peuvent être comparées et étudiées plus facilement.

L'approche ALS est mise en application dans la bibliothèque *Open Metaheuristics*. Ce projet propose un format XML pour structurer les sorties des métaheuristiques, afin de faciliter la comparaison entre les algorithmes d'optimisation. En outre, le lien entre les métaheuristiques et les problèmes peut être géré par plusieurs protocoles de communication, permettant de tester des problèmes et/ou des méthodes d'optimisation externes.



(a) La moyenne et les écarts type de l'erreur de similitude (représentant la qualité du recalage des deux images), obtenus avec un algorithme de colonies de fourmis, hybridé avec une recherche locale de Nelder-Mead, sur plusieurs images



(b) La distribution de la valeur de l'optimum pour une seule image, obtenue avec un algorithme à estimation de distribution

FIG. 3.9 – Exemple des résultats obtenus avec les tests du problème de recalage optimisé par métaheuristiques dans la plate-forme *oMetah*.

CHAPITRE 4

AMÉLIORATIONS ET TESTS INTENSIFS DE L'ALGORITHME DHCIAC

Pour trouver le plus court chemin entre la colonie et la source de nourriture, les fourmis adoptent une organisation collective particulière. En utilisant une phéromone, déposée dans l'environnement, la fourmi qui trouve de la nourriture revient à la colonie en laissant sur son chemin de retour des traces attirantes pour les autres fourmis, qui vont alors avoir tendance à suivre le chemin marqué. Les dépôts successifs de différentes fourmis auront pour effet de renforcer ce marquage. Or le chemin le plus court étant aussi le plus rapide, les fourmis le choisissant marqueront ce chemin davantage. Les fourmis étant d'autant plus attirées par un chemin que celui-ci est marqué de phéromones, la phéromone va s'amplifier sur ce chemin, au détriment des chemins plus longs. Le dépôt de phéromone est contrebalancé par l'évaporation pour empêcher une croissance infinie de la quantité de phéromone, en application des lois de rétroaction positive ou négative observées dans la nature (voir section 1.2.2).

Le premier algorithme inspiré des colonies de fourmis ACO a été proposé comme une approche multi-agent pour résoudre des problèmes d'optimisation combinatoire "difficile". Il a été appliqué ensuite à des problèmes discrets, comme les problèmes du voyageur de commerce, du routage, de communication, etc.

L'algorithme de base de *Ant Colony* est le suivant pour le problème du voyageur de commerce :

$$\text{la règle de déplacement : } P_{ij}^k(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^\alpha . (\eta_{ij})^\beta}{\sum_{l \in J_i^k} (\tau_{il}(t))^\alpha . (\eta_{ij})^\beta} & \text{si } j \in J_i^k \\ 0 & \text{si } j \notin J_i^k \end{cases}$$

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

$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \Delta\tau_{ij}(t)$
Où $\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$ m : nombre de fourmis
Avec $k = 1, \ldots, m$ fourmis, le temps $t(1 \leq t \leq t_{max})$,

$\tau_{ij}(t)$	:	l'intensité de phéromone sur une piste
η_{ij}	:	$\frac{1}{d_{ij}}$visibilité où d_{ij} : distance entre deux noeuds i et j.
J_i^k	:	liste des noeuds déjà visités par la fourmi k lorsqu'elle est sur le noeud i
P_{ij}^k	:	règle de déplacement
$L^k(t)$	:	la longueur de la tournée
Q	:	paramètre fixé
$\Delta\tau_{ij}^k(t)$	:	la quantité de phéromone sur l'ensemble du trajet de la fourmi k.

Le critère d'arrêt de l'algorithme est : la différence entre deux meilleurs points consécutifs est inférieure à 10^{-3} pendant $10.\eta$ évaluations, où η est le nombre d'agents ou fourmis. Ou bien un nombre maximum d'évaluations de la fonction objectif a été atteint.
Les paramètres qui doivent être initialisés sont les suivants :

- le nombre de fourmis η,
- ρ la persistance des spots de phéromone,
- et μ le nombre de messages initiaux.

D'autres paramètres supplémentaires sont également définis.
Les points de départ des fourmis sont distribués aléatoirement sur l'espace de recherche, $\epsilon = 10^{-14}$ le seuil au dessous duquel le spot de phéromone disparaît, ς le bruit ajouté à la position de la fourmi dans le canal piste.
Une remarque importante, que souligne Johann Dréo dans sa thèse, est la classification des algorithmes de colonies de fourmis comme algorithmes à estimation de distribution. L'échantillonnage à base d'une distribution normale correspond à la diversification. Et l'évaporation (ou bien la sélection des meilleurs individus) correspond à l'intensification.
Les métaheuristiques à population peuvent être vues comme des algorithmes manipulant un échantillonnage de la fonction objectif via des techniques de diversification et d'intensification.

Une variante introduite par Johann Dréo et al., dénommée CIAC (algorithme 4.1), consiste en l'utilisation de canaux de communication. Ce travail se focalise sur les principes de *communication*. Il propose d'ajouter aux processus stigmergiques des échanges *directs* d'informations,

en s'inspirant pour cela de la notion d'*hétérarchie*. Ainsi, une formalisation des échanges d'informations est proposée autour de la notion de canaux de communication.

Algorithme 4.1 Algorithme CIAC

1. Création des fourmis
- Distribution normale des "portées" des fourmis
- Distribuer aléatoirement les fourmis

2. Canal Piste
- Calcul du centre de gravité des spots
- Tirer au hasard une distance selon la portée de la fourmi
- Déplacement dans le sens du centre de gravité

3. Canal direct
- Lire les messages (tirage aléatoire)
- Si la valeur lue est meilleure que la valeur courante alors :
  Aller vers l'émissaire du message
- Sinon
  Envoyer le message à une fourmi
  Aller autour du point courant

4. Si des fourmis n'ont pas encore bougé alors : retour à (2)
5. Sinon : Évaporation des spots de phéromone
6. Si critère d'arrêt, alors : affichage du meilleur point trouvé
7. Sinon : retour à (2)

En effet, il existe plusieurs moyens de faire passer de l'information entre deux groupes d'individus, par exemple soit par dépôts de pistes de phéromone, soit par échanges directs. On peut définir de la sorte différents canaux de communication, représentant l'ensemble des caractéristiques du transport de l'information. Du point de vue des métaheuristiques, il y a trois caractéristiques principales :

1. Portée : le nombre d'individus mis en cause dans l'échange d'information. L'information peut, par exemple, être émise par un individu et être perçue par plusieurs autres, et inversement.
2. Mémoire : la persistance de l'information dans le système. L'information peut rester un certain temps dans le système ou n'être que transitoire.
3. Intégrité : les modifications engendrées par l'utilisation du canal de communication. L'information peut varier dans le temps, ou être biaisée lors de sa transmission.

L'information passant par un canal de communication peut être n'importe quelle information d'intérêt, comme par exemple la valeur et/ou la position d'un point de l'espace de recherche.

4.1 Algorithme *Dynamic Hybrid Continuous Interacting Ant Colony* (DHCIAC)

L'algorithme DHCIAC (figure 4.1) (*Dynamic Hybrid Continuous Interacting Ant Colony*) [36, 139, 140] est un algorithme multi agents, basé sur l'exploitation de plusieurs canaux de communication. Cet algorithme utilise la méthode des colonies de fourmis [33] pour la recherche globale, et le simplexe dynamique de *Nelder & Mead* [161] pour la recherche locale. Il a été adapté pour l'optimisation des fonctions dynamiques continues. Les fourmis disposent de deux moyens de communication. Elles peuvent déposer des spots de phéromone dans l'espace de recherche, ou utiliser un canal de communication direct (la communication se fait entre fourmis directement).

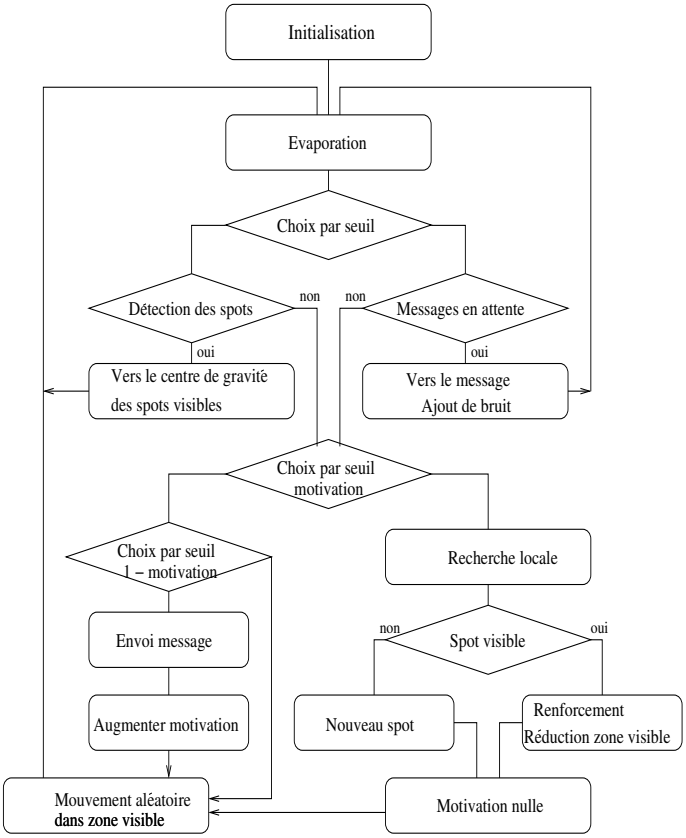


FIG. 4.1 – Principe de l'algorithme *Dynamic Hybrid Continuous Interacting Ant Colony* (DHCIAC).

1. Le canal stigmergique fait appel à des spots de phéromone, déposés sur l'espace de recherche, qui vont être plus ou moins attractifs pour les fourmis artificielles, selon leurs concentrations et leurs distances. Les caractéristiques du canal stigmergique sont donc

les suivantes : la portée est à son maximum, toutes les fourmis peuvent potentiellement prendre en compte l'information ; il y a utilisation de mémoire, puisque les spots persistent sur l'espace de recherche ; enfin l'information évolue avec le temps, puisque les spots s'évaporent. L'information portée par un spot contient implicitement la position d'un point et explicitement la valeur de l'amélioration trouvée par la fourmi ayant déposé le spot.

2. Le canal direct est implémenté sous la forme d'échange de messages entre deux individus. Une fourmi artificielle possède une pile de messages reçus et peut en envoyer à une autre fourmi. La portée de ce canal est de un, puisque seule une fourmi reçoit le message ; la mémoire est implémentée dans la pile de messages que la fourmi mémorise ; enfin l'information (ici un couple position/valeur d'un point) n'est pas altérée au cours du temps.

Chaque fourmi choisit entre ces deux canaux selon un seuil. La "motivation" est une valeur numérique qui change au cours de l'exécution. Il faut noter qu'une fourmi ne perçoit de son environnement qu'une zone appelée la zone visible. La recherche locale se fait selon le simplexe dynamique [161]. Pour plus d'informations, on pourra consulter [45].

4.2 Réglage des paramètres

Après des séries de test empiriques sur l'ensemble des fonctions de test décrites en section 2.4 et dans l'annexe A, les différents paramètres de l'algorithme ont été fixés comme suit.
L'environnement de recherche est de dimension $2D$, et l'intervalle de recherche est [-100,100],
Le nombre de fourmis a été fixé à 10, 20, 40, 80 pour chaque fonction, ce paramètre est critique lorsque le nombre de dimensions du problème augmente.
Les agents doivent choisir entre répondre à des messages d'autres fourmis ou effectuer des recherches locales par la méthode du simplexe, la probabilité de ce choix a été fixée à 0.5, avec un écart-type de 0.2.
La persistance de la phéromone ρ, si elle est choisie élevée, va piéger et attirer les fourmis vers des optimums locaux. La valeur choisie de ρ est de 0.9. La valeur minimale à partir de laquelle la phéromone va disparaître est de 0.0001.
Les déplacements des fourmis ont une valeur de 0.5. Pour le simplexe de Nelder & Mead, le nombre maximal d'itérations est de 100. Et pour DHCIAC de 50000. L'algorithme s'arrête après 10000 itérations, s'il n'y a aucune amélioration.

σ est le pourcentage de la taille de l'espace de recherche qui va être exploré, pour déterminer l'écart-type de la distribution normale des portées des fourmis. Cette valeur va déterminer la façon selon laquelle l'espace de recherche va être exploré. Une "petite" valeur entraîne une mauvaise diversification ou exploration, et une "très grande" entraîne une dispersion non fructueuse, puisque les fourmis seront dispersées dans l'espace de recherche, et la recherche sera aléatoire.

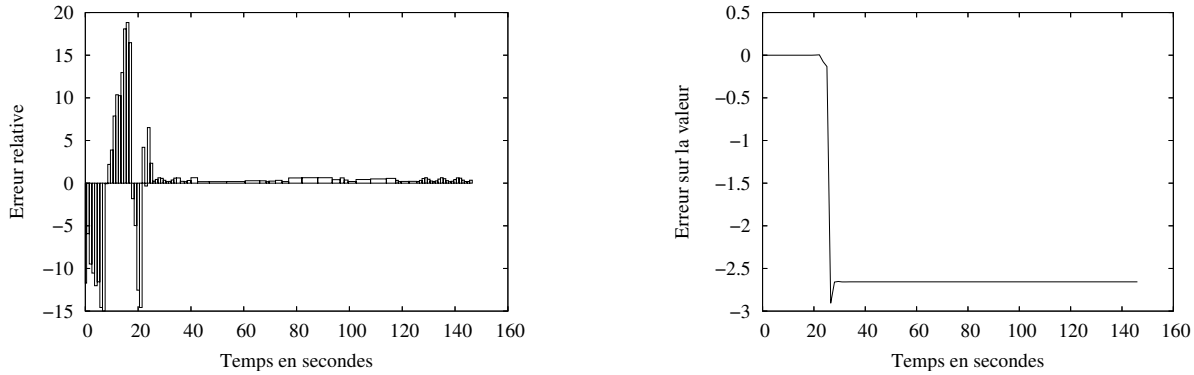
Les coefficients pour le simplexe de Nelder & Mead sont définis classiquement comme suit : coefficient de réflexion $\rho = 1$, coefficient d'expansion $\gamma = 0.5$, coefficient de contraction $\chi = 2$, coefficient de raccourcissement $\delta = 0.5$. Le simplexe que nous utilisons est le simplexe dynamique décrit dans la section 2.3.1, ce dernier n'utilise que le coefficient de réflexion.

4.3 Résultats

Pour tester le nouvel algorithme DHCIAC proposé dans la section 4.1, nous avons utilisé une batterie de tests de fonctions dynamiques définie en section 2.4, basée sur des fonctions classiques. Cette batterie de fonctions est énumérée en annexe A.

4.3.1 Problème 1 : AbPoP

La fonction objectif testée est AbPoP *All but Position Periodic* (Annexe A) basée sur la fonction statique Morrison.



(a) Erreur relative sur la position X_1

(b) Erreur relative en valeur de l'optimum.

FIG. 4.2 – Erreur relative sur AbPoP.

Sur la figure 4.2 (a), l'erreur relative à la position de l'optimum varie périodiquement dans le temps. On voit des oscillations de la valeur de l'erreur jusqu'à l'itération 25, puis l'erreur se

stabilise pour se rapprocher de la valeur nulle. Sur la figure 4.2 (b), l'erreur relative à la valeur de l'optimum diminue après 25 itérations pour se stabiliser à une valeur de -4.7.

Nombre d'agents	Erreur moyenne sur la valeur	Écart-type sur la valeur
10	4.21	3.62
20	4.15	3.69
40	4.33	3.78
80	**4.09**	**3.86**

TAB. 4.1 – Variation de la valeur en fonction du nombre d'agents (fourmis) sur AbPoP.

Nombre d'agents	Erreur moyenne sur la position	Écart-type sur la position
10	0.15, -0.67	3.69, 2.35
20	0.09, -0.71	3.76, 2.45
40	0.32, -0.61	3.85, 2.42
80	**0.04, -0.8**	3.99, 2.56

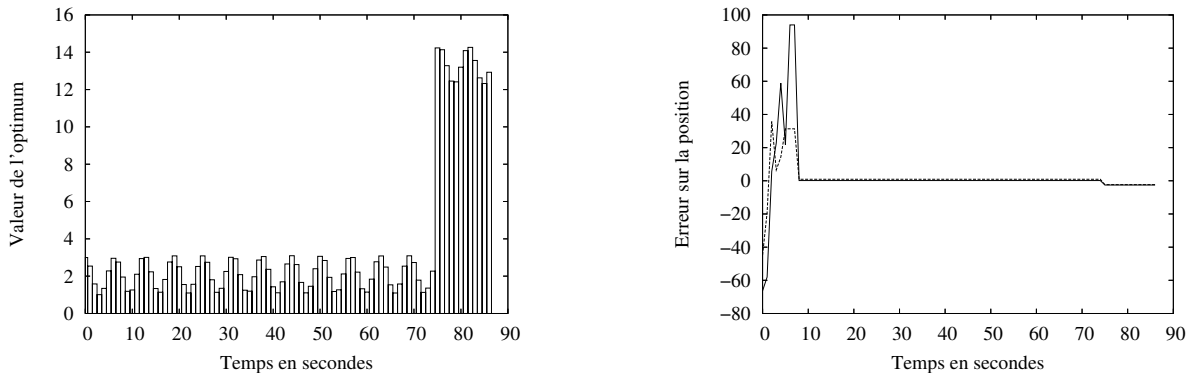
TAB. 4.2 – Variation de la position en fonction du nombre d'agents (fourmis) sur AbPoP.

Les tableaux 4.1 et 4.2 montrent les différentes valeurs de l'erreur moyenne et de l'écart-type en valeur et en position de l'optimum, pour un nombre de fourmis de 10, 20, 40, 80. La meilleure valeur de l'erreur en valeur correspond à un nombre de 80 agents avec un écart-type de 3.86. Et pour la position, la meilleure valeur correspond également à 80 agents, avec un écart-type de valeur de (3.99, 2.56).
La première observation concernant la fonction AbPoP est que DHCIAC arrive à trouver la valeur de l'optimum après un certain nombre d'itérations, par contre le suivi en valeur est moins performant. Mais DHCIAC conserve une distance fixe à la valeur de l'optimum. La meilleure performance correspond à 80 fourmis, avec des petites valeurs d'écart-type.

4.3.2 Problème 2 : AbVP

La fonction de test est AbVP *All but Value Periodic* (Annexe A) qui se base sur la fonction statique Morrison.

Sur la figure 4.3 (a), l'erreur relative à la position de l'optimum varie périodiquement dans le temps. La figure montre une oscillation de l'erreur entre les valeurs 1 et 3 jusqu'à l'itération 75, où l'erreur augmente à une valeur de 13. Sur la figure 4.3 (b), chaque courbe est l'erreur relative en position sur chaque dimension. L'erreur oscille entre les valeurs -60 et 100 jusqu'à l'itération 10, pour se stabiliser à 0.



(a) Erreur sur la valeur de l'optimum.

(b) Erreur relative en position.

FIG. 4.3 – Erreur relative sur AbVP.

Nombre d'agents	Erreur moyenne sur la valeur	Écart-type sur la valeur
10	9.99	3.36
20	9.79	3.52
40	**9.73**	**3.49**
80	9.89	3.37

TAB. 4.3 – Variation de la valeur en fonction du nombre d'agents (fourmis) sur AbVP.

Les tableaux 4.3 et 4.4 montrent les différentes valeurs de l'erreur moyenne et de l'écart-type en valeur et en position de l'optimum pour un nombre de fourmis de 10, 20, 40, 80. La meilleure valeur de l'erreur en valeur correspond à un nombre de 40 agents, avec un écart-type de 3.49. Et, pour la position, la meilleure valeur correspond également à 40 agents, avec un écart-type de valeur de (2.54, 2.62).

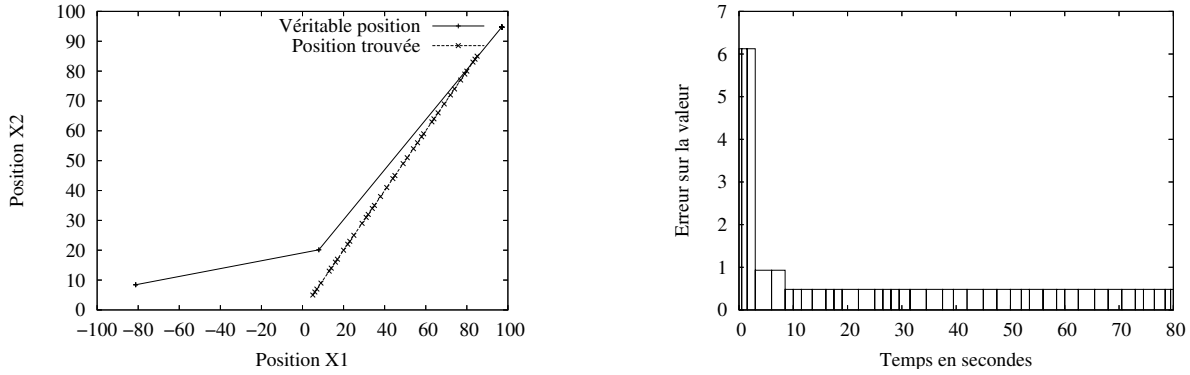
Pour la fonction de test AbVP, DHCIAC trouve la position de l'optimum après un petit nombre d'itérations et arrive à le suivre de très près, par contre, pour la valeur de l'optimum, l'erreur relative varie périodiquement avec l'optimum ; ce qui signifie que DHCIAC arrive à trouver la valeur de l'optimum, mais non pas à le suivre périodiquement. La meilleure performance correspond à 40 fourmis, avec des petites valeurs d'écart-type.

Nombre d'agents	Erreur moyenne sur la position	Écart-type sur la position
10	-0.42, -0.45	2.35, 2.44
20	-0.26, -0.29	2.55, 2.61
40	**-0.21, -0.27**	**2.54, 2.62**
80	-0.39, -0.43	2.35, 2.42

TAB. 4.4 – Variation de la position en fonction du nombre d'agents (fourmis) sur AbVP.

4.3.3 Problème 3 : APhL

La fonction de test est APhL *All Phase Linear* (Annexe A) basée sur la fonction statique Martin-Gady. La fonction APhL représente une discontinuité sur la position de l'optimum.



(a) La variation en position.

(b) Erreur sur la valeur.

FIG. 4.4 – Erreur sur APhL.

La figure 4.4 (a) montre la variation en position de l'optimum en $2D$ ainsi que la véritable position de l'optimum. La position réelle de l'optimum varie linéairement, et la position trouvée converge vers cette position. La figure 4.4 (b) montre l'erreur sur la valeur de l'optimum. (La véritable valeur de l'optimum reste constante avec le temps). L'erreur relative diminue de la valeur 6 à 1, pour atteindre une valeur 0.5 après 8 itérations.

Nombre d'agents	Erreur moyenne sur la valeur	Écart-type sur la valeur
10	4.61	52.09
20	2.53	16.51
40	**1.97**	**9.14**
80	2.041	4.14

TAB. 4.5 – Variation de la valeur en fonction du nombre d'agents (fourmis) sur APhL.

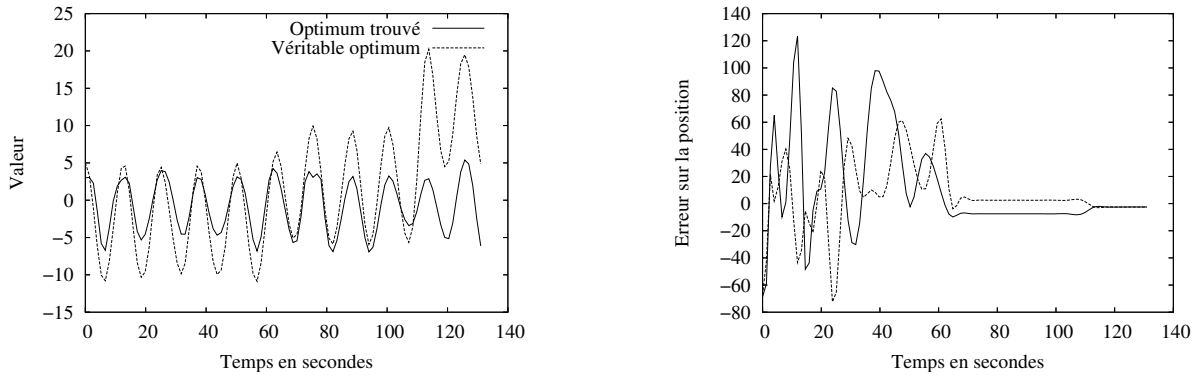
Nombre d'agents	Erreur moyenne sur la position	Écart-type sur la position
10	55.51, 24.44	57.05, 38.16
20	38.57, 8.08	53.75, 31.99
40	**31.49, 15.55**	**52.76, 31.36**
80	32.59, 10.80	56.76, 29.26

TAB. 4.6 – Variation de la position en fonction du nombre d'agents (fourmis) sur APhL.

Les tableaux 4.5 et 4.6 montrent les différentes valeurs de l'erreur moyenne et de l'écart-type en valeur et en position de l'optimum pour un nombre de fourmis de 10, 20, 40, 80. La meilleure valeur de l'erreur en valeur correspond à un nombre de 40 agents avec un écart-type de 9.14. Et, pour la position, la meilleure valeur correspond également à 40 agents, avec un écart-type de valeur de 52.76, 31.36.
Pour la fonction APhL, la convergence vers la position de l'optimum est lente. Mais les écart-types en position et en valeur sont importants, ce qui est dû à la discontinuité de la structure de la fonction APhL. La meilleure performance correspond à 40 fourmis, avec des petites valeurs d'écart-types.

4.3.4 Problème 4 : OVP

La fonction de test est OVP *Optimum Value Periodic* (Annexe A) basée sur la fonction statique Morrison.



(a) Variation de la valeur.

(b) Erreur en position.

FIG. 4.5 – Erreur sur OVP

Sur la figure 4.5 (a), on voit la variation de la véritable valeur de l'optimum au cours du temps, avec la valeur trouvée de l'optimum. La valeur de l'optimum varie périodiquement (voir tableau 4.15), l'algorithme suit la valeur de l'optimum jusqu'à l'itération 110, où la distance entre l'optimum trouvé et le véritable optimum augmente. La figure 4.5 (b) représente l'erreur en position sur chaque dimension. Cette erreur oscille entre 0 et 100, pour se stabiliser à 0 après 60 itérations.

Les tableaux 4.7 et 4.8 montrent les différentes valeurs de l'erreur moyenne et de l'écart-type en valeur et en position de l'optimum pour un nombre de fourmis de 10, 20, 40, 80. La meilleure valeur de l'erreur en valeur correspond à un nombre de 80 agents avec un écart-type

Nombre d'agents	Erreur moyenne sur la valeur	Écart-type sur la valeur
10	0.6	4.19
20	0.42	4.25
40	0.38	4.18
80	**0.37**	**4.17**

TAB. 4.7 – Erreur sur la valeur en fonction du nombre de fourmis sur OVP.

Nombre d'agents	Erreur moyenne sur la position	Écart-type sur la position
10	-3.23, -3.16	1.56, 1.64
20	-3.27, -3.15	1.72, 1.77
40	-3.29, -3.18	1.7, 1.79
80	**-3.12, -3.03**	**1.59, 1.63**

TAB. 4.8 – Erreur sur la position en fonction du nombre de fourmis sur OVP.

de 4.17. Et, pour la position, la meilleure valeur correspond également à 80 agents, avec un écart-type de valeur de (1.59, 1.63).
Pour la fonction de test OVP, DHCIAC arrive à trouver et à suivre périodiquement l'optimum en valeur et en position X_1 et X_2. Les variations brusques de l'erreur au début de la courbe sont dues au fait qu'un des optimums locaux de OVP change pour devenir un optimum global. La meilleure performance correspond à 80 fourmis.

4.3.5 Problème 5 : OPoL

La fonction de test dynamique est OPoL *Optimum Position Linear* (Annexe A) basée sur la fonction statique Easom.

La figure 4.6 représente le suivi de la position de l'optimum, qui varie linéairement avec différents nombres de fourmis (10, 20, 40, 80 fourmis). Le point de départ de l'algorithme, ainsi que la variation du vrai optimum, sont indiqués sur les courbes. La position de l'optimum s'éloigne après le lancement (ou le point de départ), de l'optimum, pour revenir et suivre linéairement l'optimum. Avec un nombre de 10 fourmis, la convergence est plus rapide, mais le suivi n'est pas constant. Avec 80 fourmis, la convergence est plus lente, mais le suivi est assuré.

Les tableaux 4.9 et 4.10 montrent les différentes valeurs de l'erreur moyenne et de l'écart-type en valeur et en position de l'optimum, pour un nombre de fourmis de 10, 20, 40, 80. La meilleure valeur de l'erreur en valeur correspond à un nombre de 10 agents avec un écart-type de 0. Et, pour la position, la meilleure valeur correspond à 80 agents, avec un écart-type de valeur de (15.54, 16.34).

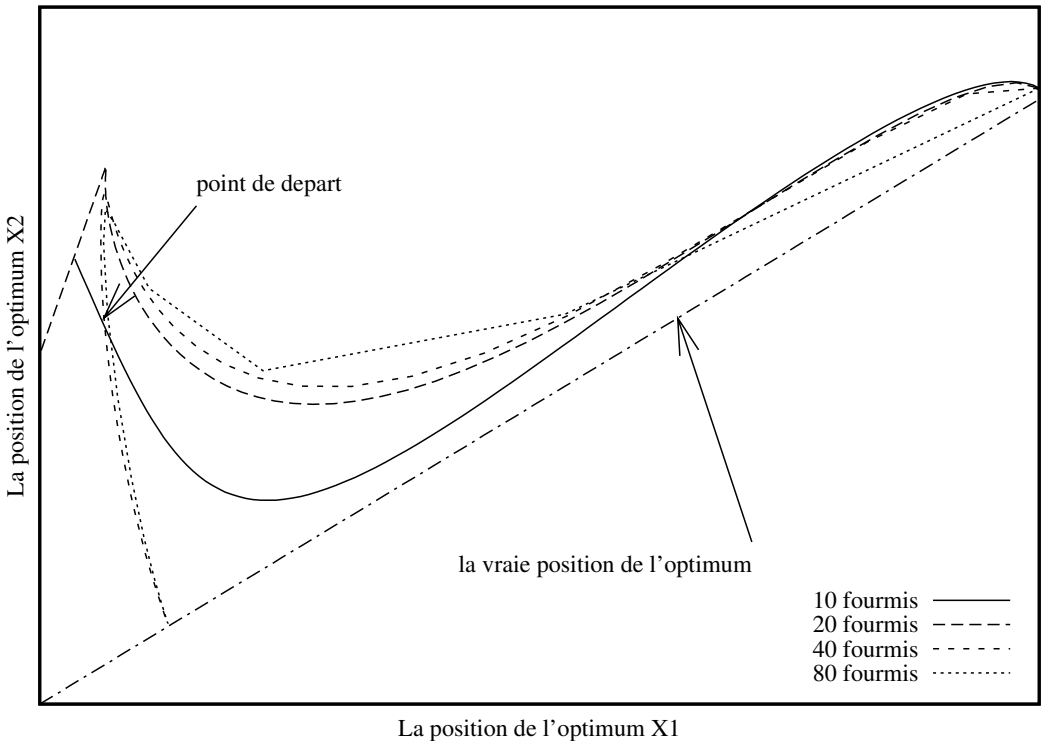


FIG. 4.6 – Suivi de la position de l'optimum sur OPoL.

Nombre d'agents	Erreur moyenne sur la valeur	Écart-type sur la valeur
10	**6.54E-005**	**0**
20	1.33E-004	0
40	2.23E-004	0.01
80	4.85E-004	0.01

TAB. 4.9 – Variation de la valeur en fonction du nombre d'agents (fourmis) sur OPoL.

Pour OPoL les résultats des tests obtenus sont satisfaisants, avec un suivi de la position de l'optimum assuré après un certain nombre d'oscillations au début de la courbe. Et la meilleure performance de DHCIAC correspond à 10 fourmis pour la valeur, et 80 fourmis pour la position.

4.3.6 Problème 6 : AVP

La fonction de test est AVP *All Value Periodic* (Annexe A) basée sur la fonction statique Shekel.

Sur la figure 4.7, le graphe montre l'erreur en valeur sur la fonction AVP. L'erreur oscille entre les valeurs -9.5 à -11.5.

Nombre d'agents	Erreur moyenne sur la position	Écart-type sur la position
10	-18, -18.79	21.34, 20.15
20	-18.86, -18.66	18.24, 18.68
40	-15.8, -15.86	16.72, 17.28
80	**-14.86, -14.99**	**15.54, 16.34**

TAB. 4.10 – Variation de la position en fonction du nombre d'agents (fourmis) sur OPoL.

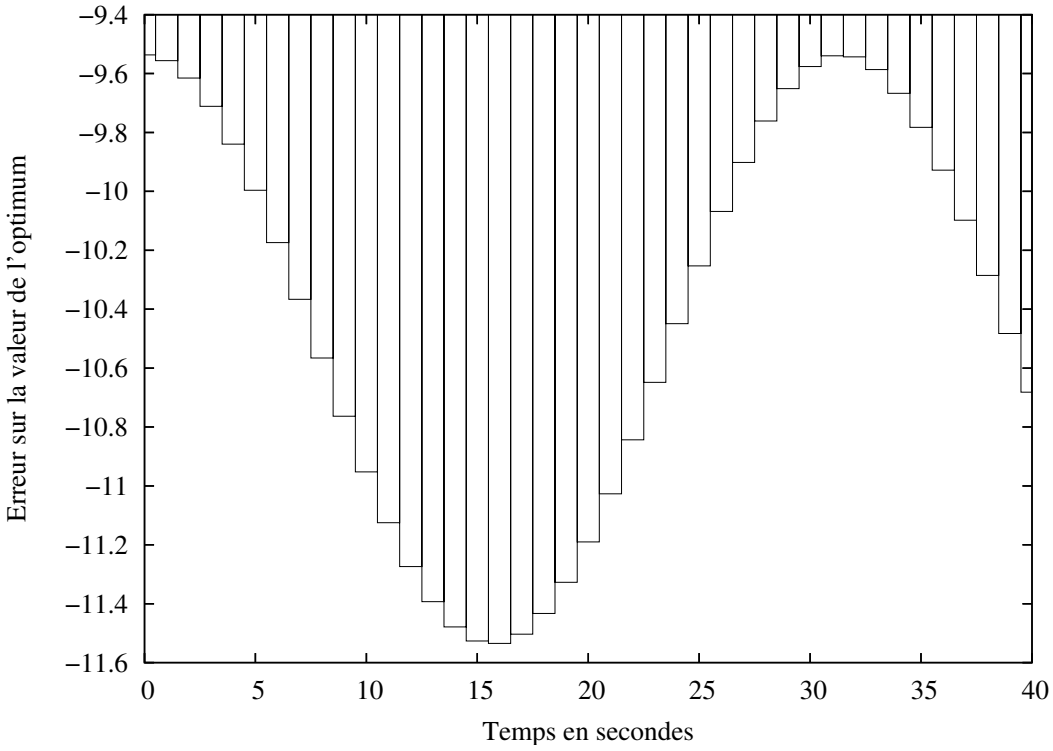


FIG. 4.7 – Erreur en valeur sur AVP.

Les tableaux 4.11 et 4.12 montrent les différentes valeurs de l'erreur moyenne et de l'écart-type en valeur et en position de l'optimum, pour un nombre de fourmis de 10, 20, 40, 80. La meilleure valeur de l'erreur en valeur correspond à un nombre de 10 agents, avec un écart-type de 0.14. Et, pour la position, la meilleure valeur correspond à 40 agents, avec un écart-type de valeur de (63.26, 56.5).

Pour la fonction de test AVP, DHCIAC n'arrive pas à suivre la valeur de l'optimum. Mais les écart-types obtenus sont faibles sur les tests effectués en valeur et en position.

Nombre d'agents	Erreur moyenne sur la valeur	Écart-type sur la valeur
10	**10.62**	**0.14**
20	10.67	0.19
40	10.68	0.18
80	10.63	0.12

TAB. 4.11 – Variation de la valeur en fonction du nombre d'agents (fourmis) sur AVP.

Nombre d'agents	Erreur moyenne sur la position	Écart-type sur la position
10	-92.95, -97.91	50.1, 40.16
20	-82.37, -90.28	61.7, 51.01
40	**-81.07, -86.65**	**63.26, 56.5**
80	-88.34, -95.4	56.42, 45.33

TAB. 4.12 – Variation de la position en fonction du nombre d'agents (fourmis) sur AVP.

4.3.7 Problème 7 : APoL

La fonction de test est APoL *All Position Linear* (Annexe A) basée sur B2, la position de l'optimum varie linéairement dans le temps.

La figure 4.8 montre le suivi en $3D$ de la position de l'optimum. La valeur trouvée s'éloigne, puis se rapproche de l'optimum à des intervalles réguliers. Et la distance qui la sépare de l'optimum diminue de plus en plus.

Nombre d'agents	Erreur moyenne sur la valeur	Écart-type sur la valeur
10	**2.42**	**2.05**
20	2.49	2.08
40	2.54	2.07
80	2.45	2.08

TAB. 4.13 – Variation de la valeur en fonction du nombre d'agents (fourmis) sur APoL.

Les tableaux 4.13 et 4.14 montrent les différentes valeurs de l'erreur moyenne et de l'écart-type en valeur et en position de l'optimum, pour un nombre de fourmis de 10, 20, 40, 80. La meilleure valeur de l'erreur en valeur correspond à un nombre de 10 agents, avec un écart-type de 2.05. Et, pour la position, la meilleure valeur correspond également à 10 agents, avec un écart-type de valeur de (2.96, 2.88).
Les résultats procurés par DHCIAC sur la fonction de test ApoL sont satisfaisants, comme le montre la figure 4.8, avec des petites valeurs d'écart-types en valeur et en position.

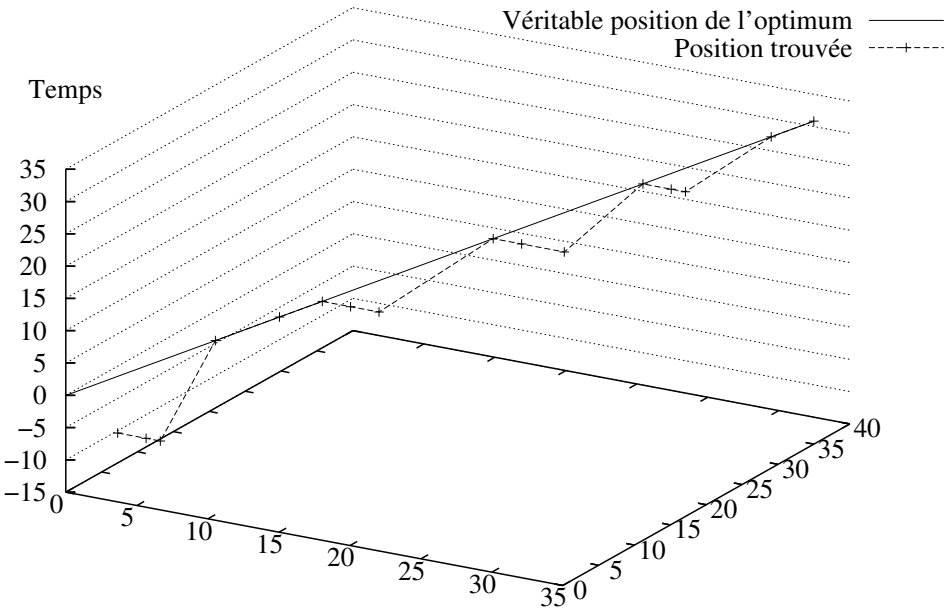


FIG. 4.8 – Suivi de la position de l'optimum sur APoL.

Nombre d'agents	Erreur moyenne sur la position	Écart-type sur la position
10	**-5.8, -5.9**	**2.96, 2.88**
20	-5.92, -5.99	3.05, 2.93
40	-5.94, -6.08	2.98, 2.95
80	-5.81, -5.91	2.89, 2.84

TAB. 4.14 – Variation de la position en fonction du nombre d'agents (fourmis) sur APoL.

4.3.8 Taux de réussite

Le tableau 4.15 montre l'ensemble des fonctions de test avec les fonctions qui définissent la variation de la valeur et de la position de l'optimum global, la fonction du temps associée, ainsi que le taux de réussite. Chaque taux représente la valeur de réussite de l'algorithme sur 100 lancements. La réussite signifie que l'optimum trouvé converge vers le véritable optimum, et le suit de près. L'algorithme montre une meilleure performance sur la méthode APoL, où la position de l'optimum varie linéairement ; avec un taux de réussite de 83 %, et AbPoP où la fonction objectif change, mais pas la position de l'optimum ; avec un taux de réussite de 68 %, et une mauvaise performance pour la fonction APhL où la position de l'optimum présente une discontinuité (voir APhL dans l'annexe A), avec un taux de réussite de 13%.

Fonction dynamique	**Position**	**Valeur**	**T(t)**	**Taux de réussite**
AbPoP	6+T(t),6	2	$\cos(t)$	51
AbVP	6,6	2+T(t)	$\cos(t)$	68
OVP	2.5,-2.5 ou -2.5,2.5	min(1+T(t),3)	$8.\cos(0.5t)$	53
APhL	5+T(t),5+T(t)	0	t	13
ADL	1,..,1	0	t	19
AVP	-	T(t)-10.53641	$\cos(0.2t)$	44
APoL	T(t),T(t)	0	t	82

TAB. 4.15 – Taux de réussite.

4.4 Comparaison

Malgré l'intérêt croissant pour l'optimisation dans le cas dynamique, aucune synthèse sur la comparaison de différents algorithmes n'a été faite dans la littérature. En outre, la plupart des résultats trouvés dans la littérature concernent des problèmes discrets (nous rappelons que nos recherches portent sur le domaine continu). Nous avons comparé notre algorithme avec la méthode de Monte-Carlo décrite dans la section 2.3.3, et le simplexe dynamique décrit dans la section 2.3.1, sur l'ensemble des fonctions de test. Les comparaisons sont effectuées sur la valeur moyenne de l'erreur en valeur et en position. Une autre comparaison a été faite avec une variante de la méthode d'essaims particulaires (voir section 2.3.4), sur une version modifiée de la fonction de test AbPoP.

	Erreur moyenne sur la valeur	Écart-type sur la valeur
DHCIAC	4.09	3.86
Monte-Carlo	4.38	4.97
Simplexe	9.52	57.39

TAB. 4.16 – Comparaison sur AbPoP en valeur.

	Erreur moyenne sur la position	Écart-type sur la position
DHCIAC	0.04, -0.8	3.99, 2.56
Monte-Carlo	32.51, -2.13	40.41, 38.18
Simplexe	-4.82, 1.14	58.45, 30.51

TAB. 4.17 – Comparaison sur AbPoP en position.

Dans les tableaux 4.16 et 4.17, les trois méthodes DHCIAC, Monte-Carlo et le simplexe dynamique sont comparées, sur la fonction de test AbPoP. Le nombre maximal d'itérations est

de 5000 pour les trois algorithmes, et le nombre de fourmis utilisées pour DHCIAC est 80. DHCIAC a une performance supérieure aux deux autres méthodes. L'erreur en valeur est de 4.09 (3.86) et en position 0.04, -0.8 (3.99, 2.56).

	Erreur moyenne sur la valeur	Écart-type sur la valeur
DHCIAC	9.73	3.49
Monte-Carlo	25.35	31.29
Simplexe	19.23	57.35

TAB. 4.18 – Comparaison sur AbVP en valeur.

	Erreur moyenne sur la position	Écart-type sur la position
DHCIAC	-0.21, -0.27	2.54, 2.62
Monte-Carlo	-14.67, 0.0048	33.41, 30.26
Simplexe	-4, -2.32	58.65, 0.32

TAB. 4.19 – Comparaison sur AbVP en position.

Dans les tableaux 4.18 et 4.19, les trois méthodes DHCIAC, Monte-Carlo et le simplexe dynamique sont comparées, sur la fonction de test AbVP. Le nombre maximal d'itérations est de 5000 pour les trois algorithmes, et le nombre de fourmis pour DHCIAC est 40. DHCIAC a une performance supérieure aux deux autres méthodes. L'erreur en valeur est de 9.73 (3.49) et en position -0.21, -0.27 (2.54, 2.62).

	Erreur moyenne sur la valeur	Écart-type sur la valeur
DHCIAC	0.38	4.18
Monte-Carlo	6.18	6.70
simplexe	9.09	57.38

TAB. 4.20 – Comparaison sur OVP en valeur.

Dans les tableaux 4.20 et 4.21, les trois méthodes DHCIAC, Monte-Carlo et le simplexe dynamique sont comparées, sur la fonction de test OVP. Le nombre maximal d'itérations est de 5000 sur les trois algorithmes, et le nombre de fourmis pour DHCIAC est 80. DHCIAC a une performance supérieure aux deux autres méthodes. L'erreur en valeur est de 0.38 (4.18) et en position -3.29, -3.18 (1.7, 1.79).

Dans les tableaux 4.22 et 4.23, les trois méthodes DHCIAC, Monte-Carlo et le simplexe dynamique sont comparées, sur la fonction de test APhL. Le nombre maximal d'itérations est de 5000 sur les trois algorithmes, et le nombre de fourmis de DHCIAC est 40. DHCIAC a

	Erreur moyenne sur la position	Écart-type sur la position
DHCIAC	-3.29, -3.18	1.7, 1.79
Monte-Carlo	7.55, -3.72	31.59, 27.07
simplexe	-4.48, 58.4	7.5, 5

TAB. 4.21 – Comparaison sur OVP en position.

	Erreur moyenne sur la valeur	Écart-type sur la valeur
DHCIAC	1.97	9.14
Monte-Carlo	31.66	53.35
Simplexe	2.4	13.98

TAB. 4.22 – Comparaison sur APhL en valeur.

une performance supérieure aux deux autres méthodes en valeur. L'erreur en valeur est de 1.97 (9.17). Et le simplexe dynamique a une meilleure erreur en position -1.16, -9.98 (98.41, 13.14).

Dans les tableaux 4.24 et 4.25, les trois méthodes DHCIAC, Monte-Carlo et le simplexe dynamique sont comparées, sur la fonction de test AVP. Le nombre maximal d'itérations est de 5000 sur les trois algorithmes, et le nombre de fourmis de DHCIAC est 40. DHCIAC a une performance supérieure aux deux autres méthodes en valeur. L'erreur en valeur est de 10.62 (0.14). Et la méthode Monte-Carlo a une meilleure erreur en position 35.84, 22.88, mais avec un écart-type supérieur aux deux autres méthodes. DHCIAC a le plus petit écart-type en position.

Dans les tableaux 4.26 et 4.27, les trois méthodes DHCIAC, Monte-Carlo et le simplexe dynamique sont comparées, sur la fonction de test OPoL. Le nombre maximal d'itérations est de 5000 sur les trois algorithmes, et le nombre de fourmis de DHCIAC est 80. DHCIAC a une performance meilleure que les deux autres méthodes en valeur. L'erreur en valeur est de 6.54E-005 (0). Et la méthode Monte-Carlo a une meilleure erreur en position 1.03, 0.82 (0.81, 0.72).

Dans les tableaux 4.28 et 4.29 les trois méthodes DHCIAC, Monte-Carlo et le simplexe dynamique sont comparées, sur la fonction de test APoL. Le nombre maximal d'itérations est de 5000 sur les trois algorithmes, et le nombre de fourmis de DHCIAC est 10. DHCIAC a

	Erreur moyenne sur la position	Écart-type sur la position
DHCIAC	31.49, 15.55	52.76, 31.36
Monte-Carlo	40.08, -8.19	53.38, 16.73
Simplexe	-1.16, 9.98	98.41, 13.14

TAB. 4.23 – Comparaison sur APhL en position.

	Erreur moyenne sur la valeur	Écart-type sur la valeur
DHCIAC	10.62	0.14
Monte-Carlo	11.38	0.74
Simplexe	11.9	13.77

TAB. 4.24 – Comparaison sur AVP en valeur.

	Erreur moyenne sur la position	Écart-type sur la position
DHCIAC	-92.95, -97.91	50.1, 40.16
Monte-Carlo	35.84, 22.88	71.38, 75.89
Simplexe	264.87,1941.78	94.57, 13.45

TAB. 4.25 – Comparaison sur AVP en position

une performance meilleure que les deux autres méthodes en valeur. L'erreur en valeur est de 2.42(2.05). L'erreur en position est de -5.8, -5.9 (2.96, 2.88).

Une autre comparaison a été faite avec un algorithme d'essaims particulaires (section 2.3.4). La fonction de test sur laquelle nous avons comparé les performances est basée sur la fonction statique Morrison, avec la fonction du temps suivante :

$$Y_i = A.Y_{(i-1)}.(1 - Y_{i-1})$$

Pour effectuer la comparaison, nous avons modifié la fonction de test AbPoP, la nouvelle fonction est définie de la façon suivante (voir [98]) :

$$f(x, y) = max_{i,N}[H_i - R_i * \sqrt{(X - X_i)^2 + (Y - Y_i)^2}]$$

Avec Y_i la fonction du temps, $N = 3$, $Hbase = 3$, $Hrange = 3$, $Rbase = 2$, $Rrange = 5$, $A = 1.2$.

$$H_i \in [Hbase, Hbase + Hrange]$$

$$R_i \in [Rbase, Rbase + Rrange]$$

	Erreur moyenne sur la valeur	Écart-type sur la valeur
DHCIAC	6.54E-005	0
Monte-Carlo	410.18	1451.05
Simplexe	8.74	57.54

TAB. 4.26 – Comparaison sur OPoL en valeur.

	Erreur moyenne sur la position	Écart-type sur la position
DHCIAC	-18, -18.79	21.34, 20.15
Monte-Carlo	1.03, 0.82	0.81, 0.72
Simplexe	-4.65, 6.9	58.4, 0.5

TAB. 4.27 – Comparaison sur OPoL en position.

	Erreur moyenne sur la valeur	Écart-type sur la valeur
DHCIAC	2.42	2.05
Monte-Carlo	30.40	87.46
Simplexe	8.33	75.65

TAB. 4.28 – Comparaison sur APoL en valeur.

L'environnement de test est en $2D$ borné sur [-1.0,1.0], le nombre d'itérations maximal des deux algorithmes est 500 itérations, et les résultats obtenus sont la moyenne sur 50 lancements. Les tests sont effectués avec un nombre d'agents (particules et fourmis) de 30, 60, 90, 120, 150. Les résultats des tests montrent une variation quasi-linéaire de l'erreur moyenne et de l'écart-type des deux méthodes en fonction du nombre d'agents, la méthode d'essaims particulaires a une meilleure performance que DHCIAC sur la version modifiée de la fonction AbPoP.

4.5 Conclusion

L'algorithme DHCIAC a été testé sur un ensemble de fonctions de test. Chaque test effectué est la moyenne de 100 lancements de l'algorithme. A chaque lancement, l'initialisation du générateur de nombres aléatoires est différente, afin d'éviter les problèmes liés au choix de la population initiale. La performance a été mesurée par le calcul de la distance euclidienne moyenne entre l'optimum trouvé et le véritable optimum.
La première observation faite est que l'algorithme arrive à suivre la position de l'optimum à une distance presque constante, pour des fonctions de test périodiques. Comme c'est le cas pour la fonction AbPoP, où l'écart à l'optimum est presque nul. Et lorsqu'on s'éloigne de l'optimum,

	Erreur moyenne sur la position	Écart-type sur la position
DHCIAC	-5.8, -5.9	2.96, 2.88
Monte-Carlo	-25.27, -25.22	18.61, 18.47
Simplexe	-6.22, 1057.89	33.28, 1865.78

TAB. 4.29 – Comparaison sur APoL en position.

N. d'agents	Err. moy. et Écart-type (PSO)	Err. moy. et Écart-type (DHCIAC)
30	0.1(0.13)	0.17(0.32)
60	0.07(0.11)	0.14(0.29)
90	0.06(0.09)	0.12(0.22)
120	0.05(0.08)	0.1(0.13)
150	0.04(0.07)	0.06(0.09)

TAB. 4.30 – Comparaison avec PSO.

le suivi reste valable, comme c'est le cas pour AbVP (voir figure 4.3 (a) page 75) et pour OVP (voir figure 4.5 (a)). Cela est dû au fait que la distance entre les optimums est constante, et que le simplexe dynamique, hybridé avec la méthode de colonie de fourmis, converge relativement vite vers ces nouveaux optimums. Les résultats pour la valeur sont meilleurs pour le suivi que pour la position ; par contre, en matière de détection, c'est l'inverse.
Pour les fonctions où la variation est linéaire, la détection et le suivi de l'optimum sont satisfaisants, en valeur et en position (voir figures 4.4, 4.6 et 4.8). Pour la fonction APhL l'erreur moyenne est petite, par contre l'écart-type est grand, ce qui signifie une dispersion autour de l'optimum. Cela est dû à la discontinuité sur la position de l'optimum et à l'absence d'un bassin d'attraction autour de l'optimum.

Sur le taux de réussite de l'algorithme (voir tableau 4.15) les meilleures performances concernent les fonctions avec variation de la valeur et de la position, et les plus mauvaises concernent la fonction avec variation de phase. En fait, il suffit que l'initialisation aléatoire se fasse sur une surface plane pour que le simplexe dynamique se comporte d'une façon chaotique, les réflexions successives qu'utilise le simplexe dynamique ne mènent pas à de meilleures valeurs.

Deuxièmement, une étude sur l'influence de nombre de fourmis sur la performance de l'algorithme montre que la variation de ce nombre a peu d'influence, à partir d'une certaine limite.

Dans la littérature, très peu de résultats existent sur des environnements dynamiques continus. Pour confronter l'algorithme aux méthodes existantes, nous avons effectué les mêmes tests sur le simplexe dynamique, et la méthode de Monte-Carlo. Notre algorithme a une meilleure performance sur la totalité des fonctions de test par rapport à ces deux méthodes. On note que même une méthode aléatoire comme celle de Monte-Carlo peut avoir des résultats satisfaisants. En fait, la méthode de Monte-Carlo ne se réinitialise pas, et garde en mémoire les meilleures valeurs, donc l'erreur va forcément converger, et au pire rester constante. Cette méthode n'utilise en quelque sorte que la diversification.

Une comparaison a été faite avec une variante des essaims particulaires utilisant la technique de sous-groupes répartis sur l'espace de recherche. L'intervalle de recherche est [-1,1]. Les résultats sont assez proches des valeurs présentées dans la littérature. La meilleure performance de la variante des essaims particulaires est due à l'utilisation explicite de la diversification et à la répartition des particules localement sur les différents optimums.

Les résultats de test sont prometteurs et prouvent que les méthodes de colonies de fourmis sont bien adaptées aux environnements dynamiques. L'algorithme a une meilleure performance que la méthode de Monte-Carlo et le simplexe dynamique, et des performances assez comparables à celles de la troisième méthode (une variante des essaims particulaires).

Une approche qui peut être interessante pour l'amélioration de la performance de DHCIAC, est l'utilisation explicite de la diversification dans les colonies de fourmis, pour augmenter les chances de trouver l'optimum lorsqu'il change.

CHAPITRE 5

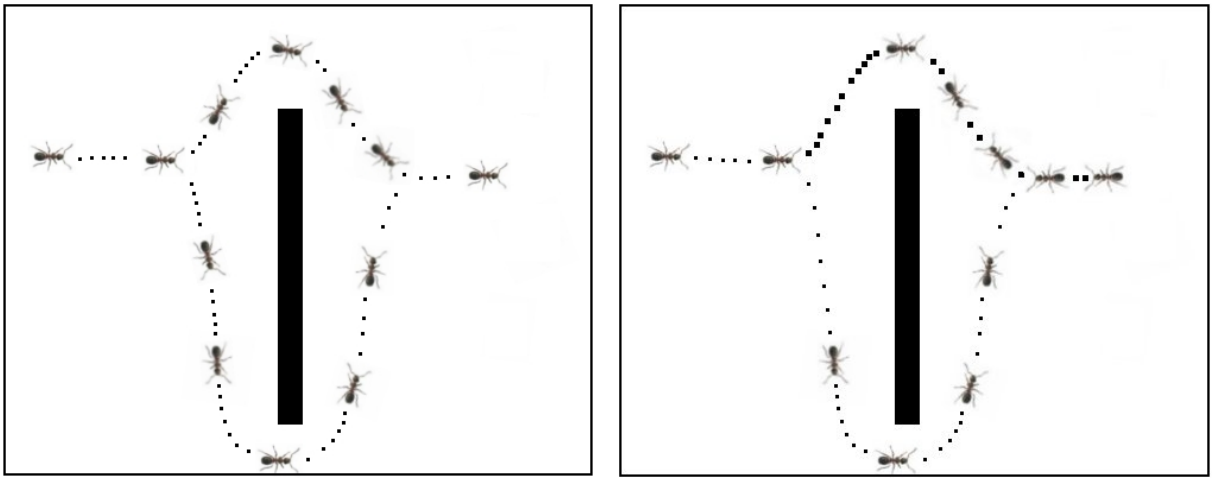
ÉLABORATION D'UN ALGORITHME DE COLONIE DE FOURMIS AVEC "CHARGE ÉLECTROSTATIQUE" POUR L'OPTIMISATION CONTINUE DYNAMIQUE

5.1 Introduction

Quelques algorithmes de colonie de fourmis (voir figure 5.1) pour l'optimisation continue dynamique ont été décrits dans la littérature. Par exemple, dans [44, 139] l'algorithme DH-CIAC (*Dynamic Hybrid Continuous Interacting Ant Colony*), est présenté comme un algorithme multi-agent, basé sur l'exploitation de deux canaux de communication. Cependant, beaucoup de recherche reste à faire pour obtenir une méthode générale. Nous présentons dans ce chapitre une nouvelle méthode d'optimisation continue dynamique, basée sur la version classique du *Ant Colony Optimization* (ACO). Une charge électrostatique répulsive/attractive est attribuée à chaque fourmi, afin de maintenir une certaine diversification dans l'espace de recherche. Pour adapter l'ACO au cas continu, nous avons utilisé des distributions de probabilité continues.

5.2 Le dilemme exploration/exploitation

Les problèmes d'optimisation dynamiques sont des problèmes où la fonction objectif change au cours du temps. L'algorithme doit dans ce cas s'adapter pour trouver les optimums de la fonction qui change. Deux approches naïves pour traiter ce genre de problèmes consistent, pour l'une, à redémarrer l'algorithme quand il y a un changement dans l'environnement. Or, si les



(a) Au début de la recherche, la probabilité que les fourmis prennent le chemin le plus court ou le plus long est de 50%.

(b) Les fourmis qui arrivent par le chemin le plus court vont retourner plus rapidement. Pour cette raison, la probabilité de prendre le chemin le plus court est plus élevée.

FIG. 5.1 – Un exemple qui montre comment une colonie de fourmis trouve le chemin le plus court. Dans notre cas, nous avons seulement deux chemins avec différentes longueurs, entre le nid et une source de nourriture.

changements sont faibles et que les optimums ne changent pas beaucoup, cette approche n'est pas optimale et présente une perte de temps importante. L'autre approche naïve consiste à ne rien faire et à continuer l'exécution de l'algorithme. Dans ce cas, le manque de diversité de la population, qui a déjà convergé vers un optimum, ne permettra pas à celle-ci d'explorer suffisamment l'espace pour trouver les nouveaux optimums.

Un problème classique en optimisation est celui du compromis entre exploitation et exploration de l'espace de recherche. En fait, il faut pouvoir équilibrer entre profiter immédiatement des connaissances acquises et les approfondir pour améliorer le gain : (1) cela implique une convergence prématurée, on risque d'être piégé dans des optimums locaux et (2) une exploration exhaustive, qui est généralement coûteuse en temps de calcul. L'utilisation des agents, à qui nous attribuons des charges électrostatiques positives/négatives (dont les valeurs varient au cours de l'exécution), donne à notre méthode une bonne capacité d'exploration, ainsi qu'une bonne capacité d'exploitation. La capacité d'équilibrer entre exploration et exploitation de cette méthode est due au phénomène de répulsion / attraction entre fourmis. En développant une zone d'influence autour d'elles pour défendre leur optimum, les fourmis induisent une répulsion, qui disperse les autres fourmis dans l'espace de recherche. Nous allons maintenant décrire la nouvelle méthode.

5.3 *Charged ANt colony for Dynamic Optimization* (CANDO) : les fourmis chargées pour l'optimisation continue dynamique

Dans les problèmes en variables discrètes, le nombre de composants du problème est fini, donc les valeurs de phéromone peuvent être attribuées directement aux composants de la solution. Une solution déterminée à un problème discret existe, même si la recherche peut être coûteuse en temps de calcul. Mais, pour des problèmes continus, les valeurs de phéromone ne peuvent pas être attribuées directement aux composants de la solution. L'approche que nous utilisons est une méthode de diversification (voir [14]), elle consiste à attribuer à chaque fourmi une charge électrostatique, $a_i = \sum_{l=1;l\neq i}^{k} a_{il}$

$$a_{il} = \begin{cases} \dfrac{Q_i Q_l}{|x_i - x_l|^3}(x_i - x_l) & \text{si} \quad r_c \leq |x_i - x_l| \leq r_p \\ \dfrac{Q_i Q_l}{{r_c}^2 |x_i - x_l|}(x_i - x_l) & \text{si} \quad |x_i - x_l| < r_c \\ 0 & \text{si} \quad r_p < |x_i - x_l| \end{cases}$$

où Q_i et Q_l sont les charges initiales des fourmis i et l respectivement, $|x_i - x_l|$ est la distance Euclidienne, r_p et r_c sont les rayons de "perception" et de "noyau" respectivement (voir figure 5.2). Le rayon de "perception" est attribué à chaque fourmi, alors que le rayon de "noyau" est le rayon de perception de la meilleure fourmi trouvée dans l'itération courante. Les valeurs des charges peuvent être positives ou négatives. Nos fourmis artificielles sont dispersées dans l'espace de recherche (voir figure 5.3), les valeurs des charges changent pendant l'exécution de l'algorithme, en fonction de la qualité de la meilleure solution trouvée.

L'adaptation au domaine continu que nous employons a été présentée par Krzysztof Socha dans [128]. L'algorithme itère sur une population de fourmis finie et constante de k individus, où chaque fourmi représente une solution ou une variable X^i, avec $i = \{1, > ..., n\}$, pour un problème de dimension n (voir figure 5.4). L'ensemble des fourmis peut être présenté comme suit

On désigne par k le nombre de fourmis et par n la dimension du problème. Au début, les individus de la population sont initialisés avec des valeurs aléatoires. Ce qui correspond à

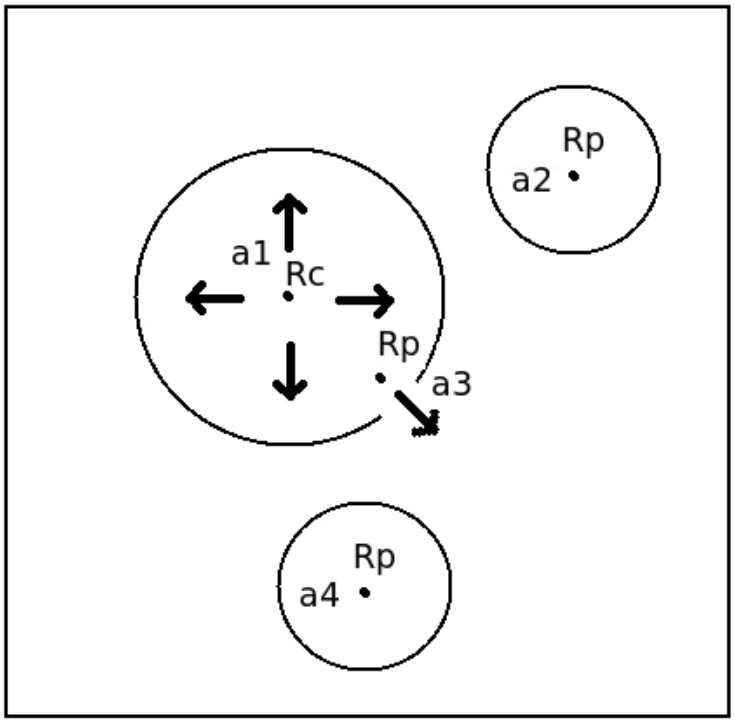
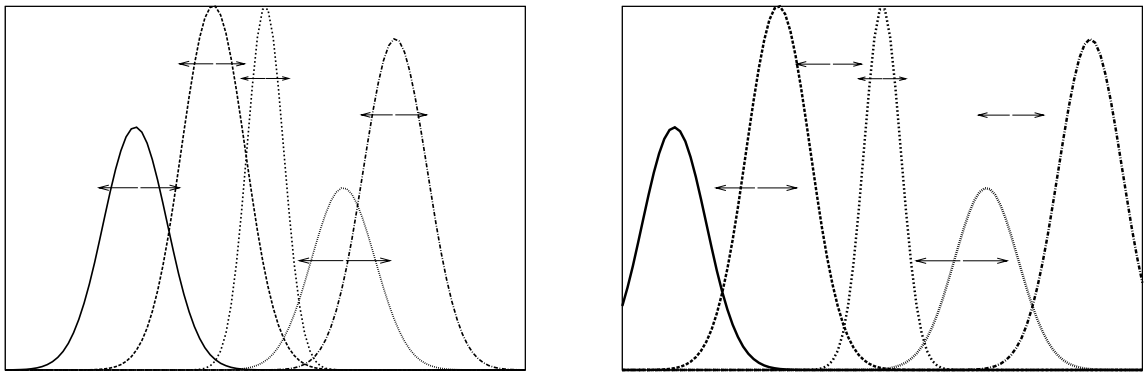


FIG. 5.2 – Une représentation de la répulsion des fourmis du "noyau" de rayon R_c de la fourmi ayant la meilleure valeur. Chaque fourmi a un rayon de "perception" R_p.



(a) (b)

FIG. 5.3 – Une charge électrostatique est attribuée à chaque fourmi ; en pratique nous attribuons une distribution gaussienne. Ce qui résulte en un changement de la figure 5.3(a) en la figure 5.3(b).

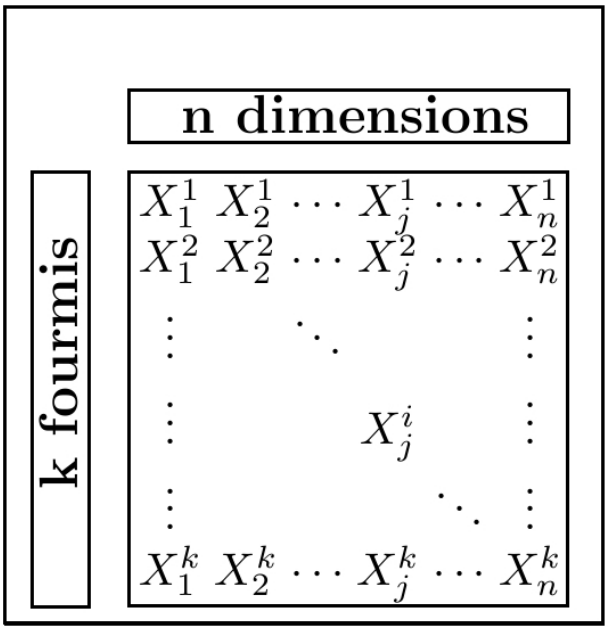


FIG. 5.4 – Chaque fourmi construit une solution complète, qui comporte n dimensions.

l'étape d'initialisation de phéromone utilisée pour les problèmes discrets. À chaque itération, les nouvelles solutions trouvées sont ajoutées à la population et un nombre égal des plus mauvaises est enlevé ; ceci correspond, à la mise à jour de phéromone (dépôt/évaporation) dans le cas discret classique. L'objectif final est de biaiser la recherche vers les meilleures solutions. La construction d'une solution est faite par composants, comme dans l'algorithme original ACO. Pour une dimension simple (par exemple la dimension j), une fourmi i choisit seulement une valeur (la $j^{ème}$ valeur du vecteur $< X_1^i, X_2^i, \ldots, X_j^i, \ldots, X_n^i >$). Chaque fourmi a une valeur de charge, et une distribution gaussienne. Le choix d'une seule fourmi sur une seule dimension est fait comme suit : une fourmi choisit une distribution gaussienne dans son rayon de perception, suivant la probabilité :

$$p_j^i = \frac{w_j^i}{\sum_{l=1}^{k} w_l^i}$$

Une fois une distribution gaussienne choisie, la fourmi i produit une valeur aléatoire, selon la distribution :

$$G(X_j^i) = \frac{1}{\sigma_j \sqrt{2\pi}} \cdot e^{-\frac{(x-\mu_j)^2}{2.\sigma_j^2}}$$

où μ et σ sont les vecteurs de la moyenne et de l'écart-type respectivement. Pour un problème de dimension n, la construction de la solution est faite par dimension, ce qui signifie que chaque composant de la solution représente exactement une dimension. Tant que l'algorithme procède sur une dimension, les autres dimensions sont laissées à part.

L'algorithme final [141, 142, 143] est le suivant (Algorithme 5.1) :

Algorithme 5.1 Algorithme de fourmis chargées pour le cas continu dynamique CANDO

Tant que critère d'arrêt non atteint **faire**
  De 1 à m fourmis faire
    Calculer la valeur a_i de la i ème fourmi
    Déplacement des fourmis selon les charges des voisins
    $s^0 = \phi$
    Répéter sur les n dimensions
      Choisir une seule distribution $G(X_j^i)$, parmi les voisins selon p_j^i
      Choisir aléatoirement la valeur X_j^i, en utilisant la $G(X_j^i)$ choisie
      $s^i = s^i \bigcup \{X_j^i\}$
    Fin
    Mise à jour des charges des fourmis, selon la meilleure solution trouvée
    Mise à jour des poids (phéromones), selon la meilleure solution trouvée
  Fin
  Choisir la meilleure solution, parmi les m fourmis
  Choisir la meilleure solution, parmi les solutions anciennes et courantes
Fin tant que

5.4 Résultats et discussion

Nous avons testé notre algorithme CANDO sur les fonctions de tests définies précédemment en utilisant la plate-forme de test *Open Metaheuristics* (voir [38] et [42] pour plus d'information).

5.4.1 Problème 1 : AbPoP

Sur la figure 5.5 (a), l'erreur relative à la valeur de l'optimum diminue après 75 itérations pour se stabiliser à une valeur de -7.5. Sur la figure 5.5 (b), l'erreur relative à la position de l'optimum varie périodiquement dans le temps. On voit des oscillations de la valeur de l'erreur jusqu'à l'itération 85, puis l'erreur se stabilise pour se rapprocher de la valeur nulle.

La première observation concernant la fonction AbPoP est que CANDO arrive à trouver la valeur de l'optimum après un certain nombre d'itérations, par contre le suivi en valeur est moins performant. Mais CANDO conserve une distance constante à la valeur de l'optimum.

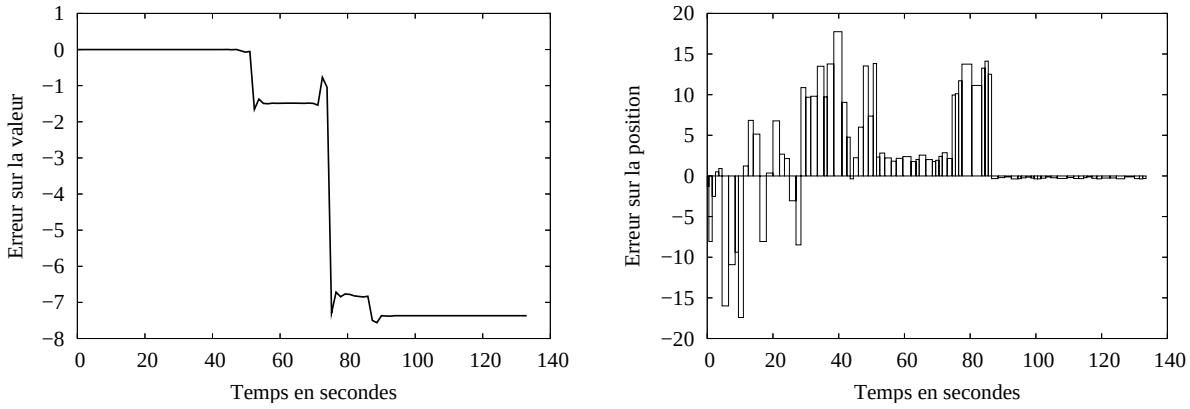


FIG. 5.5 – L'erreur sur la valeur et la position pour la fonction dynamique, AbPoP *All but Position Periodic*, basée sur la fonction statique *Morrison*.

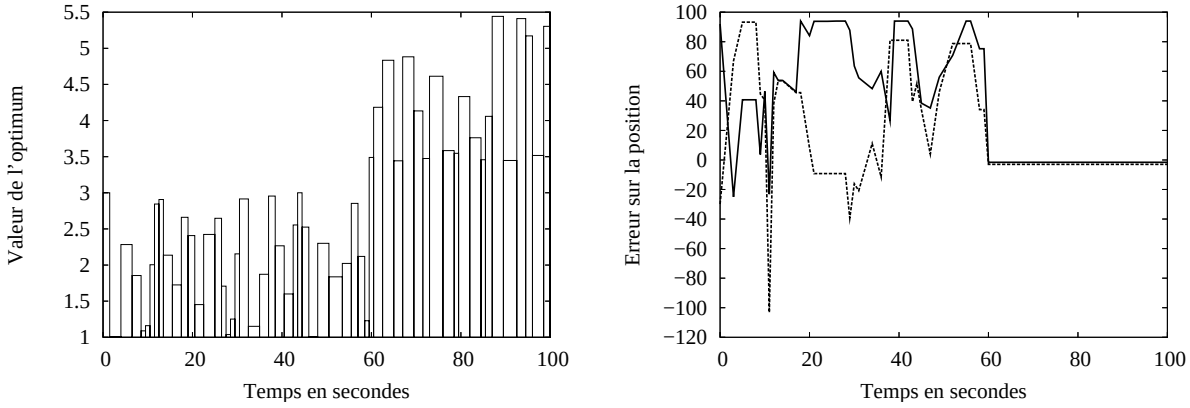


FIG. 5.6 – L'erreur sur la valeur et la position pour la fonction de test dynamique, AbVP *All but Value Periodic*, basée sur la fonction statique *Martin-Gady*.

5.4.2 Problème 2 : AbVP

Sur la figure 5.6 (a), l'erreur relative à la position de l'optimum varie périodiquement dans le temps. La figure montre une oscillation de l'erreur entre les valeurs 0 et 3.5 jusqu'à l'itération 60, où l'erreur augmente à une valeur de 5.5. Sur la figure 5.6 (b), chaque courbe représente l'erreur relative en position sur chaque dimension. L'erreur oscille entre les valeurs -100 et 90 jusqu'à l'itération 60, pour se stabiliser à 0.

Pour la fonction de test AbVP, CANDO trouve la position de l'optimum après un certain nombre d'itérations et arrive à le suivre de très près ; par contre, pour la valeur de l'optimum, l'erreur relative varie périodiquement avec l'optimum ; ce qui signifie que CANDO arrive à trouver la valeur de l'optimum, mais non pas à le suivre périodiquement.

5.4.3 Problème 3 : OVP

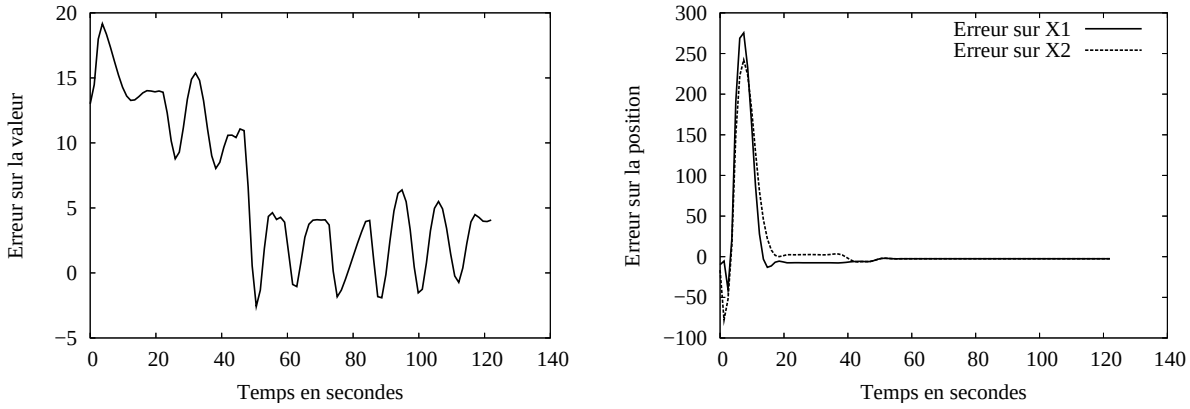


FIG. 5.7 – L'erreur sur la valeur et la position pour la fonction dynamique, OVP *Optimum Value Periodic*, basée sur la fonction statique *Morrison*.

Sur la figure 5.7 (a), on voit la variation de la distance à la valeur trouvée de l'optimum. La valeur de l'optimum varie périodiquement, l'algorithme s'approche de plus en plus de cette valeur jusqu'à l'itération 45, où l'algorithme commence à osciller entre 5 et 2.5 autour de cette valeur. La figure 5.7 (b) représente l'erreur en X_1 et X_2 de la position trouvée de l'optimum sur chaque dimension. Cette erreur oscille entre 75 et 275, pour se stabiliser à 0 après 15 itérations.

Pour la fonction de test OVP, CANDO arrive à trouver et à suivre périodiquement l'optimum en valeur et en position X_1 et X_2. Les variations brusques de l'erreur au début de la courbe sont dues au fait qu'un des optimums locaux de OVP change pour devenir un optimum global.

5.4.4 Comparaison

Les tableaux 5.1, 5.2 et 5.3 représentent une comparaison entre CANDO, eACO et DHCIAC, des différentes valeurs de l'erreur moyenne et de l'écart-type en valeur et en position X_1 et X_2 de l'optimum respectivement, pour huit fonctions de test dynamiques. Les meilleures valeurs sont représentées en gras. En valeur et en position de l'optimum, CANDO a de meilleures performances que eACO et DHCIAC pour les fonctions APhL, OVP et AVP, tandis que DHCIAC est plus performant que les deux autres méthodes pour les fonctions AbPoP, AbVP, OPoL et APoL.

	CANDO	**eACO**	**DHCIAC**
AbPoP	4,27(4,02)	4,34(4,15)	**4,09(3,86)**
AbVP	11,54(5,2)	12,03(5,45)	**9,73(3,49)**
APhL	**0,29(4,07)**	0,39(4,1)	0,38(4,18)
OVP	**1,78(8,09)**	1,84(9,11)	1,97(9,14)
OPoL	12,3(0,37)	13,61(0,42)	**10,62(0,14)**
AVP	**4,77E-005(0)**	5,43E-005(0)	6,54E-005(0)
APoL	2,87(2,24)	2,91(2,4)	**2,42(2,05)**

TAB. 5.1 – Comparaison de l'erreur relative et de l'écart-type de la valeur pour CANDO, eACO et DHCIAC.

	CANDO	**eACO**	**DHCIAC**
AbPoP	0,043(4,29)	0,06(4,50)	**0,04(3,99)**
AbVP	-0,29(2,98)	-0,37(3,01)	**-0,21(2,54)**
APhL	**-3,04(1,56)**	-3,37(1,9)	-3,29(1,7)
OVP	**24,01(43,14)**	28,8(47,9)	31,49(52,76)
OPoL	-98,01(55,43)	-98,9(56,2)	**-92,95(50,1)**
AVP	**-17(19,6)**	-17,39(19,7)	-18(21,34)
APoL	-7,02(3,13)	-7,19(3,25)	**-5,8(2,96)**

TAB. 5.2 – Comparaison de l'erreur relative et de l'écart-type de la position X_1 pour CANDO, eACO et DHCIAC.

L'ajout des charges aux fourmis permet une amélioration de la méthode sur la totalité des fonctions dynamiques. Les tests ont été refaits pour l'algorithme *extended Ant Colony Optimization* (eACO), pour permettre une comparaison avec notre approche. En effet eACO était testé seulement sur des fonctions statiques dans la littérature. L'approche multi-agent utilisée a permis une amélioration des performances de la méthode de colonies de fourmis par rapport à l'approche classique. Cette amélioration est due principalement à la diversification continue dans le temps dans l'espace de recherche. Les charges évitent l'encombrement autour des solutions trouvées, ce qui empêche une convergence prématurée ; cela est nécessaire dans les deux

	CANDO	**eACO**	**DHCIAC**
AbPoP	-0,91(2,74)	-0,97(2,81)	**-0,8(2,56)**
AbVP	-0,36(2,69)	-0,39(2,64)	**-0,27(2,62)**
APhL	**-3,02(1,51)**	-3,24(1,63)	-3,18(1,79)
OVP	**13,7(28,4)**	14,02(30,05)	15,55(31,36)
OPoL	-98,1(43,06)	-98,9(45)	**-97,91(40,16)**
AVP	**-16,5(18,3)**	-18,84(21,3)	-18,79(20,15)
APoL	-6,7(3,05)	-7,04(3,27)	**-5,9(2,88)**

TAB. 5.3 – Comparaison de l'erreur relative et de l'écart-type de la position X_2 pour CANDO, eACO et DHCIAC.

cas statique et dynamique. CANDO a de meilleures performances que DHCIAC sur les fonctions qui représentent plusieurs optimums ; en fait, plusieurs groupes de fourmis entourent les différents optimums, ce qui permet un meilleur suivi du changement de l'environnement. Pour DHCIAC, la rapidité de la convergence de la recherche locale (simplexe de Nelder Mead) permet de meilleures performances sur les fonctions avec un seul optimum. Nous rappelons que CANDO est une méthode de colonies de fourmis qui utilise la construction de solution d'une façon itérative, et n'utilise pas la communication entre fourmis, contrairement à DHCIAC.

5.5 Conclusion

Nous avons présenté dans ce chapitre un nouvel algorithme de colonies de fourmis *Charged ANt colony for Dynamic Optimization* (CANDO) dédié à l'optimisation continue dynamique. Très peu de travaux ont été effectués pour adapter les algorithmes de colonies de fourmis aux problèmes dynamiques continus, la plupart d'entre eux traitent des problèmes à variables discrètes. Pour traiter des problèmes continus dynamiques, nous avons choisi des fourmis simples et réactives, la solution finale étant obtenue par un travail collectif seulement. Le nouvel algorithme consiste à attribuer à chaque fourmi une charge électrostatique répulsive/attractive, qui varie selon la qualité de la solution trouvée. L'adaptation de l'algorithme de colonie de fourmis aux problèmes continus est faite en remplaçant la distribution de probabilité discrète par une distribution continue. Cette approche est intéressante, parce qu'elle est très proche de l'ACO original, qui modélise la façon dont les fourmis réelles trouvent la solution optimale dans la nature.

Les résultats de test sont prometteurs et prouvent que notre nouvelle méthode CANDO est bien adaptée aux environnements dynamiques. L'algorithme a de meilleures performances que

DHCIAC sur les fonctions qui présentent plusieurs optimums, et permet une amélioration de la méthode sur la totalité des huit fonctions dynamiques par rapport à la méthode eACO.

Conclusion générale et perspectives

Dans cette thèse, nous avons abordé le sujet de l'optimisation continue dynamique. Après avoir exposé les méthodes d'optimisation statiques existant dans la littérature, nous avons décrit quelques méthodes d'optimisation dynamique.

Dans un premier temps, nous avons contribué à l'élaboration d'une nouvelle plate-forme de test des métaheuristiques, qui montre que les métaheuristiques à population peuvent être vues comme des algorithmes manipulant un échantillonnage d'une distribution de probabilité, représentant la fonction objectif d'un problème d'optimisation. Dans cette plate-forme, ces algorithmes traitent l'échantillon itérativement, grâce à trois processus : l'apprentissage, l'intensification et la diversification. Cette nouvelle approche, appelée ALS, peut être employée pour mettre en application les métaheuristiques, tout en facilitant leur conception, leur analyse et leur comparaison. L'approche ALS est mise en application dans la bibliothèque *Open Metaheuristics*. Le lien entre les métaheuristiques et les problèmes peut être géré par plusieurs protocoles de communication, permettant de tester des problèmes et/ou des méthodes d'optimisation externes.

Ensuite, l'algorithme DHCIAC a été testé sur un ensemble de fonctions de test. Nous avons constaté que l'algorithme arrive à suivre la position de l'optimum à une distance presque constante, pour des fonctions de test périodiques. Et lorsqu'on s'éloigne de l'optimum, le suivi reste valable. Cela est dû au fait que la distance entre les optimums est constante, et que le simplexe dynamique hybridé avec la méthode de colonie de fourmis converge relativement vite vers ces nouveaux optimums. Pour les fonctions où la variation est linéaire, la détection et le suivi de l'optimum sont satisfaisants en valeur et en position. L'algorithme a des taux de réussite élevés pour les fonctions avec variation de la valeur et de la position, et les plus mauvais résultats concernent la fonction avec variation de phase. En fait, il suffit que l'initialisation aléatoire se fasse sur une surface plane pour que le simplexe dynamique se comporte d'une façon chaotique,

les réflexions successives qu'utilise le simplexe dynamique ne mènent pas à de meilleures valeurs. L'étude de l'influence du nombre de fourmis sur la performance de l'algorithme montre que la variation de ce nombre a peu d'influence, à partir d'une certaine limite. L'algorithme a de meilleures performances que la méthode de Monte-Carlo et le simplexe dynamique, et des performances assez comparables à celles de la troisième méthode (une variante des essaims particulaires).

Nous avons introduit un nouvel algorithme de colonies de fourmis *Charged ANt colony for Dynamic Optimisation* CANDO, dédié à l'optimisation continue dynamique. En fait, très peu de travaux ont été décrits dans la littérature, concernant l'adaptation des algorithmes de colonies de fourmis aux problèmes dynamiques continus, la plupart d'entre eux traitent des problèmes à variables discrètes. Les travaux existants, comme DHCIAC, utilisent des canaux de communication directe entre fourmis, et ne construisent pas la solution globale d'une façon itérative. Pour notre nouvelle méthode CANDO, nous avons choisi des fourmis simples et réactives, la solution finale étant obtenue par un travail collectif seulement. Le nouvel algorithme consiste à attribuer à chaque fourmi une charge électrostatique répulsive/attractive, qui varie selon la qualité de la solution trouvée. Les résultats de test et de comparaisons sont prometteurs et prouvent que les méthodes de colonies de fourmis, et plus particulièrement CANDO, sont bien adaptées aux environnements dynamiques.

Plusieurs pistes sont engagées pour prolonger notre travail de thèse.

Nous envisageons en premier lieu d'améliorer les performances de notre algorithme CANDO, en mettant en place une hybridation avec un algorithme de recherche locale, comme la méthode de Nelder & Mead. Nous souhaitons également expérimenter une hybridation avec d'autres métaheuristiques, en particulier un algorithme d'optimisation par essaim particulaire et un algorithme à estimation de distribution, ces deux approches étant développées dans le cadre d'une autre thèse en cours au laboratoire.

Plusieurs applications de CANDO à des problèmes de traitement d'images biomédicales sont en projet, pour le recalage ou la segmentation d'images. Notre méthode sera utilisée dans la recherche des meilleurs paramètres des transformations géométriques requises pour le recalage.

Nous envisageons aussi de tester notre méthode d'optimisation en segmentation, pour identifier au mieux les contours dans des images médicales.

Enfin, une application de l'optimisation dynamique est en projet avec une équipe du LiSSi, qui s'intéresse à la segmentation d'images cardiaques en mouvement (segmentation de type $3D + t$).

ANNEXE A

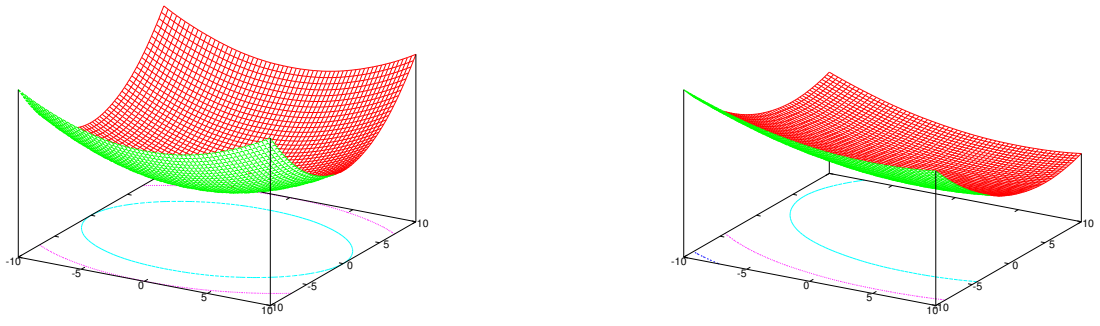
FONCTIONS DE TEST POUR LES PROBLÈMES D'OPTIMISATION DYNAMIQUE

APoL

APoL *All Position Linear*, basée sur la fonction statique B2. La fonction du temps est $T(t) = t$

$$f(\vec{x}, T(t)) = (x_1 - T(t))^2 + 2 \cdot (x_2 - T(t))^2 + 0.7$$

$$-0.3 \cdot \cos(3\pi - (x_1 - T(t))) - 0.4 \cdot \cos(4.\pi - (x_2 - T(t)))$$



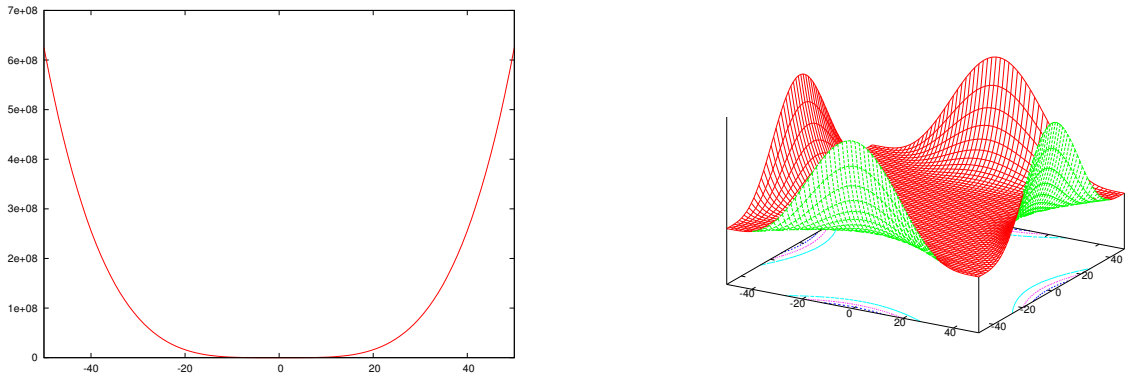
(a) instant t0 (b) instant t1

FIG. A.1 – APoL

ADL

ADL *All Dimension Linear*, basée sur la fonction statique Rosenbrock. La fonction du temps est $T(t) = t$.

$$\sum_{i=1}^{N-1}[100.(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2]$$

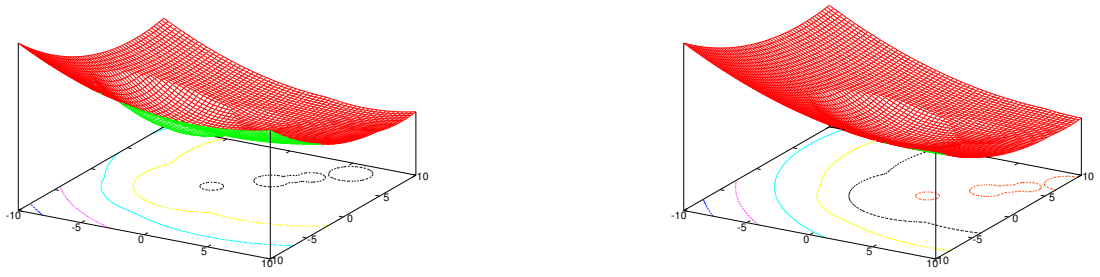


(a) instant t0 (b) instant t1

FIG. A.2 – ADL

AbPoP

AbPoP *All but Position Periodic*, basée sur la fonction statique Morrison. La fonction du temps est $T(t) = \cos(t)$.



(a) instant t0 (b) instant t1

FIG. A.3 – AbPoP

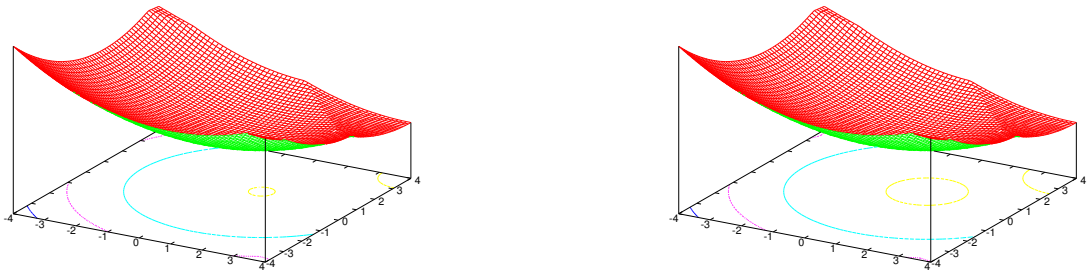
$$D(x_1, x_2, t) = min. \sum_{i=1}^{5} \left(\left(-H_i + R_i . \left((x_1 - P_{i,1}\left(T(t)\right))^2 + (x_2 - P_{i,2})^2 \right) \right) \right)$$

$$H = \{ \ 2, \ 3, \ 5, \ 7, \ 9 \ \}$$
$$R = \{ \ 2, \ 5, \ 7, \ 3, \ 6 \ \}$$

$$P = \begin{Bmatrix} 6+T(t) & 5+T(t) & 1+T(t) & 8+T(t) & 4+T(t) \\ 6 & 5 & 1 & 8 & 4 \end{Bmatrix}$$

AbVP

AbVP *All but Value Periodic*, basée sur la fonction statique Morrison. La fonction du temps est $T(t) = \cos(t)$.



(a) instant t0 (b) instant t1

FIG. A.4 – AbVP

$$D(x_1, x_2, t) = min. \sum_{i=1}^{5} \left(\left(-H_i\left(T(t)\right) + R_i . \left((x_1 - P_{i,1})^2 + (x_2 - P_{i,2})^2 \right) \right) \right)$$

$$H = \{ \ 2 + T(t), \ 3 + T(t), \ 5 + T(t), \ 7 + T(t), \ 9 + T(t) \ \}$$
$$R = \{ \ 2, \ 5, \ 7, \ 3, \ 6 \ \}$$

$$P = \begin{Bmatrix} 6 & 5 & 1 & 8 & 4 \\ 6 & 5 & 1 & 8 & 4 \end{Bmatrix}$$

OVP

OVP *Optimum Value Periodic*, basée sur la fonction statique Morrison. La fonction du temps est $T(t) = 8.\cos(0.5t)$.

(a) instant t0 (b) instant t1

FIG. A.5 – OVP

$$D(x_1, x_2, t) = min. \sum_{i=1}^{5} \left(\left(-H_i + R_i . \left((x_1 - P_{i,1})^2 + (x_2 - P_{i,2})^2 \right) \right) \right)$$

$$H = \{ \quad 1, \quad 3, \quad 5, \quad 7 + T(t), \quad 9 \quad \}$$
$$R = \{ \quad 1, \quad 1, \quad 1, \quad 1, \quad 1 \quad \}$$
$$P = \begin{Bmatrix} 2.5 & -2.5 & -5 & 0 & -4 \\ -2.5 & 2.5 & 5 & 0 & -4 \end{Bmatrix}$$

AVP

AVP *All Value Periodic*, basée sur la fonction statique Shekel. La fonction du temps est $T(t) = \cos(0.2t)$.

$$D(x) = \sum_{i=1}^{n} \frac{1}{(x - a_i)^T . (x - a_i) + c_i}$$

avec :

$$x = (x_1, x_2, x_3, x_4)^T$$

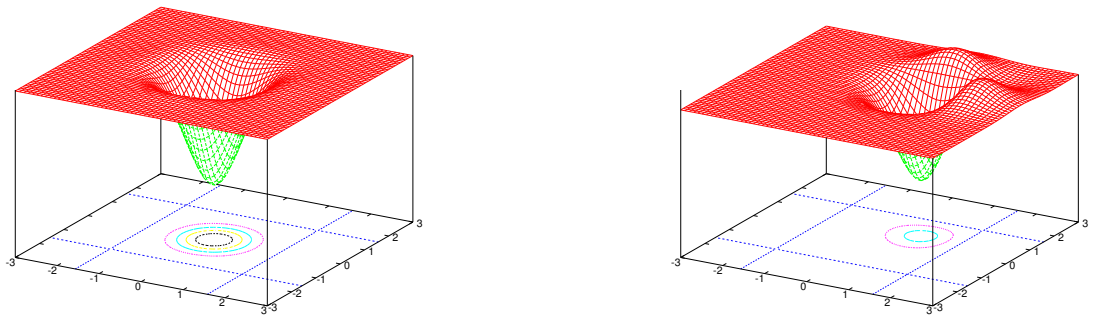
$$a_i = (a_i^1, a_i^2, a_i^3, a_i^4)^T$$

et :

i	a_i^T				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

OPoL

OPoL *Optimum Position Linear*, basée sur la fonction statique Easom. La fonction du temps est $T(t) = t$.



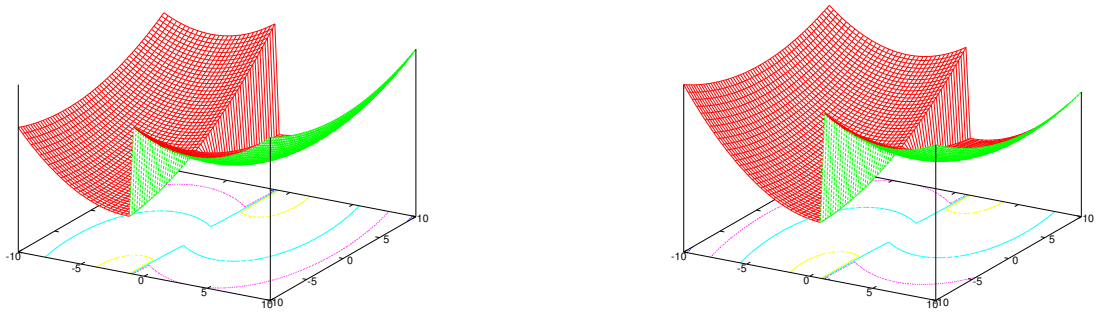
(a) instant t0 (b) instant t1

FIG. A.6 – OPoL

$$\text{Easom}(x_1, x_2) = \cos(x_1).\cos(x_2).exp^{-(x_1-t)^2-(x_2-t)^2}$$

APhL

APhL *All Phase Linear*, basée sur la fonction statique MartinGady. La fonction du temps est $T(t) = t$.



(a) instant t0 (b) instant t1

FIG. A.7 – APhL

$$(X_1 - X_2)^2 + (\frac{X_1 + X_2 - 10}{3})^2$$

avec

$$X_1 = Xp_1 - T(t)$$

$$X_2 = Xp_2 - T(t)$$

Xp_1, Xp_2 coordonnées polaires

$$T(t) = t$$

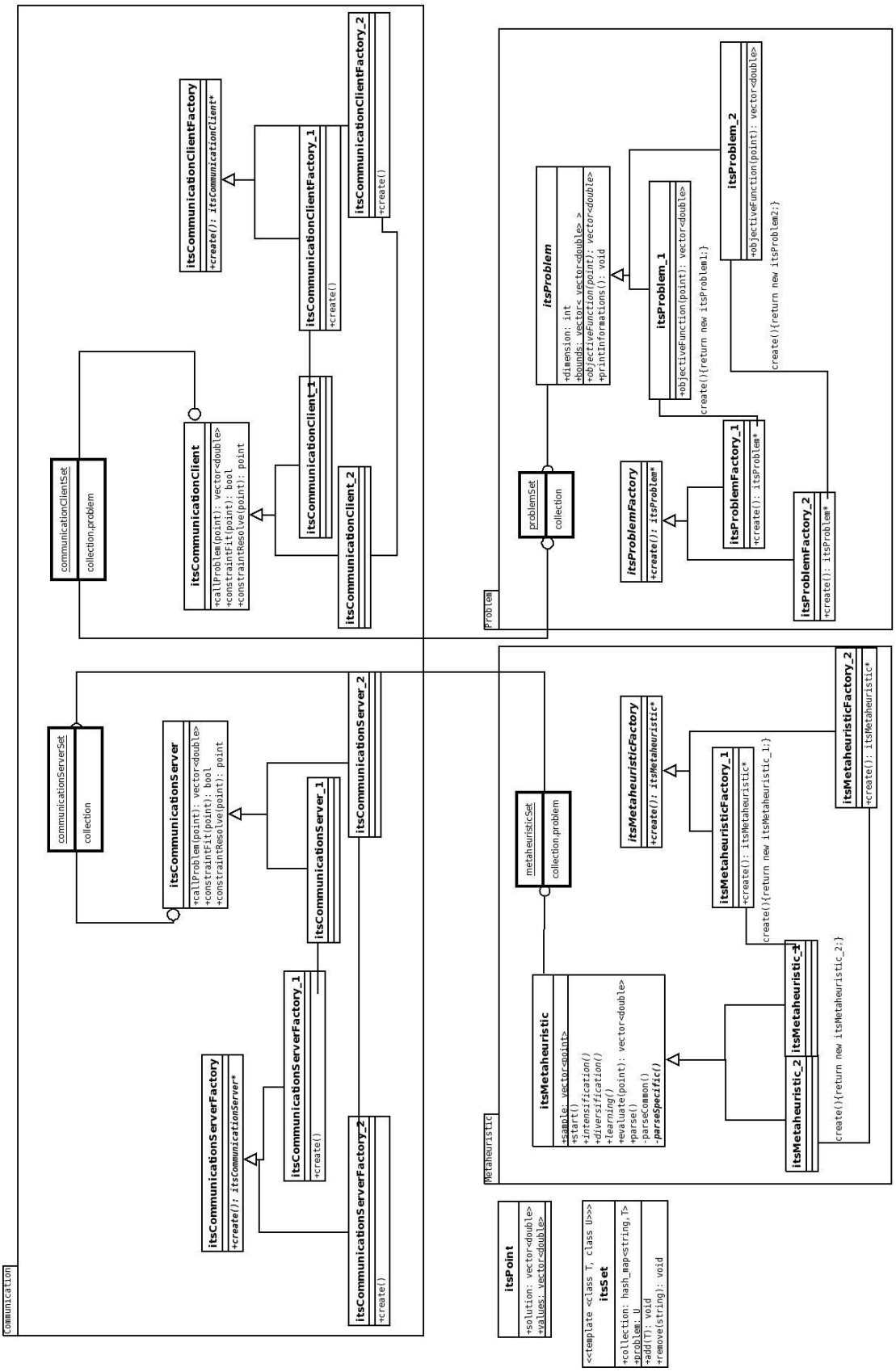


FIG. A.8 – Diagramme UML complet de la plate-forme oMetah.

Références bibliographiques

[1] E. H. L. Aarts and P. J. M. Van Laarhoven. Statistical cooling : a general approach to combinatorial optimisation problems. *Philips Journal of Research*, 40 :193–226, 1985.

[2] E. Alba and E. Bengoetxea. *Estimation of Distribution Algorithms, A new tool for Evolutionnary Computation*. Kluwer Academic Publishers, 2002.

[3] H. C. Andersen. An Investigation into Genetic Algorithms, and the Relationship between Speciation and the Tracking of Optima in Dynamic Functions. Honours thesis, Queensland University of Technology, Brisbane, Australia, 1991.

[4] V. S. Aragon and S. C. Esquivel. An evolutionary algorithm to track changes of optimum value locations in dynamic environments. *Journal of Computer Science and Technology*, 4(3) :127–134, 2004.

[5] S. Baluja. Population-based Incremental Learning : A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, 1994.

[6] S. Baluja. Genetic Algorithms and Explicit Search Statistics. *Advances in Neural Information Processing Systems*, 9 :319–325, 1997.

[7] S. Baluja and R. Caruana. Removing the Genetics from the Standard Genetic Algorithm. In A. Prieditis and S. Russel, editors, *International Conference on Machine Learning*, pages 38–46, Lake Tahoe, California, 1995. Morgan Kaufmann.

[8] S. Baluja and S. Davies. Fast probabilistic modeling for combinatorial optimization. In *Fifteenth National Conference on Artificial Intelligence, Tenth Conference on Innovative Applications of Artificial Intelligence*, Madison, Wisconsin, 1998.

[9] C. N. Bendtsen. *Optimization of Non-Stationary Problems with Evolutionary Algorithms and Dynamic Memory*. PhD thesis, University of Aarhus, Department of Computer Science Ny Munkegade 8000 Aarhus C, 2001.

[10] C. N. Bendtsen and T. Krink. Dynamic Memory Model for Non-Stationary Optimization. In *Congress on Evolutionary Computation*, pages 145–150. IEEE, 2002.

[11] A. Berro. *Optimisation multiobjectif et stratégie d'évolution dynamique*. PhD thesis, Université Paul Sabatier. Toulouse, 2001.

[12] C. Bierwirth, H. Kopfer, D. C. Mattfeld, and I. Rixen. Genetic Algorithm based Scheduling in a Dynamic Manufacturing Environment. In *Proc. of IEEE Conference on Evolutionary Computation*. IEEE Press, 1995.

[13] C. Bierwirth and D. C. Mattfeld. Production Scheduling and Rescheduling with Genetic Algorithms. *Evolutionary Computation*, 7(1) :1–17, 1999.

[14] T. M. Blackwell. Particle Swarms and Population Diversity I : Analysis. In J. Branke, editor, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 9–13, 2003.

[15] T. M. Blackwell. Particle Swarms and Population Diversity II : Experiments. In J. Branke, editor, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 14–18, 2003.

[16] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence, From Natural to Artificial Systems*. Oxford University Press, 1999.

[17] P. Bosman and D. Thierens. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In M. Muehlenbein and A. Rodriguez, editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO-2000*, pages 197–200, San Francisco, California, 2000. Morgan Kauffmann.

[18] P. Bosman and D. Thierens. IDEAs based on the normal kernels probability density function. Technical Report UU-CS-2000-11, Utrecht University, 2000.

[19] P. A. N. Bosman and D. Thierens. An algorithmic framework for density estimation based evolutionary algorithm. Technical Report UU-CS-1999-46, Utrech University, 1999.

[20] J. Branke. Evolutionary Approaches to Dynamic Environments - updated survey. In *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, 2001.

[21] J. Branke. Evolutionary Approaches to Dynamic Optimization Problems – Introduction and Recent Trends. In J. Branke, editor, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, 2003.

[22] S. Camazine, J. L. Deneubourg, N. R. Francks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press and Oxford, 2001.

[23] G. D. Caro and M. Dorigo. AntNet : Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9 :317–365, 1998.

[24] W. Cedeno and V. R. Vemuri. On the Use of Niching for Dynamic Landscapes. In *International Conference on Evolutionary Computation*. IEEE, 1997.

[25] V. Cerny. Thermodynamical approach to the traveling salesman problem : an efficient simulation algorithm. *J. of Optimization Theory and Applications*, 45(1) :41–51, 1985.

[26] H. G. Cobb. An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.

[27] H. G. Cobb and J. J. Grefenstette. Genetic Algorithms for Tracking Changing Environments. In *International Conference on Genetic Algorithms*, pages 523–530. Morgan Kaufmann, 1993.

[28] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In F. Varela and P. Bourgine, editors, *Proceedings of ECAL'91 - First European Conference on Artificial Life*, pages 134–142, Paris, France, 1992. Elsevier Publishing.

[29] Y. Cooren, J. Dréo, W. Tfaili, and P. Siarry. Open metaheuristic : une plateforme de développement et de test pour les "métaheuristiques" d'optimisation. In *3èmes Journées Doctorales d'Automatique*, Angers, 2006.

[30] M. Creutz. Microcanonical Monte Carlo simulation. *Physical Review Letters*, 50(19) :1411–1414, 1983.

[31] P. M. C. De Oliveira. Broad Histogram : An Overview. *arxiv :cond-mat/0003300v1*, 2000.

[32] K. Deb and P. N. Suganthan. Special Session on Real-Parameter Optimization. In *IEEE Congress on Evolutionary Computation*, Sept. 2005.

[33] M. Dorigo and L. M. Gambardella. Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1) :53–66, 1997.

[34] M. Dorigo and L. M. Gambardella. Guest editorial special on ant colony optimization. *IEEE Transactions on evolutionary computation*, 6(4) :317–319, 2002.

[35] R. Dorne and C. Voudouris. *Metaheuristics : computer decision-making*, chapter HSF : the iOpt's framework to easily design metaheuristic methods, pages 237–256. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[36] J. Dréo. *Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical*. PhD thesis, université Paris12. Paris, 2004.

[37] J. Dréo, J.-P. Aumasson, and W. Tfaili. Open Metaheuristics. http ://ometah.berlios.de, 2005.

[38] J. Dréo, J. P. Aumasson, W. Tfaili, and P. Siarry. Adaptive learning search, a new tool to help comprehending metaheuristics. *International Journal on Artificial Intelligence Tools*, 16(3) :483–505, 2007.

[39] J. Dréo, J. Nunes, P. Truchetet, and P. Siarry. Retinal angiogram registration by estimation of distribution algorithm. In *6th IFAC Symposium on Modelling and Control in Biomedical Systems (IFAC 2006)*, 2006.

[40] J. Dréo, J.-C. Nunes, and P. Siarry. Robust rigid registration of retinal angiograms through optimization. *Computerized Medical Imaging and Graphics*, 30(8) :453–463, 2006.

[41] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles. Paris, 2003.

[42] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Metaheuristics for Hard Optimization, Methods and Case Studies*. Springer, 2005.

[43] J. Dréo, A. Pétrowski, P. Siarry, and E. D. Taillard. *Metaheuristics for hard optimization*. Springer, 2006.

[44] J. Dréo and P. Siarry. Dynamic Optimization Through Continuous Interacting Ant Colony. In M. Dorigo and al., editors, *ANTS 2004*, LNCS 3172, pages 422–423, 2004.

[45] J. Dréo and P. Siarry. An ant colony algorithm aimed at dynamic continuous optimization. *Applied Mathematics and Computation*, 181(1) :457–467, 2007.

[46] J. Eggermont and T. Lenaerts. Dynamic Optimization using Evolutionary Algorithms with a Case-based Memory. In *the 14th Belgium Netherlands Artificial Intelligence Conference*, 2002.

[47] J. E. Eiben, A. E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[48] A. M. Ferrenberg and R. H. Swendsen. Optimized Monte Carlo Data Analysis. *Physical Review Letters*, 63 :1195, 1989.

[49] A. Fink and S. Voß. Generic metaheuristics application to industrial engineering problems. *Computers and Industrial Engineering*, 37(1-2) :281–284, 1999.

[50] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artifical Intelligence through Simulated Evolution*. Wiley, 1966.

[51] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns : Abstraction and reuse of object-oriented design. *Lecture Notes in Computer Science*, 707 :406–431, 1993.

[52] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison Wesley Professional, 1994.

[53] S. M. Garrett and J. H. Walker. Genetic Algorithms : Combining Evolutionary and 'Non'-Evolutionary Methods in Tracking Dynamic global optima. In *Genetic and Evolutionary Computation Conference*, pages 359–374. Morgan Kaufmann, 2002.

[54] A. Gaspar. Secundary Immune Response for Evolutionary Time Dependent Optimization. Technical Report 02-08, DIUF, PAI group, University of Fribourg (Switzerland), 2002.

[55] A. Gaspar and P. Collard. There is A Life beyond Convergence : Using a Dual Sharing to Adapt in Time Dependent Optimization. In *Congress on Evolutionary Computation*, volume 3, pages 1867 – 1874. IEEE, 1999.

[56] A. Ghosh, S. Tsutsui, and H. Tanaka. Function Optimization in Nonstationary Environment using Steady State Genetic Algorithms with Aging of Individuals. In *IEEE International Conference on Evolutionary Computation*, pages 666–671, 1998.

[57] D. E. Goldberg. *Genetic Algorithms*. Adison-Wesley, 1989.

[58] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine learning*. Addison-Wesley, 1989.

[59] J. J. Grefenstette. Genetic algorithms for changing environments. In R. Maenner and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 137–144. North Holland, 1992.

[60] J. J. Grefenstette. Evolvability in Dynamic Fitness Landscapes : A Genetic Algorithm Approach. In *Congress on Evolutionary Computation*, volume 3, pages 2031–2038. IEEE, 1999.

[61] J. J. Grefenstette and C. L. Ramsey. An Approach to Anytime Learning. In D. Sleeman and P. Edwards, editors, *International conference on Machine Learning*, pages 189–195. Morgan Kaufmann, 1992.

[62] M. Guntsch and M. Middendorf. Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP. In *EvoWorkshop*, pages 213–222, 2001.

[63] M. Guntsch and M. Middendorf. Applying Population Based ACO to Dynamic Optimization Problems. In *Third International Workshop ANTS*, pages 111–122. Springer Verlag, LNCS 2463, 2002.

[64] M. Guntsch and M. Middendorf. A Population Based Approach for ACO. In *2nd European Workshop on Evolutionary Computation in Combinatorial Optimization*, pages 72–81. Springer Verlag, LNCS 2279, 2002.

[65] M. Guntsch, M. Middendorf, and H. Schmeck. An Ant Colony Optimization Approach to Dynamic TSP. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 860–867, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[66] G. Harik. Linkage learning in via probabilistic modeling in the EcGA. Technical Report 99010, IlliGAL, 1999.

[67] G. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. In *IEEE Conference on Evolutionary Computation*, pages 523–528, 1998.

[68] W. K. Hastings. Monte Carlo sampling method using Markov chains and their applications. *Biometrika*, 57, 1970.

[69] P. S. Heck and S. Ghosh. A study of synthetic creativity : Behavior modeling and simulation of an ant colony. *IEEE Intelligent Systems*, 15 :58–66, 2000.

[70] E. M. Hendrix and O. Klepper. On Uniform Covering, Adaptive Random Search and Raspberries. *Journal of Global Optimization*, 18(2) :143–163, 2000.

[71] E. M. Hendrix, P. Ortigosa, and I. Garcìa. On success rates for controlled random search. *Journal of Global Optimization*, 21(3) :239–263, 2001.

[72] J. H. Holland. Outline for logical theory of adaptive systems. *J. Assoc. Comput. Mach.*, 3 :297–314, 1962.

[73] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, 1992.

[74] R. Hoshyar, S. H. Jamali, and C. Locus. Ant colony algorithm for finding good interleaving pattern in turbo codes. *IEE Proc-commun*, 147(5) :257–262, 2000.

[75] S.-J. Huang. Enhancement of Hydroelectric Generation Scheduling Using Ant Colony System based optimization Approaches. *IEEE Transactions on energy conversion*, 16(3) :296–301, 2001.

[76] K. Hukushima and K. Nemoto. Exchange Monte Carlo method and application to spin glass simulations. *Journal of the Physical Society of Japan*, 65 :1604–1608, 1996.

[77] T. Hussain, D. Montana, and G. Vidaver. Evolution-based deliberative planning for cooperating unmanned ground vehicles in a dynamic environment. In *Genetic and Evolutionary Computation Conference*, LNCS, pages 1017–1029. Springer, 2004.

[78] Y. Iba. Population Annealing : An approach to finite-temperature calculation. In *Joint Workshop of Hayashibara Foundation and SMAPIP*. Hayashibara Forum, 2003.

[79] M. Jenkinson and S. Smith. A global optimisation method for robust affine registration of brain images. *Medical Image Analysis*, 5 :143–156, 2001.

[80] M. Jones. *An Object-Oriented Framework for the Implementation of Search Techniques*. PhD thesis, University of East Anglia, 2000.

[81] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer. Evolving objects : A general purpose evolutionary computation library. In *Artificial Evolution*, pages 231–244, 2001.

[82] J. Kertesz and I. Kondor, editors. *Advances in Computer Simulation*, chapter Introduction To Monte Carlo Algorithms. Springer-Verlag, 1998.

[83] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598) :671–680, 1983.

[84] K. Krishnakumar. Micro-Genetic Algorithms for Stationary and Non-Stationary Function Optimization. In *Intelligent Control and Adaptive Systems, Proc. of the SPIE*, volume 1196, pages 289–296, 1989.

[85] F. Liang and W. H. Wong. Evolutionary Monte Carlo : Application to C_p Model Sampling and Change Point Theorem. *Statistica Sinica*, 10, 2000.

[86] A. Liekens, H. Eikelder, and P. Hilbers. Finite Population Models of Dynamic Optimization with Alternating Fitness Functions. In J. Branke, editor, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 19–23, 2003.

[87] S.-C. Lin, E. D. Goodman, and W. F. Punch. A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems. In T. Bäck, editor, *Seventh International Conference on Genetic Algorithms*, pages 481–488. Morgan Kaufmann, 1997.

[88] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering*, 11(5) :769–778, 1999.

[89] D. C. Mattfeld and C. Bierwirth. Minimizing Job Tardiness : Priority Rules vs. Adaptive Scheduling. In I. C. Parmee, editor, *Adaptive Computing in Design and Manufacture*, pages 59–67. Springer, 1998.

[90] D. Merkle and M. Middendorf. Ant Colony Optimization with the Relative Pheromone Evaluation Method. In *3rd European Workshop on Scheduling and Timetabling and 3rd European Workshop on Evolutionary Methods for AI Planning*, pages 325–333. LNCS 2279, 2002.

[91] D. Merkle and M. Middendorf. Studies on the Dynamics of Ant Colony Optimization Algorithms. In *the Genetic and Evolutionary Computation Conference*, pages 105–112. (GECCO), New York, 2002.

[92] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4) :333–346, 2002.

[93] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21 :1087–1092, 1953.

[94] N. Monmarché, E. Ramat, G. Dromel, M. Slimane, and G. Venturini. On the similarities between AS, BSC and PBIL : toward the birth of a new meta-heuristics. E3i 215, Université de Tours, 1999.

[95] N. Monmarché, N. Ramat, L. Desbarat, and G. Venturini. Probabilistic search with genetic algorithms and ant colonies. In A. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop*, pages 209–211, 2000.

[96] R. Montemanni, D. H. Smith, and S. M. Allen. An ants algorithm for the minimum-span frequency-assignment problem with interference. *IEEE Transactions on Vehicular Technology*, 51(5) :949–953, 2002.

[97] R. Morrison. Performance Measurement in Dynamic Environments. In J. Branke, editor, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 5–8, 2003.

[98] R. W. Morrison and K. A. De Jong. A Test Problem Generator for Non-Stationary Environments. In *Congress on Evolutionary Computation*, volume 3, pages 2047–2053. IEEE, 1999.

[99] R. W. Morrison and K. A. De Jong. Triggered Hypermutation Revisited. In *Congress on Evolutionary Computation*, pages 1025–1032, 2000.

[100] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. *Lecture Notes in Computer Science 1411 : Parallel Problem Solving from Nature*, PPSN IV :178–187, 1996.

[101] T. Nanayakkara, K. Watanabe, and K. Izumi. Evolving in Dynamic Environments Through Adaptive Chaotic Mutation. In *Third International Symposium on Artificial Life and Robotics*, volume 2, pages 520–523, 1999.

[102] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7 :308–313, 1965.

[103] L. Nemes and T. Roska. A cnn model of oscillation and chaos in ant colonies : a case study. *IEEE Transactions on Circuits and Systems : Fundamental Theory and Applications*, 42(10) :741–745, 1995.

[104] M. E. J. Newman and R. G. Palmer. Error estimation in the histogram Monte Carlo method. *arxiv :cond-mat/98043006*, 1998.

[105] S. Nolfi and D. Parisi. Learning to adapt to changing environments in evolving neural networks. Technical report, Institute of Psychology C.N.R. - Roma, 1995.

[106] G. Ochoa, C. Mädler-Kron, R. Rodriguez, and K. Jaffe. Assortative mating in genetic algorithms for Dynamic Problems. In F. Rothlauf et al., editors, *Applications of Evolutionary Computing*, volume 3449 of *LNCS*, pages 617–622. Springer, 2005.

[107] Y. Okamoto and U. H. E. Hansmann. Thermodynamics of helix-coil transitions studied by multicanonical algorithms. *Journal Physical Chemistry*, 99 :11276–11287, 1995.

[108] F. Oppacher and M. Wineberg. The Shifting Balance Genetic Algorithm : Improving the GA in a Dynamic Environment. In *Genetic and Evolutionary Computation Conference (GECCO)*, volume 1, pages 504–510. Morgan Kaufmann, San Francisco, 1999.

[109] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an Ant Colony Optimization Algorithm. *IEEE Transactions on Evolutionary Computing*, 6(4) :321–332, 2002.

[110] D. Parrott and X. Li. A Particle Swarm Model for tracking Multiple Peaks in a Dynamic Environment using Speciation. In *Congress on Evolutionary Computation*, pages 98–103. IEEE, 2004.

[111] V. Ramos, C. Fernandes, and A. C. Rosa. Social Cognitive Maps, Swarm Collective Perception and Distributed Search on Dynamic Landscapes. Soumis à : Journal of New Media in Neural and Cognitive Science, 2005.

[112] C. L. Ramsey and J. J. Grefenstette. Case-based initialization of genetic algorithms. In S. Forrest, editor, *International Conference on Genetic Algorithms*, pages 84–91. Morgan Kaufmann, 1993.

[113] I. Rechenberg. *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment Library Translation, 1965.

[114] M. Resende. Greedy randomized adaptive search procedures (GRASP). Technical Report TR 98.41.1, AT&T Labs-Research, 2000.

[115] N. Ritter, R. Owens, J. Cooper, R. H. Eikelboom, and P. P. V. Saarloos. Registration of stereo and temporal images of the retina. *IEEE Trans. On Medical Imaging*, 18 :404–418, 1999.

[116] C. Ronnewinkel, C. Wilke, and T. Martinetz. Genetic Algorithms in Time-Dependent Environments. In *Theoretical Aspects of Evolutionary Computing*. Springer, 2000.

[117] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[118] R. Salomon and P. Eggenberger. Adaptation on the Evolutionary Time Scale : A Working Hypothesis and Basic Experiments. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *3rd European Conference on Artificial Evolution*, number 1363 in LNCS, pages 251–262. Springer, 1997.

[119] J. Sarma and K. D. Jong. The Behavior of Spatially Distributed Evolutionary Algorithms in Non-Stationary Environments. In W. B. et al., editor, *GECCO*, volume 1, pages 572 –578. Morgan Kaufmann, San Francisco, California, 1999.

[120] T. Sasaki and M. Tokoro. Adaptation toward Changing Environments : Why Darwinian in Nature ? In P. Husbands and I. Harvey, editors, *European Conference on Artificial Life*. MIT Press, 1997.

[121] T. Sasaki and M. Tokoro. Adaptation under Changing Environments with Various Rates of Inheritance of Acquired Characters. In X. Y. et al., editor, *Simulated Evolution and Learning*, number 1585 in LNCS, pages 34–41. Springer, 1998.

[122] T. Sasaki and M. Tokoro. Evolving Learnable Neural Networks under Changing Environments with Various Rates of Inheritance of Acquired Characters : Comparison between Darwinian and Lamarckian Evolution. *Artificial Life*, 5(3) :203–223, 1999.

[123] L. Schönemann. On the Influence of Population Sizes in Evolution Strategies in Dynamic Environments. In J. Branke, editor, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 29–33, 2003.

[124] R. Schoonderwoerd, O. E. Holland, J. L. Bruten, and L. J. M. Rothkrantz. Ant-Based Load Balancing in Telecommunications Networks. *Adaptive Behavior*, (2) :169–207, 1996.

[125] K. M. Sim and W. H. Sun. Ant colony optimization for routing and load-balancing : survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics-Part A : Cybernetics*, 33(5) :560–572, 2003.

[126] A. Simões and E. Costa. Using biological inspiration to deal with dynamic environments. In *Proceedings of the Seventh International Conference on Soft Computing (MENDEL01)*, Brno, Czech Republic, 2001.

[127] J. E. Smith and F. Vavak. Replacement Strategies in Steady State Genetic Algorithms : Dynamic Environments. *Journal of Computing and Information Technology*, 7(1) :49–59, 1999.

[128] K. Socha. ACO for Continuous and Mixed-Variable Optimization. In M. Dorigo and al., editors, *ANTS 2004*, Springer LNCS 3172, pages 25–36, 2004.

[129] C. Solnon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computing*, 347-357(4) :11, 2002.

[130] Y.-H. Song and M. R. Irving. Optimization techniques for electrical power systems Part 2 Heuristic optimization methods. *Power Engineering Journal*, pages 151–160, 2001.

[131] P. D. Stroud. Kalman-Extended Genetic Algorithm for Search in Nonstationary Environments with Noisy Fitness Evaluations. *IEEE Transactions on Evolutionary Computation*, 5(1) :66–77, 2001.

[132] T. Stützle and M. Dorigo. A short Convergence Proof for a Class of Ant Colony Optimization Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(4) :358–365, 2002.

[133] R. Subrata and A. Y. Zomaya. A comparison of three artificial life techniques for reporting cell planning in mobile computing. *IEEE Transactions on Parallel and Distributed Systems*, 14(2) :142–153, 2003.

[134] G. Syswerda. Simulated Crossover in Genetic Algorithms. In L. D. Whitley, editor, *Second workshop on Foundations of Genetic Algorithms*, pages 239–255, San Mateo, California, 1993. Morgan Kaufmann.

[135] E. D. Taillard. A statistical test for comparing success rates. In *Metaheuristic international conference MIC'03*, Kyoto, Japan, Aug. 2003.

[136] E. D. Taillard, L. M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive Memory Programming : A Unified View of Meta-Heuristics. *European Journal of Operational Research*, 135(1) :1–16, 1998.

[137] J.-H. Teng and Y.-H. Liu. A Novel ACS-Based Optimum Switch Relocation Method. *IEEE Transactions on Power systems*, 18(1) :113–120, 2003.

[138] W. Tfaili. Optimisation continue dynamique, algorithme de colonie de fourmis. Technical report, LISSI, E.A. 3956 Université Paris 12, 2005.

[139] W. Tfaili, J. Dréo, and P. Siarry. Fitting of an ant colony approach to dynamic optimization through a new set of test functions. *International Journal of Computational Intelligence Research*, 3(3) :205–218, 2007.

[140] W. Tfaili, J. Dréo, and P. Siarry. Un algorithme de colonie de fourmis pour l'optimisation continue dynamique. In *JDMACS-JNMACS*, Lyon, septembre 2005.

[141] W. Tfaili and P. Siarry. Un algorithme de colonie de fourmis "chargées" pour l'optimisation continue dynamique. In *FRANCORO V - ROADEF*, Grenoble, 2007.

[142] W. Tfaili and P. Siarry. Using diversification in a new ant colony algorithm aimed at dynamic optimization. In *The Seventh Metaheuristics International Conference MIC.*, Montreal, 2007.

[143] W. Tfaili and P. Siarry. A new charged ant colony algorithm for continuous dynamic optimization. *Applied Mathematics and Computation*, 197(2) :604–613, 2008.

[144] R. Tinos and A. de Carvalho. A genetic algorithm with gene dependent mutation probability for non-stationary optimization problems. In *Congress on Evolutionary Computation*, volume 2, 2004.

[145] E. Triki, Y. Collette, and P. Siarry. A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. *European Journal of Operational Research*, 166 :77–92, 2005.

[146] K. Trojanowski and Z. Michalewicz. Evolutionary Optimization in Non-Stationary Environments. *Journal of Computer Science and Technology*, 1(2) :93–124, 2000.

[147] K. Trojanowski, Z. Michalewicz, and J. Xiao. Adding Memory to the Evolutionary Planner/Navigator. In *IEEE Intl. Conference on Evolutionary Computation*, pages 483–487, 1997.

[148] D. Tschumperlé. Cimg library. Available at `http ://cimg.sourceforge.net/`, 2005.

[149] R. K. Ursem. Multinational GA Optimization Techniques in Dynamic Environments. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Genetic and Evolutionary Computation Conference*, pages 19–26. Morgan Kaufmann, 2000.

[150] R. K. Ursem, B. Filipič, and T. Krink. Exploring the Performance of an Evolutionary Algorithm for Greenhouse Control. In V. Glavinić, V. H. Dobrić, and D. Šimić, editors, *Proceedings of the 24th International Conference on Information Technology Interfaces ITI 2002*, pages 429–434, 2002.

[151] J. I. van Hemert and J. A. L. Poutre. Dynamic Routing Problems with Fruitful Regions : Models and Evolutionary Computation. In X. Yao et al., editor, *Parallel Problem Solving from Nature*, volume 3242 of *LNCS*, pages 692–701. Springer, 2004.

[152] K. Verbeek and A. Nowé. Colonies of learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 32(6) :347–357, 2002.

[153] S. Voß and D. L. Woodruff. *Optimization Software Class Libraries*. OR/CS Interfaces Series. Kluwer, Dordrecht, 2002.

[154] K. Weicker. An Analysis of Dynamic Severity and Population Size. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN VI)*, volume 1917 of *LNCS*. Springer, 2000.

[155] K. Weicker and N. Weicker. On Evolution Strategy Optimization in Dynamic Environments. In *Congress on Evolutionary Computation*, volume 3, pages 2039–2046, 1999.

[156] K. Weicker and N. Weicker. Dynamic rotation and partial visibility. In *Congress on Evolutionary Computation*, pages 1125–1131, 2000.

[157] O. Wendt and W. König. Cooperative Simulated Annealing : How Much Cooperation is Enough ? Technical Report 97-19, Institute of Information Systems, Goethe University, Frankfurt, 1997.

[158] Wikipédia. Mediawiki : l'encyclopédie libre. http ://fr.wikipedia.org, 2007.

[159] C. O. Wilke. Evolution in time-dependent fitness landscapes. Technical Report IR-INI 98-09, Institut für Neuroinformatik, Ruhr-Universität Bochum, 1998.

[160] M. Wineberg and F. Oppacher. Enhancing the GA's Ability to Cope with Dynamic Environments. In W. et al., editor, *Genetic and Evolutionary Computation Conference*, pages 3–10. Morgan Kaufmann, 2000.

[161] Q. Xiong and A. Jutan. Continuous optimization using a dynamic simplex method. *Chemical Engineering Science*, 58(16) :3817–3828, 2003.

[162] K. Yamasaki. Dynamic Pareto Optimum GA against the Changing Environments. In *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 47–50, 2001.

[163] S. Yang. Constructing dynamic test environments for genetic algorithms based on problem difficulty. In *Congress on Evolutionary Computation*, volume 2, pages 1262–1269, 2004.

[164] H. Yu, A. S. Wu, K.-C. Lin, and G. Schiavone. Adaptation of Length in a Nonstationary Environment. In E. Cantu-Paz, editor, *Genetic and Evolutionary Computation Conference*, volume 2724 of *LNCS*, pages 1541–1553. Springer, 2003.