République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider - Biskra
Faculté des Sciences et des Sciences de l'Ingénieur
Département d'Informatique

Mémoire

en vue de l'obtention du diplôme de Magister en informatique

Option : Intelligence Artificielle et Systèmes Distribués

Adaptation des métaheuristiques aux problèmes d'optimisation à variables continues

Présenté par :

FAREH Abdelhak

Membres du jury :

Dr. BENMOHAMMED Mohamed	**Professeur à l'Université de Constantine**	**Président**
Dr. KAZAR Okba	**Maître de conférences à l'Université de Biskra**	**Rapporteur**
Dr. BILAMI Azeddine	**Maître de conférences à l'Université de Batna**	**Examinateur**
Dr. CHERIF Foudil	**Maître de conférences à l'Université de Biskra**	**Examinateur**

Dédicace

À mes parents

À mon épouse

À mes frères et mes sœurs

Remerciements

Je remercie, tout d'abord, dieu qui m'a éclairé le chemin du savoir.

Ensuite, je tiens à remercier mon encadreur **Dr. KAZAR Okba** pour ses conseils, ses orientations et son aide.

Je remercie également les membres de jury, **Pr. BENMOHAMMED Mohamed**, **Dr. BILAMI Azeddine** et **Dr. CHERIF Foudil**, d'avoir accepté de juger ce travail.

Enfin, je présente mes remerciements à toutes les personnes qui m'ont encouragé à construire et élaborer ce mémoire.

Résumé

Les métaheuristiques ont été conçues, à l'origine, pour résoudre les problèmes d'optimisation discrète. Le travail présenté dans le cadre de ce mémoire consiste à adapter spécialement la méthode d'optimisation par essaim de particules aux problèmes à variables continues. Cette métaheuristique est l'une des méthodes de la large catégorie des méthodes d'intelligence en essaim. Elle simule le comportement social des essaims d'oiseaux et des bancs de poissons et repose sur la notion d'essaim de particules.

L'algorithme d'essaim de particules se base sur la collaboration des particules entre elles. Il est initialisé par une population de solutions potentielles aléatoires, interprétées comme des particules se déplaçant dans l'espace de recherche. Chaque particule est attirée vers sa meilleure position découverte par le passé ainsi que vers la meilleure position découverte par les particules de son voisinage.

L'algorithme est enfin appliqué dans le cadre d'un problème de télécommunication concernant la planification des réseaux locaux sans-fil. Ce problème est similaire à celui posé depuis quelques années dans les réseaux cellulaires et largement traité dans la littérature.

La planification d'un réseau local sans-fil est un problème d'optimisation dont les variables sont données par l'ensemble des configurations possibles des points d'accès et les objectifs par la description mathématique des services que le réseau doit offrir.

Mots clés : métaheuristique, optimisation continue, essaim particulaire, réseaux sans-fil, planification.

Abstract

The metaheuristics have been designed, initially, to solve discrete optimization problems. The work presented in this thesis is to adapt specially the method of particle swarm optimization with problems of continuous variables. This metaheuristic is one of the broad category of swarm intelligence methods. It simulates the social behavior of swarms of birds and fish and is based on the concept of particles swarm.

The particle swarm algorithm is based on the collaboration between the particles themselves. It is initialized by a population of random potential solutions interpreted as particles moving in the research space. Each particle is attracted to his best position discovered by the past as well as to the best position discovered by the particles in its neighborhood.

The algorithm is finally applied in the context of a problem on telecommunication, regarding planning of local wireless networks. This problem is similar to that presented in the recent years in the cellular networks and widely covered in the literature. Planning a local wireless network is an optimization problem, its variables are given by all possible access points configurations and goals in the mathematical description of the services that the network has to offer.

Keywords : metaheuristic, continuous optimization, particle swarm, wireless network, planning.

مـــوجـــــز

لقد صممت الإستدلاليات في الأصل لحل مشاكل الإستمثال المنفصلة لكن العمل المقدم في هذه المذكرة يقوم على تكييف طريقة الإستمثال بسرب الجزئيات – بصفة خاصة – مع المشاكل ذات المتغيرات المستمرة. هذه الإستدلالية هي إحدى طرق الفئة الواسعة من طرق الذكاء بالسرب و هي تحاكي السلوك الاجتماعي لأسراب الطيور و مجموعات الأسماك كما أنها ترتكز على مفهوم سرب الجزئيات.

خوارزمية سرب الجزئيات تستند على التعاون بين هذه الجزئيات، إذ أنها تهيء بمجموعة من الحلول المحتملة بصفة عشوائية ممثلة بجزئيات تنتقل في فضاء البحث و كل جزيء ينجذب لأفضل موقع اكتشفه في الماضي وكذلك لأفضل موقع اكتشفته الجزئيات المجاورة.

الخوارزمية بعد ذلك طبقت في سياق مشكلة من مشاكل الإتصالات وهي تتعلق بتخطيط الشبكات المحلية اللاسلكية، هذا الإشكال مشابه لذلك الذي طرح منذ عدة سنوات في اطار الشبكات الخلوية و الذي تمت دراسته بشكل واسع.

تخطيط شبكة محلية لاسلكية هو مشكل استمثال إذ تعطى المتغيرات بمجموعة التشكيلات الممكنة لنقاط الإتصال و الأهداف بالوصف الرياضي للخدمات التي يجب أن تقدمها الشبكة.

الكلمات المفتاحيـــة : إستدلالية، إستمثال مستمر، سرب جزيئات، شبكات لاسلكية، تخطيط.

Table des matières

Introduction générale

Une terminologie considère que les *méta-heuristiques* sont une forme d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) pour lesquels on ne connaît pas de méthode classique plus efficace. Le terme *méta* est donc pris au sens où les algorithmes peuvent regrouper plusieurs heuristiques. On rencontre cette définition essentiellement dans la littérature concernant les algorithmes évolutionnaires, où elle est utilisée pour désigner une spécialisation.

Les métaheuristiques sont souvent inspirées à partir des systèmes naturels, qu'ils soient pris en physique (cas du recuit simulé), en biologie de l'évolution (cas des algorithmes génétiques) ou encore en éthologie (cas des algorithmes de colonie de fourmis ou de l'optimisation par essaim de particules).

En pratique, elles ne devraient être utilisées que sur des problèmes ne pouvant être optimisés par des méthodes mathématiques. Les métaheuristiques sont souvent employées en optimisation combinatoire, mais on en rencontre également pour des problèmes continus ou mixtes (problèmes à variables discrètes et continues).

Ensuite, l'un des principaux buts des recherches en vie artificielle consiste à examiner comment les créatures naturelles se comportent comme un essaim, et de développer des nouvelles techniques d'optimisation utilisant l'analogie de ces comportements **[FAR07]**. Ces comportements collectifs s'inscrivent tout à fait dans la théorie de l'auto-organisation. Pour résumer, chaque individu utilise l'information locale à laquelle il peut accéder sur le déplacement de ses plus proches voisins pour décider de son propre déplacement. Des règles très simples comme "rester proche des autres individus", "aller dans la même direction", "aller à la même vitesse" suffisent pour maintenir la cohésion du groupe tout entier, et pour susciter des comportements collectifs complexes et adaptés.

Dans ce cadre, l'optimisation par essaim de particules ("Particle Swarm Optimization", *PSO*) est issue d'une analogie avec les comportements collectifs de déplacements d'animaux. La métaphore a de plus été largement enrichie de notions de socio-psychologie. En effet, chez certains groupes d'animaux, comme les bancs de poissons, on peut observer des dynamiques de déplacements relativement complexes, alors que les individus eux-mêmes n'ont accès qu'à des informations limitées, comme la position et la vitesse de leurs plus proches voisins. On peut par exemple observer qu'un banc de poissons est capable d'éviter un prédateur : d'abord en se divisant en deux groupes, puis en reformant le banc originel, tout en maintenant la cohésion du banc.

Les auteurs de la méthode d'optimisation par essaim de particules se sont inspirés de ces comportements en s'appuyant sur la théorie de la socio-psychologie du traitement de l'information et des prises de décisions dans les groupes sociaux. La méthode en elle-même met en jeu des groupes de *particules* sous forme de vecteurs se déplaçant dans l'espace de recherche. Chaque particule est caractérisée par sa *position* et un vecteur de changement de position (appelé *vélocité* ou *vecteur vitesse*). À chaque itération, la particule se déplace. La socio-psychologie suggère que des individus se déplaçant sont influencés par leur comportement passé et par celui de leurs voisins (voisins dans le réseau social et non nécessairement dans l'espace). On tient donc compte, dans la mise à jour de la position de chaque particule, de la direction de son mouvement, sa vitesse, sa meilleure position et la meilleure position de ses voisins. Ici, la mémoire est structurée au niveau local, entre particules voisines, à chaque itération chaque particule n'évolue qu'en fonction de ses proches voisins, et non pas selon l'état global de la population à l'itération précédente **[DRS03]**.

D'un autre côté, bien que les problèmes d'optimisation difficile en variables continues soient très courants en ingénierie mais ils sont peu étudiés en recherche opérationnelle. La plupart des algorithmes d'optimisation sont, en effet, proposés dans le domaine combinatoire. Dès lors, l'adaptation de ces métaheuristiques combinatoires aux problèmes continus est profitable à un grand nombre de problèmes réels.

Alors, les réseaux locaux sans-fil (*wireless LAN* ou *wLAN*) permettent de transmettre des données par les ondes électromagnétiques à l'intérieur de bâtiments et si besoin, à l'échelle d'un ensemble de bâtiments (campus, site industriel…). Ils appartiennent à la famille des réseaux sans-fil qui sont classifiés selon l'étendue de leur zone de service.

Le déploiement de réseaux locaux sans-fil présente des contraintes plus complexes à gérer pour obtenir des conditions de transmission optimales. La réception du message est tributaire de la qualité du lien radio qui elle même est contrainte par la configuration de l'environnement (position et nature des murs, présence de meubles…) et la position de l'émetteur. Il est clair que le choix de l'emplacement et des caractéristiques des émetteurs radio, compte tenu de la description de l'environnement, est primordial pour le bon fonctionnement d'un réseau sans-fil. Ce choix est le cœur du problème de planification radio **[RUN05]**.

Enfin, le but principal de notre mémoire est de montrer comment planifier un réseau local sans-fil en utilisant l'optimisation par essaim de particules.

Ce mémoire se compose de quatre chapitres. Le premier chapitre présente quelques types de méthodes d'optimisation ou bien de métaheuristiques tels que le recuit simulé, la recherche tabou, les algorithmes génétiques, les colonies de fourmis et l'essaim de particules dont chacune le principe de base et l'algorithme de fonctionnement en plus de quelques notions spécifiques.

Le deuxième chapitre décrit un problème d'optimisation en général ainsi que son processus avant de passer une vue sur le comportement de quelques métaheuristiques parmi les citées ci-dessus à un problème type très connu dans le monde d'optimisation difficile qui est le problème du voyageur de commerce.

Le troisième chapitre dresse un état de l'art sur les travaux d'adaptation de quelques métaheuristiques aux problèmes d'optimisation à variables continues.

Le quatrième chapitre sera consacré à une présentation du problème de planification de réseaux locaux sans-fil, ses caractéristiques, ses objectifs et surtout ses formulations.

L'implémentation de notre approche et l'étude comparative des résultats fera l'objet de la dernière partie de ce chapitre. Pour cela, l'environnement de développement, les choix techniques, la description du modèle et la présentation du logiciel seront tous configurés dans cette partie.

Nous terminons le mémoire par une conclusion générale qui synthétise notre projet et présente les perspectives de notre travail.

Chapitre 1 :

Généralité sur les métaheuristiques

I. Introduction

Nous présentons dans ce chapitre le cadre général des métaheuristiques qui sont apparues au début des années 1980 dont le but commun est de résoudre au mieux les problèmes d'optimisation difficile.

Le mot "heuristique" vient du grec *heurein* (découvrir) et qualifie tout ce qui sert à la découverte, à l'invention et à la recherche.

En effet, les métaheuristiques sont des stratégies qui permettent d'explorer l'espace de recherche efficacement afin de déterminer les solutions presque optimales.

Parmi les différentes métaheuristiques, nous allons uniquement nous intéresser aux métaheuristiques modernes telles que le recuit simulé, la recherche tabou, les algorithmes génétiques, les algorithmes de colonie de fourmis et les algorithmes d'essaim de particules.

II. Vue générale

Les ingénieurs se heurtent quotidiennement à des problèmes technologiques à complexité sans cesse grandissante, qui surgissent dans des secteurs très divers, comme dans le traitement des images, la conception de systèmes mécaniques, la planification et l'exploitation des réseaux électriques, le contrôle des processus, … etc. Le problème à résoudre peut fréquemment être exprimé sous la forme générale d'un problème d'optimisation, dans lequel on définit une fonction objectif, ou fonction de coût, que l'on cherche à minimiser (ou maximiser) vis-à-vis des paramètres concernés. Par exemple, dans le célèbre problème du voyageur de commerce, on cherche à minimiser la distance de la tournée d'un « voyageur de commerce », qui doit visiter un certain nombre de villes, avant de retourner à la ville de départ. La définition du problème d'optimisation est souvent complétée par la définition de *contraintes* : tous les paramètres (ou variables de décisions) de la solution proposée doivent respecter ces contraintes, faute de quoi la solution n'est pas *réalisable*.

Il existe de nombreuses méthodes d'optimisation "classiques" pour résoudre de tels problèmes, applicables lorsque certaines conditions mathématiques sont satisfaites tel que : la programmation linéaire qui traite efficacement le cas où la fonction objectif, ainsi que les contraintes s'expriment linéairement en fonction des variables de décision. Dans le cas où la fonction objectif et ses contraintes ne sont pas linéaires, la programmation non linéaire

est applicable. Malheureusement, les situations rencontrées en pratique comportent souvent une ou plusieurs complications, qui mettent en défaut ces méthodes : par exemple, la fonction objectif peut être non homogène, ou même ne peut pas s'exprimer analytiquement en fonction des paramètres ; ou encore, le problème peut exiger la considération simultanée de plusieurs objectifs contradictoires.

L'arrivée d'une nouvelle classe de méthodes d'optimisation, nommées *métaheuristiques*, marque une grande révolution dans le domaine de l'optimisation. En effet, celles-ci s'appliquent à toutes sortes de problèmes combinatoires, et elles peuvent également s'adapter aux problèmes continus. Ces méthodes, qui comprennent notamment la méthode du recuit simulé, les algorithmes génétiques, la méthode de recherche tabou, les essaims de particules et les algorithmes de colonie de fourmis, … etc. sont apparues, à partir des années 1980, avec une ambition commune : résoudre au mieux les problèmes d'optimisation difficile. Elles ont en commun, les caractéristiques suivantes **[BOU07]**:

- Elles sont, au moins pour une partie, stochastiques : cette approche permet de faire face à l'explosion combinatoire des possibilités.
- Elles sont d'origine combinatoire : elles ont l'avantage, décisif dans le cas continu, d'être directes, c'est-à-dire qu'elles ne recourent pas au calcul, souvent problématique, des gradients de la fonction objectif.
- Elles sont inspirées par des analogies : avec la physique (recuit simulé, diffusion simulée, … etc.), avec la biologie (algorithmes génétiques, recherche tabou, … etc.) ou avec l'éthologie (colonie de fourmis, essaim de particules, … etc.).
- Elles sont capables de guider, dans une tache particulière, une autre méthode de recherche spécialisée (par exemple, une autre heuristique, ou une méthode d'exploration locale).
- Elles partagent aussi les mêmes inconvénients : les difficultés de réglage des paramètres mêmes de la méthode, et le temps de calcul élevé.

Pour l'optimisation d'une fonction continue, ces méthodes d'optimisation peuvent être adaptées moyennant des transformations plus au moins aisées, en inventant une nouvelle topologie. Chaque paramètre doit être *discrétisé* de façon individuelle. La difficulté majeure réside dans la détermination de la taille optimale du pas de discrétisation et de sa direction (résultant des variables sur lesquelles on agit).

Le choix de la loi de discrétisation est un compromis entre deux situations extrêmes :

- si le pas est trop petit, on n'explore qu'une région limitée de l'espace des configurations, et l'algorithme risque d'être piégé dans un minimum local ;
- si le pas est trop grand, la recherche devient quasiment aléatoire.

La meilleure solution peut consister à élaborer une topologie adaptative.

Les métaheuristiques comportent souvent plusieurs paramètres contrôlant les différents opérateurs et l'influence du ou des processus stochastiques. L'efficacité d'une métaheuristique dépend du choix de ses paramètres de contrôle. Ce réglage est complexe, surtout quand le nombre de paramètres est élevé et quand la plage de variation de chacun de ces paramètres est étendue. Les différents paramètres sont généralement corrélés, ce qui rend encore plus difficile leur réglage. Enfin pour un jeu de paramètres de contrôle donnés, l'aspect stochastique fait que les résultats varient d'une exécution à l'autre **[CHE99]**.

III. Notions fondamentales

III.1. Définition

Les métaheuristiques sont apparues dans les années 1980 et forment une famille d'algorithmes d'optimisation dont le but est la résolution des problèmes d'optimisation difficile.

Étymologiquement parlant ce mot est composé dans un premier temps du préfixe « méta » qui signifie « au-delà » ou « plus haut » en grec puis de « heuristique » (heuriskein) qui signifie « trouver ». Cette décomposition permet facilement de comprendre le premier but de ces algorithmes : trouver des solutions à des problèmes en utilisant plusieurs (méta) heuristiques **[LAP06]**.

III.2. Présentation des heuristiques

Pour rappel, les heuristiques sont des règles empiriques simples qui, à la différence des algorithmes, ne se basent non pas sur des analyses scientifiques parfois trop complexes (car nécessitant la définition et l'assignation de nombreux éléments), mais sur l'expérience et les relations accumulées au fils des résultats. Plus simplement ces règles utilisent les résultats passés et leurs analogies afin d'optimiser leurs recherches futures en examinant d'abord les cas les plus plausibles.

Par définitions, les heuristiques disposent d'une simplicité et donc d'une rapidité dans leur exécution plus élevée que les algorithmes classiques. Ces règles s'appliquant à un ensemble particulier la recherche des faits ce voit simplifiée et accélérée (moins de possibilité). D'où une analyse des situations améliorées. Mais une méthode heuristique trop simplifiée ou au contraire trop générale peut conduire à des biais cognitifs, générant des erreurs de décisions.

L'utilisation de plusieurs de ces éléments simples (les heuristiques) afin de créer des éléments plus complexes (les métaheuristiques) permet donc de réduire considérablement l'ensemble de recherche global de l'algorithme. On peut même parfois définir cet ensemble dans un ensemble de moindre complexité que celui de l'algorithme (Par exemple l'algorithme est défini sur R et les solutions possibles uniquement sur N).

L'une de leur caractéristique principale et à première (seulement) vue défaut, dont hérite également les métaheuristiques, est qu'ils peuvent dans

certains cas ne pas proposer la solution optimale au problème. Mais un résultat s'y rapprochant d'assez près pour qu'il soit considéré comme correct, on parle alors de garantie de performance. Cet état arrive lorsque, par exemple, la résolution parfaite et optimale du problème nécessite un temps trop élevé ou tout simplement infini. Cet état de faits est parfaitement illustré par l'effet Tétris, qui dit qu'une perception hâtive (voir fausse) peut se montrer plus efficace qu'une analyse exacte mais tardive **[LAP06]**.

III.3. Présentation des métaheuristiques

Les métaheuristiques sont des algorithmes d'optimisation de type stochastiques et progressant vers un optimum par échantillonnage d'une fonction objectif dont le but est la résolution de problèmes d'optimisation difficile.

Un problème d'optimisation est un problème à partir duquel on peut définir une ou plusieurs fonctions objectifs permettant la différenciation d'une bonne solution face à une mauvaise : concrètement ces fonctions objectifs parcours l'ensemble des solutions possibles de l'espace de recherche local et sont, à chaque itérations, comparées à des optimums précédemment définis. Leur égalité (ou presque égalité dans le cas d'une garantie de performance) conduit alors à l'état final. À la solution. Le principe même d'une métaheuristique est de minimiser ou de maximiser ces fonctions afin de réduire les solutions possibles et par la même occasion le temps d'exécution.

Lorsqu'une seule valeur est associée à une seule fonction objectif on parle de problème *mono-objectif.*

Dans le cas contraire on parle naturellement d'un problème *multi-objectif.*

On utilise le terme métaheuristique car ces algorithmes regroupent en réalité plusieurs heuristiques. Cette utilisation d'heuristiques implique qu'on ne peut pas réellement classer les métaheuristiques dans la catégorie des algorithmes d'intelligence artificielle puisqu'ils sont principalement guidés par le hasard : on peut d'ailleurs les comparer à un aveugle cherchant (et trouvant) son chemin à tâtons. Cependant ils sont souvent combinés à d'autres algorithmes afin d'en accélérer la convergence en obligeant, entre autre, le métaheuristique à ne pas prendre en compte les solutions trop « extravagantes ».

Une notion à bien cerner découlant de l'utilisation d'heuristiques est l'ensemble des solutions possibles ou espace global. Car bien que les métaheuristiques fonctionnent plus ou moins de façon hasardeuse cette tare (qui en réalité est un avantage) est compensée par la diminution de l'espace de travail local à chaque itération **[LAP06]**.

III.4. Fonctionnement général des métaheuristiques

Un autre avantage des métaheuristiques est que leur utilisation ne nécessite pas de connaissances particulières sur le problème d'optimisation à résoudre. En pratique il suffit simplement d'associer une ou plusieurs variables (fonctions objectifs) à une ou plusieurs solutions (optimums).

Du fait de leur facilité de programmation et de manipulation les métaheuristiques sont souvent employées à la résolution de problèmes d'optimisation ayant résisté aux méthodes de résolution classiques (telles que les méthodes déterministes).

L'exécution des métaheuristiques se déroule en trois phases :

1. Diversification
2. Intensification
3. Mémoire

La diversification regroupe les actions ayant pour but de récolter des informations sur le problème dans un ensemble défini (ou espace local) lors de l'intensification.

L'intensification vise à utiliser les informations de la mémoire et celles récoltées lors de la phase de diversification afin de définir les meilleurs espaces de recherche locaux futurs et de les faire parcourir de façon optimale par les fonctions objectifs.

La mémoire est le support de l'apprentissage permettant à l'algorithme de ne tenir compte que des zones où l'optimum est susceptible de se trouver et de garder en mémoire les résultats passés.

Les métaheuristiques progressent itérativement et alternativement entre les phases de diversification, d'intensification et d'apprentissage. La phase originale est souvent choisie aléatoirement puis l'algorithme continue jusqu'à ce qu'un critère d'arrêt (fonctions objectifs = optimums) soi atteint.

Les métaheuristiques sont souvent inspirées par des systèmes naturels et utilisées dans de nombreux domaine : en physique (recuit simulé), en biologie de l'évolution (algorithmes évolutionnaires et génétiques) ou en éthologie (algorithmes de colonie de fourmis) **[LAP06]**.

III.4.1. Fonctionnement des métaheuristiques à parcours

Les métaheuristiques les plus classiques sont ceux fondés sur la notion de parcours. Dans ce type d'algorithme ce dernier fait évoluer (parcourir) une seule fonction objectif sur l'espace de recherche local à chaque itération puis la compare aux optimums. La compréhension de la notion de voisinage (approche locale de la topologie, plus globale) est alors nécessaire.

Les plus connues dans cette classe sont le recuit simulé et la recherche avec tabous **[LAP06]**.

III.4.2. Fonctionnement des métaheuristiques à populations

Ces métaheuristiques utilisent un échantillonnage des fonctions objectif comme base d'apprentissage. Ces dernières deviennent alors extrêmement importantes lors du choix de l'espace de recherche local.

Dans cette famille les métaheuristiques utilisent la notion de population : ils manipulent un ensemble de solutions en parallèle. Chaque élément de ladite

population parcours un certain nombre de solutions dans l'ensemble local. On peut donc en quelque sorte les assimiler à des métaheuristiques à métaparcours.

Du fait de ces métaparcours, la différence entre les métaheuristiques à parcours et les métaheuristiques à population est parfois floue. En effet si l'on considère par exemple un recuit simulé dans lequel la température baisse par paliers l'algorithme manipulera alors à chaque palier (itération) un ensemble de points (population). Il s'agit simplement d'une méthode d'échantillonnage particulière.

Parmi les algorithmes inclus dans cette classification on peut citer les algorithmes génétiques et les algorithmes de colonie de fourmis **[LAP06]**.

III.4.3. Fonctionnement des métaheuristiques à méthodes implicites

Dans la famille des métaheuristiques à population l'échantillonnage se fait sur une base aléatoire et peut donc être décrite via une distribution de probabilité. Selon l'approche voulue deux choix de méthode s'offrent à nous : implicites et explicites.

Lors de l'utilisation des méthodes implicites, avec les algorithmes génétiques par exemple, la distribution de probabilité n'est pas connue ou n'est pas utilisée : le choix de l'échantillonnage entre deux itérations ne suit pas une loi donnée, mais est fonction de règles locales **[LAP06]**.

III.4.4. Fonctionnement des métaheuristiques à méthodes explicites

Les métaheuristiques basées sur les méthodes explicites utilisent une distribution de probabilité choisie à chaque itération. C'est le cas, entre autre, des algorithmes à estimation de distribution qui, comme leur nom l'indique, estime à chacune de leur itération et via une distribution de probabilité (issue des fonctions objectifs) l'espace de recherche local optimal **[LAP06]**.

IV. Métaheuristiques modernes

Parmi les différentes métaheuristiques, nous allons nous intéresser aux principales métaheuristiques modernes :

IV.1. Méthode du recuit simulé

Le recuit simulé trouve ses origines dans la thermodynamique. Cette méthode est issue d'une analogie entre le phénomène physique de refroidissement lent d'un corps en fusion, qui le conduit à un état solide, de basse énergie. Il faut baisser lentement la température, en marquant des paliers suffisamment longs pour que le corps atteigne l'"équilibre thermodynamique" à chaque palier de température. Pour les matériaux, cette basse énergie se manifeste par l'obtention d'une structure régulière, comme dans les cristaux et l'acier.

L'analogie exploitée par le recuit simulé consiste à considérer une fonction f à minimiser comme fonction d'énergie, et une solution x peut être considérée comme un état donné de la matière dont $f(x)$ est l'énergie. Le recuit simulé exploite généralement le critère défini par l'algorithme de Metropolis

(**Figure 1.1**) pour l'acceptation d'une solution obtenue par perturbation de la solution courante.

Pour une "température" T donnée, à partir d'une solution courante x, on considère une transformation élémentaire qui changerait x en $s(x)$. Si cette perturbation induit une diminution de la valeur de la fonction objectif f, $\Delta f = f(s(x)) - f(x) < 0$, elle est acceptée. Dans le cas contraire, si $\Delta f = f(s(x)) - f(x) \geq 0$, la perturbation est acceptée tout de même avec une probabilité :

$$p = exp\frac{-\Delta f}{T}$$

SI $f(s(x)) \leq f(x)$
  $f(x) = f(s(x))$
  $x = s(x)$
SINON
  $p = exp\frac{-\Delta f}{T}$
  $r =$ solution aléatoire entre [0,1]
  SI $r \leq p$
    $f(x) = f(s(x))$
    $x = s(x)$
  FIN SI

Figure 1.1 : pseudocode de la règle de Metropolis.

Le paramètre de contrôle T est la "température" du système, qui influe sur la probabilité d'accepter une solution plus mauvaise. À une température élevée, la probabilité d'acceptation d'un mouvement quelconque tend vers 1 : presque tous les changements sont acceptés. L'algorithme équivaut alors à une marche aléatoire dans l'espace des configurations. Cette température est diminuée lentement au fur à mesure du déroulement de l'algorithme pour simuler le processus de refroidissement des matériaux, et sa diminution est suffisamment lente pour que l'équilibre thermodynamique soit maintenu. Nous présentons dans la **Figure 1.2** l'algorithme du recuit simulé.

L'efficacité du recuit simulé dépend fortement du choix de ses paramètres de contrôle, dont le réglage reste très empirique.

Les principaux paramètres de contrôle sont les suivants **[CHE99]**:

- la valeur initiale de la température,
- la fonction de décroissance de la température,
- le critère de changement de palier de température,
- les critères d'arrêt.

$x =$ solution aléatoire
$f(x) =$ valeur de la fonction

$f_{min} = f(x)$
$x_{min} = x$
$T =$ initialiser température (assez élevée)
REPETER
  REPETER
    générer un voisin $s(x) \in$ voisinage $S(x)$
    appliquer la règle de Metropolis
    SI $f(x) < f_{min}$
      $f_{min} = f(x)$
      $x_{min} = x$
    FIN SI
  JUSQU'A équilibre thermodynamique atteint
  $T =$ décroître température
JUSQU'A conditions d'arrêt satisfaites

Figure 1.2 : pseudocode de l'algorithme du recuit simulé.

Pour le calcul de la température de départ, plusieurs méthodes ont été proposées. Une des méthodes est basée sur l'observation de la variation moyenne de la fonction *f*. À partir d'une solution initiale x_0 on génère, par transformations élémentaires aléatoires, un certain nombre de solutions x'_0 (environ 50 à 100) telles que $f(x'_0) > f(x_0)$, et on calcule la variation moyenne $|< \Delta f >|_{init}$. Une température initiale T_{init} est calculée de façon à accepter au départ une certaine proportion p_{init} de mouvements dégradant la fonction *f*. Pour une température initiale "moyenne", la valeur de p_{init} est de 0.5. La valeur de T_{init} est déduite de la formule suivante :

$$p_{init} = exp \frac{-|< \Delta f >|_{init}}{T_{init}}$$

Le rôle de la température *T* au cours du processus de recuit simulé est très important. Une forte décroissance de température risque de piéger l'algorithme dans un minimum local, alors qu'une faible décroissance au début du processus entraîne une convergence très lente de l'algorithme. Un compromis pour adapter la décroissance de la température à l'évolution du processus consiste à utiliser une variation logarithmique. La loi logarithmique de décroissance de la température, qui assure la convergence théorique du recuit simulé, est la suivante:

$$T_k = \frac{\mu}{Log(1+k)}$$

où *k* est le nombre de paliers de température effectuée, et μ une constante positive. En pratique, on adopte souvent une décroissance géométrique

$T_{k+1} = \alpha T_k$, avec $(0 < \alpha < 1)$, car la loi précédente induit un temps de calcul prohibitif.

Pour le changement de palier de température, on peut simplement spécifier un nombre de transformations, acceptées ou non, au bout duquel la température est abaissée **[CHE99]**.

IV.2. Méthode de recherche tabou

La recherche tabou est une méthode originalement développée par Glover et indépendamment par Hansen, sous l'appellation de "*steepest ascent mildest descent*". Elle est basée sur des idées simples, mais elle est néanmoins très efficace.

Cette méthode combine une procédure de recherche locale avec un certain nombre de règles et de mécanismes permettant à celle-ci de surmonter l'obstacle des optima locaux, tout en évitant de cycler. Elle a été appliquée avec succès pour résoudre de nombreux problèmes difficiles d'optimisation combinatoire : problèmes de routage de véhicule, problèmes d'affectation quadratique, problèmes d'ordonnancement, problèmes de coloration de graphes, … etc. **[CHE99]**

IV.2.1. Principe de base

Dans une première phase, la méthode de recherche tabou peut être vue comme une généralisation des méthodes d'amélioration locales. En effet, en partant d'une solution quelconque x appartenant à l'ensemble de solutions X, on se déplace vers une solution $s(x)$ située dans le voisinage $[S(x)]$ de x. Donc l'algorithme explore itérativement l'espace de solutions X.

Afin de choisir le meilleur voisin $s(x)$ dans $S(x)$, l'algorithme évalue la fonction objectif f en chaque point $s(x)$, et retient le voisin qui améliore la valeur de la fonction objectif f, ou au pire celui qui la dégrade le moins.

L'originalité de la méthode de recherche tabou, par rapport aux méthodes locales, qui s'arrêtent dès qu'il n'y a plus de voisin s(x) permettant d'améliorer la valeur de la fonction objectif f, réside dans le fait que l'on retient le meilleur voisin, même si celui-ci est plus mauvais que la solution d'où l'on vient. Ce critère autorisant les dégradations de la fonction objectif évite à l'algorithme d'être piégé dans un minimum local. Mais il induit un risque de cyclage. En effet, lorsque l'algorithme a quitté un minimum quelconque par acceptation de la dégradation de la fonction objectif, il peut revenir sur ses pas, à l'itération suivante.

Pour régler ce problème, l'algorithme a besoin d'une mémoire pour conserver pendant un moment la trace des dernières meilleures solutions déjà visitées. Ces solutions sont déclarées *tabou*, d'où le nom de la méthode. Elles sont stockées dans une liste de longueur L donnée, appelée *liste tabou*. Une nouvelle solution n'est acceptée que si elle n'appartient pas à cette liste tabou. Ce critère d'acceptation d'une nouvelle solution évite le cyclage de l'algorithme, durant la visite d'un nombre de solutions au moins égal à la

longueur de la liste tabou, et il dirige l'exploration de la méthode vers des régions du domaine de solutions non encore visitées. Le pseudocode de l'algorithme tabou classique est présenté dans la **Figure 1.3**.

La liste tabou est généralement gérée comme une liste "*circulaire*" : on élimine à chaque itération la solution tabou la plus ancienne, en la remplaçant par la nouvelle solution retenue.

Mais le codage d'une telle liste est encombrant, car il faudrait garder en mémoire tous les éléments qui définissent une solution. Pour pallier cette contrainte, on remplace la liste tabou de solutions interdites par une liste de "*transformations interdites*", en interdisant la transformation inverse d'une transformation faite récemment. **[CHE99]**

IV.2.2. Critère d'aspiration

Le remplacement de la liste tabou des solutions visitées par la liste des transformations élémentaires $\{x, s(x)\}$ conduit non seulement à l'interdiction de revenir vers des solutions précédentes (on évite le cyclage court), mais aussi vers un ensemble de solutions dont plusieurs peuvent ne pas avoir été visitées jusqu'ici. Il est donc primordial de corriger ce défaut et de trouver un moyen de lever l'interdiction de l'acceptation d'une transformation élémentaire $\{x, s(x)\}$ déjà effectuée (donc appartenant à la liste tabou), sous un certain critère, appelé *critère d'aspiration*. Cette correction permet aussi de revenir à une solution déjà visitée et de redémarrer la recherche dans une autre direction.

$x =$ solution aléatoire
$f_{min} = f(x)$
$x_{min} = x$

TABOU = liste de solutions $s(x)$ de longueur L
TABOU = VIDE

REPETER
  générer un N-échantillon TEL QUE $s_i(x) \in$ voisinage $S(x)$
  et $\{x, s_i(x)\} \notin$ TABOU
  $f(s(x)) = \min_{1 \le i \le N} [f(s_i(x))]$
  ajouter $(\{x, s_i(x)\}, \text{TABOU})$
  $x = s(x)$
  SI $f(x) < f_{min}$
    $f_{min} = f(x)$
    $x_{min} = x$
  FIN SI
JUSQU'A conditions d'arrêt satisfaites

Figure 1.3 : pseudocode de la méthode de recherche tabou.

Le critère d'aspiration le plus simple et le plus couramment utilisé consiste à tester si la solution produite de statut tabou présente un coût inférieur à celui de la meilleure solution trouvée jusqu'à présent. Si cette situation se produit, le statut tabou de la solution est levé. Ce critère est évidemment très sévère, il ne devrait pas être vérifié très souvent, donc il apporte peu de changements à la méthode. D'autres critères d'aspiration plus complexes peuvent être envisagés. L'inconvénient de recourir trop souvent à l'aspiration est qu'elle peut détruire, dans une certaine mesure, la protection offerte par la liste tabou vis-à-vis du cyclage.

Notons que, dans le cas d'une liste tabou de solutions, le concept de critère d'aspiration n'est pas intéressant. Toute annulation du statut tabou d'une solution se trouvant dans la liste tabou pourrait conduire l'algorithme au cyclage **[CHE99]**.

IV.2.3. Intensification

L'intensification consiste à approfondir la recherche dans certaines régions du domaine, identifiées comme susceptibles de contenir un optimum global. Cette intensification est appliquée périodiquement, et pour une durée limitée. Pour mieux intensifier la recherche dans une zone bien localisée, plusieurs stratégies sont proposées dans la littérature.

La plus simple consiste à retourner à l'une des meilleures solutions trouvée jusqu'à présent, puis de reprendre la recherche à partir de cette solution, en réduisant la longueur de la liste tabou pour un nombre limité d'itérations. Dans ce cas, on adapte la procédure de recherche tabou, en élargissant le voisinage de la solution courante (en augmentant la taille de l'échantillon $S(x)$), tout en diminuant le pas des transformations. On peut aussi remplacer simplement l'heuristique tabou par une autre méthode plus puissante, ou mieux adaptée, pour une recherche locale **[CHE99]**.

IV.2.4. Diversification

La diversification permet à l'algorithme de bien explorer l'espace des solutions, et d'éviter que le processus de recherche ne soit trop localisé et laisse de grandes régions du domaine totalement inexplorées. La plus simple des stratégies de diversification consiste à interrompre périodiquement l'acheminement normal de la procédure tabou, et à la faire redémarrer à partir d'une autre solution, choisie aléatoirement, ou "intelligemment". Une autre méthode consiste à biaiser la fonction d'évaluation f, en introduisant un terme qui pénalise les transformations effectuées fréquemment, afin de favoriser des transformations nouvelles. Ce type de stratégie de diversification peut être utilisé de façon continue, sans interrompre la procédure de recherche tabou.

En résumé, la diversification et l'intensification sont des concepts complémentaires, qui enrichissent la méthode de recherche tabou et la rendent plus robuste et plus efficace **[CHE99]**.

IV.3. Algorithmes génétiques

Les principes fondamentaux de ces algorithmes ont été exposés par Holland. Ces algorithmes s'inspirent du fonctionnement de l'évolution naturelle, notamment la sélection de Darwin, et la procréation selon les règles de Mendel.

La sélection naturelle, que Darwin appelle l'élément "propulseur" de l'évolution, favorise les individus d'une population qui sont le mieux adaptés à un *environnement*. La sélection est suivie de la procréation, réalisée à l'aide de croisements et de mutations au niveau du patrimoine génétique des individus (ou "génotype"), constitué d'un ensemble de gènes. Ainsi deux individus "parents", qui se croisent, transmettent une partie de leur patrimoine génétique à leurs descendants. Le génotype de l'enfant fait que celui-ci est plus au moins bien adapté à l'environnement. S'il est bien adapté, il a une plus grande chance de procréer dans la génération future. Au fur et à mesure des générations, on sélectionne les individus les mieux adaptés, et l'augmentation du nombre des individus bien adaptés fait évoluer la population entière **[CHE99]**.

IV.3.1. Principe de base

Dans les algorithmes génétiques, on essaie de simuler le processus d'évolution d'une population. On part d'une population de N solutions du problème représentées par des individus. Cette population choisie aléatoirement est appelée population parent. Le degré d'*adaptation* d'un individu à l'environnement est exprimé par la valeur de la fonction de coût $f(x)$, où x est la solution que l'individu représente. On dit qu'un individu est d'autant mieux adapté à son environnement, que le coût de la solution qu'il représente est plus faible. Au sein de cette population, intervient alors la *sélection* au hasard d'un ou deux parents, qui produisent une nouvelle solution, à travers les *opérateurs génétiques*, tels que le *croisement* et la *mutation*. La nouvelle population, obtenue par le choix de N individus parmi les populations parent et enfant, est appelée génération suivante. En itérant ce processus, on produit une population plus riche en individus mieux adaptés. La **Figure 1.4** présente le principe d'un algorithme génétique de base.

Cet algorithme comporte trois phases distinctes **[CHE99]**:

- la production de la population d'individus la mieux adaptée pour contribuer à la reproduction de la génération suivante (version artificielle de la sélection naturelle) ; elle peut être mise en œuvre sous plusieurs formes algorithmiques ;
- la phase de reproduction, qui exploite essentiellement les opérateurs de croisement et de mutation ;
- la stratégie de remplacement des populations parent et enfant par la génération suivante. Elle pourra être mise en œuvre sous plusieurs formes.

POP, *POP'* et *QUALITE* : tableaux de taille *N*
POP = initialiser population
$f(x) = \min_{1 \le i \le N} [f(x_i)]$

$f_{min} = f(x)$
$x_{min} = x$

REPETER
  QUALITE (évaluer la population *POP*)
  REPETER (phase de reproduction génétique)
    sélection
    croisement
    mutation
  JUSQU'A *POP'* remplie
  POP = sélectionner nouvelle population (*POP*,*POP'*)
  $f(x) = \min_{1 \le i \le N} [f(x_i)]$
  SI $f(x) < f_{min}$
    $f_{min} = f(x)$
    $x_{min} = x$
  FIN SI
JUSQU'A conditions d'arrêt satisfaites

Figure 1.4 : pseudocode d'un algorithme génétique de base.

IV.3.2. Codage

Plusieurs codes d'informations sont utilisés. Les plus fréquemment utilisés sont le code *binaire naturel* et le code *binaire de Gray*. Plus récemment le codage "*réel*" a fait son apparition **[CHE99]**.

⇒ *Codage binaire*

Ce codage consiste, pour un individu donné, à concaténer toutes ses variables codées en binaire. La chaîne binaire 1000|0110|1101, par exemple, correspond à un individu défini par 3 variables (8, 6, 13) en codage binaire naturel sur 4 bits chacune. Ce codage binaire présente plusieurs avantages : alphabet minimum, facilité de mise en place d'opérateurs génétiques et existence de résultats théoriques. Néanmoins, ce codage présente trois inconvénients majeurs **[CHE99]**:

- Les performances de l'algorithme sont diminuées lorsque la longueur de la chaîne augmente.
- Deux nombres décimaux voisins (exemple : 7 et 8) peuvent être très éloignés dans le codage binaire naturel (1000 et 0111) : falaise de Hamming. Ce problème peut être réglé en remplaçant le code binaire naturel par le code binaire de Gray.
- La dissymétrie entre le 0 (plus fréquent) et le 1.

⇒ *Codage réel*

Ce codage consiste simplement à la concaténation des variables x_i d'un individu x. Exemple un individu $x(25,31,8)$ est codé 25|31|8. Ce codage présente des avantages majeurs. Il est plus précis que le codage binaire et l'espace de recherche est le même que l'espace du problème. Il a le mérite d'être simple à utiliser, l'évaluation de la fonction de coût est plus rapide. Le codage réel évite de faire le transcodage du binaire naturel ou de Gray vers les réels à chaque évaluation. Néanmoins, il possède deux inconvénients, son alphabet est infini, et il a besoin d'opérateurs appropriés **[CHE99]**.

IV.3.3. Calcul de la qualité

On détermine la population d'individus les mieux adaptés. On calcule la qualité de chaque individu de la population pour déterminer sa probabilité de sélection. Plus la qualité est élevée, mieux l'individu est adapté. On obtient cette qualité à partir de la fonction de coût f par une transformation telle que **[CHE99]**:

$$Qualité(x) = f_{max} - f(x)$$

f_{max} devra être choisi de manière à ce que la valeur de *Qualité(x)* reste toujours positive.

IV.3.4. Opérateurs de reproduction

La phase de reproduction exploite principalement deux opérateurs : le croisement et la mutation. Elle comporte aussi l'opération de sélection et l'opération de production de la génération suivante, à partir des populations parent et enfant **[CHE99]**.

⇒ *Sélection*

La sélection consiste à choisir les paires d'individus qui vont participer à la reproduction de la population future. La fonction de sélection calcule une probabilité de sélection pour chaque individu, en fonction de sa qualité et de la qualité de tous les autres individus dans la population **[CHE99]**.

⇒ *Croisement*

Le principal opérateur agissant sur la population de parents est le croisement, qui est appliqué avec une certaine probabilité, appelée taux de croisement *Pc* (typiquement proche de l'unité). Le croisement consiste à choisir deux individus représentés par leurs chaînes de gènes, tirés au hasard dans la population courante, et à définir aléatoirement un ou plusieurs points de croisement. Les nouvelles chaînes sont alors créées en interchangeant les différentes parties de chaque chaîne. La **Figure 1.5** montre une telle opération sur deux individus représentés par leurs chaînes codées sur 8 bits. Cet opérateur permet de bien explorer le domaine de variation des individus, et de diriger la recherche vers des régions intéressantes de l'espace d'étude en utilisant la connaissance déjà présente dans la population courante **[CHE99]**.

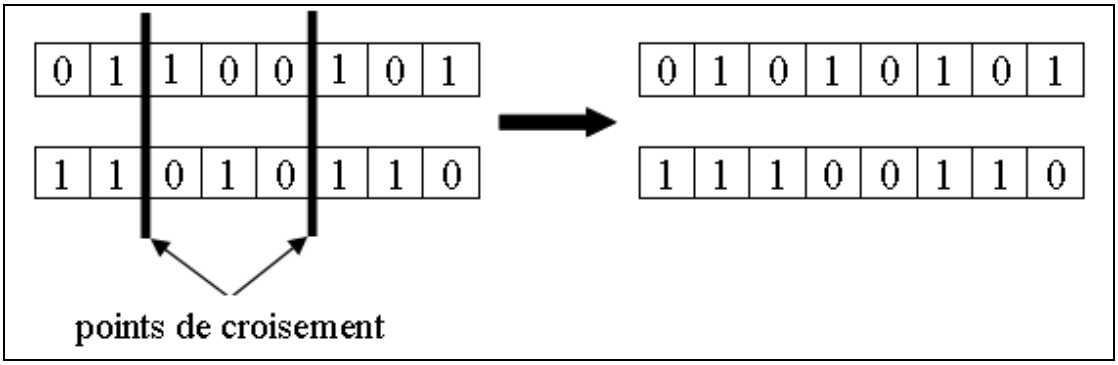


Figure 1.5 : l'opérateur de croisement dans le codage binaire.

⇒ *Mutation*

L'opération de mutation protège les algorithmes génétiques des pertes prématurées d'informations pertinentes. Elle permet d'introduire une certaine information dans la population, qui a pu être perdue lors de l'opération de croisement. Ainsi elle participe au maintien de la diversité, utile à une bonne exploration du domaine de recherche. L'opérateur de mutation s'applique avec une certaine probabilité, appelée taux de mutation *Pm*, typiquement compris entre 0.05 et 0.10. Ce faible taux de mutation permet de dire que la mutation est considérée comme un mécanisme d'adaptation secondaire pour les algorithmes génétiques. Dans le codage binaire, la mutation consiste à changer un bit 1 par le bit 0 et vice versa, pour chaque bit de la chaîne, avec la probabilité *Pm*. La **Figure 1.6** représente l'opération de mutation pour un individu représenté par une chaîne binaire codée sur 8 bits. Une autre façon de faire consiste à choisir un individu à muter avec la probabilité *Pm*, et à changer un bit de cet individu choisi au hasard. Cette seconde méthode est plus intéressante, car la probabilité de mutation *Pm* est indépendante de la longueur du codage des variables des individus.

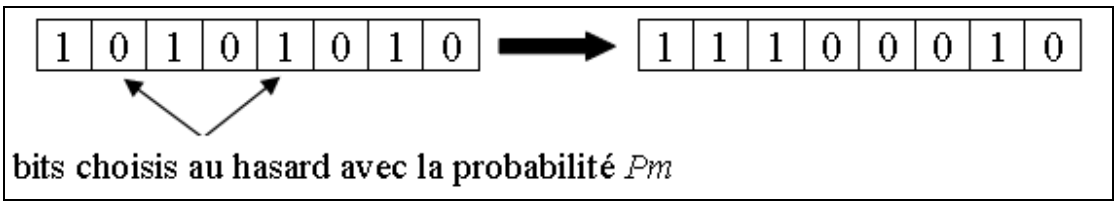


Figure 1.6 : l'opérateur de mutation dans le codage binaire.

L'algorithme génétique fait évoluer une population. La sélection réduit la diversité de la population, tandis que les opérateurs génétiques, croisement et mutation, augmentent cette diversité **[CHE99]**.

IV.3.5. Quelques résultats théoriques

Le choix d'un codage adéquat est un élément critique dans l'efficacité d'un algorithme génétique. Il existe deux types de difficultés dans le choix d'un codage **[CHE99]**.

⇒ *Principe des alphabets minimaux*

Un codage doit pouvoir être adapté au problème de façon à limiter au mieux la taille de l'espace de recherche, et aussi de façon que les nouveaux individus engendrés par les opérateurs de recherche soient significatifs le plus souvent possible **[CHE99]**.

⇒ *Principe de pertinence des schémas*

Une théorie qui décrit l'évolution des solutions partielles, appelées *schémas*, dans une population d'une génération à l'autre est introduite par Holland. Par exemple, la chaîne 1101 appartient aux régions 11**, *10*, 11*1, … etc. : les * désignent indifféremment un 1 ou un 0. L'efficacité d'un algorithme génétique dépend des schémas associés aux bonnes régions (i.e. contenant un nombre élevé de bonnes solutions). Le principal résultat de cette théorie est que d'une génération à une autre, la répétition des croisements accroît les bons motifs dans la population de manière exponentielle, à condition que ces bons motifs soient courts. Le facteur de croissance dépend beaucoup des paramètres de l'algorithme tels que le taux de croisement et le taux de mutation. La multiplication de ces bons motifs appelés *blocs de construction* accroît évidemment les chances de trouver la solution optimale recherchée.

Les algorithmes génétiques ont relativement peu de fondements théoriques. Il n'existe aucune garantie que la méthode trouve la solution optimale, bien que des travaux récents donnent des preuves de convergence. Ces travaux tiennent peu compte de l'influence de l'opérateur de croisement, qui est réduit parfois à sa plus simple forme, afin de simplifier l'analyse de l'algorithme uniquement en fonction de la mutation et de la sélection. Il est difficile de faire l'analyse d'un algorithme génétique dans sa globalité, mais en revanche on peut s'intéresser à l'un de ses opérateurs séparément **[CHE99]**.

IV.4. Optimisation par colonie de fourmis

Le principe de l'optimisation par colonie de fourmis est apparu au début des années 1990. Il est dû aux chercheurs M. Dorigo, V. Maniezzo et A. Colorni qui expliquent leur théorie dans un article fondateur **[MDC96]**. Article dans lequel ils proposent une nouvelle approche pour l'optimisation stochastique combinatoire et mettent en avant la rapidité de leur nouvelle méthode à trouver des solutions acceptables tout en évitant des convergences prématurées. Ils qualifient leur méthode de versatile (elle peut s'appliquer à des versions similaires d'un même problème), robuste et bien sûr basée sur une population d'individus.

L'optimisation par colonie de fourmis s'inspire du comportement des fourmis lorsque celles-ci sont à la recherche de nourriture. Les fourmis en se déplaçant déposent des phéromones, substances olfactives et volatiles. Chaque fourmi se dirige en tenant compte des phéromones qui sont déposées par les autres membres de la colonie. Les fourmis choisissent leur chemin de manière probabiliste. Comme les phéromones s'évaporent progressivement, le choix probabiliste que prend une fourmi pour choisir son chemin évolue

continuellement. Ce mécanisme de choix des chemins peut être illustré par l'exemple **Figure 1.7**.

À T = 0, 32 fourmis sont en D et 32 autres en B. Le choix du chemin à suivre entre H et C est complètement aléatoire puisqu'il n'y a encore aucune phéromone sur ce chemin. Chaque groupe de fourmis se divise donc en deux parties égales qui déposent chacune 16 phéromones sur leur chemin. À T = 1, 32 nouvelles fourmis se présentent en D et 32 autres en B. Mais pendant ce temps, les 16 fourmis parties de D en passant par C sont arrivées en B, de même les 16 autres parties de B en passant par C sont en D. Il y a donc 16 + 16 / 2 = 24 phéromones qui indiquent le chemin en passant par C. Ces 16 / 2 = 8 phéromones sont dues à l'évaporation. De l'autre côté, les 16 fourmis parties de D et les 16 autres parties de B se retrouvent en H. Il n'y a donc que 16 / 2 = 8 phéromones qui indiquent en D comme en C le chemin en passant par H. Les nouvelles fourmis vont donc majoritairement choisir le chemin par C. C'est ce que l'on voit à T = 2.

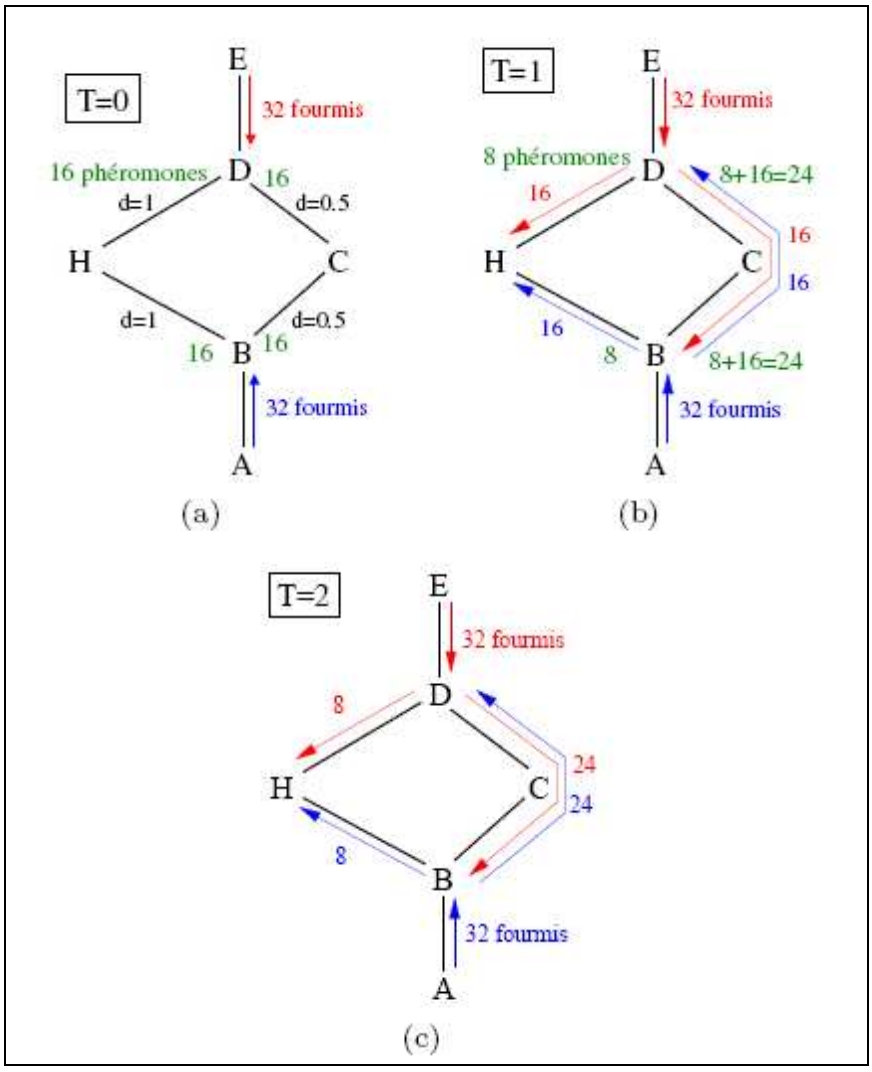


Figure 1.7 : un exemple de la stratégie des colonies de fourmis, le contournement d'obstacles.

(a) Le schéma initial avec les distances, 32 fourmis arrivent en D et en B. Pas de phéromones, les fourmis choisissent une direction de manière aléatoire.
(b) À t = 1, évaporation des phéromones, arrivée en B des fourmis passées par C.
(c) Les fourmis sont allées majoritairement dans la plus courte branche.

Dans l'exemple précédent, les phéromones s'évaporaient de moitié à chaque nouveau pas d'horloge. Mais le schéma de décroissance peut être différent, c'est l'un des paramètres de modulation des colonies de fourmis avec le nombre de fourmis **[BIA05]**.

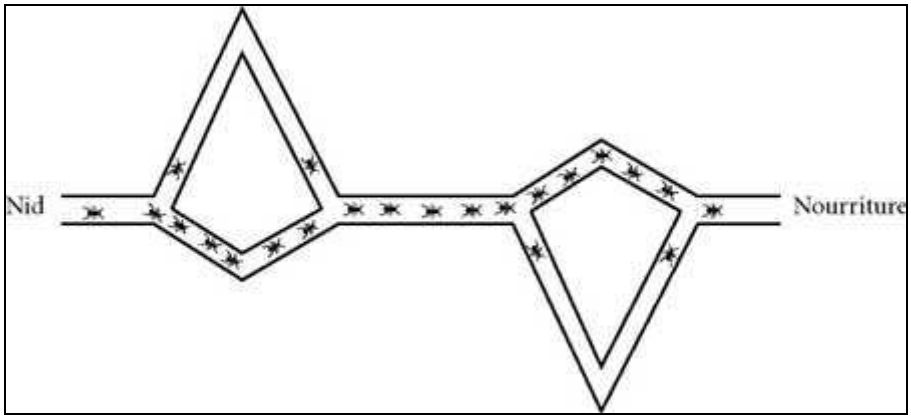


Figure 1.8 : recueil de ressources chez certaines fourmis.

IV.5. Optimisation par essaim de particules

L'optimisation par essaim de particules (OEP) est une méthode née en 1995 aux Etats-Unis sous le nom de *Particle Swarm Optimization* (PSO).

Initialement, ses deux concepteurs, Russel Eberhart (ingénieur en électricité) et James Kennedy (socio-psychologue), cherchaient à modéliser des interactions sociales entre des « agents » devant atteindre un objectif donné dans un espace de recherche commun, chaque agent ayant une certaine capacité de mémorisation et de traitement de l'information. La règle de base était qu'il ne devait y avoir aucun chef d'orchestre, ni même aucune connaissance par les agents de l'ensemble des informations, seulement des connaissances locales. Un modèle simple fut alors élaboré.

Dès les premières simulations, le comportement collectif de ces agents évoquait celui d'un essaim d'êtres vivants convergeant parfois en plusieurs sous-essaims vers des sites intéressants. Ce comportement se retrouve dans bien d'autres modèles, explicitement inspirés des systèmes naturels. Ici, la métaphore la plus pertinente est probablement celle de l'essaim d'abeilles, particulièrement du fait qu'une abeille ayant trouvé un site prometteur sait en informer certaines de ses consœurs et que celles-ci vont tenir compte de cette information pour leur prochain déplacement. Finalement, le modèle s'est révélé être trop simple pour vraiment simuler un comportement social, mais par contre très efficace en tant qu'outil d'optimisation **[CLS04]**.

IV.5.1. Méthode de base

La version historique peut facilement être décrite en se plaçant du point de vue d'une particule. Au départ de l'algorithme, un essaim est réparti au hasard dans l'espace de recherche, chaque particule ayant également une vitesse aléatoire. Ensuite, à chaque pas de temps :

- chaque particule est capable d'évaluer la qualité de sa position et de garder en mémoire sa meilleure performance, c'est-à-dire la meilleure position qu'elle a atteinte jusqu'ici (qui peut en fait être parfois la position courante) et sa qualité (la valeur en cette position de la fonction à optimiser).
- chaque particule est capable d'interroger un certain nombre de ses congénères (ses informatrices, dont elle-même) et d'obtenir de chacune d'entre elles sa propre meilleure performance (et la qualité afférente).
- à chaque pas de temps, chaque particule choisit la meilleure des meilleures performances dont elle a connaissance, modifie sa vitesse en fonction de cette information et de ses propres données et se déplace en conséquence.

Le premier point se comprend facilement, mais les deux autres nécessitent quelques précisions. Les informatrices sont définies une fois pour toutes de la manière suivante :

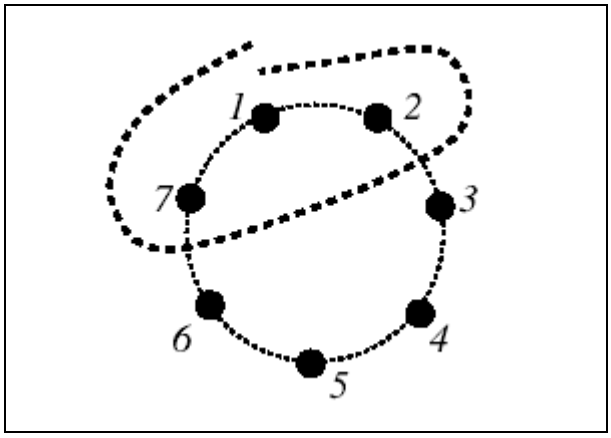


Figure 1.9 : le cercle virtuel pour un essaim de sept particules.
Le groupe d'information de taille trois de la particule 1 est composé des particules 1, 2 et 7.

On suppose que toutes les particules sont disposées (symboliquement) en cercle et, pour la particule étudiée, on inclut progressivement dans ses informatrices, d'abord elle-même, puis les plus proches à sa droite et à sa gauche, de façon à atteindre le total requis. Il y a bien sûr de nombreuses variantes, y compris celle consistant à choisir les informatrices au hasard, mais celle-ci est à la fois simple et efficace.

Une fois la meilleure informatrice est détectée, la modification de la vitesse est une simple combinaison linéaire de trois tendances, à l'aide de coefficients de confiance :

- la tendance « *volontariste* », consistant à continuer selon la vitesse actuelle,
- la tendance « *conservatrice* », ramenant plus ou moins vers la meilleure position déjà trouvée,
- la tendance « *suiviste* », orientant approximativement vers la meilleure informatrice.

Les termes « plus ou moins » ou « approximativement » font référence au fait que le hasard joue un rôle, grâce à une modification aléatoire limitée des coefficients de confiance, ce qui favorise l'exploration de l'espace de recherche **[CLS04]**.

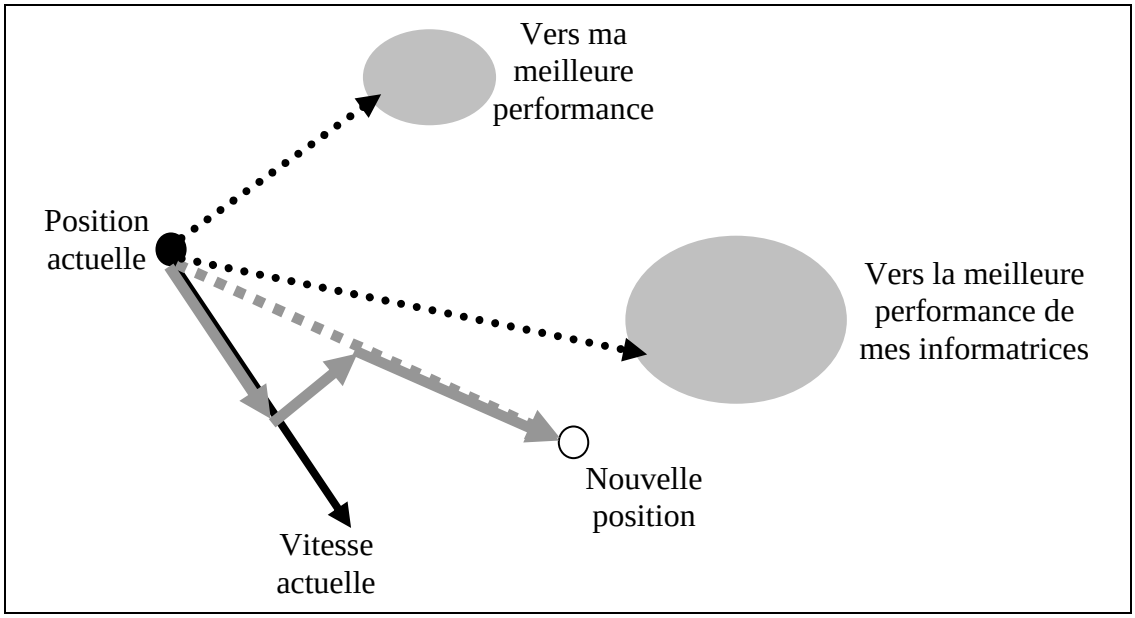


Figure 1.10 : schéma de principe du déplacement d'une particule. Pour réaliser son prochain mouvement, chaque particule combine trois tendances : suivre sa vitesse propre, revenir vers sa meilleure performance, aller vers la meilleure performance de ses informatrices.

IV.5.2. Formulation générale

La formulation générale de ce comportement est la suivante **[CLE01]**:

$$\begin{cases} v(t+1) = \lambda_1 v(t) + \lambda_2 \left(p_i(t) - x(t)\right) + \lambda_3 \left(p_g(t) - x(t)\right) & (1) \\ x(t+1) = x(t) + v(t+1) & (2) \end{cases}$$

où t est le temps, x la position de la particule, v sa vitesse, p_i sa meilleure position atteinte, p_g la meilleure des meilleures positions atteintes dans son voisinage et λ_1, λ_2, λ_3 les coefficients de confiance pondérant les trois directions possibles (*volontariste*, *conservatrice*, *suiviste*), sont choisis à chaque pas de temps au hasard dans un intervalle donné.

IV.5.3. Algorithme de base

[Initialisation]
Initialiser aléatoirement la population
[Traitement]
Répéter
  Pour chaque particule
    Évaluer la valeur de la fitness
    Modifier la vitesse en utilisant l'équation (1)
    Puis déplacer en utilisant l'équation (2)
  Fin pour
Jusqu'à ce que (*le processus converge*)

Algorithme 1.1 : algorithme de base de l'OEP.

L'algorithme s'exécute tant qu'un critère de convergence n'a pas été atteint. Cela peut être **[DUO00]**:

- Un nombre fixe d'itérations ;
- En fonction de la fitness ;
- Lorsque la variation de vitesse est proche de 0.

IV.5.4. Paramètres de l'algorithme

L'algorithme OEP comprend plusieurs paramètres de réglage qui permettent d'agir sur le compromis *exploration – exploitation*.

⇒ L'*exploration* est la capacité de tester différentes régions de l'espace à la recherche de bonnes solutions candidates.

⇒ L'*exploitation* est la capacité de concentrer la recherche autour des solutions prometteuses afin de s'approcher le plus possible de l'optimum **[TRE03]**.

Parmi ces paramètres, on peut citer **[DUO00]**:

1. Le nombre de particules ;
2. Les valeurs des coefficients λ ;
3. La taille et la définition du voisinage ;
4. La vitesse maximale.

⇒ *Voisinage*

Le voisinage constitue la structure du réseau social. Les particules à l'intérieur d'un voisinage communiquent entre-elles. Différents voisinages ont été étudiés **[DUO00]**:

- Topologie en étoile **(a)** : le réseau social est complète, chaque particule est attirée vers la meilleure particule et communique avec les autres.

- Topologie en anneau **(b)** : chaque particule communique avec *n* voisines immédiates. Chaque particule tend à se déplacer vers la meilleure dans son voisinage local.
- Topologie en rayon **(c)** : une particule "centrale" est connectée à toutes les autres. Seule cette particule centrale ajuste sa position vers la meilleure, si cela provoque une amélioration l'information est propagée aux autres.

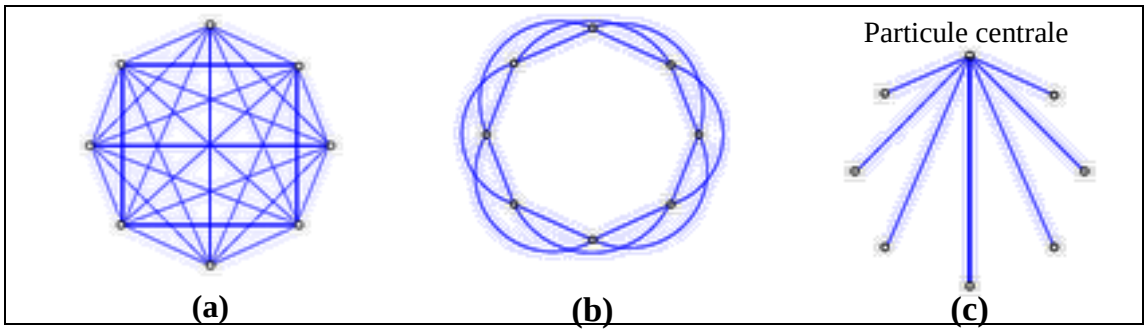


Figure 1.11 : trois topologies différentes.

Le voisinage le plus utilisé est celui dit « circulaire ».

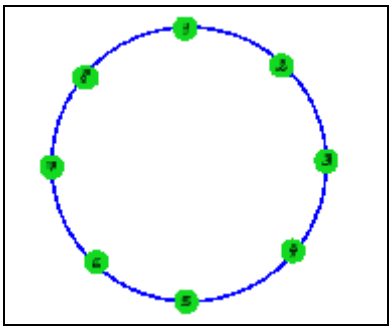


Figure 1.12 : topologie en cercle.

Les particules sont numérotées, disposées virtuellement sur un cercle et la définition des voisinages se fait une fois pour toutes en fonction des numéros **[CLE03]**.

⇒ *Vitesse maximale*

Pour éviter que les particules se déplacent trop rapidement d'une région à une autre dans l'espace de recherche, on fixe une vitesse maximale (n'est pas obligatoire) **[DUO00]**.

IV.5.5. Domaines d'application

L'OEP n'est pas adaptée à la résolution de systèmes linéaires, car il existe des méthodes spécialisées bien plus efficaces. Par contre elle peut traiter presque naturellement les problèmes multiobjectifs, ceux avec des valeurs floues (du fait de sa robustesse), ceux dont la fonction à minimiser change au cours même du processus (du fait de sa rapidité) et, enfin, les problèmes hétérogènes (dimensions non toutes de même nature) **[MCL03]**.

V. Conclusion

Nous avons vu dans ce chapitre quelques généralités sur les métaheuristiques modernes où chacune d'entre elles a des inspirations soit de la physique comme le recuit simulé qui trouve ses origines dans la thermodynamique, soit de la biologie comme les algorithmes génétiques qui s'inspirent du fonctionnement de l'évolution naturelle, notamment la sélection de Darwin et la procréation selon les règles de Mendel ou bien de l'éthologie comme les colonies de fourmis qui s'inspire du comportement des fourmis lorsque celles-ci sont à la recherche de nourriture et l'essaim de particules qui s'inspire du comportement des insectes sociaux tels que les essaims d'oiseaux ou d'abeilles et les bancs de poissons.

Dans le chapitre suivant nous introduirons le principe de fonctionnement de chacune de ces métaheuristiques pour résoudre un problème d'optimisation difficile.

Chapitre 2 :

Métaheuristiques pour l'optimisation difficile

I. Introduction

Dans la vie courante, nous sommes fréquemment confrontés à des problèmes plus ou moins complexes. Ces problèmes peuvent être exprimés sous la forme générale d'un problème d'optimisation.

Dans ce chapitre, nous décrivons le cadre de l'optimisation difficile et des métaheuristiques et nous présentons le principe de fonctionnement de chacune des métaheuristiques modernes dans le cas où on veut résoudre un problème d'optimisation difficile tel que le problème type du voyageur de commerce.

En théorie, un voyageur de commerce désire visiter un certain nombre de villes, débutant et finissant son parcours dans la même ville en visitant chacune des autres villes une et une seule fois. Il désire sélectionner la tournée qui minimise la distance totale parcourue.

II. Vue générale

Les ingénieurs et les décideurs sont confrontés quotidiennement à des problèmes de complexité grandissante, qui surgissent dans des secteurs techniques très divers, comme dans la conception de systèmes mécaniques, le traitement des images, l'électronique ou la recherche opérationnelle. Le problème à résoudre peut souvent s'exprimer comme un *problème d'optimisation* : on définit une fonction objectif, ou fonction de coût (voire plusieurs), que l'on cherche à minimiser ou à maximiser par rapport à tous les paramètres concernés. La définition du problème d'optimisation est souvent complétée par la donnée de *contraintes*. On distingue les contraintes *impératives* (ou « dures ») et les contraintes *indicatives* (ou « molles »). Tous les paramètres des solutions retenues doivent respecter les contraintes du premier type, qui peuvent aussi être vues comme définissant l'espace de recherche, faute de quoi ces solutions ne sont pas réalisables. Les contraintes indicatives doivent simplement être respectées aussi bien que possible. De nouvelles méthodes, dénommées *métaheuristiques*, comprenant notamment la méthode du recuit simulé, les algorithmes évolutionnaires, la méthode de recherche tabou, les algorithmes de colonie de fourmis… sont apparues, à partir des années 1980, avec une ambition commune : résoudre *au mieux* les problèmes dits d'*optimisation difficile*.

Pour tenter de cerner le domaine - mal défini - de l'optimisation difficile, il est nécessaire de faire la distinction entre deux types de problèmes

d'optimisation : les problèmes « discrets » et les problèmes à variables continues. Citons un exemple de chaque type, pour fixer les idées. Parmi les problèmes discrets, on trouve le célèbre problème du voyageur de commerce : il s'agit de minimiser la longueur de la tournée d'un « voyageur de commerce », qui doit visiter un certain nombre de villes, avant de retourner à la ville de départ. Dans la catégorie des problèmes continus, un exemple classique est celui de la recherche des valeurs à affecter aux paramètres d'un modèle numérique de processus, pour que ce modèle reproduise au mieux le comportement réel observé. En pratique, on rencontre aussi des « problèmes mixtes », qui comportent à la fois des variables discrètes et des variables continues.

Revenons sur la définition de l'optimisation difficile. Deux sortes de problèmes reçoivent, dans la littérature, cette appellation, non définie strictement (et liée, en fait, à l'état de l'art en matière d'optimisation) :

- certains problèmes d'optimisation discrète, pour lesquels on ne connaît pas d'algorithme exact *polynomial* (c'est-à-dire dont le temps de calcul est proportionnel à N^n, où N désigne le nombre de paramètres inconnus du problème, et n est une constante entière). C'est le cas, en particulier, des problèmes dits « NP-difficiles », pour lesquels on conjecture qu'il n'existe pas de constante n telle que le temps de résolution soit borné par un polynôme de degré n.
- certains problèmes d'optimisation à variables continues, pour lesquels on ne connaît pas d'algorithme permettant de repérer un *optimum global* (c'est-à-dire la meilleure solution possible) à coup sûr et en un nombre fini de calculs.

Des efforts ont longtemps été menés, séparément, pour résoudre ces deux types de problèmes. Dans le domaine de l'optimisation continue, il existe ainsi un arsenal important de méthodes classiques dites d'*optimisation globale*, mais ces techniques sont souvent inefficaces si la fonction objectif ne possède pas une propriété structurelle particulière, telle que la convexité. Dans le domaine de l'optimisation discrète, un grand nombre d'*heuristiques*, qui produisent des solutions proches de l'optimum, ont été développées ; mais la plupart d'entre elles ont été conçues spécifiquement pour un problème donné **[CLS04]**.

III. Problème d'optimisation

Un problème d'optimisation au sens général est défini par un ensemble de solutions possibles S, dont la qualité peut être décrite par une fonction objectif f. On cherche alors à trouver la solution s^* possédant la meilleure qualité $f(s^*)$ (par la suite, on peut chercher à minimiser ou à maximiser $f(s)$). Un problème d'optimisation peut présenter des contraintes d'égalité (ou d'inégalité) sur s, être dynamique si $f(s)$ change avec le temps ou encore multi-objectif si plusieurs fonctions objectifs doivent être optimisées.

Il existe des méthodes déterministes (dites "exactes") permettant de résoudre certains problèmes en un temps fini. Ces méthodes nécessitent

généralement un certain nombre de caractéristiques de la fonction objectif, comme la stricte convexité, la continuité ou encore la dérivabilité. On peut citer comme exemple de méthode la programmation linéaire, quadratique ou dynamique, la méthode du gradient, la méthode de Newton, … etc. **[DRE04]**

IV. Processus d'optimisation

La **Figure 2.1** présente le processus d'optimisation en trois étapes : analyse, synthèse et évaluation **[MAG06]**.

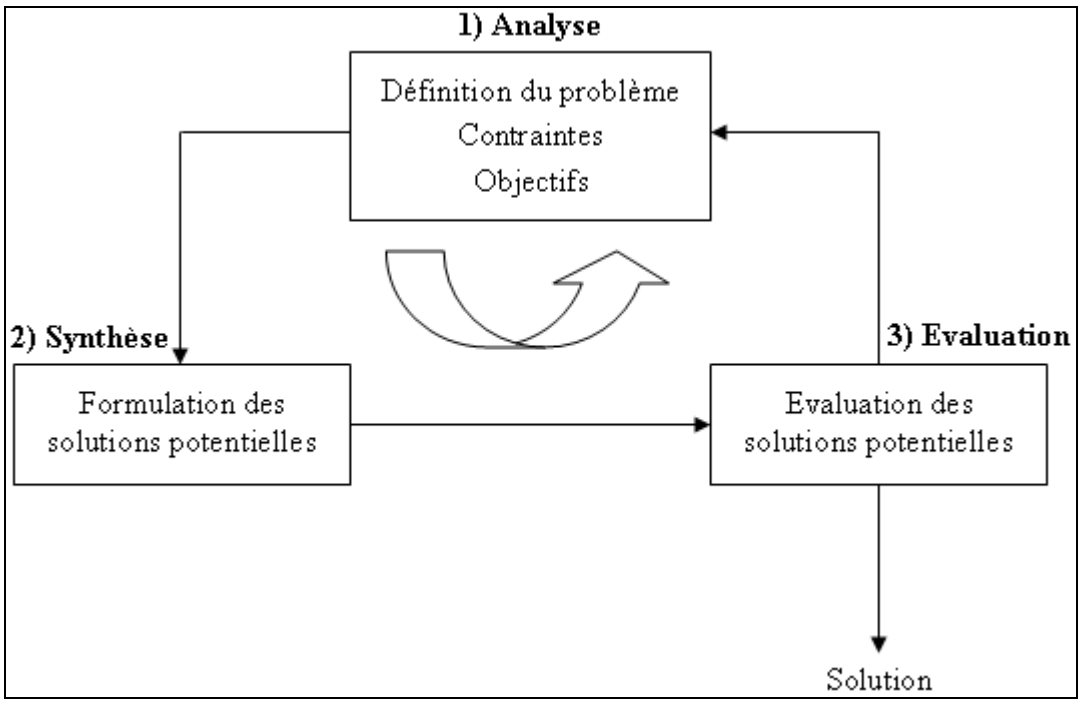


Figure 2.1 : processus d'optimisation.

Tout d'abord, il convient d'analyser le problème et d'opérer un certain nombre de choix préalables :

IV.1. Variables du problème

Quels sont les paramètres intéressants à faire varier ?

C'est à l'utilisateur de définir les variables du problème. Il peut avoir intérêt à faire varier un grand nombre de paramètres pour augmenter les degrés de liberté de l'algorithme afin de découvrir des solutions nouvelles. Autrement, s'il a une vue suffisamment précise de ce qu'il veut obtenir, il peut limiter le nombre de variables à ceux qui sont essentielles **[MAG06]**.

IV.2. Espace de recherche

Dans quelles limites faire varier ces paramètres ?

Dans certains algorithmes d'optimisation, tels que les *stratégies d'évolution*, l'espace de recherche est infini : seule la population initiale est confinée dans un espace fini. Mais dans le cas des algorithmes de type *Monte Carlo* et *génétique*, il est généralement nécessaire de définir un espace de recherche fini. Cette limitation de l'espace de recherche n'est généralement pas

problématique. En effet, ne serait-ce que pour des raisons technologiques ou informatiques (taille de la fenêtre de modélisation), les intervalles de définition des variables sont en général naturellement limités **[MAG06]**.

IV.3. Fonctions objectif

Quels sont les objectifs à atteindre ? Comment les exprimer mathématiquement ?

Un algorithme d'optimisation nécessite généralement la définition d'une fonction rendant compte de la pertinence des solutions potentielles, à partir des grandeurs à optimiser. Nous la nommerons *fonction d'adaptation f* (ou *fitness function* en terminologie anglo-saxonne). L'algorithme convergera vers un optimum de cette fonction, quelle que soit sa définition. La pertinence de la solution dépendra donc de la pertinence de la " question " posée à l'ordinateur. La fonction *f* doit donc exprimer le plus fidèlement possible le désir de l'utilisateur sous forme mathématique **[MAG06]**.

V. Optimisation difficile

Certains problèmes d'optimisation demeurent cependant hors de portée des méthodes exactes. Un certain nombre de caractéristiques peuvent en effet être problématiques, comme l'absence de convexité stricte (multimodalité), l'existence de discontinuités, une fonction non dérivable, présence de bruit, … etc.

Dans de tels cas, le problème d'optimisation est dit "difficile", car aucune méthode exacte n'est capable de le résoudre exactement en un temps "raisonnable", on devra alors faire appel à des heuristiques permettant une optimisation approchée.

L'optimisation difficile peut se découper en deux types de problèmes : les problèmes discrets et les problèmes continus. Le premier cas rassemble les problèmes de type NP-complets, tels que le problème du voyageur de commerce. Un problème "NP" est dit complet s'il est possible de le décrire à l'aide d'un algorithme polynomial sous la forme d'un sous-ensemble d'instances. Concrètement, il est "facile" de décrire une solution à un tel problème, mais le nombre de solutions nécessaires à la résolution croît de manière exponentielle avec la taille de l'instance. Jusqu'à présent, la conjecture postulant que les problèmes NP-complets ne sont pas solubles en un temps polynomial n'a été ni démontrée, ni révoquée. Aucun algorithme polynomial de résolution n'a cependant été trouvé pour de tels problèmes. L'utilisation d'algorithmes d'optimisation permettant de trouver une solution approchée en un temps raisonnable est donc courante.

Dans la seconde catégorie, les variables du problème d'optimisation sont continues. C'est le cas par exemple des problèmes d'identifications, où l'on cherche à minimiser l'erreur entre le modèle d'un système et des observations expérimentales. Ce type de problème est moins formalisé que le précédent, mais un certain nombre de difficultés sont bien connues, comme l'existence de nombreuses variables présentant des corrélations non identifiées, la présence de

bruit ou plus généralement une fonction objectif accessible par simulation uniquement. En pratique, certains problèmes sont mixtes et présente à la fois des variables discrètes et des variables continues **[DRE04]**.

VI. Métaheuristiques en optimisation

VI.1. Problème du voyageur de commerce

Un voyageur de commerce doit visiter un ensemble de villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ où les distances entre les villes sont connues. Quel chemin faut-il choisir afin de minimiser la distance parcourue ? **[AIL03]**. Une méthode d'énumération exhaustive est exclue : s'il y a *N* villes, pour la seconde étape de son périple notre voyageur a *N*-1 possibilités, pour la troisième *N*-2, … etc. Le nombre de combinaisons en fixant la première ville, est donc (*N*-1)!. Pour seulement 40 villes, cela fait à peu près $2e^{46}$ solutions à tester. En supposant que l'on dispose d'un ordinateur permettant de tester un milliard de solutions par seconde, cela nous prendrait plus de $1e^{19}$ fois l'âge de l'univers pour tester toutes les solutions ! **[MAG03]**.

La complexité du problème croît comme la factorielle du nombre de villes. Si l'on considère des permutations simples, il existe 3 628 800 permutations de 10 villes. Pour 100 villes, on passe à 10 puissance 158 **[REN03]**. Ce problème représente la classe des problèmes NP-complets. L'existence d'un algorithme de complexité polynomiale reste inconnue. Un calcul rapide de la complexité montre qu'elle est en *O(N!)* où *N* est le nombre de villes. En supposant que le temps pour évaluer un trajet est de 1 µs, le **Tableau 2.1** montre l'explosion combinatoire du problème du voyageur de commerce (on considère que le nombre de possibilités est ((*N*-1)! /2) configurations) **[TOL03]**.

Nombre de villes	Nombre de possibilités	Temps de calcul
5	12	12 µs
15	43 milliards	12 heures
25	310 e^{+21}	9,8 milliards d'années

Tableau 2.1 : nombre de possibilités de chemins et temps de calcul en fonction du nombre de villes (on suppose qu'il faut 1 µs pour évaluer une possibilité).

VI.2. Méthode du recuit simulé

Très simple et très rapide à mettre en place cette métaheuristique permet de résoudre de nombreux problèmes d'optimisation, tel que le célèbre problème du voyageur de commerce :

Une solution à l'utilisation de cet algorithme pour le système initial pourrait être générée par la méthode du voisin le plus proche. Elle consiste en la construction, via des itérations successives, d'une permutation de l'ensemble des villes : à chaque itération, on recherche la ville la plus proche de l'extrémité de la permutation déjà construite. Cette méthode donne une solution très moyenne du problème mais permet de rapidement disposer d'une solution fonctionnelle.

Dans notre exemple la notion d'énergie est remplacée par la longueur du chemin et la modification élémentaire par la permutation de 2 villes dans ce chemin **[LAP06]**.

VI.2.1. Avantages et inconvénients

Avantages :

- Très simple et très rapide à mettre en place.
- Convergence vers un optimum global démontrée via des chaînes de Markov : la prédiction du futur à partir du présent ne nécessite pas la connaissance du passé. Plus clairement cette métaheuristique ne nécessite pas de mémoire (passé) afin de trouver les espaces de recherche locaux suivants (futur). Cela signifie également que contrairement à d'autres métaheuristiques le recuit simulé peut trouver la meilleure solution si on le laisse chercher indéfiniment.

Inconvénients :

- La non utilisation de mémoire bride les possibilités.
- Il faut déterminer les paramètres à la main : température initiale, modification élémentaire... en testant divers valeurs **[LAP06]**.

VI.3. Algorithmes génétiques

Pour la résolution du problème du voyageur de commerce un algorithme génétique se caractérise par différents points :

- une instance du problème
- un espace de recherche
- le codage des points de l'espace de recherche (par un chromosome)
- la fonction d'évaluation
- les opérateurs génétiques classiques (sélection, croisement, mutation)
- diversification

L'implantation du problème demande une précision pour chacun de ces points avec les options choisies :

VI.3.1. Instance du problème

Une instance du problème du voyageur de commerce est un graphe complet de n sommets dont les arêtes sont pondérées par un coût strictement positif. L'instance sera alors implantée comme une matrice $M\,n\times n$ dont les coefficients sont strictement positifs sauf sur la première diagonale où ils sont tous nuls. M est appelé matrice de coût. Ainsi la distance entre le sommet j et le sommet i est M_{ij}.

VI.3.2. Espace de recherche

C'est l'ensemble S_n des permutations de $\{1,2,\ldots,n\}$. Un point de l'espace de recherche est une permutation.

VI.3.3. Codage des points de l'espace de recherche

Chaque points de l'espace de recherche est une permutation de $\{1,2,\ldots,n\}$. Une première idée est de coder alors chaque permutation par une chaîne de bits. Par exemple, pour $n = 8$:

011 111 000 100 001 010 101 110

représente la permutation :

3 7 0 4 1 2 5 6

Cependant après quelques hybridations, on risque d'avoir un chromosome qui ressemble à :

011 111 011 100 000 000 111

à savoir

3 7 3 4 0 0 7

ce qui ne représente plus une permutation. Il suffit alors de répartir les chromosomes comme une chaîne d'entiers et d'adapter les opérateurs génétiques à ce codage. Notons que les sommets sont répartis à partir de 0.

VI.3.4. Fonction d'évaluation

C'est ici le coût total d'une permutation. À savoir pour une permutation σ :

$$f(\sigma) = \sum_{i=0}^{n-2} M_{\sigma(i)\sigma(i+1)} + M_{\sigma(n)\sigma(0)}$$

Le dernier terme de la somme permettant de revenir au sommet initial. C'est cette fonction là que nous cherchons à minimiser.

VI.3.5. Opérateurs génétiques classiques

⇒ **Sélection**

La valeur moyenne de la fonction d'évaluation dans la population est :

$$\overline{f} = \frac{1}{m} \sum_{i=0}^{m-1} f(p_i)$$

où p_i est l'individu i de la population et m la taille de la population. Les individus sont triés où la place d'un individu p_i est proportionnel à :

$$\frac{\overline{f}}{f(p_i)}$$

Alors sélectionner *m*/2 individus pour la reproduction. Il y a aussi la possibilité d'avoir une politique d'« élitisme ». C'est à dire qu'à chaque étape de sélection le meilleur chromosome est automatiquement sélectionné.

⇒ **Croisement**

Une fois la population intermédiaire sélectionnée, on complète la population avec les enfants de la population intermédiaire. Deux chromosomes se combinant donnent naissance à deux autres chromosomes de la façon suivante :

- Un point d'hybridation est déterminé aléatoirement (entre 0 et la longueur du chromosome).
- Copier dans le premier fils les indices du premier père jusqu'au point d'hybridation.
- Compléter ensuite par les indices du deuxième chromosome père ne se trouvant pas déjà dans le fils dans l'ordre donné par le deuxième parent.
- Réitérer 2) et 3) avec le même point d'hybridation, mais en inversant le rôle des deux parents.

Par exemple, pour n = 8, on a les deux chromosomes suivants qui s'hybrident :

$$\boldsymbol{C1 = (1\ 0\ 5\ 3\ 2\ 7\ 4\ 6)}$$

$$\boldsymbol{C2 = (7\ 1\ 3\ 5\ 6\ 2\ 0\ 4)}$$

On hybride à partir du rang 5. On arrive alors aux deux nouveaux chromosomes :

$$\boldsymbol{C1 = (1\ 0\ 5\ 3\ 2\ 7\ 6\ 4)}$$

$$\boldsymbol{C2 = (7\ 1\ 3\ 5\ 6\ 0\ 2\ 4)}$$

⇒ **Mutation**

Une des méthodes les plus simples pour muter un chromosome est d'inverser les positions de deux villes. Par exemple :

$$\boldsymbol{C3 = (0\ 1\ 3\ 6\ 4\ 7\ 5\ 2)}$$

devient $\boldsymbol{C3' = (0\ 1\ 7\ 6\ 4\ 3\ 5\ 2)}$

VI.3.6. Diversification

La diversification consiste à mieux répartir les individus de la population dans l'espace de recherche. Ici, on fait un décalage vers la gauche de longueur aléatoire sur tous les individus nouvellement créés. Cela permet une meilleure diversité génétique **[TOS03]**.

VI.3.7. Avantages et inconvénients

Avantages :

- Aucune hypothèse à faire sur l'espace de recherche.

- Nombreuses méthodes disponibles.
- Solutions intermédiaires interprétables.
- Adaptation rapide à de nouveaux environnements.
- Co-évolution (tournoi), parallélisme et distribution aisés.
- Les représentations facilitent la compréhension.

Inconvénients :

- Aucune garantis de solution optimale en un temps fini.
- Initialisation de plusieurs paramètres, choix des méthodes important.
- Coût d'exécution important **[LAP06]**.

VI.4. Optimisation par colonie de fourmis

Cette méthode consiste à placer des fourmis sur différentes villes et chaque fourmi à son tour choisit une nouvelle ville de manière probabiliste en fonction de **[CAN99]**:

- La quantité de phéromone ;
- Une fonction de la distance à la prochaine ville ;
- Sa mémoire interne qui lui indique les villes déjà parcourues.

Chaque fois qu'une fourmi se déplace elle laisse une trace (*local trail updating*). Celle qui a trouvé le chemin de longueur minimum laisse une trace plus importante (*global trail updating*).

VI.4.1. Algorithme

Soit *Dist*(*i*,*j*) la distance entre les villes *i* et *j* et *Phero*(*i*,*j*) la quantité de phéromone sur le chemin entre ces deux villes, *N* le nombre de villes et *M* le nombre de fourmis.

⇒ **Initialisation**

- Initialiser le tableau des traces avec : $Phero(x, y) = T_0$ où T_0 une quantité de phéromone élémentaire calculée par : $T_0 = 1/(N * Lnn)$. La distance *Lnn* est la distance calculée par l'algorithme glouton du plus proche voisin (*nearest neighbor*) qui consiste à choisir une première ville au hasard et de construire un chemin en allant vers la ville la plus proche n'appartenant pas déjà au chemin.
- Tirer au hasard les *M* villes de départ pour les *M* fourmis.

```
- Initialisation
- Boucle principale
  Pour iter =1,.., NbIter Faire
    Pour i =1,.., N Faire
      Pour k =1,.., M Faire
        choix de la prochaine ville pour la fourmi k
        dépôt de la trace locale
      FinPour
    FinPour
    Choix de la fourmi ayant le plus court chemin
    Dépôt de la trace globale
  FinPour
```

Algorithme 2.2 : algorithme de colonie de fourmis pour résoudre le problème du voyageur de commerce.

VI.4.2. Choix de la ville

Soit q un nombre aléatoire entre 0 et 1 et soit v_k la position de la fourmi k, si $q \leq q_0$ alors choisir parmi les villes candidates u celle qui maximise :

$$\max_u \frac{Phero(v_k,u)}{Dist(v_k,u)^2}$$

sinon choisir une ville au hasard selon la loi de probabilité suivante :

$$p_k(u) = \frac{Phero(v_k,u)/Dist(v_k,u)^2}{\sum_v Phero(v_k,v)/Dist(v_k,v)}$$

VI.4.3. Trace locale

La trace locale est calculée pour chaque couple (x,y) de ville parcourue par :

$$Phero(x,y) = Phero(x,y)(1-\rho) + T_0$$

VI.4.4. Trace globale

Pour chaque couple de ville (x,y) appartenant au meilleur chemin trouvé il y a :

$$Phero(x,y) = Phero(x,y)(1-\alpha) + \frac{\alpha}{L_{meilleur}}$$

où $L_{meilleur}$ est la longueur du meilleur chemin trouvé.

VI.4.5. Efficacité de l'algorithme

L'efficacité de cet algorithme dépend de la quantité de phéromones déposées par une fourmi et la vitesse à laquelle elles s'évaporent. Ces deux paramètres sont réglés arbitrairement **[TSA03]**.

VI.4.6. Avantages et inconvénients

Avantages :

- Très grande adaptabilité.
- Parfait pour les problèmes basés sur des graphes.

Inconvénients :

- Un état bloquant peut arriver.
- Temps d'exécution parfois long.
- Ne s'applique pas à tous type de problèmes **[LAP06]**.

VI.5. Optimisation par essaim de particules

L'optimisation par essaim de particules peut facilement être adaptée à tout problème discret pour lequel on ne disposerait pas de bon algorithme spécialisé. Nous présentons en détail comment elle peut être appliquée au problème bien connu du voyageur de commerce.

VI.5.1. Espace de recherche

Soit $G = \{E_G, V_G\}$ le graphe évalué dans lequel nous cherchons des cycles Hamiltonien. E_G est l'ensemble des nœuds et V_G l'ensemble des arcs pondérés. Les nœuds du graphe sont numérotés de 1 à N et chaque élément de V_G est un triplet :

$$(i, j, w_{i,j}), i \in \{1,...,N\}, j \in \{1,...,N\}, w_{i,j} \in R$$

Comme nous cherchons des cycles, nous pouvons considérer juste les suites de N+1 nœuds, tous différents, sauf le dernier égal au premier. Ici, une telle suite est appelée un N-cycle et vue comme une "*position*". Donc l'espace de recherche est défini comme l'ensemble fini de tous les N-cycles **[CLE00]**.

VI.5.2. Fonction objectif

$$x = (n_1, n_2, ..., n_N, n_{N+1}), n_i \in E_G, n_1 = n_{N+1}$$

Une position x est acceptable, si seulement si, tous les arcs (n_i,n_{i+1}) existent. Dans le graphe, chaque arc existe comme une valeur et afin de définir la fonction "coût", une manière classique pour juste compléter le graphe consiste à créer des arcs non existants avec une valeur arbitraire l_{sup} assez grande pour assurer qu'aucune solution pourrait contenir un arc "virtuel", par exemple :

$$\begin{cases} l_{sup} > l_{max} + (N-1)(l_{max} - l_{min}) \\ l_{max} = MAX(w_{i,j}) \\ l_{min} = MIN(w_{i,j}) \end{cases}$$

Donc, chaque arc (n_i,n_{i+1}) a une valeur réel ou virtuel.

Maintenant, pour chaque position, une fonction objectif possible peut simplement être définie par :

$$f(x) = \sum_{i=1}^{N-1} w_{n_i, n_{i+1}}$$

Cette fonction objectif possède un nombre fini de valeurs et son minimum global est, en effet, la meilleure solution **[CLE00]**.

VI.5.3. Formulations

En rappel, la formulation générale du mouvement d'une particule est la suivante **[CLE00]**:

$$\begin{cases} v_i(t+1) = c_1 v_i(t) + c_2\left(p_i - x_i(t)\right) + c_3\left(p_g - x_i(t)\right) \\ x_i(t+1) = x_i(t) + v_i(t+1) \end{cases}$$

où :

$v_i(t)$: la vitesse de la particule *i* au pas de temps *t*
$x_i(t)$: la position courante de la particule *i* au pas de temps *t*
p_i : la meilleure position trouvée par la particule *i*
p_g : la meilleure position trouvée par le voisinage de la particule *i*
c_1, c_2, c_3 : les coefficients de confiance pondérant les trois directions possibles (*volontariste*, *conservatrice*, *suiviste*). Dans presque toutes les applications, les coefficients c_2 et c_3 sont choisis à chaque pas de temps au hasard dans un intervalle donné.

En pratique, nous pouvons considérer que $c_3 = c_2$ et en définissant une position intermédiaire :

$$p_{ig} = p_i + \frac{1}{2}\left(p_g - p_i\right)$$

Le système est finalement formalisé comme suit :

$$\begin{cases} v_i(t+1) = c_1 v_i(t) + c_2'\left(p_{ig} - x_i(t)\right) \\ x_i(t+1) = x_i(t) + v_i(t+1) \end{cases}$$

VI.5.4. Vitesse

Nous pouvons définir une "*vitesse*" *v* comme une permutation d'éléments ou bien une liste de transpositions quand on l'applique à une position pendant un pas de temps, elle nous donne une autre position. Alors, une vitesse est définie par **[CLE00]**:

$v = ((i_k, j_k)), i_k \in \{1,...,N\}, j_k \in \{1,...,N\}, k \uparrow_1^{\|v\|}$ où $\|v\|$ est la longueur de cette vitesse.

VI.5.5. Voisinage

Il y a beaucoup de façons pour définir un "*voisinage*", mais nous pouvons distinguer deux types **[CLE00]**:

- Le voisinage "*physique*", qui prend en considération des distances. En pratique, les distances sont recalculées à chaque pas de temps.
- Le voisinage "*social*", qui prend en considération "juste des rapports". En pratique, pour chaque particule, son voisinage est défini comme une liste de particules au commencement même et ne change pas.

VI.5.6. Processus NoHope / ReHope

D'autres options sont utiles pour améliorer le fonctionnement du système, en particulier le processus *NoHope* / *ReHope* **[CLE00]**:

Tests NoHope

- **Critère 0**

  Si une particule doit se déplacer vers une autre qui est à la distance 1, il ne se déplace pas du tout, ou bien il va exactement à la même position de celle la, selon les coefficients de confiance. Aussitôt que la taille d'essaim est plus petite que $N(N-1)$, il peut arriver que tous les mouvements calculés selon la formulation ci-dessus sont nuls. Dans ce cas, il n'y a absolument aucun espoir "NoHope" pour améliorer la meilleure solution actuelle.

- **Critère 1**

  Ce critère est défini : "l'essaim trop petit". Il doit calculer la taille d'essaim à chaque pas de temps, qui est coûteux. Aussitôt que la distance entre deux particules a tendance à devenir "trop petite", les deux particules deviennent identiques (d'abord par des positions et ensuite par des vitesses). Ainsi, à chaque pas de temps, un essaim "réduit" est calculé, dans lequel toutes les particules sont différentes. Dans ce cas, le test NoHope devient "l'essaim trop réduit", c'est-à-dire de 50%.

- **Critère 2**

  Un autre critère a été ajouté : "l'essaim trop lent". En comparant les vitesses de toutes les particules à un seuil, individuellement ou bien globalement. Dans une version de l'algorithme, ce seuil est en fait modifié à chaque pas de temps, selon le meilleur résultat obtenu et la distribution statistique des valeurs des arcs.

- **Critère 3**

  Un autre critère très simple est défini : "aucune amélioration pendant beaucoup de temps".

Processus ReHope

Aussitôt qu'il n'y a "*aucun espoir*", "*NoHope*", l'essaim est ré-étendu. Il existe deux types de processus ReHope :

- Méthode "*Simple*" : l'essaim est réinitialisé simplement.

- Méthode "*Sophistiquée*" **[CLE99]**:

  - Lazy Descent Method (LDM)

  Chaque particule va en arrière à sa meilleure position précédente et, de là, se déplace aléatoirement et lentement et s'arrête aussitôt qu'il trouve une meilleure position ou quand un nombre maximal de mouvements M_{max} est atteint. Si l'essaim actuel est plus petit que l'initial, il est complété par un nouvel ensemble de particules aléatoirement choisies.

  - Energetic Descent Method (EDM)

  Chaque particule va en arrière à sa meilleure position précédente et, de là, se déplace lentement tant qu'il trouve une meilleure position au maximum en M_{max} mouvements. Si l'essaim actuel est plus petit que l'initial, il est complété par un nouvel ensemble de particules aléatoirement choisies.

  - Local Iterative Levelling (LIL)

  Cette méthode est plus puissante et plus coûteuse. L'idée consiste en ce que, comme il y a une infinité de fonctions objectifs possibles avec le même minimum global, nous pouvons localement et temporairement employer chacune d'entre elles pour guider une particule. Pour chaque voisin immédiat *y* (à la distance 1) de la particule *x*, une fonction objectif provisoire qui estime *f(y)* est calculée, et, selon ces évaluations provisoires, *x* se déplace au meilleur voisin immédiat.

  - Adaptive ReHope Method (ARM)

  Les trois méthodes précédentes peuvent être employées automatiquement d'une façon adaptative, selon pour combien de temps (le nombre de pas de temps) la meilleure solution n'a pas été améliorée. Un exemple de stratégie est comme suit :

Nombre de pas de temps sans amélioration	Type de processus ReHope
0-1	No ReHope
2-3	Lazy Descent Method
4	Energetic Descent Method
>4	Local Iterative Levelling

Tableau 2.2 : exemple de stratégie.

VI.5.7. Extra-meilleure particule

Pour accélérer le processus, l'algorithme peut aussi utiliser une extra-particule, juste pour garder la meilleure position trouvée dès le début. Ce n'est pas absolument nécessaire, car cette position est aussi retenue comme "le meilleur précédent" dans au moins une particule, mais elle pour éviter une séquence d'itérations entre deux processus ReHope **[CLE00]**.

VI.5.8. Version parallèle et séquentielle

L'algorithme peut s'exécuter dans le mode parallèle (simulé) ou bien dans le mode séquentiel **[CLE00]**:

- Dans le mode "*parallèle*", à chaque pas de temps, des nouvelles positions sont calculées pour toutes les particules et ensuite l'essaim est généralement déplacé.
- Dans le mode "*séquentiel*", chaque particule est déplacée en un pas de temps sur une voie cyclique. Ainsi, particulièrement le meilleur voisin employé au temps t+1 ne peut pas être désormais le même comme le meilleur voisin au temps t, même si l'itération n'est pas complète.

VII. Conclusion

L'optimisation est un sujet central dans la recherche opérationnelle et un grand nombre de problèmes pouvant être décrits sous la forme de problèmes d'optimisation difficile. Les problèmes de routage de véhicule, problèmes d'affectation quadratique, problèmes d'ordonnancement, problèmes de coloration de graphes, problèmes de la recherche du plus court chemin sont, par exemple, des problèmes d'optimisation difficile.

Dans ce chapitre nous avons passé une vue générale sur le fonctionnement des métaheuristiques pour résoudre les problèmes d'optimisation difficile.

Dans le chapitre suivant, nous présenterons quelques travaux dont le sujet est l'adaptation des métaheuristiques aux problèmes d'optimisation continue.

Chapitre 3 :

Adaptation des métaheuristiques aux problèmes d'optimisation continue

I. Introduction

Depuis plusieurs années, les métaheuristiques se sont imposées comme des moyens efficaces pour la résolution de problèmes d'optimisation difficile. Plusieurs travaux de recherche ont visé à adapter ces méthodes aux problèmes à variables continues. Les variables du problème doivent être discrétisées, et le choix d'une stratégie de discrétisation économe est particulièrement crucial. Il est nécessaire de définir des solutions voisines de la solution courante, puis de sélectionner la meilleure d'entre elles.

Dans ce chapitre, nous présentons quelques travaux d'adaptation des métaheuristiques, principalement la méthode de recherche tabou, les algorithmes génétiques et la méthode d'optimisation par colonie de fourmis, aux problèmes d'optimisation à variables continues.

II. Adaptation de la méthode tabou

En utilisant la liste tabou et le critère d'aspiration, permettant à l'algorithme de revenir sur ses traces, et d'orienter l'exploration du domaine de recherche dans une autre direction, on peut obtenir une méthode efficace, lorsque l'espace des solutions présente une vallée. Mais cette méthode montre ses limites dans deux cas : lorsque le relief de l'espace des solutions présente un "plateau", la convergence de la méthode est trop lente, et la précision des solutions est médiocre ; lorsque ce relief présente des puits très profonds, l'algorithme, s'il est piégé dans l'un de ces puits, n'arrive plus à en sortir.

Pour pallier à ces deux handicaps majeurs, nous avons besoin de bien adapter le pas de mouvement en fonction du relief du domaine des solutions. Si le relief est aplani, le pas de mouvement doit être réduit, afin d'obtenir la précision voulue ; par contre, si le relief est accidenté, avec des puits profonds, le pas de mouvement doit être grand, afin d'accélérer la convergence, et de plonger rapidement au fond du puits, et afin de pouvoir quitter un puits pour explorer une autre région de l'espace des solutions. Cette adaptation du pas de mouvement est liée à deux concepts, l'*intensification* et la *diversification*. Elle confère à l'algorithme une mémoire à plus *long terme*, qui complète la mémoire à *court terme* concrétisée par la liste tabou **[RCH99]**.

II.1. Principe

Dans ce paragraphe, nous présentons l'un des algorithmes de la recherche tabou appliquée au cas des fonctions à variables continues, qui est dénommé *CPTS* (pour "*Continuous Pure Tabu Search*").

Pour améliorer les performances de la méthode de recherche tabou classique, nous avons défini trois phases essentielles :

- Une première phase, appelée "diversification", permet d'explorer un large espace des solutions, d'éviter que la recherche ne soit trop localisée et ne laisse de grandes régions du domaine totalement inexplorées. Cette phase de recherche permet de détecter plusieurs régions susceptibles de contenir un optimum global de la fonction objectif. Ces régions sont appelées "zones prometteuses".
- La seconde phase permet d'étudier de près ces zones prometteuses déjà détectées et d'en retenir la meilleure.
- La troisième phase, appelée "intensification", permet d'exploiter la meilleure zone retenue et d'approfondir la recherche dans cette zone.

Figure 3.1 : organigramme de l'algorithme CPTS.

L'adaptation de l'algorithme tabou au cas des fonctions à variables continues est basée sur les notions de "voisinage" d'une solution et de "pas" de mouvement **[RCH99]**.

II.2. Notions de "voisinage" et de "pas" de mouvement

Le choix des solutions voisines de la solution courante est important pour l'efficacité d'une heuristique itérative. La définition d'un voisinage *S(x)* de la solution courante *x* dans le cas d'une fonction à variables continues est délicate, et l'évaluation complète de ce voisinage continu est impossible. Dans la pratique, il est nécessaire de définir une stratégie d'échantillonnage de ce voisinage. La manière la plus simple d'échantillonner un voisinage déjà prédéfini consiste à générer aléatoirement un sous-ensemble discret de solutions *x'* voisines de *x* (appartenant à *S(x)*). Cette procédure n'est malheureusement pas la mieux adaptée aux problèmes à variables continues. En effet, les voisins *x'* d'une solution donnée *x*, tirés aléatoirement, peuvent se concentrer dans une même zone. Dans la littérature, plusieurs stratégies de définition du voisinage et d'échantillonnage ont été proposées. Parmi les stratégies les plus répandues, nous allons présenter celle qui subdivise l'espace des solutions en sous-espaces, celles qui utilisent un arbre binaire de partition, celles qui sélectionnent une seule ou une partie des composantes x_i de la variable *x*, en s'assurant que les autres composantes seront considérées aux itérations suivantes, avec un pas de variation Δx_i constant, ou variable dans un voisinage, pour chaque composante x_i de la variable *x* **[RCH99]**.

II.2.1. Subdivision de l'espace des solutions

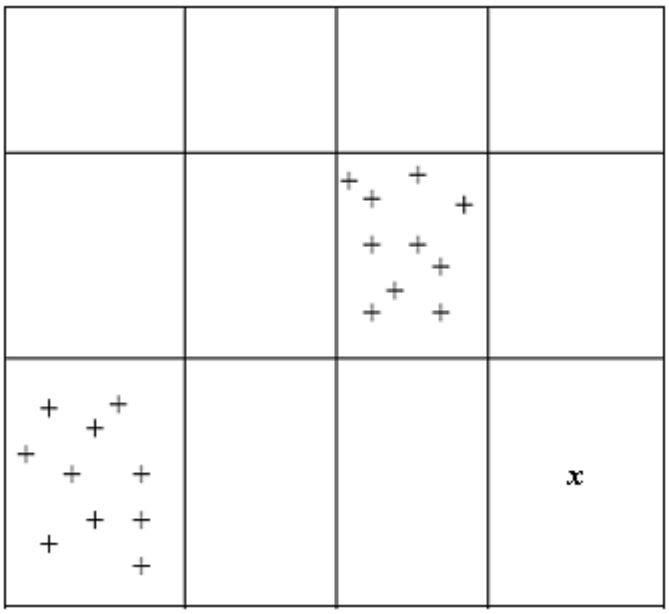


Figure 3.2 : partition de l'espace des solutions et définition du voisinage de la solution *x* (exemple dans le cas $n = 2$).

Une structure de voisinage a été introduite dans un espace continu, qu'ils ont appelée "voisinage conditionnel". L'espace solution *X* (hypercube dans R^n, $X \subset R^n$) est partitionné en cellules disjointes par division des intervalles des coordonnées x_i en p_i parties. Le vecteur de partitionnement empirique $P = (p_1,p_2,...,p_n)$ détermine un partitionnement unique de *X* en cellules *C* de tailles plus ou moins grandes, et spécifie ainsi l'adresse de chaque cellule. À chaque itération, le voisinage de la solution courante *x* est obtenu par tirage, suivant

une loi de distribution uniforme, d'un échantillon de n_s points sur n_c cellules tirées aléatoirement. La taille du voisinage de $S(x)$ est égale à $n_s.n_c$. Si un mouvement s d'une solution x vers x', de la cellule $C(x)$ vers $C(x')$, est accepté, alors l'adresse de la cellule $C(x')$, contenant la solution x', est ajoutée à la liste tabou des adresses des cellules visitées.

L'exemple précédent (P = (4,3), n_s = 10 et n_c = 2) suggère l'inconvénient de cette méthode : le nombre de cellules croît avec la dimension du problème. En outre, du fait du caractère purement aléatoire de cette méthode, une solution acceptée (non tabou), n'appartient pas nécessairement à la cellule qui contient la solution courante. Les pas de mouvement peuvent être importants et l'exploration de l'espace des solutions n'est pas ordonnée **[RCH99]**.

II.2.2. Partitionnement par arbre binaire

Une structure de données dynamique a été introduite pour représenter la structure de voisinage. L'espace des solutions X (hypercube dans R^n, $X \subset R^n$), est représenté par un arbre binaire à k degrés de liberté, où chaque niveau de l'arbre représente une partition binaire d'un de ses degrés de liberté. Chaque nœud de l'arbre pourrait être interprété comme représentant un hyperrectangle parent, et ses nœuds-fils représentent les sous hyperrectangles fils résultant de la division par 2 de l'hyperrectangle parent, suivant un degré de liberté particulier. Le voisinage de la solution courante x est obtenu par tirage, suivant une loi de distribution uniforme, d'un échantillon de n_s points sur les hyperrectangles voisins de celui qui contient x. La **Figure 3.3** illustre un exemple à deux dimensions d'un arbre de partition, et le nombre de niveaux verticaux dans l'arbre correspond aux raffinements successifs.

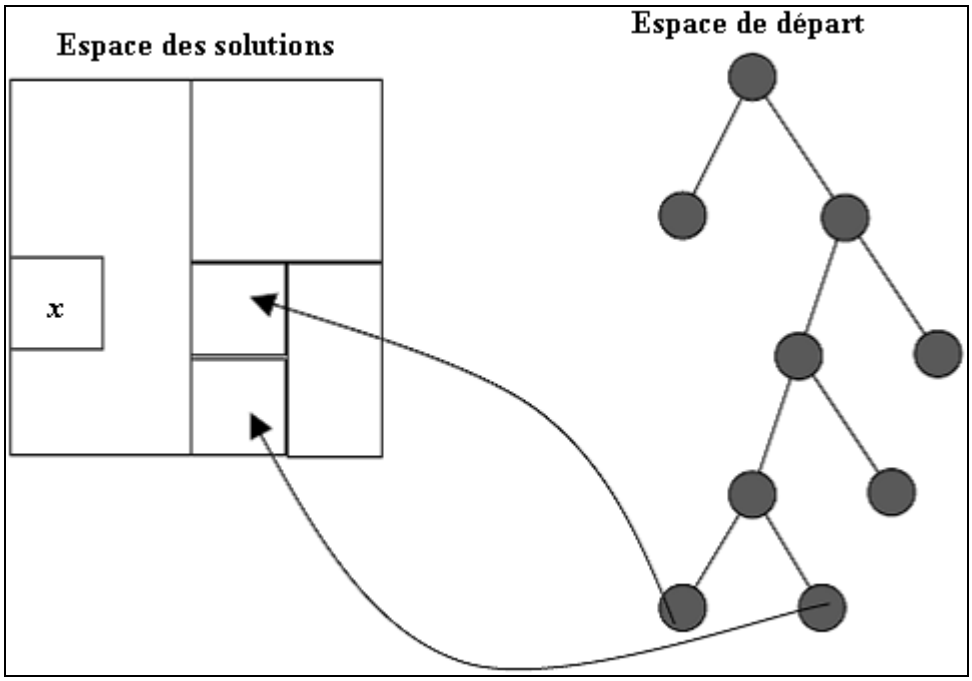


Figure 3.3 : partition de l'espace des solutions à deux dimensions en utilisant un arbre binaire.

Cette méthode est efficace lorsque le nombre de variables est réduit ; dans le cas contraire, on s'expose à une explosion combinatoire des possibilités lorsqu'on veut raffiner la recherche **[RCH99]**.

II.2.3. Mouvement par modification d'une seule composante

Le mouvement dans le voisinage de la solution courante consiste à modifier une par une les composantes de cette solution, et à choisir le meilleur voisin. Le pas du mouvement dans chaque direction est constant. Cela revient à subdiviser l'espace des solutions en hyperrectangles égaux suivant chacune de ses cordonnées, à partir d'une solution *x*, centre de l'un des hyperrectangles, et à définir comme voisins possibles les centres des hyperrectangles voisins. La **Figure 3.4** montre un tel exemple pour une fonction à deux variables **[RCH99]**.

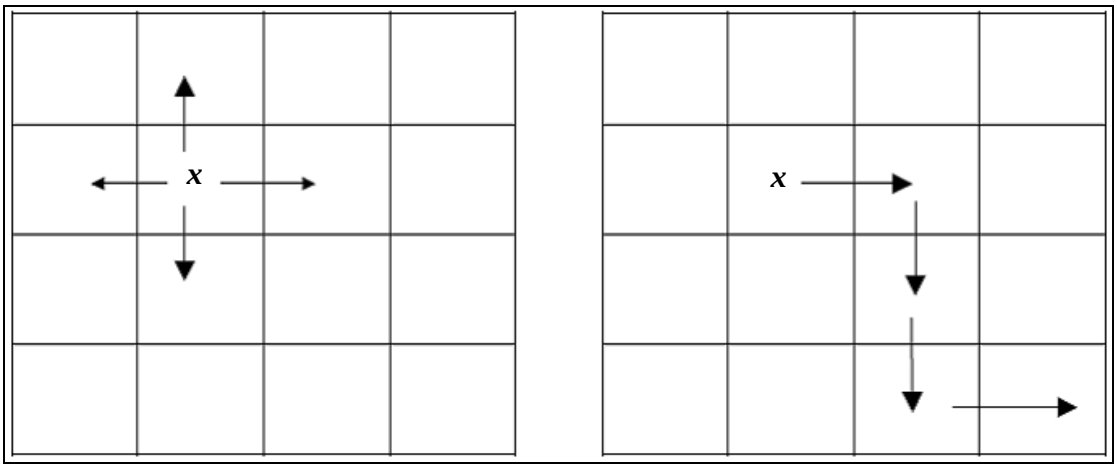


Figure 3.4 : mouvements possibles et exemple de succession de mouvements à l'intérieur de l'espace des solutions.

II.2.4. Notion de voisinage par topologie de boules

Un concept de "boule" est utilisé pour définir le voisinage. Une boule $B(x,\varepsilon)$ est centrée sur x avec un rayon ε ; elle contient tous les points x' tels que : $//x' - x// = \varepsilon$ (le symbole //...// est utilisé pour exprimer la norme euclidienne). Pour obtenir une exploration homogène de l'espace de voisinage, un ensemble de boules centrées a été considéré sur la solution courante *x*, avec des rayons $h_0,h_1,...,h_\eta$. Ainsi un espace partitionné en "couronnes" concentriques $C_i(x,h_{i-1},h_i)$ a été obtenu, telles que :

$$C_i(x,h_{i-1},h_i) = \{x' / h_{i-1} \le // x' - x // \le h_i\}$$

Les η voisins de x ont été obtenus par sélection aléatoire d'un point à l'intérieur de chaque couronne C_i, pour i variant de 1 à η. Dans *CPTS*, nous remplaçons les boules par des hyperrectangles pour la partition du voisinage de la solution courante (**Figure 3.5**), et nous générons des voisins de la solution courante x par modification d'une partie des composantes x_i en nous assurant que les autres composantes seront considérées aux générations suivantes de voisins. Le pas de variation Δx_i pour chaque composante x_i de la variable x est choisi de telle manière que le voisin généré se trouve dans la zone hyperrectangulaire voulue. C'est cette facilité qui nous a incités à utiliser un

voisinage hyperrectangulaire au lieu de couronnes circulaires : en utilisant les coordonnées cartésiennes, il est mathématiquement plus facile de sélectionner un point à l'intérieur d'une zone hyperrectangulaire donnée que de sélectionner un point à l'intérieur d'une couronne limitée par deux boules concentriques **[RCH99]**.

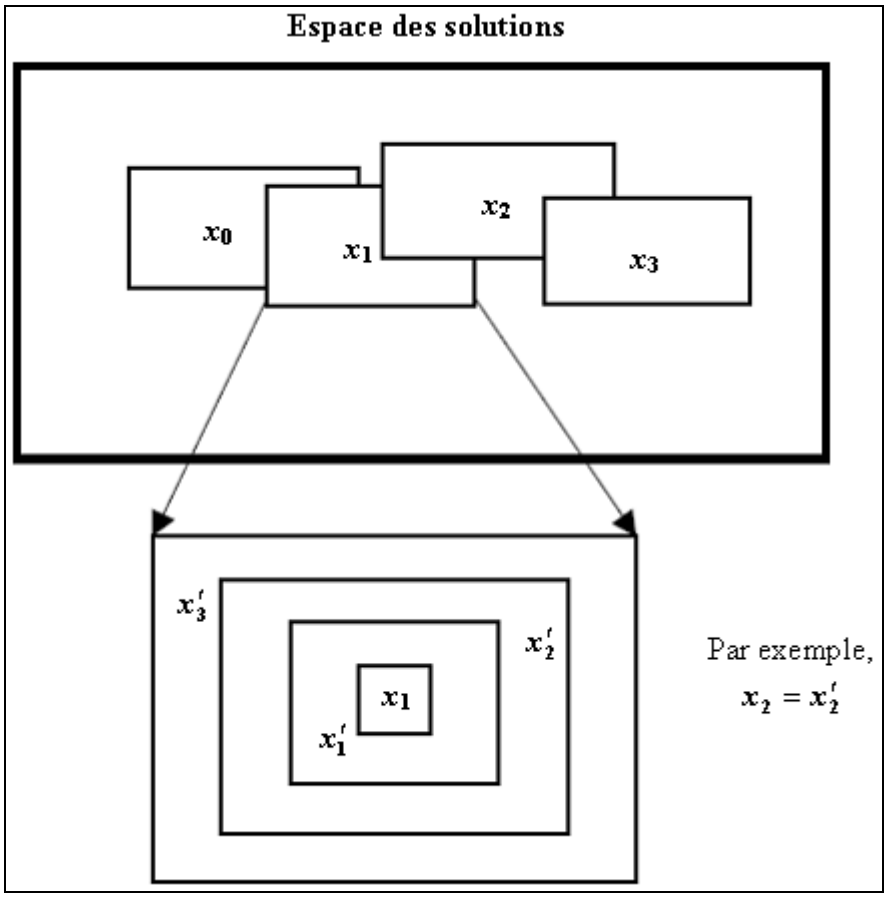


Figure 3.5 : principe de la partition du voisinage de la solution courante mise en œuvre dans CPTS.

II.3. Taille de la liste tabou

La taille de la liste tabou a une grande importance, notamment pour la convergence de la méthode. Il n'existe pas de règle permettant de déterminer une valeur idéale. Le réglage empirique de ce paramètre nécessite plusieurs expérimentations. Pour une application bien définie, après un certain nombre d'expérimentations, on peut déterminer un ordre de grandeur de la taille de la liste, ou la cerner dans un intervalle $[L_{min},L_{max}]$.

Selon les expériences, si la taille de la liste tabou est trop petite, elle ne permet pas d'éviter les cycles, alors que, si elle est trop grande, on risque de "manquer" très fréquemment des solutions intéressantes, on obtient une stagnation de la recherche. Pour que la méthode de recherche tabou soit plus robuste, il est donc intéressant de gérer dynamiquement la taille de la liste tabou **[RCH99]**.

II.4. Présentation détaillée de l'algorithme tabou continu (CPTS)

Deux types de listes ont été définis : la liste tabou et la liste prometteuse. Durant la phase de diversification, la liste prometteuse est de taille moyenne, avec un rayon de voisinage des solutions prometteuses plus grand, et la liste tabou est de grande taille, avec un rayon de voisinage plus réduit. La grande taille de la liste tabou protège l'algorithme contre le cyclage, et évite d'explorer à nouveau une région située autour d'un point récemment visité ; le rayon de voisinage réduit des solutions taboues permet d'avoir une bonne précision. Le grand rayon de voisinage des solutions prometteuses évite à l'algorithme le retour au voisinage des zones prometteuses déjà détectées et le contraint à explorer d'autres régions du domaine de recherche.

Après avoir localisé les zones prometteuses et déterminé la meilleure zone parmi les zones localisées, la phase d'intensification commence avec une liste tabou vide, de taille plus petite que celle utilisée dans la phase de diversification. Lorsque la taille de la liste tabou est fixe, elle est généralement gérée comme une file FIFO ("*First In First Out*"). Pour ajouter une solution dans cette liste, les solutions présentes sont décalées dans un sens, et la place libérée est occupée par la nouvelle solution. Si la liste est pleine, la plus ancienne solution est éliminée.

Ainsi que deux types de boules ont été définies : la boule tabou et la boule prometteuse. On dit qu'un point appartient à la liste tabou ou à la liste prometteuse, s'il est situé dans un voisinage des points appartenant à la liste tabou ou dans un voisinage des points appartenant à la liste prometteuse. La liste tabou contient les dernières solutions retenues par l'algorithme, et la liste prometteuse contient les minima locaux ou globaux rencontrés lors de l'exploration **[RCH99]**.

II.4.1. Diversification

L'idée de la diversification est d'orienter la recherche vers de nouvelles régions non encore explorées, qui peuvent être intéressantes. La stratégie de diversification la plus simple consiste à interrompre périodiquement la procédure et à la faire redémarrer à partir d'une nouvelle solution choisie aléatoirement, ou située dans une zone non visitée jusqu'ici.

Dans cet algorithme, cette phase est assurée grâce à l'utilisation de deux listes : une liste tabou et une liste prometteuse. La liste tabou de taille L_t est initialement vide, on initialise la liste prometteuse en générant un nombre L_p (taille de la table des "solutions prometteuses") de points bien dispersés sur le domaine des solutions. Une solution tirée aléatoirement n'est acceptée dans la liste prometteuse que si elle n'appartient au voisinage d'aucune solution déjà sélectionnée. La liste prometteuse ainsi initialisée procure le seuil d'acceptation d'une nouvelle zone prometteuse : ce seuil est égal à la moyenne des valeurs que prend la fonction objectif sur les points de cette liste. Après cette phase d'initialisation, un point de départ x est tiré aléatoirement dans le domaine des solutions, et *CPTS* démarre sa phase de diversification en utilisant la procédure de recherche tabou contrôlée par les deux listes de points (liste tabou et liste

prometteuse). En partant du point courant *x*, *CPTS* génère un nombre η de voisins de *x* : un point est sélectionné à l'intérieur de chaque hyperrectangle autour de la solution courante.

Chaque voisin est accepté s'il n'appartient ni à la liste tabou, ni à la liste prometteuse. Le meilleur voisin est ajouté à la liste tabou, et devient le nouveau point courant, même s'il est plus mauvais que le précédent. Une nouvelle solution prometteuse est détectée à chaque fois qu'une détérioration "inacceptable" de la fonction objectif apparaît à l'intérieur de la zone de voisinage centrée autour de la solution courante *x*.

TabouList : liste des solutions visitées récemment, de longueur L_t
PromList : liste des minima locaux ou globaux, de longueur L_p
PromList = génération de L_p solutions, bien dispersées sur le domaine de recherche
TabouList = VIDE
x = solution aléatoire

$f_{min} = f(x)$
$x_{min} = x$

REPETER
  générer un η-échantillon TEL QUE $s_j(x) \in$ voisinage $S(x)$
  et $s_j(x) \notin$ *TabouList* et $s_j(x) \notin$ *PromList*
  $f(x) = \min_{1 \le j \le \eta}[f(s_j(x))]$
  $x = s_j(x)$
  ajouter(*x*,*TabouList*)
  SI $f(x) < f_{min}$
    $f_{min} = f(x)$
    $x_{min} = x$
  SINON
    SI (*détection d'une zone prometteuse*)
      remplacer le plus mauvais point de *PromList* par $\{x, f(x)\}$
    FIN SI
  FIN SI
JUSQU'A conditions d'arrêt satisfaites

Figure 3.6 : pseudocode de la phase de diversification.

Cette solution prometteuse définit une nouvelle "zone prometteuse", elle est comparée à la plus mauvaise solution de la liste prometteuse, et la meilleure des deux est conservée. Après l'ajout de cette solution à la liste prometteuse, la valeur du seuil d'acceptation d'une nouvelle solution se remet à jour. L'utilisation de deux listes stimule la recherche de solutions loin de la solution courante et permet ainsi de bien couvrir l'espace de recherche et d'identifier plusieurs zones prometteuses susceptibles de contenir un bon optimum. Après un nombre *MaxEval* d'évaluations successives, sans détection de zone

prometteuse, l'algorithme arrête la phase de diversification et passe le relais à la phase suivante, qui est le choix de la meilleure zone prometteuse. Le pseudocode de la phase de diversification est donné sur la **Figure 3.6 [RCH99]**.

II.4.2. Sélection de la meilleure zone prometteuse

Pour choisir la meilleure zone prometteuse parmi celles déjà localisées, le processus possède trois étapes. Premièrement, la moyenne des valeurs de fonction objectif *f* des solutions présentes dans la liste prometteuse est calculée. Deuxièmement, toutes les solutions pour lesquelles la valeur de la fonction objectif est supérieure à cette valeur moyenne sont éliminées. Troisièmement, la liste prometteuse est traitée ainsi réduite de la manière suivante. Le rayon des boules taboues et la taille du voisinage hyperrectangulaire sont divisés par deux. Pour chaque solution prometteuse restante, η solutions voisines sont générées et la meilleure est sélectionnée. La solution prometteuse et la meilleure solution générée sont comparées ; la meilleure des deux est conservée dans la liste prometteuse, en remplacement de l'ancienne solution prometteuse. Après avoir balayé toute la liste prometteuse, l'algorithme élimine la plus mauvaise solution contenue dans cette liste prometteuse. Ce procédé est réitéré après avoir réduit encore de moitié le rayon des boules tabous et la taille du voisinage hyperrectangulaire. L'algorithme s'arrête lorsqu'il ne reste plus qu'une solution dans la liste prometteuse **[RCH99]**.

PromList : liste des centres de toutes les zones prometteuses localisées, de longueur L_p

$$\overline{f} = \frac{1}{L_p}\sum_{i=1}^{L_p} f(x_i)$$

REPETER
  SI $\overline{f} < f(x_i)$
    éliminer $\left(\{x_i, f(x_i)\}, PromList\right)$
  FIN SI
JUSQU'AU balayage complet de la liste prometteuse
REPETER
  REPETER
    générer un η-échantillon TEL QUE $s_j(x_i) \in$ voisinage $S(x_i)$
    $f(s(x_i)) = \min_{1 \le j \le \eta}[f(s_j(x_i))]$
    SI $f(s(x_i)) < f(x_i)$
      remplacer dans *PromList* le $\{x_i, f(x_i)\}$ par $\{s(x_i), f(s(x_i))\}$
    FIN SI
    JUSQU'AU balayage complet de la liste prometteuse
    éliminer le plus mauvais $\{x_i, f(x_i)\}$ dans *PromList*
JUSQU'A ce qu'il ne reste plus qu'une seule zone prometteuse

Figure 3.7 : pseudocode de la phase de la sélection de la meilleure zone prometteuse.

II.4.3. Intensification

L'objectif de l'intensification est d'approfondir la recherche dans une région susceptible de contenir de bonnes solutions. Plusieurs méthodes peuvent être envisagées pour aller dans cette direction. L'une est d'effectuer une intensification en repartant de la meilleure solution trouvée jusqu'ici avec une liste tabou vide de longueur réduite par rapport à celle utilisée durant la phase d'intensification. La première étape de la phase d'intensification consiste à diminuer la taille de la liste tabou, et à la réinitialiser (liste vide). On définit un nouveau domaine de recherche réduit autour de la meilleure solution trouvée précédemment et un nouveau rayon : respectivement,

$$X = X / \rho_{neigh} \text{ et } \varepsilon_t = \varepsilon_t / \rho_{neigh}$$

où ρ_{neigh} est le coefficient de réduction. Cette solution est prise comme solution courante et la procédure de recherche tabou redémarre : génération de η voisins n'appartenant pas à la liste tabou, sélection du meilleur voisin et insertion de celui-ci dans la liste tabou.

NbRed : nombre de réductions successives du domaine de recherche
$X = X / \rho_{neigh}$ réduction du domaine de recherche
$\varepsilon_t = \varepsilon_t / \rho_{neigh}$ réduction du rayon des boules tabous
TabouList = liste tabou de longueur L_t'
TabouList = VIDE
x = centre de la meilleure zone trouvée précédemment
$f_{min} = f(x)$
$x_{min} = x$
NbRed = 0

REPETER
  générer un η-échantillon TEL QUE $s_j(x) \in$ voisinage $S(x)$
  et $s_j(x) \notin$ *TabouList*
  $f(x) = \min_{1 \le j \le \eta}[f(s_j(x))]$
  $x = s_j(x)$
  ajouter(*x*,*TabouList*)
  SI $f(x) < f_{min}$
    $f_{min} = f(x)$
    $x_{min} = x$
    NbRed = 0
  SINON
    NbRed = *NbRed* + 1
    réduction de moitié du domaine hyperrectangulaire de recherche
    réduction de moitié du rayon des boules tabous
  FIN SI
JUSQU'A conditions d'arrêt satisfaites

Figure 3.8 : pseudocode de la phase d'intensification.

La taille de voisinage hyperrectangulaire et le rayon des boules tabous sont réduits progressivement, par division par deux après un certain nombre d'itérations successives. Les critères d'arrêt portent sur le nombre maximal d'itérations successives *MaxIter*, sans aucune amélioration de la valeur de la fonction objectif, et le nombre maximal de réductions successives *MaxRed* du voisinage hyperrectangulaire et du rayon de la boule tabou, sans aucune amélioration de la valeur de la fonction objectif. Le pseudocode de la phase d'intensification est donné sur la **Figure 3.8 [RCH99]**.

II.5. Présentation de l'algorithme hybride tabou-polytope (CHTS)

II.5.1. Principe

L'introduction de deux listes dans la phase de diversification permet à l'algorithme de bien explorer l'espace de recherche. Au cours de la diversification, la liste tabou de grande taille évite en principe le cyclage de l'algorithme, et la liste prometteuse de taille moyenne interdit à l'algorithme de retourner dans des zones prometteuses déjà localisées. L'intensification permet ensuite d'exploiter la meilleure zone prometteuse détectée.

CPTS est efficace pour explorer un large domaine de recherche et détecter une "vallée" prometteuse, mais prend beaucoup de temps pour intensifier la recherche dans cette vallée. Pour la tâche d'intensification, les algorithmes de recherche locale, tels que la méthode du polytope de Nelder-Mead, convergent plus rapidement et avec une meilleure précision. Pour améliorer la vitesse de convergence de l'algorithme *CPTS*, un algorithme hybride combinant un algorithme de recherche tabou et la méthode du polytope de Nelder-Mead, appelée communément la méthode du "simplex" (*SS*) est développé. Ce nouvel algorithme est appelé ("*Continuous Hybrid Tabu Search*") *CHTS*. Une variante de cet algorithme, appelée *CTSS* pour "*Continuous Tabu Search Simplex*".

Cette nouvelle méthode hybride utilise l'algorithme de recherche tabou pour la diversification et la méthode locale du polytope de Nelder-Mead, pour l'intensification.

La transformation de l'algorithme *CPTS* en *CHTS* nécessite la modification de certains paramètres dans la phase de diversification, la suppression des paramètres de contrôle qui interviennent dans la phase d'intensification de *CPTS* et l'introduction de quelques nouveaux paramètres de contrôle liés à *SS* **[RCH99]**:

- introduction des coefficients (α,β,γ) qui déterminent les pas de mouvement dans la méthode de Nelder-Mead ;
- introduction d'un nouveau critère d'arrêt lié à la méthode du polytope.

II.5.2. Description de l'algorithme

Dans *CPTS* on commençait par localiser toutes les zones prometteuses, on les stockait dans la liste prometteuse, puis on revenait pour les exploiter (phase de sélection de la meilleure zone prometteuse et phase d'intensification). Dans *CHTS*, à chaque localisation d'une zone prometteuse par la recherche tabou, on

pratique immédiatement l'intensification avec la méthode du polytope de Nelder-Mead. Lorsque la précision souhaitée est atteinte, on ajoute la solution obtenue à la liste prometteuse, en remplacement de la plus mauvaise solution contenue dans cette liste, et le processus reprend jusqu'au critère d'arrêt. Dans cet algorithme il y a une alternance permanente entre la phase de diversification et la phase d'intensification. Sur la **Figure 3.9** on donne le pseudocode de cet algorithme **[RCH99]**.

TabouList : liste tabou de longueur L_t
PromList : liste des solutions prometteuses de longueur L_p
PromList = génération de L_p solutions bien dispersées sur le domaine de recherche
TabouList = VIDE

$\overline{f} = \frac{1}{L_p}\sum_{i=1}^{L_p} f(x_i)$ seuil de dégradation inacceptable de la fonction objectif
x = solution aléatoire
$f_{min} = f(x)$
$x_{min} = x$

REPETER *DIVERSIFICATION*
  générer un η-échantillon TEL QUE $s_j(x) \in$ voisinage $S(x)$
    et $s_j(x) \notin$ *TabouList* et $s_j(x) \notin$ *PromList*
  $f(x) = \min_{1 \le j \le \eta}[f(s_j(x))]$
  $x = s_j(x)$
  ajouter(x,*TabouList*)
  SI $f(x) < f_{min}$
    $f_{min} = f(x)$
    $x_{min} = x$
  SINON
    SI (*détection d'une zone prometteuse*) *INTENSIFICATION*
      $X = X / \rho_{neigh}$ réduction du domaine de recherche
      construction du polytope de départ
      après transformations géométriques,
      remplacer le plus mauvais point dans *PromList* par $\{x, f(x)\}$
      $\overline{f} = \frac{1}{L_p}\sum_{i=1}^{L_p} f(x_i)$ nouveau seuil d'acceptation
    FIN SI
  FIN SI
JUSQU'A conditions d'arrêt satisfaites (plus de zones prometteuses)

Figure 3.9 : pseudocode de l'algorithme *CHTS*.

II.5.3. Diversification

Le principe utilisé dans la phase de diversification de *CHTS* est le même que dans la diversification de *CPTS*. La liste tabou est choisie assez grande (de l'ordre de 20). Les définitions du voisinage et du pas de mouvement sont les mêmes que dans *CPTS*. La liste prometteuse contient des points dispersés sur tout l'espace des solutions ; ces points sont considérés comme les centres de régions prometteuses. La liste prometteuse fournit aussi le seuil d'acceptation d'une solution comme centre d'une nouvelle zone prometteuse. La détection d'une zone prometteuse se fait de la même manière que dans *CPTS* **[RCH99]**.

II.5.4. Intensification à l'intérieur d'une zone prometteuse

À chaque détection d'une solution considérée comme centre d'une nouvelle zone prometteuse, un nouveau domaine de recherche centré autour de ce point est construit. Chaque côté du domaine initial est réduit d'un facteur donné, appelé facteur de réduction ρ_{red}. Le polytope de départ S_0 avec le meilleur point trouvé précédemment est construit, soit x_1, et les points x_i choisis dans le nouvel espace des solutions, de telle manière qu'ils forment une base engendrant un espace géométrique, généralement une base orthogonale. La procédure du polytope démarre la recherche en utilisant les différents mouvements (réflexion, contraction, extension et rétrécissement).

L'algorithme accomplit la tâche d'exploitation à l'intérieur de la zone prometteuse, jusqu'à ce que les conditions d'arrêt soient atteintes. Dans *CHTS*, deux critères ont été implémentés :

- le premier est relatif au nombre maximal d'itérations *MaxIter* de la boucle de la procédure du polytope ;
- le second est une mesure de déplacement du polytope d'une itération k à l'itération suivante (k+1) :

$$\frac{1}{n}\sum_{i=1}^{n}\left\|x_i^k - x_i^{k+1}\right\| < \varepsilon$$

où x_{k+1} est le sommet remplaçant le sommet x_k à l'itération (k+1), et ε est un nombre réel positif donné.

Le meilleur point du polytope final, après arrêt de la procédure de *SS*, est ajouté à la liste prometteuse en remplacement de la plus mauvaise solution contenue dans cette liste. Cette solution est considérée comme la nouvelle solution courante. Après la mise à jour du seuil d'acceptation d'une solution comme centre d'une nouvelle zone prometteuse, la procédure de diversification redémarre avec une liste tabou vide **[RCH99]**.

II.5.5. Critères d'arrêt

Après plusieurs expérimentations, deux critères d'arrêt ont été implémentés pour l'algorithme :

- le nombre maximal d'évaluations successives *MaxEval*, sans détection de zone prometteuse, qui est égal à 5 fois le nombre de variables du problème,
- le nombre maximal d'itérations *MaxIter*, qui est égal à 50 fois le nombre de variables du problème.

Le critère lié au nombre de mouvements sans détection d'une zone prometteuse est directement dépendant de la qualité des zones prometteuses déjà détectées, puisqu'une solution n'est acceptée comme centre d'une nouvelle zone prometteuse que si la valeur de la fonction objectif en cette solution est inférieure au seuil d'acceptation **[RCH99]**:

$$\overline{f} = \frac{1}{L_p} \sum_{i=1}^{L_0} f(x_i)$$

III. Adaptation de l'algorithme génétique

Une variante de l'algorithme génétique adapté au cas continu est appelée *CPGA* (pour "*Continuous Pure Genetic Algorithm*").

POP, *POP'* et *QUALITE* : tableaux de taille *N*
POP = génération de la population initiale
(f_{min} , x_{min}) = meilleur point dans la population parent

REPETER
  DIVERSIFICATION
  QUALITE = (évaluer la population *POP*)
  REPETER
    sélection
    recombinaison
    mutation
  JUSQU'A *POP'* remplie
  POP = sélectionner nouvelle population, issue de (*POP*,*POP'*)
  SI (détection de la zone prometteuse)
    INTENSIFICATION
    réduction du domaine de recherche
    réduction de la taille de la population
    réduction de la probabilité de mutation
    réduction du rayon de voisinage des individus
    génération de la nouvelle population
  FIN SI
JUSQU'A condition d'arrêt satisfaite

Figure 3.10 : pseudocode de l'algorithme *CPGA*.

Pour améliorer les performances de l'algorithme génétique de base, deux phases essentielles ont été définies. Une première phase, appelée "diversification", permet d'explorer un large espace des solutions, afin de déterminer la région susceptible de contenir un optimum : cette région est appelée "zone prometteuse". La seconde phase, appelée "intensification", permet d'exploiter la zone prometteuse déjà détectée et d'approfondir la

recherche dans cette zone. La **Figure 3.10** donne le pseudocode de cet algorithme **[RAC99]**.

III.1. Principe de base de l'algorithme génétique pur continu (CPGA)

III.1.1. Diversification

Après avoir engendré la population initiale, *CPGA* démarre la procédure de diversification exploitant les opérateurs de reproduction génétique. Lorsqu'une "zone prometteuse" est détectée (après un certain nombre de générations sans amélioration du meilleur point connu), l'algorithme arrête la procédure de reproduction, et débute l'intensification.

Pour ne pas être pénalisé par le temps nécessaire à l'évaluation d'une grande population, durant l'intensification, il est intéressant de réduire cette population au fur et à mesure que l'on s'approche d'une région susceptible de contenir un optimum global **[RAC99]**.

III.1.2. Intensification

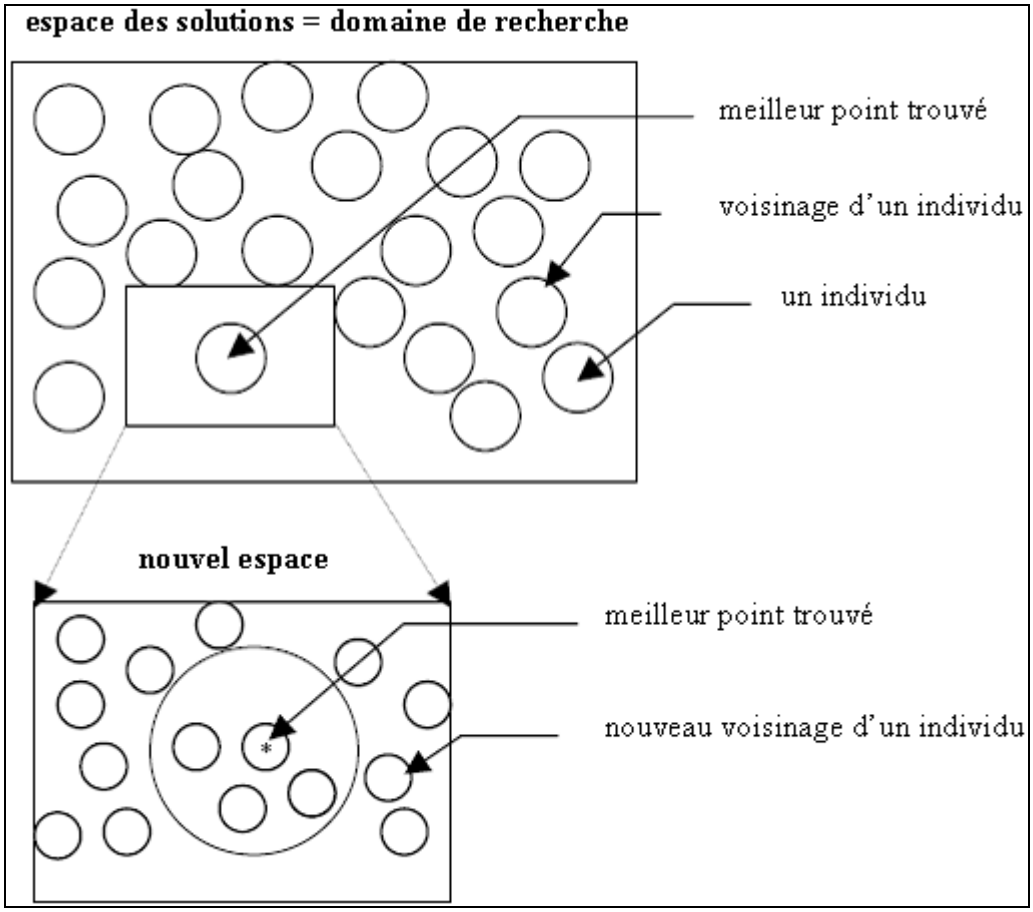


Figure 3.11 : gestion de l'espace des solutions, de la taille de la population et du voisinage.

Dans la phase d'intensification, une zone réduite autour du meilleur point trouvé précédemment est définie, une nouvelle population est générée de taille plus petite que la précédente à l'intérieur de cette nouvelle zone prometteuse, suivant le même principe que la génération de la population initiale, et le

processus de reproduction génétique reprend. La réduction de l'espace d'étude s'accompagne de la réduction du voisinage des individus, des pas de mouvement des opérateurs génétiques qui tiennent compte du domaine d'étude, et de la probabilité de mutation. Le facteur de réduction de l'espace d'étude est le même que celui du voisinage. La **Figure 3.11** montre le schéma de gestion du domaine de recherche, de la taille de la population ainsi que le voisinage des individus **[RAC99]**.

III.2. Génération de la population initiale

Pour une meilleure efficacité de la recherche, le choix de la taille de la population initiale et des individus constituant celle–ci est important pour la convergence de l'algorithme. Dans la phase de diversification, pour bien couvrir l'espace de recherche, il est intéressant de partir avec une population initiale de plus grande taille présentant une distribution uniforme des individus sur l'espace des solutions.

Dans la littérature, la population initiale est choisie le plus souvent aléatoirement, mais néanmoins, un autre choix de la population initiale fondé sur le calcul de l'entropie **[RAC99]**.

III.2.1. Théorie de l'entropie d'information

Les individus d'une population sont d'autant plus éloignés les uns des autres que leur entropie d'information est plus grande. Cette entropie exprime, entre les individus de cette population, une distance euclidienne pour le codage réel ou une distance de Hamming pour le codage binaire. Une population de N individus à M variables est acceptée si son entropie moyenne d'information :

$$H(N) = \frac{1}{M}\sum_{j=1}^{M} H_j(N)$$

est supérieure à un certain seuil.

L'entropie $H_j(N)$ sur la variable j est donnée par :

$$H_j(N) = \sum_{i=1}^{N}\sum_{k=1}^{N} -P_{ik}.log\,P_{ik}$$

où P_{ik} désigne la portabilité :

$$P_{ik} = \left(1 - \left|x_{ij} - x_{jk}\right|\right) / \left(b_j - a_j\right)$$

x_{ji} la composante j de l'individu i, et $x_{ji} \in [a_j,b_j]$ **[RAC99]**.

III.2.2. Méthode du voisinage

Afin de couvrir d'une manière homogène tout l'espace des solutions, et éviter d'avoir de nombreux individus dans la même région, l'algorithme génère une population de grande taille, et définit un "voisinage" pour chaque individu généré. Le type de voisinage proposé utilise la notion de "boule". Une boule $B(x,\varepsilon)$, centrée sur x avec le rayon ε, contient tous les points x' tels que : $//x' - x// = \varepsilon$ (le symbole //...// utilisé représente la norme euclidienne).

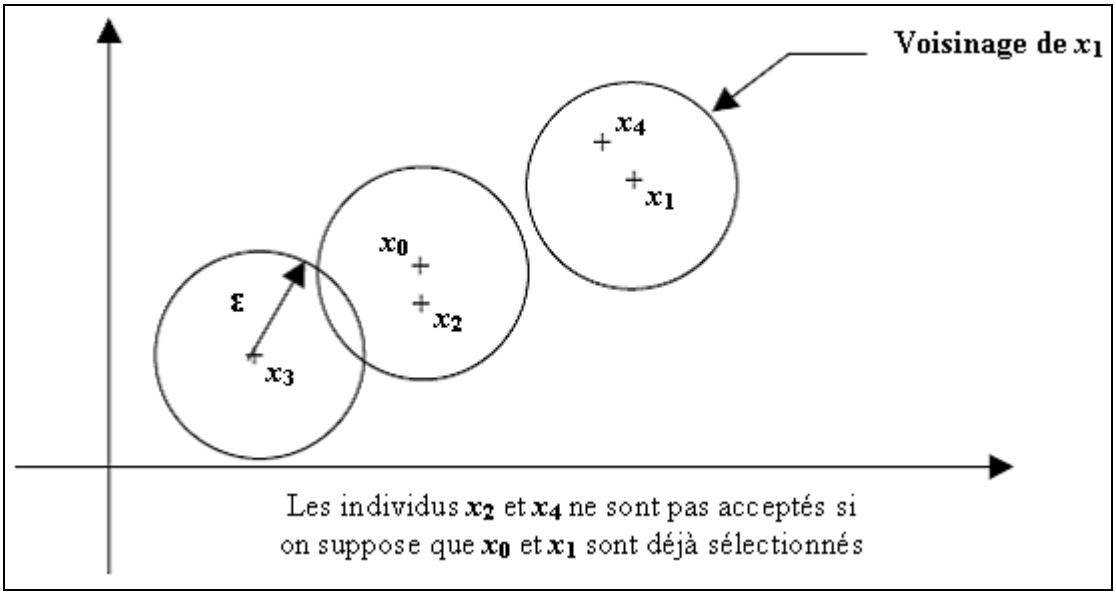


Figure 3.12 : génération des voisins de la solution courante.

Un individu généré est accepté dans la population initiale, s'il n'appartient au voisinage d'aucun individu déjà sélectionné (**Figure 3.12**, dans le cas d'un espace à deux dimensions) **[RAC99]**.

III.3. Fonction d'adaptation

La qualité d'une solution est représentée par une valeur positive dite "valeur d'adaptation". La *fonction d'adaptation* ou fonction "*fitness*", associe une *valeur d'adaptation* à chaque individu, afin de déterminer le nombre de fois qu'il sera sélectionné pour la procréation. Cette fonction d'adaptation dépendant du problème à résoudre, elle conditionne l'efficacité de l'algorithme génétique. Les individus ayant les meilleures adaptations sont reproduits plus souvent que les autres, en remplacement des moins bons. La pression sélective *SP* est définie comme étant le rapport de la probabilité du meilleur individu qui sera sélectionné à la probabilité moyenne de sélection de tous les individus. On pourrait la définir comme étant le nombre espéré de sélections du meilleur individu.

On considère *Nind* le nombre d'individus dans une population, *Pos* la position d'un individu dans cette population (l'individu le moins bon à la position *Pos* = 1, et le meilleur individu à la position *Pos* = *Nind*) et *SP* la pression sélective.

Deux types de rangements de la population sont considérés, le rangement linéaire et le rangement non linéaire **[RAC99]**:

III.3.1. Rangement linéaire

$$Fit(Pos) = 2 - SP + \frac{2(SP-1)(Pos-1)}{Nind-1}$$

Le rangement linéaire permet des valeurs de la pression sélective $SP \in [1,2]$.

III.3.2. Rangement non linéaire

$$Fit(Pos) = \frac{Nind.x^{Pos-1}}{\sum_{i=1}^{Nind} x^{i-1}}$$

où x est calculé comme la racine du polynôme suivant :

$$0 = (SP-1).x^{Nind-1} + SP.x^{Nind-2} + ... + SP.x + SP$$

Le rangement non linéaire permet des valeurs de la pression sélective $SP \in [1, Nind\text{-}2]$.

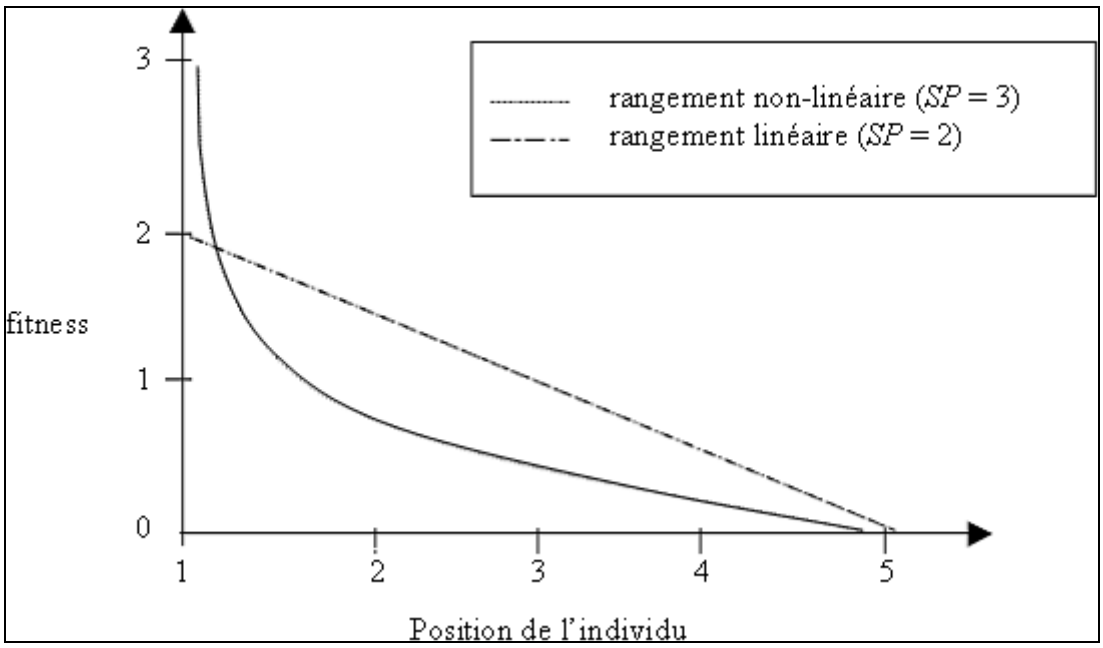


Figure 3.13 : comparaison de la "*fitness*" par rangement pour une population de 5 individus.

La probabilité de chaque individu qui sera sélectionné est le rapport de sa "fitness" à la "fitness" totale de la population.

III.3.3. Simple fonction "fitness"

La valeur de la fonction fitness d'un individu *i* dépend uniquement de la valeur de sa fonction objectif *f* et de la valeur maximale de la fonction objectif dans la population.

$$Fit(i) = Max(f) - f(i)$$

III.4. Sélection

La sélection consiste à choisir la population d'individus pour la procréation. Le nombre de fois où un individu est sélectionné pour former cette population dépend de sa *valeur d'adaptation*, qui n'est pas forcément sa fonction objectif **[RAC99]**.

III.4.1. Sélection proportionnelle

Supposons que notre population se compose de *n* individus, dont les fonctions d'adaptation *Fit(j)* sont calculées. Par une technique de sélection proportionnelle, la probabilité de sélection p_i en pour-cent d'un individu *i* est donnée par l'expression suivante :

$$P_i = \frac{Fit(i)}{\sum_{j=1}^{n} Fit(j)}$$

Le **Tableau 3.1** donne les probabilités espérées de sélection de chaque individu *i* pour une population de 5 individus. Nous remarquons que l'individu 5, qui possède la plus forte valeur de la fonction objectif (ce qui se traduit par une valeur de "fitness" nulle), ne possède aucune chance d'être retenu pour la phase de reproduction.

Individu	Fonction d'adaptation	p_i
1	249	0.19
2	576	0.42
3	180	0.13
4	321	0.26
5	0	0
Total	1326	1.00

Tableau 3.1 : les probabilités de sélection des individus de la population.

Chaque individu de la population présente occupe un secteur de la roulette de surface proportionnelle à son adaptation. La roulette biaisée correspondante est représentée dans la **Figure 3.14 [RAC99]**.

Figure 3.14 : roulette dont les secteurs sont des surfaces proportionnelles à l'adaptation des individus.

"Roulette Wheel Selection" (RWS)

Cette méthode est aussi appelée "*stochastic sampling with replacement*". Elle est directement dérivée de la sélection proportionnelle. Les individus sont représentés par des segments de droite de longueur égale à la valeur de leur "fitness" (portion de cercle). La **Figure 3.15** schématise une telle représentation. Pour sélectionner un individu susceptible de se reproduire, on tire au hasard un

nombre compris entre 0 et 1, et on repère cette valeur. Pour générer une population de 4 individus, on fait 4 tirages. L'individu 2 qui est le mieux adapté a été tiré deux fois alors que l'individu 3 n'a pas été retenu pour la phase de reproduction **[RAC99]**.

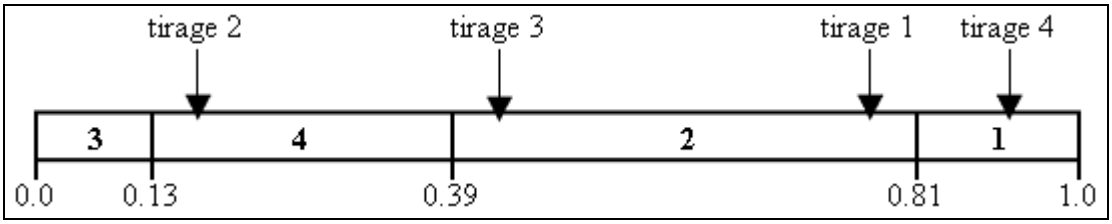


Figure 3.15 : représentation par segments de droite de la sélection par roulette.

"Stochastic Universal Sampling" (SUS)

La représentation est la même que pour *RWS*, on considère toujours un segment de droite partitionné en autant de zones qu'il y a d'individus dans la population. Les tailles de ces zones sont proportionnelles à l'adaptation. On considère *NPointeur* le nombre d'individus à sélectionner, alors la distance *d* entre les pointeurs est 1/*NPointeur*, et la position du premier pointeur est donnée par un nombre choisi aléatoirement entre [0,1/*NPointeur*]. Les individus ainsi sélectionnés sont représentés par un ensemble de points équidistants (**Figure 3.16**).

$$x_i = x + i.d$$

avec *i* variant de 1 au nombre (*NPointeur*-1) d'individus à générer **[RAC99]**.

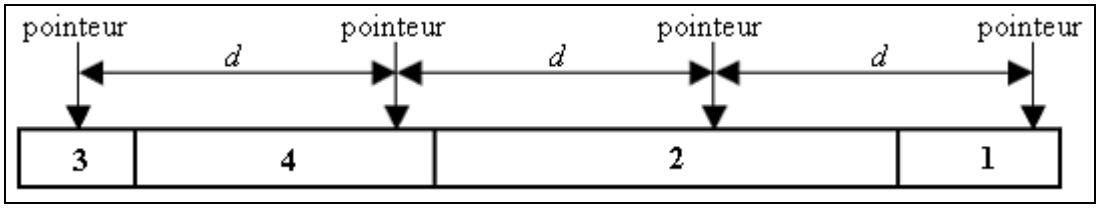


Figure 3.16 : échantillonnage universel stochastique (SUS).

III.4.2. Sélection par tournois

La sélection consiste à choisir aléatoirement un certain nombre d'individus dans la population, et à sélectionner pour la reproduction celui qui a la plus grande adaptation. Au cours d'une génération, il y a autant de tournois que d'individus à remplacer. La pression de sélection est ajustée par le nombre de participants au tournoi. Choisir de nombreux participants (grande taille du tournoi) conduit à une forte pression de sélection, car un individu moyen aura moins de chances d'être sélectionné. On réduit ainsi la diversité dans la recherche : environ 50% des individus de la population sont perdus lorsque la taille du tournoi est égale à 5. La **Figure 3.17** illustre les propriétés de la sélection par tournois. Elle montre la variation de l'"intensité de sélection" et de la "perte de diversité" en fonction de la taille *T* du tournoi (nombre d'individus qui participent à un tournoi). L'intensité de sélection et la perte de diversité ont défini comme suit **[RAC99]** :

L'intensité de sélection est donnée par :

$$IntSel(T) = \left(2\left(\log T - \log\left((4.14\log(T))^{1/2}\right)\right)\right)^{1/2}$$

La perte de diversité est donnée par :

$$PertDiv(T) = T^{\frac{1}{r-1}} - T^{\frac{r}{r-1}}$$

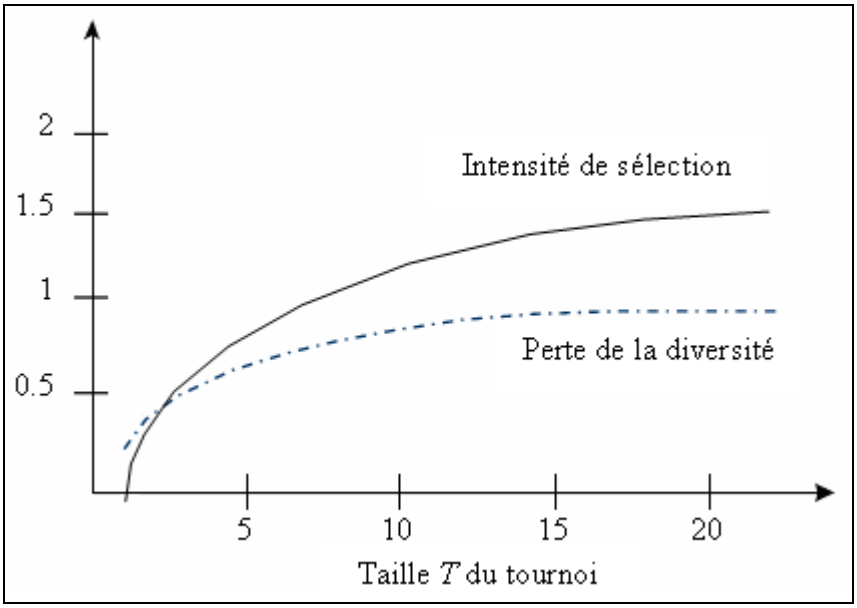


Figure 3.17 : propriétés de la sélection par tournois.

III.5. Recombinaison

L'opérateur de recombinaison dépend évidemment du type de codage utilisé. Par exemple, dans le codage binaire on rencontre des croisements simples ou multiples **[RAC99]**.

III.5.1. Recombinaison discrète

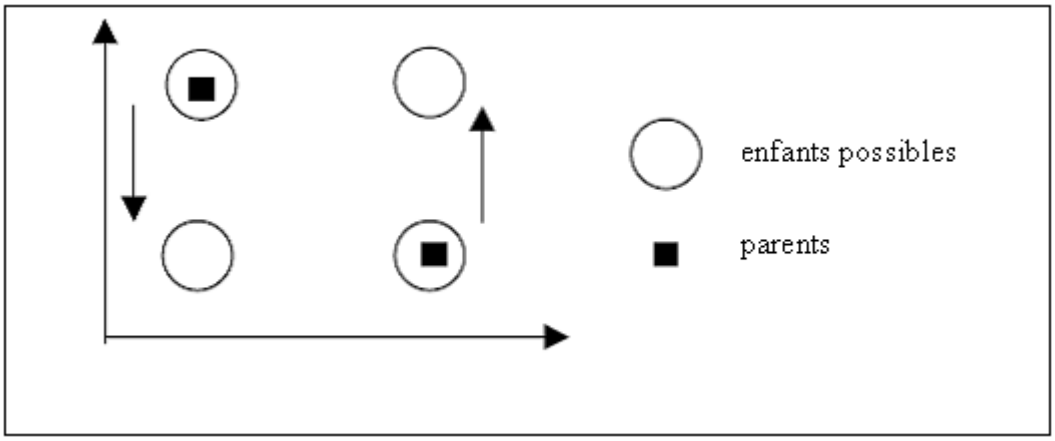


Figure 3.18 : recombinaison discrète.

Ce type de recombinaison, représenté sur la **Figure 3.18**, consiste à échanger les valeurs des variables entre individus. On considère les deux

parents *x* et *y*. Pour chaque variable de l'enfant généré, le parent qui contribue est choisi aléatoirement avec une même probabilité.

Ainsi la recombinaison discrète génère des enfants occupant les coins d'un hypercube défini par les parents. Ce mode de recombinaison est aussi utilisé pour les variables codées en binaire **[RAC99]**.

III.5.2. Recombinaison intermédiaire

Cette recombinaison est uniquement appliquée au codage réel. Dans cette méthode (**Figure 3.19**), les enfants sont générés quelque part autour ou entre les valeurs des variables des parents. Un enfant est produit par la règle suivante :

$$enfant = x + \alpha(y - x)$$

où α est un facteur d'échelle choisi aléatoirement sur l'intervalle [-*d*,1+*d*] pour chaque variable. Dans la recombinaison intermédiaire, *d* = 0, pour une recombinaison intermédiaire étendue, *d* > 0. Un bon choix est obtenu pour *d* = 0.25 **[RAC99]**.

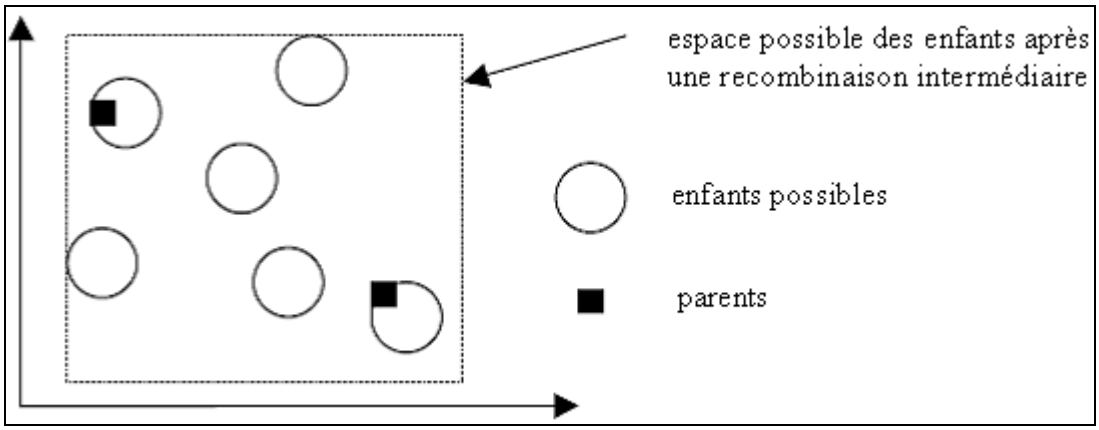


Figure 3.19 : recombinaison intermédiaire étendue.

III.5.3. Recombinaison "ligne"

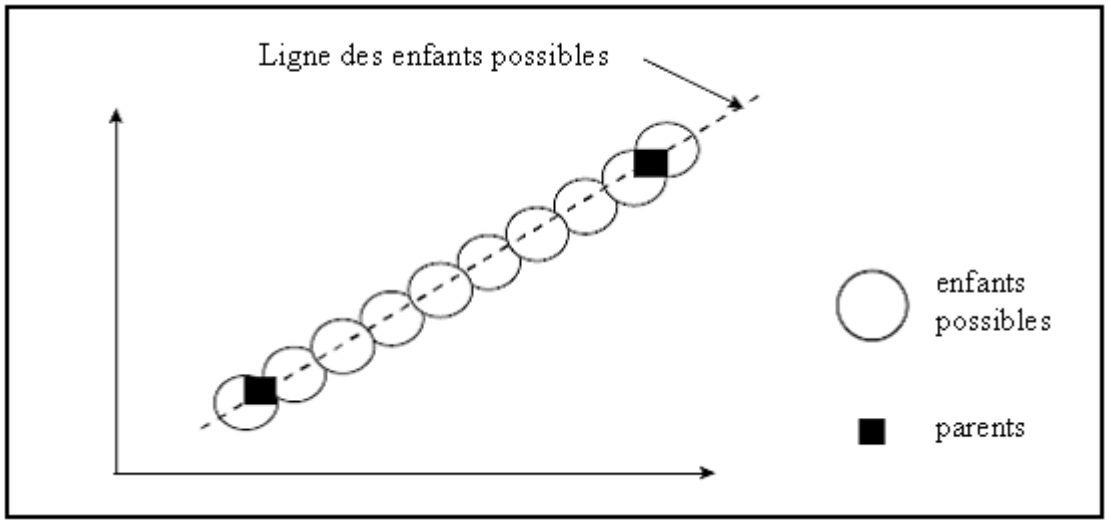


Figure 3.20 : recombinaison "ligne".

Cette méthode est identique à la méthode recombinaison intermédiaire, sauf que le facteur d'échelle est constant pour toutes les variables. La **Figure 3.20** montre ce type de recombinaison **[RAC99]**.

III.5.4. Recombinaison hybride

Ce type de recombinaison est inspiré du croisement en codage binaire, adapté au codage réel. Dans le codage binaire, lors de l'opération de croisement, les bits situés avant le point de croisement ne sont pas affectés, et ceux situés après le point de croisement sont interchangés entre les deux individus concernés par l'opération. Dans le codage réel, afin de simuler aussi fidèlement que possible l'opération de croisement, nous procédons de la manière suivante : pour déterminer le point de croisement, on tire au hasard un entier *i* compris entre 0 et le nombre de variables de l'individu. Toutes les composantes situées à gauche de ce point de croisement ne sont pas affectées, celles situées après le point de croisement sont interchangées entre les deux individus concernés. Pour les composantes *x(i)* et *y(i)* des deux individus *x* et *y* situées à la position *i*, nous opérons comme suit : nous soustrayons une quantité Δx de *x(i)* et nous l'ajoutons à *y(i)*, puis nous soustrayons une quantité Δy de *y(i)* et nous l'ajoutons à *x(i)*. Ces quantités Δx et Δy sont déterminées en tirant deux entiers aléatoires *M1* et *M2* compris entre 1 et 1000, et nous les calculons comme suit **[RAC99]** :

$$\Delta x = \frac{UpBd(i) - LowBd(i)}{M_1} \text{ et } \Delta y = \frac{UpBd(i) - LowBd(i)}{M_2}$$

Les nouvelles valeurs des cordonnées sont :

$$x'(i) = x(i) + (\Delta y - \Delta x) \text{ et } y'(i) = y(i) + (\Delta x - \Delta y)$$

III.6. Mutation

Après l'opération de recombinaison, intervient celle de la mutation. Pour chaque individu, nous tirons au hasard un nombre compris entre 0 et 1, et nous le comparons à la probabilité de mutation. Si ce nombre est supérieur à cette probabilité, alors l'individu n'est pas candidat à la mutation, sinon la mutation est pratiquée de la manière suivante : nous tirons aléatoirement un entier *i* compris entre 0 et le nombre de variables de l'individu, et un autre entier *M* compris entre 1 et 1000, et nous déterminons les variations suivantes :

$$\Delta x = \frac{UpBd(i) - LowBd(i)}{M}$$

UpBd(i) et *LowBd(i)* sont respectivement la borne supérieure et inférieure de la composante *x(i)* de l'individu *x* candidat à la mutation.

$$x'(i) = x(i) \pm k.\Delta x$$

où *x'(i)* est la nouvelle valeur, *k* est un coefficient inférieur à l'unité **[RAC99]**.

III.7. Remplacement et réinsertion (stratégies élitistes)

Dans la littérature, plusieurs méthodes d'insertion sont évoquées. Par exemple, la réinsertion globale peut consister à remplacer systématiquement la population des parents par la population des enfants (pure réinsertion), ou à former la nouvelle génération à partir des meilleurs individus sélectionnés parmi les deux populations parent et enfant (réinsertion élitiste).

Une autre méthode de détermination de la nouvelle population consiste à comparer le meilleur individu de la population enfant à celui de la population parent. Si le premier est meilleur que le second, la population parent est remplacée directement par la population enfant, sinon le meilleur individu de la population parent est conservé en l'insérant dans la population enfant, le plus mauvais individu de cette dernière est remplacé, et la population ainsi composée remplace la population parent. La taille de la population de la nouvelle génération est toujours égale à la taille de la population de départ **[RAC99]**.

POP et *POP'* : tableaux de taille N
POP = population parent
$(f_{opt}, x_{opt}) = (f_{min}, x_{min})$ = meilleur point dans la population parent

REPRODUCTION (*POP'* = population enfant)
(f'_{min}, x'_{min}) = meilleur point dans la population enfant

SI $f_{min} < f'_{min}$
 localiser (f'_{max}, x'_{max}) le plus mauvais point dans *POP'*
 $(f'_{max}, x'_{max}) = (f_{min}, x_{min})$
SINON
 $(f_{opt}, x_{opt}) = (f'_{min}, x'_{min})$
FIN SI
POP = POP'

Figure 3.21 : pseudocode de la procédure de détermination de la nouvelle population.

III.8. Présentation de l'algorithme hybride génétique-polytope (CHGA)

III.8.1. Principe

Dans cet algorithme, deux concepts largement utilisés dans la recherche tabou ont été introduits : la diversification et l'intensification. La phase de diversification consiste à un démarrage avec une large population, bien répartie sur tout l'espace de solutions et une forte probabilité de mutation ; une zone prometteuse susceptible de contenir un optimum global est détecté. L'intensification consiste à explorer cette zone prometteuse, après génération d'une nouvelle population de taille réduite. La réduction de l'espace de recherche, ainsi que de la taille de la population, s'accompagne d'une réduction du voisinage des individus et de la probabilité de mutation.

Comme l'algorithme *CPTS*, l'algorithme *CPGA* est efficace pour explorer un large domaine de recherche, et détecter une "vallée" prometteuse, mais il est trop gourmand pour intensifier la recherche dans cette vallée. Pour améliorer la vitesse de convergence de l'algorithme *CPGA*, un algorithme hybride combinant un algorithme génétique et la méthode du polytope de Nelder-Mead *SS* a été développé. Ce nouvel algorithme appelé *CHGA* "*Continuous Hybrid Genetic Algorithm*" utilise la phase de diversification de *CPGA* pour localiser une zone prometteuse, et intensifie la recherche dans cette zone en utilisant la recherche locale *SS*.

La transformation de l'algorithme *CPGA* en *CHGA* nécessite les modifications suivantes **[RAC99]**:

- la suppression des paramètres de gestion de la population (*IPopSize*, *FPopSize* et $\Delta_{PopSize}$) et leur remplacement par une taille de population unique et constante *PopSize*, égale à *IPopSize* ;
- l'introduction d'un nouveau critère de détection de la zone prometteuse ;
- l'introduction des coefficients (α,β,γ) qui déterminent les pas de mouvement dans la méthode de Nelder-Mead ;
- l'introduction d'un nouveau critère d'arrêt lié à la méthode du polytope ;
- l'introduction d'une nouvelle loi de variation de la probabilité de mutation, qui dépend du nombre de générations *NbGen*, au lieu du nombre de réductions de domaine de recherche. En effet, dans ce nouvel algorithme, une seule réduction de domaine apparaît lors du passage de la phase de diversification à la recherche par le polytope de Nelder-Mead. La nouvelle loi de variation de la probabilité, représentée sur la **Figure 3.22**.

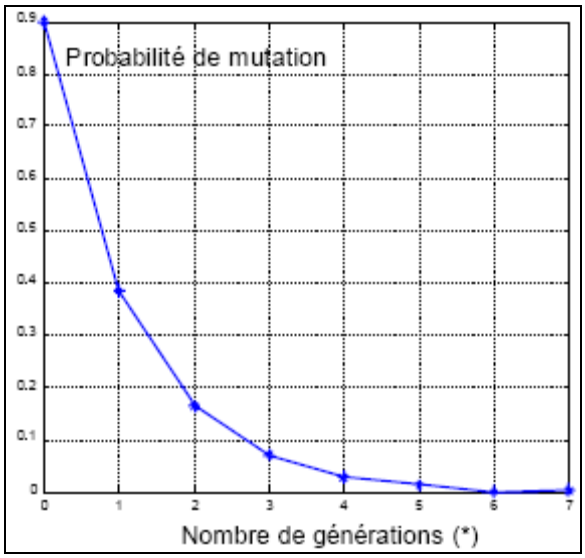


Figure 3.22 : variation de la probabilité de mutation en fonction du nombre de générations.

Cette nouvelle loi s'exprime de la manière suivante :

$$Pmut(k) = IPmut.exp\left(-\frac{k}{\mu}\right) \text{ et } \mu = \frac{NbGen}{Log(IPmut / 0.001)}$$

où *IPmut* et *Pmut(k)* représentent respectivement la probabilité de mutation initiale et la probabilité de mutation à une génération donnée *k*.

III.8.2. Description de l'algorithme

Sur la **Figure 3.23** donnant le pseudocode de l'algorithme *CHGA*, nous retrouvons la phase de diversification implémentée dans *CPGA*, et une nouvelle forme d'intensification utilisant *SS* **[RAC99]**.

```
POP, POP' et QUALITE : tableaux de taille N
POP = génération de la population initiale
( f_min , x_min ) = meilleur point dans la population parent

DIVERSIFICATION
 REPETER
   QUALITE = (évaluer la population POP)
   REPETER
     sélection
     recombinaison
     mutation
   JUSQU'A POP' remplie
   POP = sélectionner nouvelle population, issue de (POP,POP')
 JUSQU'A détection de la zone prometteuse

INTENSIFICATION
 réduction du domaine de recherche
 construction du polytope initial
 REPETER
   Mouvements géométriques
 JUSQU'A condition d'arrêt satisfaite
```

Figure 3.23 : pseudocode de l'algorithme *CHGA*.

III.8.3. Diversification

La phase de diversification de *CHGA* repose sur le même principe que celle de *CPGA*. On part d'une large population, dispersée uniformément sur tout l'espace des solutions ; une nouvelle population est produite à partir de la population précédente. Des individus enfants sont produits par sélection, recombinaison, et mutation des individus parents, appartenant à la génération précédente, les valeurs des fonctions objectifs de ces individus enfants sont calculées, une stratégie de remplacement de la population parent par la population enfant est opérée. Une nouvelle population est ainsi produite, et le processus est réitéré. L'exploration s'arrête lorsqu'une des conditions suivantes est vérifiée :

- un nombre donné de générations successives *MaxGen* sans détection d'une zone prometteuse ;

- une précision donnée relative aux individus générés est obtenue : la plus grande distance entre le meilleur point trouvé *BestPoint* et les individus générés Ind_i est plus petite qu'un rayon de voisinage donné σ_{abs} :

$$\frac{1}{PopSize}\sum_{i=1}^{PopSize} // Ind_i - BestPoint// \leq \sigma_{abs}$$

(le symbole //...// utilisé représente la distance euclidienne).

La valeur de σ_{abs} est typiquement égale au dixième de la distance δ (le plus petit domaine de variation des variables du problème). Les deux paramètres σ_{abs} et *MaxGen* sont les principaux paramètres influençant la convergence de l'algorithme **[RAC99]**.

III.8.4. Intensification à l'intérieur d'une zone prometteuse

On prend le meilleur point trouvé dans la phase précédente, et l'on construit un nouveau domaine de recherche centré autour de ce point. Chaque côté du domaine initial est réduit d'un facteur donné, soit ρ_{red} La méthode de Nelder-Mead permet de construire le polytope de départ. Ce polytope est obtenu en tenant compte du meilleur point trouvé précédemment, désigné par x_1, et les autres points x_i sont choisis dans le nouvel espace des solutions, de telle manière qu'ils forment une base engendrant l'espace de variation des paramètres du problème.

La procédure *SS* démarre ensuite la recherche **[RAC99]**.

III.8.5. Critères d'arrêt

L'algorithme accomplit la tâche d'exploitation à l'intérieur de la zone prometteuse jusqu'à ce que les conditions d'arrêt soient atteintes. Dans *CHGA*, on a deux critères d'arrêt :

- le premier est relatif au nombre maximal d'itérations (boucle de la procédure *SS*) *MaxIter* ;
- le second critère est une mesure de déplacement du polytope *s* d'une itération *k* à l'itération suivante (*k*+1) :

$$\frac{1}{n}\sum_{i=1}^{n}\left\|s_i^k - s_i^{k+1}\right\|^2 < \varepsilon$$

où s_i^{k+1} est le sommet *i* du polytope *s* remplaçant le sommet s_i^k à l'itération (*k*+1), et ε est un nombre réel positif donné.

Le meilleur point du polytope final, après arrêt de la procédure de *SS*, tient lieu de résultat final de l'algorithme **[RAC99]**.

IV. Adaptation de la méthode d'optimisation par colonie de fourmis

IV.1. Problèmes d'adaptation

Les métaheuristiques sont bien souvent élaborées pour des problèmes combinatoires, mais il existe une classe de problèmes souvent rencontrée en

ingénierie, où la fonction objectif est à variables continues et pour lesquels les métaheuristiques peuvent être d'un grand secours (fonction non dérivable, multiples minimums locaux, grand nombre de variables, non-convexité, … etc.). Plusieurs tentatives pour adapter les métaheuristiques de colonie de fourmis au domaine continu sont apparues.

Outre les problèmes classiques d'adaptation d'une métaheuristique, les algorithmes de colonie de fourmis posent un certain nombre de problèmes spécifiques. Ainsi, le principal problème vient si l'on se place dans le formalisme ACO avec une construction de la solution composant par composant. En effet, un problème continu peut - selon la perspective choisie - présenter une infinité de composants, le problème de la construction est difficilement soluble dans ce cas. La plupart des algorithmes s'inspirent donc des caractéristiques d'auto-organisation et de mémoire externe des colonies de fourmis, laissant de côté la construction itérative de la solution ; cependant, depuis peu, le caractère probabiliste du formalisme ACO commence à être employé.

Dans la littérature, il y a plusieurs algorithmes de colonie de fourmis pour l'optimisation continue : *CACO*, un algorithme hybride non baptisé, *API*, ACO pour l'optimisation continue et *CACS* **[DRE04]**.

IV.2. Algorithme CACO

Le premier de ces algorithmes, tout naturellement nommée *CACO* "*Continuous Ant Colony Algorithm*", utilise deux approches : un algorithme de type évolutionnaire sélectionne et croise des régions d'intérêt, que des fourmis explorent et évaluent. Une fourmi sélectionne une région avec une probabilité proportionnelle à la concentration en phéromone de cette région, de la même manière que - dans le "Ant System" -, une fourmi sélectionnerait une piste allant d'une ville à une autre :

$$Pi(t) = \frac{\tau_i^{\alpha}(t).\eta_i^{\beta}(t)}{\sum_{j=1}^{N} \tau_j^{\alpha}(t).\eta_i^{\beta}(t)}$$

où N est le nombre de régions et $\eta_i^{\beta}(t)$ est utilisé pour inclure une heuristique spécifique au problème. Les fourmis partent alors du centre de la région et se déplacent selon une direction choisie aléatoirement, tant qu'une amélioration de la fonction objectif est trouvée.

Le pas de déplacement utilisé par la fourmi entre chaque évaluation est donné par :

$$\delta r(t,R) = R.\left(1 - u^{\left(1-\frac{t}{T}\right)^{c}}\right)$$

où R est le diamètre de la région explorée, $u \in [0,1]$ un nombre aléatoire, T le nombre total d'itérations de l'algorithme et c un paramètre de refroidissement (permettant de réduire le pas à chaque itération). Si la fourmi a trouvé une meilleure solution, la région est déplacée de façon à ce que son centre coïncide avec cette solution, et la fourmi augmente la quantité de phéromone de la région proportionnellement à l'amélioration trouvée (appelée ici "*fitness*"). L'évaporation des "pistes" se fait classiquement en fonction d'un coefficient ρ.

Des modifications ont été apportées pour améliorer les performances de l'algorithme original. Ainsi, en plus des fourmis "locales" de *CACO*, des fourmis "globales" vont explorer l'espace de recherche (**Figure 3.24**) pour éventuellement remplacer les régions peu intéressantes par de nouvelles régions non explorées. Les régions sont également affectées d'un âge, qui augmente si aucune amélioration n'est découverte. De plus, le paramètre t dans le pas de recherche des fourmis $\delta r(t,R)$ est défini par l'âge de la région explorée.

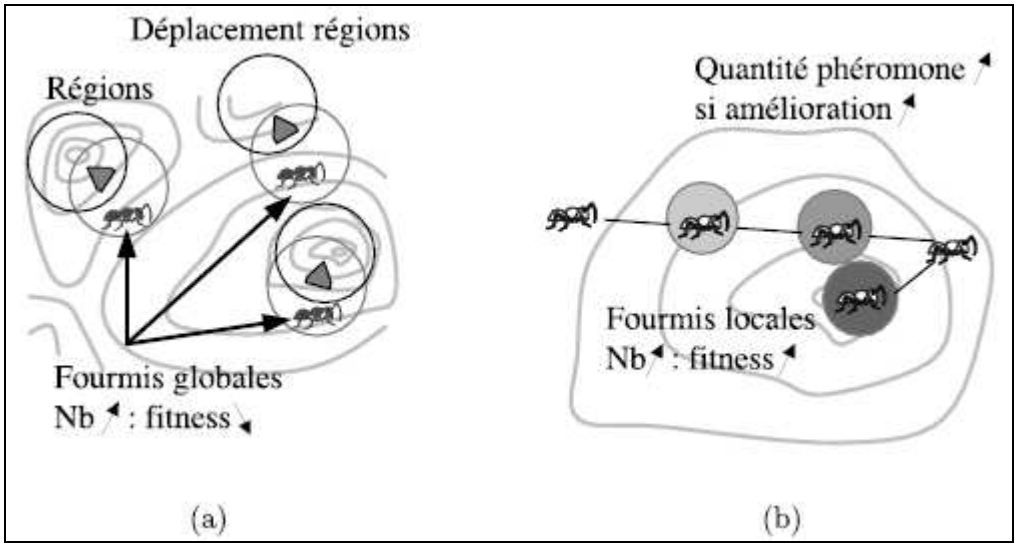


Figure 3.24 : algorithme *CACO* - les fourmis globales (a) participent au déplacement des régions que les fourmis locales (b) évaluent.

Une refonte de l'algorithme a été opérée par d'autres auteurs en vue de relier plus finement *CACO* avec le paradigme des colonies de fourmis et d'abandonner la liaison avec l'algorithme évolutionnaire. On parle ainsi par exemple de *diffusion* pour définir la création de nouvelles régions **[DRE04]**.

IV.3. Méthode hybride

Une approche semblable - utilisant à la fois une approche de colonie de fourmis et d'algorithme évolutionnaire - a été proposée par Ling et al.. L'idée principale de cette méthode est de considérer les écarts entre deux individus sur chaque dimension comme autant de parties d'un chemin où les phéromones sont déposées, l'évolution des individus étant prise en charge par des opérateurs de mutation et de croisement. D'un certain point de vue, cette méthode tente donc de reproduire le mécanisme de construction de la solution, composant par composant.

La méthode procède précisément selon l'**Algorithme 3.1**. Chaque fourmi x_i de la population contenant m individus est considérée comme un vecteur à n dimensions. Chaque élément $x_{i,e}$ de ce vecteur peut donc être considéré comme un candidat à l'élément $x^*_{i,e}$ de la solution optimale. L'idée est d'utiliser le chemin entre les éléments $x_{i,e}$ et $x_{j,e}$ - notée (i,j) - pour déposer une piste de phéromone dont la concentration est notée $\tau_{ij}(t)$ au pas de temps t.

1. À chaque itération, sélectionner pour chaque fourmi une valeur initiale dans le groupe de valeurs candidates avec la probabilité :
$$P_{ij}^k(t) = \frac{\tau_{ij}(t)}{\sum \tau_{ir}(t)}$$
2. Utiliser des opérateurs de mutation et de croisement sur les m valeurs, afin d'obtenir m nouvelles valeurs ;
3. Ajouter ces nouvelles valeurs au groupe de valeurs candidates pour le composant $x_{i,e}$;
4. Les utiliser pour former m solutions de la nouvelle génération ;
5. Calculer la "*fitness*" de ces solutions ;
6. Quand m fourmis ont parcouru toutes les arêtes, mettre à jour les pistes de phéromone des valeurs candidates de chaque composant par :
$$\tau_{ij}(t+1) = (1-\rho)\tau_{ir}(t) + \sum \tau_{ij}^k$$
7. Si la $k^{ème}$ fourmi choisit la $j^{ème}$ valeur candidate du groupe de composants, alors :
$$\Delta\tau_{ij}^k(t+1) = Wf_k$$
sinon :
$$\Delta\tau_{ij}^k = 0$$
en désignant par W une constante et par f_k la "*fitness*" de la solution trouvée par la $k^{ème}$ fourmi ;
8. Effacer les m valeurs ayant les plus basses intensités de phéromone dans chaque groupe de candidats.

Algorithme 3.1 : un algorithme de colonie de fourmis hybride pour le cas continu.

Les auteurs ont proposé une version "adaptative", où les probabilités de mutation et de croisement sont variables. Malheureusement, cet algorithme n'a pas encore été complètement testé **[DRE04]**.

IV.4. Algorithme API

Dans tous les algorithmes évoqués jusqu'ici, le terme "colonie de fourmis" s'entend de par l'utilisation de la stigmergie comme processus d'échange d'information.

Il existe cependant un algorithme adapté au cas continu qui s'inspire du comportement de fourmis primitives (ce qui ne veut pas dire *inadaptées*) de l'espèce *Pachycondyla apicalis*, et qui ne fait pas usage de la communication indirecte par pistes de phéromone : l'algorithme *API*.

Dans cette méthode, on commence par positionner un nid aléatoirement sur l'espace de recherche, puis des fourmis sont distribuées aléatoirement dans l'espace. Ces fourmis vont alors explorer localement leur "site de chasse" en évaluant plusieurs points dans un périmètre donné (**Figure 3.25**). Chaque fourmi mémorise le meilleur point trouvé. Si, lors de l'exploration de son site de chasse, elle trouve un meilleur point, alors elle reviendra sur ce site, sinon, après un certain nombre d'explorations, elle choisira un autre site. Une fois les explorations des sites de chasse terminées, des fourmis tirées au hasard comparent deux à deux (comme peuvent le faire les fourmis réelles durant le comportement de "*tandemrunning*", qu'on pourrait traduire par "course en couple") leurs meilleurs résultats, puis mémorisent le meilleur des deux sites de chasse.

Le nid est finalement réinitialisé sur le meilleur point trouvé après un temps donné, la mémoire des sites des fourmis est remise à zéro, et l'algorithme effectue une nouvelle itération **[DRE04]**.

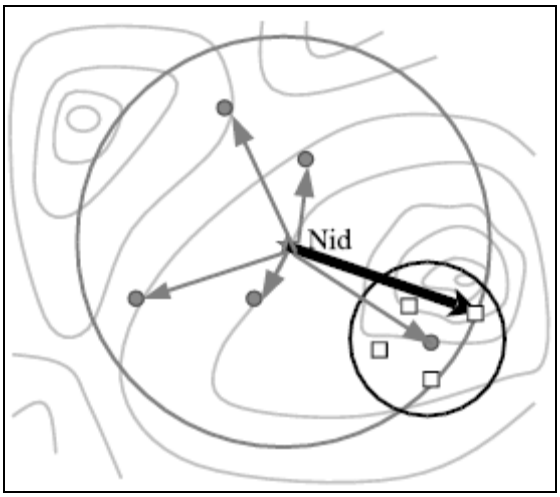


Figure 3.25 : algorithme *API* - une méthode à démarrage multiple inspirée par une espèce de fourmi primitive. Les fourmis (cercles pleins) explorent des sites de chasse (petits carrés) dans un périmètre (grand cercle) autour du nid. Le nid est déplacé sur le meilleur point au moment de la réinitialisation (flèche en trait gras).

IV.5. ACO pour l'optimisation continue

Cet algorithme tente de maintenir la construction itérative des solutions dans le cas de variables continues, en adoptant un point de vue différent des précédents. En effet, il prend le parti de considérer que les composants de toutes les solutions sont formés par les différentes variables optimisées. De plus, plutôt que de considérer l'algorithme du point de vue de la fourmi, il prend le parti de se placer au niveau de la colonie, les fourmis n'étant plus que des points à évaluer.

Dans cette méthode, dénommée simplement "ACO pour l'optimisation continue", on tire aléatoirement une population de fourmis dans une distribution de probabilité à chaque itération. De cet ensemble de points ne sont conservés que les meilleurs points, qui servent alors à construire une "meilleure" distribution de probabilité.

La distribution de probabilité utilisée est un "amalgame pondéré de noyaux normaux", soit un ensemble de distributions normales combinées :

$$P(x) = \sum_{j=1}^{k} w_j . \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right)$$

en désignant par k le nombre de noyaux utilisés, μ et σ^2 la moyenne et la variance d'un noyau et w_i la pondération.

Chaque distribution n'est utilisée que sur une variable, sans dépendance avec les autres. La modification des distributions est nommée "mise à jour de phéromone" et consiste à renforcer ou à diminuer l'influence des noyaux correspondant aux solutions. Le principe de la méthode est présenté sur l'**Algorithme 3.2 [DRE04]**.

Construire la distribution de probabilité initiale : $\tau_i^0 = P_i^0(x_i), i \in \{1...n\}$
Tant que (critère d'arrêt non atteint) :
  Pour chaque fourmi, de α = 1 à m :
    Pour chaque variable, de i = 1 à n :
      Choisir aléatoirement une valeur x_i selon la distribution $P_i(x_i)$
      Ajouter à la solution en construction : $S^a = \{s_1^a,...,s_{i-1}^a\} \cup \{x_i\}$
    Fin
  Fin
  Mémoriser les k meilleures solutions trouvées : $S^* = \{s_1^*,...,s_k^*\}$
  Reconstruire la distribution de probabilité selon les meilleures solutions : $\tau = P(S^*)$
Fin

Algorithme 3.2 : algorithmes du type colonie de fourmis, utilisant l'estimation de distribution pour l'optimisation en variables continues.

IV.6. Algorithme CACS

Cette méthode appelée "*Continuous Ant Colony System*" est très proche de la précédente, bien qu'ayant été présentée simultanément. En effet, dans *CACS* comme dans ACO pour l'optimisation continue, le cœur de l'algorithme consiste à faire évoluer une distribution de probabilité.

Le principe de la méthode est le même que celui présenté sur l'**Algorithme 3.2**. Dans CACS, la distribution utilisée est dite "normale", mais

la formule de la distribution diffère légèrement de la formule classique (équation 1) et la variance utilisée est en fait un nouvel indice de dispersion (équation 2).

$$P(x) = e^{-\frac{(x-x_{min})^2}{2\sigma^2}} \quad (1)$$

en désignant par x_{min} le mode de la distribution et σ^2 l'indice de dispersion suivant :

$$\sigma^2 = \frac{\sum_{j=1}^{m} \frac{1}{f_j - f_{min}} (x_j - x_{min})^2}{\sum_{j=1}^{m} \frac{1}{f_j - f_{min}}} \quad (2)$$

en désignant par m le nombre de fourmis, f_j la valeur de la fonction associée à la fourmi j, et f_{min} la meilleure valeur trouvée.

Dans *CACS*, la seule distribution utilisée est centrée sur le mode de la distribution de l'itération précédente, et non sur la moyenne **[DRE04]**.

V. Conclusion

Dans les deux algorithmes fondés sur la méthode tabou, *CPTS* et *CHTS*, la phase de diversification est assurée par les listes tabou et prometteuse. La longueur de la liste tabou, qui est relativement élevée, ne permet pas à l'algorithme de revenir sur les solutions récemment visitées, et permet d'éviter en principe le cyclage. La liste prometteuse rassemblant les boules prometteuses déjà localisées évite à l'algorithme de revenir dans ces régions et stimule ainsi la recherche vers d'autres régions non visitées. Durant la phase d'intensification, la taille de la liste tabou, ainsi que les rayons de voisinage, sont réduits. Les deux algorithmes tabou sont directement inspirés de la méthode tabou combinatoire, et adaptés au cas des variables continues.

Dans les deux algorithmes génétiques *CPGA* et *CHGA*, pour obtenir la diversification de la recherche, l'algorithme est démarré avec une large population, bien répartie sur tout l'espace des solutions, et avec une grande probabilité de mutation. Un opérateur de croisement est implémenté et aussi une méthode de génération de population nouvelle qui évite le piégeage de l'algorithme dans un minimum local. Afin de converger rapidement vers une solution globale, lorsqu'une zone prometteuse est localisée, l'algorithme passe à la phase d'intensification, en réduisant le domaine de recherche autour de la zone prometteuse détectée, la taille de la population, ainsi que la probabilité de mutation. Afin d'adapter facilement les algorithmes génétiques aux problèmes à variables continues, le codage réel est utilisé.

Pour l'adaptation des algorithmes de colonie de fourmis, les deux premiers types d'algorithmes étaient en fait plus ou moins hybridés avec un

algorithme de type évolutionnaire, et qu’un troisième n’utilisait pas la métaphore "classique" des colonies de fourmis. Les deux derniers algorithmes utilisent une approche différente des précédents, plus centrée sur l’aspect probabiliste que comportemental des algorithmes à colonies de fourmis.

Afin d’examiner l’adaptation de l’une des métaheuristiques modernes qui est l’optimisation par essaim de particules aux problèmes d’optimisation à variables continues, nous appliquerons, dans le chapitre suivant, cette méthode sur le problème de planification de réseaux locaux sans-fil.

Chapitre 4 :

Conception & Implémentation

I. Introduction

Dans le premier chapitre, nous avons donné des généralités sur les métaheuristiques les plus connues telle quelle l'optimisation par essaim de particules, ensuite, nous avons présenté, dans le deuxième chapitre, les applications de ces approches aux problèmes d'optimisation difficile tel que le voyageur du commerce.

Dans ce travail, nous avons développé notre système en se basant sur les mécanismes de l'optimisation par essaim de particules mais cette fois dans le domaine de l'optimisation continue et nous avons choisi, comme application, le problème de planification de réseaux sans fil.

La planification d'un réseau sans-fil est un problème d'optimisation dont les variables sont données par l'ensemble des configurations possibles des points d'accès et les objectifs par la description mathématique des services que le réseau doit offrir.

Dans ce chapitre, nous présentons tout d'abord la problématique, l'environnement et les choix techniques retenus pour développer notre système. Ensuite, nous présentons les résultats obtenus et nous discutons, enfin, ces résultats.

II. Planification de réseaux wLAN

II.1. Déploiement de réseaux sans-fil

Avec l'expansion des réseaux wLAN depuis le début des années 2000, un nombre sans cesse croissant d'intégrateurs réseaux s'investissent dans le déploiement de réseaux locaux sans-fil. Avec leur expertise, ils ont mis en place un processus de déploiement en 3 étapes :

1. Mesure et analyse du site ;
2. Installation du réseau : positionnement des AP et allocation des canaux ;
3. Tests de fonctionnement.

L'analyse du site client passe par la mise en place temporaire de point d'accès de test et par la mesure de leur couverture radio. Ces mesures sont alors analysées par l'expert radio qui choisit les positions permettant de couvrir la totalité de la surface. Or, pour s'assurer de la fourniture du service dans tout le bâtiment, les experts ont tendance à surestimer volontairement le nombre de points d'accès.

Cette surestimation du nombre de points d'accès a pour effet de rendre l'allocation de canaux très délicate. En effet, la taille des zones de recouvrement entre les différentes zones de services des points d'accès est alors trop importante pour trouver facilement une solution au problème d'allocation de canaux. Cette surestimation a donc un effet naturel de dégradation du service.

Des travaux ont proposé et analysé différents choix que peut faire l'installateur lors du déploiement. Ces choix permettent de le guider sur le positionnement des points d'accès et le choix de leurs caractéristiques. Néanmoins, très vite, des algorithmes de planification automatique basés sur la prédiction de la couverture radio ont été proposés. En effet, la nature empirique du déploiement radio ne permet pas de faire face à la complexité intrinsèque du problème. La principale difficulté qui ne peut être prise en compte par un déploiement manuel est l'antagonisme des deux objectifs que sont la couverture radio et la minimisation des interférences **[RUN05]**.

II.2. Variables et paramètres du problème wLAN

II.2.1. Nombre de points d'accès

La minimisation du nombre de points d'accès n'est plus un objectif essentiel du déploiement wLAN. C'est pourquoi beaucoup de travaux fixent le nombre de points d'accès dans les algorithmes de recherche. Ces travaux se distinguent de la suite par deux caractéristiques : l'objectif est d'assurer la couverture avec un nombre de points d'accès fixé à l'avance. Le choix délicat du nombre de points d'accès relève alors de l'expertise de l'installateur qui va planifier le réseau.

Pourtant, lorsque le nombre de points d'accès est surestimé, la configuration de points d'accès obtenue conduit à un fonctionnement fortement dégradé par les interférences entre canaux. Il est nécessaire de choisir un nombre juste de points d'accès. C'est pourquoi, les travaux considèrent le nombre de points d'accès comme une variable du problème **[RUN05]**.

II.2.2. Position des points d'accès

La principale variable du problème wLAN est la position physique des points d'accès. Pour chaque point d'accès, il est nécessaire de déterminer ses coordonnées (x,y,z). La troisième variable z n'est pas définie quand les prédictions de couverture radio sont planaires. Selon la formulation du problème wLAN, ces trois variables de position sont soit continues, soit discrètes **[RUN05]**.

II.2.3. Paramètres antennaires

Les points d'accès wLAN sont équipés d'antennes moins évoluées que les stations de base GSM.

Les paramètres antennaires wLAN sont l'azimut ψ^k et la puissance d'émission P^k.

Stamatelos et Ephremides font partie des premiers chercheurs à avoir proposé une formulation du problème de planification wLAN en 1996. Ils ont pris en compte la puissance d'émission P^k en définissant une variable de position $p = (p_x,p_y)$ continue. Bahri et Chamberland, Wong et al. recherchent à la fois la position et la puissance d'émission du nombre minimal de points d'accès. L'ensemble des puissances d'émission est discret pour eux.

Les seuls travaux qui mettent en œuvre la directivité des points d'accès sont proposés par Aguado et al.. L'obtention de la direction d'émission est traitée comme un problème à part entière où l'on recherche l'orientation optimale des N émetteurs dans un ensemble discret d'orientations possibles. Ce scénario d'optimisation considère que la position, le nombre et la puissance des points d'accès sont déjà fixés **[RUN05]**.

En résumé. une configuration de points d'accès se compose toujours :

- du nombre de points d'accès : N
- de la position de chaque point d'accès : $(p^1,...,p^k,...,p^N)$

Les variables suivantes peuvent être exploitées :

- la puissance d'émission des points d'accès : $(P_E^1,...,P_E^k,...,P_E^N)$
- l'azimut des points d'accès : $(\psi^1,...,\psi^k,...,\psi^N)$

II.3. Scénarios de planification

Le comportement des ondes en environnement fermé est bien plus complexe qu'en espace libre. Il peut apparaître des zones très mal couvertes du fait de l'atténuation due aux murs ou de phénomènes d'interférences. Pour obtenir un réseau fiable, il faut au moins pouvoir y accéder dans toute la zone de service. C'est pourquoi il est important de positionner au mieux la ou les stations de base.

Avant de choisir le positionnement des antennes, il faut clairement définir quels sont les buts à atteindre. Est-ce que l'on veut simplement accéder au service sur toute la surface ou préfère-t-on un champ uniformément réparti ? Le positionnement ne se fera pas de la même façon dans les deux cas. Dans le premier cas, on cherchera simplement à obtenir un niveau de champ supérieur à la sensibilité du récepteur. Dans le deuxième cas, on essaiera de diminuer au maximum l'écart entre la valeur la plus faible et la plus élevée du niveau de champ pour tendre vers un niveau de couverture moyen.

On peut aussi introduire la notion de qualité de service. Pour cela, on peut définir une topologie permettant d'assurer un débit plus important dans certaines pièces. Mais le débit et le niveau de couverture ne sont pas les seuls critères variables. On peut essayer de diminuer les interférences en minimisant le recouvrement entre différentes stations de base. Ainsi, on diminue le bruit dû aux interférences.

On peut aussi rechercher pour chacun de ces critères, le nombre minimal de stations de base et leur position dans le plan **[RUN02]**.

II.4. Objectifs de la planification

Au-delà de l'objectif initial d'assurer la couverture radio en tout point, la planification doit permettre d'optimiser le réseau relativement à des critères plus élaborés. Les objectifs de planification sont définit pour obtenir une certaine qualité de services de la part du réseau wLAN. Un objectif de planification est défini comme suit **[RUN05]**:

Définition : *Un objectif de planification est une mesure des performances du réseau sans-fil à optimiser pour un type de service donné.*

II.4.1. Objectifs de couverture radio

Le premier service que doit offrir un réseau est la fourniture d'accès. Pour un réseau sans-fil, du fait de la nature du médium radio utilisé, il est nécessaire de garantir un accès aux utilisateurs sur toute la surface couverte en proposant un canal de communication de qualité.

Bahri et Chamberland, Wong et al., Prommak et al., Amaldi et al. décrivent le problème de planification à l'aide d'un programme linéaire. L'objectif principal de planification est soit la minimisation du nombre de points d'accès, soit l'obtention d'une qualité de service adéquate.

Frühwirt et Brisset, Anderson et McGeehan et He et al. proposent de maximiser le nombre de points du plan couvert. Frühwirt et Brisset et He et al. travaillent avec un nombre de points d'accès constant tandis qu'un Anderson et McGeehan minimisent le nombre de points d'accès. Le même objectif est présenté par Stamatelos et Ephremides sous la forme d'une minimisation du nombre de points de test non couverts avec N fixe.

Mateus et al. définissent un critère de couverture qui maximise la somme des puissances reçues en chaque point de test. Maksuriwong et al. proposent de maximiser le rapport signal sur bruit moyen obtenu pour tout le plan. Ces deux critères ne garantissent pas l'absence de trous de couverture. En effet, la somme des puissances reçues ne rendent pas compte de la présence ou non de ces trous.

Sherali et al. proposent un critère unique qui minimise à la fois l'atténuation moyenne du signal et l'atténuation maximale de chaque point de test **[RUN05]**.

II.4.2. Objectifs de recouvrement et d'interférences

Le deuxième critère de planification communément rencontré a rapport aux interférences. En effet, la dégradation du débit fourni par le canal radio peut être liée à un rapport signal sur interférences trop faible.

Il est possible d'optimiser l'utilisation des ressources spectrales en répartissant au mieux les canaux affectés aux points d'accès présents sur le site. L'allocation de ces canaux est un problème d'optimisation combinatoire qui

peut se réduite au problème NP-complet de coloriage de graphes. Il est communément référencé par l'acronyme *FAP* ("*Frequency Assignment Problem*").

En planification wLAN, le nombre de canaux et de points d'accès présents est moins important et des heuristiques plus simples donnent de bons résultats pour le *FAP*.

Quelques travaux, dont ceux de Mathar et al. ont proposé de réaliser de façon conjointe à la fois l'allocation des canaux et la planification du réseau. Il en est de même en planification wLAN où Bahri et Chamberlain, Mateus et al., Molina et Gonzalez, Lee et al., Prommak et al. et Wertz et al. proposent de rajouter des variables permettant d'allouer les canaux aux points d'accès dans le processus de planification.

Les travaux de Stamatelos et Ephremides et d'Aguado et al. définissent un objectif principal qui a pour but de réduire le recouvrement entre les zones de service des points d'accès sans pour autant réaliser l'allocation des canaux. Si la taille des zones de recouvrement entre des points d'accès voisins est faible en surface, il sera bien plus aisé d'allouer des canaux par la suite.

Stamatelos et Ephremides proposent de minimiser à la fois la taille de la surface non couverte et la taille de la zone interférée.

Le problème de planification cherche à sélectionner des solutions qui couvrent tout l'environnement et qui minimisent le recouvrement entre les zones de service des points d'accès **[RUN05]**.

II.4.3. Objectifs de trafic et de qualité de service

Seuls les travaux les plus récents définissent des problématiques de qualité de service. Ils répondent à la nécessité de fournir des réseaux sans-fil qui garantissent une bande passante aux utilisateurs.

Adickes et al. ont proposé en 2002 d'intégrer un objectif de *QoS* pour un ensemble de points, les *TTP* ("*Traffic Test Points*"), qui possèdent une demande de trafic que le réseau doit fournir. Ce placement des points de trafic s'inspire des travaux de planification cellulaire. Dans ce modèle, la capacité des points d'accès n'est pas prise en compte et il est possible de trouver des solutions où un point d'accès ne pourra fournir assez de bande passante à tous les *TTP* qu'il couvre.

Les travaux ultérieurs formulent le problème à l'aide d'un programme linéaire et s'assurent que le trafic total demandé par l'ensemble des *TTP* couverts par un point d'accès n'excède pas sa capacité.

En 2004, Molina et Gonzalez combinent un critère de couverture et un critère de trafic. Ce dernier vérifie également que la demande des *TTP* n'excède pas la capacité du point d'accès fixée. De même, Bahri et Chamberland imposent la même contrainte de trafic dans un programme linéaire qui minimise le nombre de points d'accès.

Les deux travaux proposés par Lee et al. et Amaldi et al. sont les premiers à avoir choisi de définir comme objectif principal d'optimisation un critère de qualité de service. Ils ont défini un modèle qui a pour objectif de répartir le mieux possible entre les *TTP* le débit fourni par les points d'accès. La couverture et le recouvrement sont pris en compte par la mise en place de contraintes.

Lee et al. définissent la même contrainte de capacité sur les points d'accès pour obtenir une répartition uniforme du trafic sur ces points d'accès.

Amaldi et al. se placent dans le contexte des réseaux wLAN et considèrent qu'un seul canal est utilisé dans le réseau pour tous les points d'accès. La mesure de capacité du réseau est la somme des probabilités d'accès pour tous les utilisateurs du réseau. L'objectif présenté par Amaldi et al. est de maximiser la capacité du réseau sous la contrainte de couverture totale **[RUN05]**.

II.5. Formulations du problème wLAN

Le problème de planification peut être défini soit à l'aide d'une formulation continue soit à l'aide d'une formulation discrète.

II.5.1. Approche continue

La formulation continue du problème wLAN est basée sur la définition d'un espace continu des variables du problème. Ainsi, trois variables (x,y,z) de position, une variable représentant la puissance d'émission p et une variable d'azimut ψ peuvent être définies pour chaque point d'accès. Si l'on cherche à planifier N points d'accès, la dimension de l'espace des solutions vaut $D = 5.N$.

Une solution est représentée par un vecteur :

$$X = [(x_1, y_1, z_1, p_1, \psi_1), \ldots, (x_i, y_i, z_i, p_i, \psi_i), \ldots, (x_N, y_N, z_N, p_N, \psi_N)]$$

Si le problème ne comporte pas de contrainte, une fonction d'évaluation unique est définie par :

$$\hat{x} = Arg\,min_{x \in \Omega^0} \; f(X), \quad f : \mathrm{R}^D \rightarrow \mathrm{R} \text{ avec } D \text{ le nombre de variables}$$

Pour pouvoir prendre en compte les contraintes d'un problème d'optimisation, un terme de pénalisation est rajouté à la fonction de coût $f(X)$. Ce terme de pénalisation pondère l'évaluation de $f(X)$ quand la contrainte n'est pas satisfaite. Ainsi, une contrainte d'inégalité $c_i(X) \geq 0$ qui s'applique au point de test i pour la solution X est incluse dans la fonction de coût comme suit :

$$P(X, \mu) = f(X) + \mu \sum_{i=1}^{Np} max(0, c_i(X))$$

Le coefficient de pénalisation μ permet de discriminer de façon continue des solutions qui respectent la contrainte $c_i(x)$ de celles qui ne la respectent pas. Avec une formulation continue, la minimisation du nombre de points d'accès ne peut s'envisager avec une seule étape d'optimisation. Pour pouvoir ajuster le nombre de points d'accès, il est nécessaire d'appliquer plusieurs fois la même

heuristique de résolution en ajustant intelligemment le nombre de points d'accès à chaque lancement **[RUN05]**.

II.5.2. Approche combinatoire

Dans une approche combinatoire du problème wLAN, un ensemble de positions candidates de points d'accès, référencé par E_{AP} et de taille finie M, est déterminé soit par l'installateur soit automatiquement par l'heuristique d'optimisation. Un point d'accès candidat numéro k est désigné par sa position b_l, sa puissance P^k et son azimut ψ^k.

Chaque candidat est défini dans le processus d'optimisation par une variable binaire x_k qui vaut 1 si le site candidat fait partie de la solution courante et 0 sinon. Ainsi, une solution issue d'une formulation discrète est donnée par le vecteur binaire des x_k, $k \in [0,M]$:

$$(x_1,\ldots,x_k,\ldots,x_M)$$

L'heuristique d'optimisation sélectionne le meilleur sous-ensemble de N points d'accès candidats selon les objectifs définis au préalable. Un ensemble de contrainte d'égalité ($c_i(x)$, $i \in [1,I]$ ou d'inégalité ($d_j(x)$, $j \in [1,J]$) peuvent être définies. La formulation récurrente du problème d'optimisation est réalisée à l'aide d'un programme mathématique qui présente une fonction d'évaluation principale à optimiser et une série de contraintes, linéaires ou non. Un exemple générique de programme mathématique est donné par :

$$\hat{x} = Arg\,min\, f(X)$$
$$\begin{cases} c_i(x) \geq 0 & i \in [1,I] \\ d_j(x) = 0 & j \in [1,J] \end{cases}$$

La formulation discrète est la formulation qui a été la plus souvent exploitée dans la littérature relative au problème wLAN **[RUN05]**.

II.6. Métaheuristiques pour le problème wLAN

Nous allons présenté dans cette section quelques métaheuristiques qui ont été utilisées pour résoudre le problème de planification wLAN.

II.6.1. Recuit simulé

Cette métaheuristique a fait parti des toutes premières méthodes testées pour la planification wLAN en 1994, par Anderson et McGeehan. Le voisinage de la solution de coût $f(s)$ est défini par l'ensemble des solutions qui peuvent être atteintes par des changements aléatoires des paramètres du système pour arriver à une nouvelle configuration s' de coût $f(s')$. Les paramètres du système sont les positions en x et y dans le plan 2D de chaque point d'accès. L'amplitude des changements est régie par la température T du système à l'aide d'une distribution Gaussienne à moyenne nulle et de variance T.

La température initiale est initialisée à 80 mètres pour un environnement de taille 650 x 650 mètres et celle-ci diminue de 10 mètres toutes les 100 itérations. Pour chaque test effectué, l'algorithme trouve une configuration très

proche de l'optimum. Les tests ont été réalisés sur un PC Pentium 60 MHz et ont duré 25 heures en moyenne. Avec la puissance de calcul de l'époque, l'utilisation d'une métaheuristique est synonyme d'un temps de traitement conséquent. De plus, pour pouvoir obtenir de tels temps de calcul, il a fallu définir un environnement de test très simple et utiliser un modèle de prédiction de couverture radio empirique.

Une version plus complexe d'algorithme de recuit a été implantée par Kamenetsky et Unbehaun. Ils ont choisi de définir 5 niveaux de température et une probabilité d'acceptation d'une solution dégradée qui suit l'équation :

$$P_a(T) = min\left(1, exp\left(-y\frac{F(S_1)/F(S_0)-1}{(T/T_0)^2}\right)\right)$$

S_0 est la solution initiale et S_1 la solution courante de test. Ils ont défini le même type de voisinage qu'Anderson et McGeehan où des déplacements aléatoires des points d'accès sont réalisés à l'intérieur d'une zone circulaire. Le rayon de ce cercle devient de plus en plus petit avec l'évolution de la température **[RUN05]**.

II.6.2. Recherche Tabou

Bahri et Chamberland ont implanté une procédure de recherche tabou. Une solution voisine s' de la solution s est obtenue en modifiant la valeur d'une seule de ses variables. La liste tabou stocke les sites candidats qui ont été modifiés lors des T dernières itérations. Cette liste présente une taille variable, choisie aléatoirement selon une loi uniformément répartie dans l'intervalle [5,10]. La taille de cet intervalle est fixée expérimentalement.

L'algorithme a été appliqué à un environnement de test conséquent : un immeuble de 10 étages où chaque étage a une superficie de 5000 m^2. 400 sites candidats sont répartis uniformément dans l'immeuble. L'algorithme a été lancé pour 8 instances du problème où le nombre de points de test varie entre 100 et 800. La recherche est stoppée au bout de 100 itérations. Une borne inférieure de la fonction de coût est calculée pour le problème avec le solveur *CPLEX*. Ils observent que la solution tabou est relativement éloignée de la borne inférieure du problème quand le nombre de points de test est élevé pour un réseau de type 802.11a.

Il est important de noter que l'algorithme tabou effectue également la recherche des canaux fréquentiels. Comme la borne inférieure du problème complet n'a pas pu être calculée avec *CPLEX*, la borne représentée est obtenue sans allocation fréquentielle. Les auteurs attribuent à cette hypothèse la différence de qualité entre la solution tabou et la borne inférieure de *CPLEX*.

Les tests menés avec un réseau de type 802.11b n'ont pas permis de trouver de solution réalisable pour les instances qui présentent un nombre élevé de points de test (400, 500 et 600). C'est l'allocation fréquentielle qui a rendu la recherche plus difficile car ils n'ont utilisé que les 4 canaux disjoints de 802.11b pour le *FAP* **[RUN05]**.

II.6.3. Algorithmes génétiques

Adickes et al. ont choisi d'implanter un algorithme génétique. Dans l'implantation de leur algorithme génétique, ils définissent un chromosome comme une solution $\{(x_1,y_1),\ldots,(x_n,y_n)\}$ où chaque gène (x_i,y_i) donne la position d'un point d'accès. La population initiale est générée en choisissant aléatoirement la position des points d'accès de chacun des 30 individus de la population.

Adickes et al. utilisent une formulation multi-objectif hiérarchique de leur problème. À partir de règles sur la hiérarchie de leurs 3 fonctions d'évaluations, ils vont trier les individus de la population courante et leur assigner un poids unique f_i. Une mesure de la qualité totale de la population est définie comme suit :

$$F = \sum_{i=1}^{N_{pop}} f_i$$

Ils en déduisent une probabilité de sélection $p_i = f_i / F$ utilisée pour sélectionner les parents qui généreront une nouvelle partie de la population par recombinaison. La nouvelle population contient également les deux meilleures solutions. La mutation est appliquée à l'ensemble de la nouvelle population avec une probabilité $p_m = 0.1$.

Adickes et al. ont réalisé 3 tests pour valider les performances de leur algorithme. Le premier test compare leurs résultats sur un environnement présenté par Tang et al. de taille 75 x 30 mètres. Leur algorithme génétique trouve également une solution qui totalise 100% de couverture, mais avec une meilleure atténuation moyenne que Tang et Al. Pour le second test, ils recherchent exhaustivement l'optimum pour le placement de 2 points d'accès sur le même environnement et montrent que leur algorithme trouve bien la même solution. Le dernier test compare les solutions de la simulation à celles qui ont été proposées par des ingénieurs radio avec leur processus de déploiement empirique. Dans tous les cas, la solution proposée par l'algorithme présente des caractéristiques bien meilleures.

Aguado et Al et Ji et al. ont également implanté une version simplifiée d'algorithme génétique pour comparer ses performances à d'autres heuristiques. Les deux travaux montrent que les solutions obtenues sont proches des solutions optimales mais que leur recherche nécessite plus d'évaluations de fonction. Aguado et Al concluent également qu'une implantation plus élaborée de l'algorithme devrait fournir de meilleurs résultats, notamment en utilisant une méthode où plusieurs sous-populations sont traitées en parallèle et où une communication inter-populations permet de propager les individus performants **[RUN05]**.

III. Environnement de développement

C++ Builder est l'environnement de développement basé sur C++ proposé par Borland. Fort du succès de Delphi, Borland a repris la philosophie,

l'interface et la bibliothèque de composants visuels de ce dernier pour l'adapter depuis le langage Pascal Orienté Objet vers C++ répondant ainsi à une large faction de programmeurs peu enclins à l'utilisation du Pascal qu'ils jugent quelque peu dépassé.

Tout d'abord Builder C++ est un outil *RAD* ("*Rapid Application Development*"), c'est à dire tourné vers le développement rapide d'applications sous Windows.

En un mot, C++ Builder permet de réaliser de façon très simple l'interface des applications et de relier aisément le code utilisateur aux événements Windows, quelle que soit leur origine (souris, clavier, événement système, … etc.)

Pour ce faire, C++ Builder repose sur un ensemble très complet de *composants visuels* prêts à l'emploi. La quasi totalité des contrôles de Windows (boutons, boîtes de saisies, listes déroulantes, menus et autres barres d'outils) y sont représentés, regroupés par famille. Leurs caractéristiques sont éditables directement dans une fenêtre spéciale intitulée *éditeur d'objets*. L'autre volet de cette même fenêtre permet d'associer du code au contrôle sélectionné.

Il est possible d'ajouter à l'environnement de base des composants fournis par des sociétés tierces et même d'en créer soit même **[SGE05]**.

IV. Choix techniques

Les algorithmes de notre approche ont été implémentés sur un PC qui possède les caractéristiques suivantes :

- Microprocesseur Intel Pentium 4 de vitesse 1.7 GHZ
- Une mémoire RAM de 512 MO
- Disque dur de 80 GO

V. Description du modèle

Notre approche permet de planifier ou bien positionner les différents points d'accès d'un réseau local sans fil sur un espace qui se définit par sa longueur, sa largeur et aussi son hauteur.

Les éléments de base de notre approche sont décrits comme suit **[CLE00]**:

1. **Espace de recherche**

   L'espace de recherche S est défini comme l'ensemble fini de toutes les variables (x,y,z) :

   $$S = \{s_i\}$$

2. **Fonction objectif**

   Une fonction objectif f sur S, dans un ensemble de valeurs dont les meilleures positions sont les états de la solution :

   $$S \xrightarrow{f} M = \{m_i\}$$

3. **Distance**

   Dans le cas d'un voisinage physique, une distance *d* a été définie dans l'espace de recherche.

   Soient x_1 et x_2 deux positions. La distance entre ces positions est définie par :

$$d(x_1, x_2) = \|x_2 - x_1\|$$

   Nous notons que :

$$\|x_2 - x_1\| = \|x_1 - x_2\|$$
$$\|x_2 - x_1\| = 0 \Leftrightarrow x_1 = x_2$$

4. **Position d'une particule**

   Une position est définie par trois variables de position (*x*,*y*,*z*) et une variable représentant la puissance d'émission *p* :

$$x' = (x, y, z, p), (x, y, z) \in (X, Y, Z)$$

5. **Vitesse d'une particule**

   C'est un opérateur *v* ; l'application de ce dernier à une position pendant un pas de temps, donne une autre position. Donc, ici, c'est une permutation d'éléments, c'est-à-dire une liste de transpositions. La longueur de cette liste est $\|v\|$. Une vitesse est alors définie par :

$$v = ((i_k, j_k)), k \uparrow_1^{\|v\|} \text{ et } \|v\| \leq 3$$

   Ce qui signifie "changer les variables (i_1, j_1), puis les variables (i_2, j_2), … etc." et enfin les variables $(i_{\|v\|}, j_{\|v\|})$. Une vitesse nulle est une vitesse équivalente à $\varnothing$, la liste vide.

6. **Opposé d'une vitesse**

$$\begin{aligned} \neg v &= ((i_k, j_k)), k \downarrow_1^{\|v\|} \\ &= \left(\left(i_{\|v\|-k+1}, j_{\|v\|-k+1}\right)\right), k \uparrow_1^{\|v\|} \end{aligned}$$

   Cela signifie "faire les mêmes transpositions que dans *v*, mais dans l'ordre inverse".

   Il est facile de vérifier que $\neg\neg v = v$ et $v \oplus \neg v \cong \varnothing$ (*addition "vitesse plus vitesse"*)

7. **Opérations**

   L'application des équations de base de l'optimisation par essaim de particules nécessite la définition des opérations suivantes :

Mouvement (addition) "position plus vitesse"

Soient p_1 une position et v une vitesse. La position $p_2 = p_1 + v$ est trouvée en appliquant la première transposition de v à p_1, alors le deuxième au résultat, … etc. :

$$(position, vitesse) \xrightarrow{plus} position$$

Soustraction "position moins position"

Soient p_1 et p_2 deux positions. La différence $p_2 \ominus p_1$ est définie comme la vitesse v. Ce qui signifie "trouver une vitesse v qui est une fois appliquée à p_1 donne p_2" :

$$(position, position) \xrightarrow{moins} vitesse$$

Addition "vitesse plus vitesse"

Soient v_1 et v_2 deux vitesses. L'addition $v_1 \oplus v_2$ est considérée comme la liste des transpositions qui contient d'abord ceux de v_1, suivi par ceux de v_2 :

$$(vitesse, vitesse) \xrightarrow{plus} vitesse$$

Particulièrement, cette opération est définie pour que :

$$v \oplus \neg v = \varnothing$$
$$\|v_1 \oplus v_2\| \le \|v_1\| + \|v_2\|$$
$$\|v_1 \oplus v_2\| \le 3$$
$$v_1 \oplus v_2 \ne v_2 \oplus v_1$$

Multiplication "coefficient fois vitesse "

Soit c un coefficient réel et v une vitesse :

$$(réel, vitesse) \xrightarrow{fois} vitesse$$

Il y a des cas différents, selon la valeur de c.

- **Cas $c = 0$:** $cv = \varnothing$
- **Cas $c \in$] 0,1] :** $cv = ((i_k, j_k)), k \uparrow_1^{\|cv\|}$ et $\|cv\| \le 3$
- **Cas $c > 1$**

  Cela signifie que : $c = k + c', k \in N^*, c' \in [0,1[$

  Donc : $cv = \underbrace{v \oplus v \oplus \ldots \oplus v}_{k \text{ fois}} \oplus c'v$

- **Cas $c < 0$**

  En écrivant $cv = (-c)\neg v$, l'un des cas précédents va être considéré.

Nous notons que $v_1 \cong v_2 \Rightarrow cv_1 \cong cv_2$ si c est un entier, mais ce n'est pas toujours vrai, dans le cas général.

L'architecture de notre système est décrite comme suit :

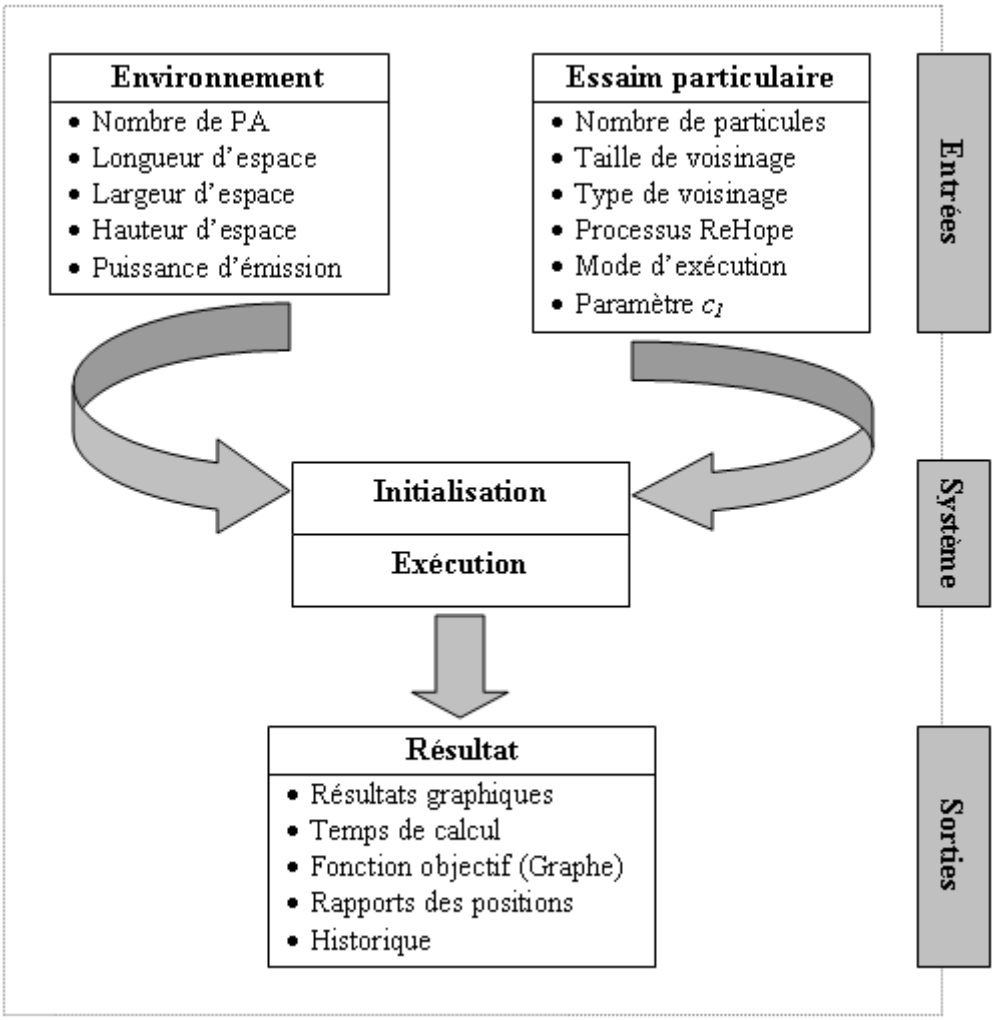


Figure 4.1 : modèle du système.

Cette modélisation est basée sur l'algorithme de base suivant :

[**Initialisation**]
Initialiser aléatoirement la population
[**Exécution**]
Répéter
  Évaluer la valeur de la fonction objectif
  Pour chaque particule
    1. Modifier la vitesse en utilisant l'équation :
       $v(t+1) = c_1 v(t) + c_2\left(p_i(t) - x(t)\right) + c_3\left(p_g(t) - x(t)\right)$
    2. Puis déplacer en utilisant l'équation :
       $x(t+1) = x(t) + v(t+1)$
  Fin pour
Jusqu'à ce que (*le processus converge*)

Algorithme 4.1 : algorithme de base du système.

La fonction objectif de notre système prend deux formes, l'une lorsque la puissance d'émission de différents points d'accès est homogène :

$$min\ f(x) = \frac{1}{N}\sum_{i=1}^{N} P_i$$

et l'autre lorsque la puissance d'émission est non homogène :

$$min\ f(x) = \frac{1}{N}\sum_{i=1}^{N} max\left(0, \overline{P} - P_i\right)$$

où

- t : le temps ;
- x : la position de la particule ;
- v : sa vitesse ;
- p_i : sa meilleure position atteinte ;
- p_g : la meilleure des meilleures positions atteintes dans son voisinage ;
- c_1, c_2, c_3 : les coefficients de confiance pondérant les trois directions possibles (*volontariste*, *conservatrice*, *suiviste*), sont choisis à chaque pas de temps au hasard dans un intervalle donné ;
- N : nombre de points d'accès fixé ;
- P_i : puissance d'émission de chaque point d'accès et $\overline{P}$ est la puissance moyenne des puissances d'émission.

L'algorithme s'exécute tant qu'un critère de convergence n'a pas été atteint et ceci est le nombre fixe d'itérations.

VI. Présentation du logiciel

Le logiciel, une fois lancé, se présente sous la forme d'une fenêtre principale possédant une barre de menu, barre d'outils, zone de travail et une barre d'état.

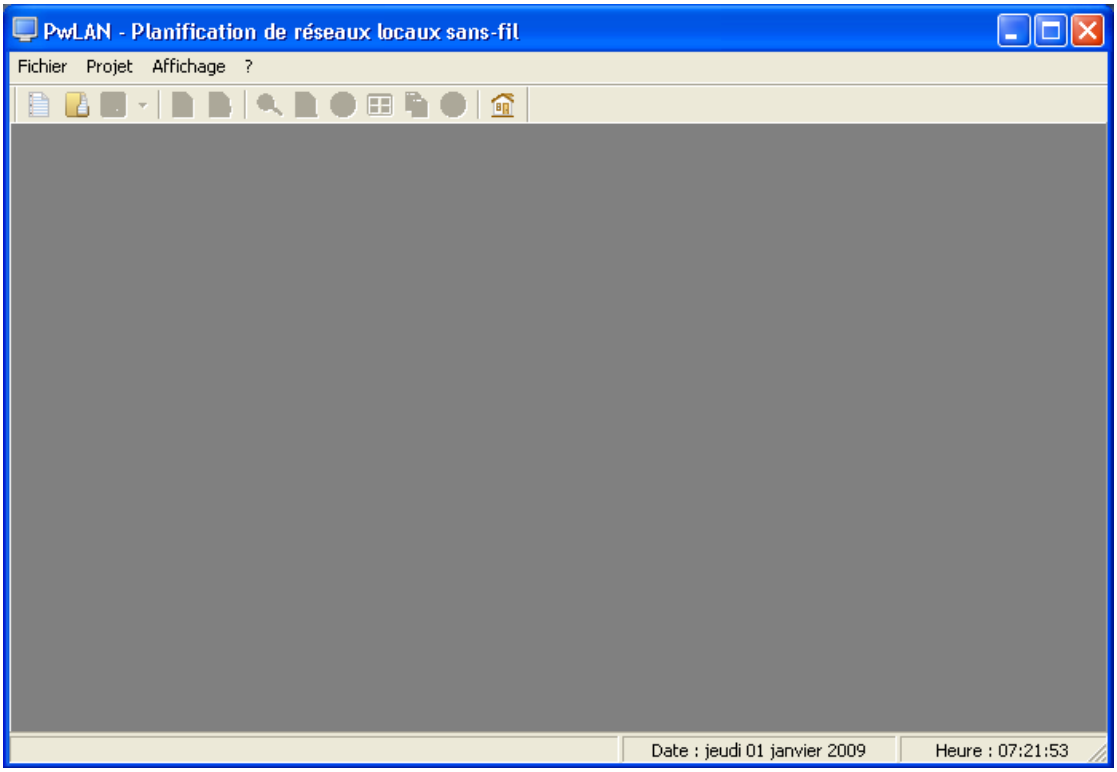


Figure 4.2 : fenêtre principale du logiciel.

La barre de menu contient les différentes commandes :

- Menu *Fichier* :

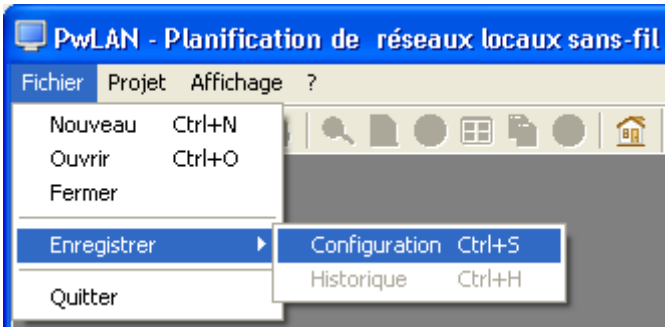


Figure 4.3 : menu "*Fichier*".

Commande	**Tâche**
Nouveau	Introduire une nouvelle configuration à travers la fenêtre « **Nouveau…** ».
Ouvrir	Ouvrir une configuration existe déjà sous la forme d'un fichier texte.
Fermer	Fermer l'exercice actuel.
Enregistrer	Enregistrer soit la configuration actuelle soit l'historique de la dernière exécution.
Quitter	Quitter l'application.

- Menu *Projet* :

Figure 4.4 : menu "*Projet*".

Commande	Tâche
Initialiser	Lancer la première étape de notre système qui est l'initialisation.
Exécuter	Déclencher le traitement ou bien l'exécution qui est la deuxième étape du système.

- Menu *Affichage* :

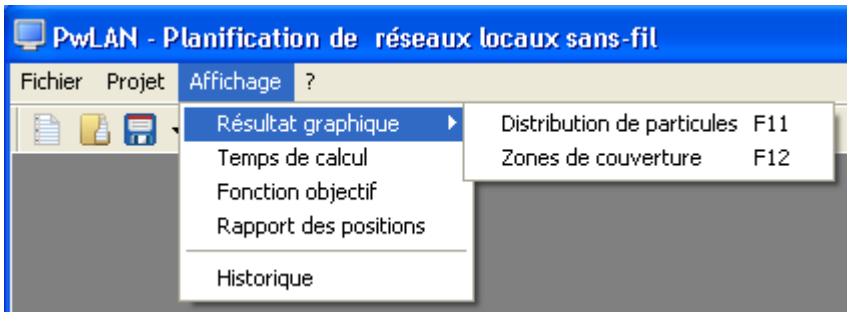


Figure 4.5 : menu "*Affichage*".

Commande	Tâche
Résultat graphique	Afficher soit la distribution de particules soit les zones de couverture des points d'accès sous une forme graphique.
Temps de calcul	Afficher le temps de calcul de notre système en secondes.
Fonction objectif	Afficher la fonction objectif par rapport le nombre d'itérations sous une forme de graphe.
Rapport des positions	Afficher les coordonnées des points d'accès avant et après l'exécution.
Historique	Ouvrir le fichier texte qui contient l'historique de l'exercice en cours.

- Menu ? :

Figure 4.6 : menu "?".

Commande	Tâche
Aide	Ouvrir les rubriques d'aide du logiciel (**Figure 4.7**).
À propos	Afficher la fenêtre « **À propos…** » du logiciel (**Figure 4.8**).

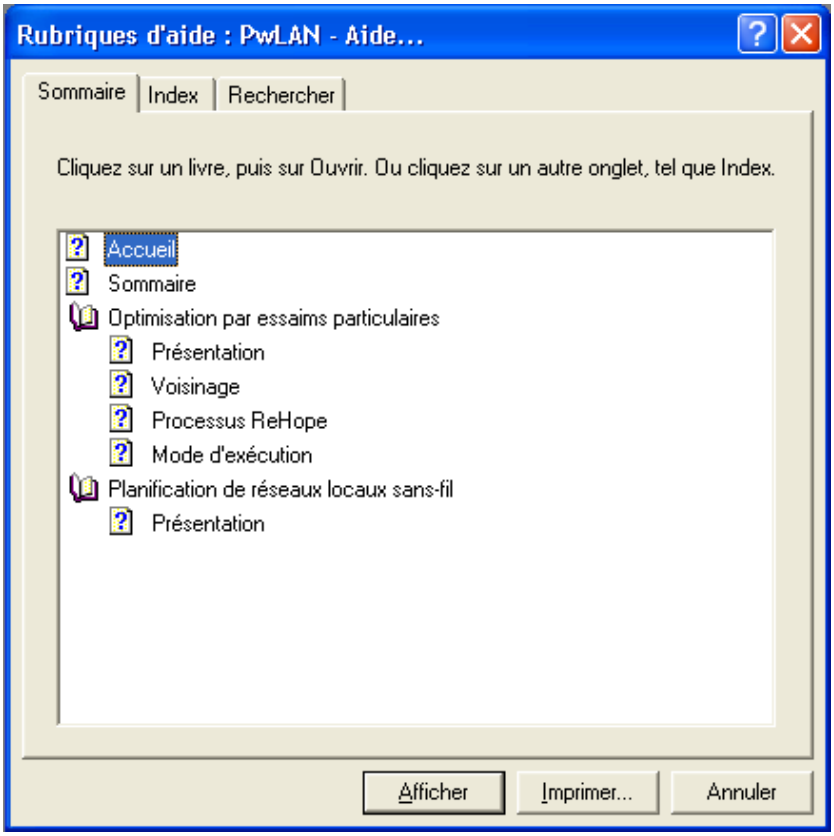


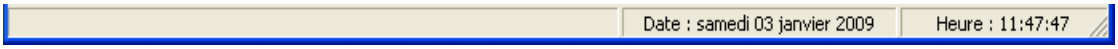
Figure 4.7 : rubriques d'aide.

Figure 4.8 : fenêtre "*À propos*".

Sur la barre d'outils s'existe toutes les commandes nécessaires :

Icône	Commande	Icône	Commande
	*Nouveau*		*Zones de couverture*
	*Ouvrir*		*Temps de calcul*
	*Enregistrer*		*Fonction objectif*
	*Initialiser*		*Rapport des positions*
	*Exécuter*		*Historique*
	*Distribution de particules*		*À propos*

La barre d'état est divisée en trois zones :



1) La première est une zone d'état pour indiquer s'il y a un exercice en cours ;
2) La deuxième pour afficher la date du jour ;
3) La dernière pour afficher l'heure.

VII. Étude de cas

Pour l'essai de notre logiciel, nous proposons une étude de cas avec la configuration suivante :

- Nombre de points d'accès : **10 points**
- Longueur du bloc : **100 mètres**
- Largeur du bloc : **80 mètres**
- Hauteur du bloc : **60 mètres**
- Puissance d'émission : **Homogène**
- Nombre de particules : **100 particules**
- Taille de voisinage : **50 voisins**
- Type de voisinage : **Social**
- Processus *ReHope* : **Simple**
- Mode d'exécution : **Parallèle**
- Paramètre c_1 : **0.50**

Nous saisissons cette configuration sur les champs de saisie de la fenêtre « **Nouveau…** » :

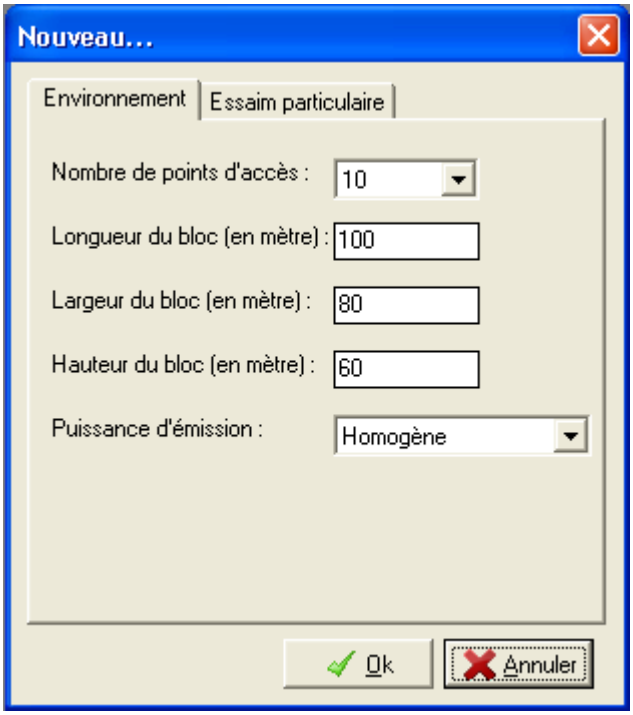


Figure 4.9 : saisie de configuration en face *Environnement*.

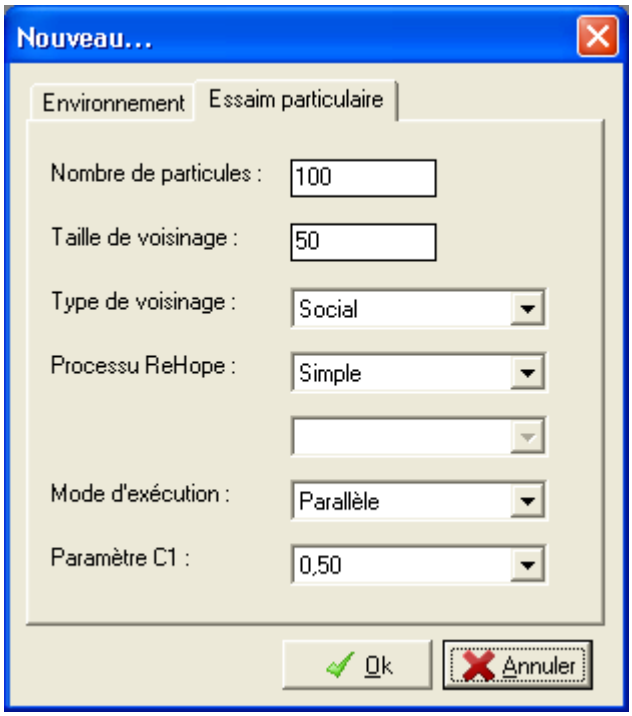


Figure 4.10 : saisie de configuration en face *Essaim particulaire*.

ou bien, nous ouvrons une configuration déjà enregistrée dans un fichier texte :

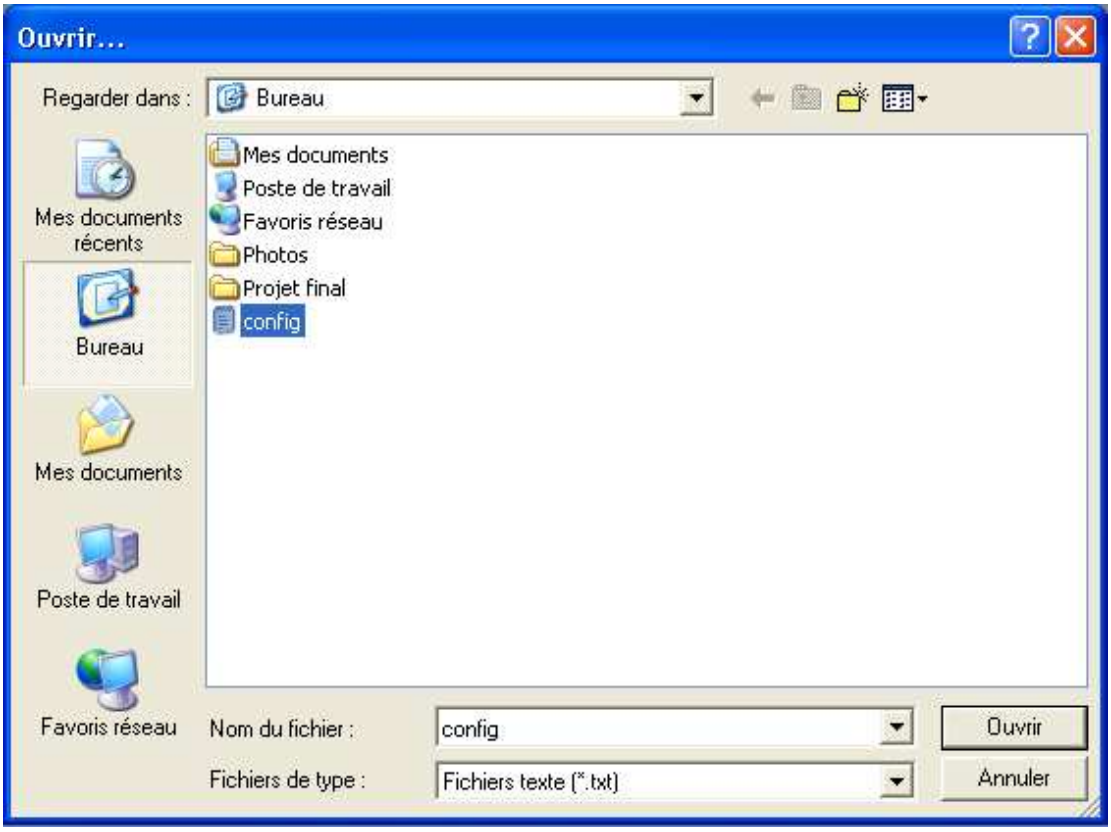


Figure 4.11 : ouvrir une configuration enregistrée dans un fichier.

Ensuite, nous cliquons sur la commande *Initialiser* ou sur la touche *F8* pour lancer la phase d'initialisation de notre système. Un message s'affiche lorsque cette phase est terminée :

Figure 4.12 : message affiché une fois l'initialisation terminée.

Ainsi que pour la phase d'exécution, nous cliquons sur la commande *Exécuter* ou sur la touche *F9* et lorsque l'exécution est terminée, un message est affiché :

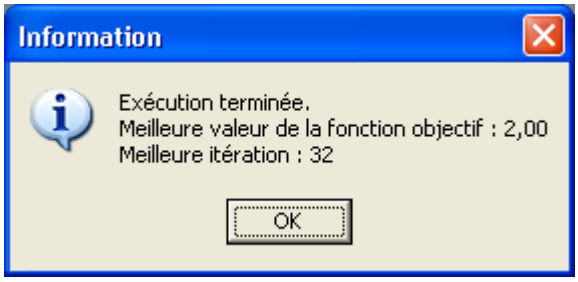


Figure 4.13 : message affiché une fois l'exécution terminée.

Ce message nous indique que l'exécution est terminée comme nous affiche la meilleure valeur de la fonction objectif après l'exécution et la meilleure itération aussi.

VIII. Résultats d'application

Une fois l'exécution est terminée, nous pouvons consulter les différents modes de résultats tels que :

- **Distribution de particules :**

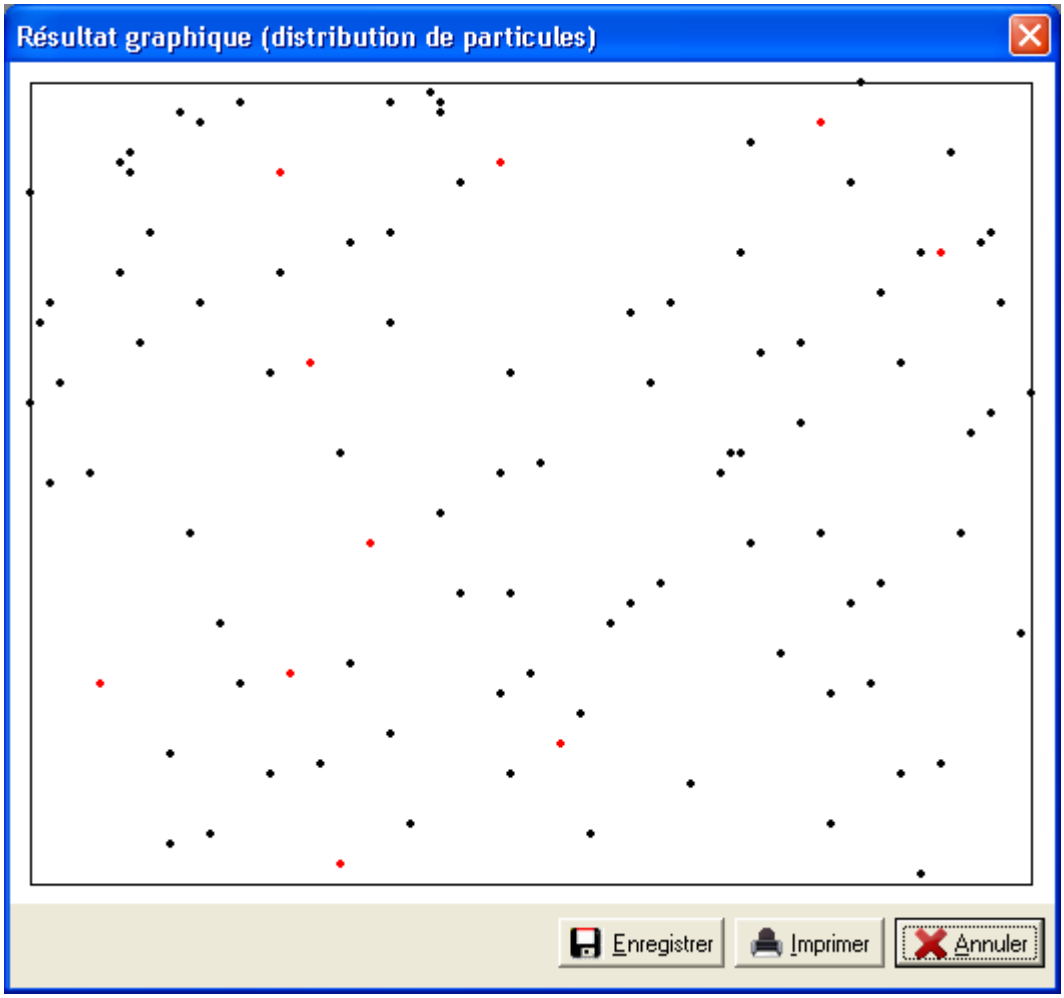


Figure 4.14 : distribution graphique de particules après l'exécution.

Les points rouges représentent les points d'accès et les autres particules sont représentées par des points noirs.

- **Zones de couverture :**

Chaque point d'accès a une zone de couverture, ces zones sont tracées par des cercles bleus autour des points d'accès qui sont à leurs tours représentés par des points rouges.

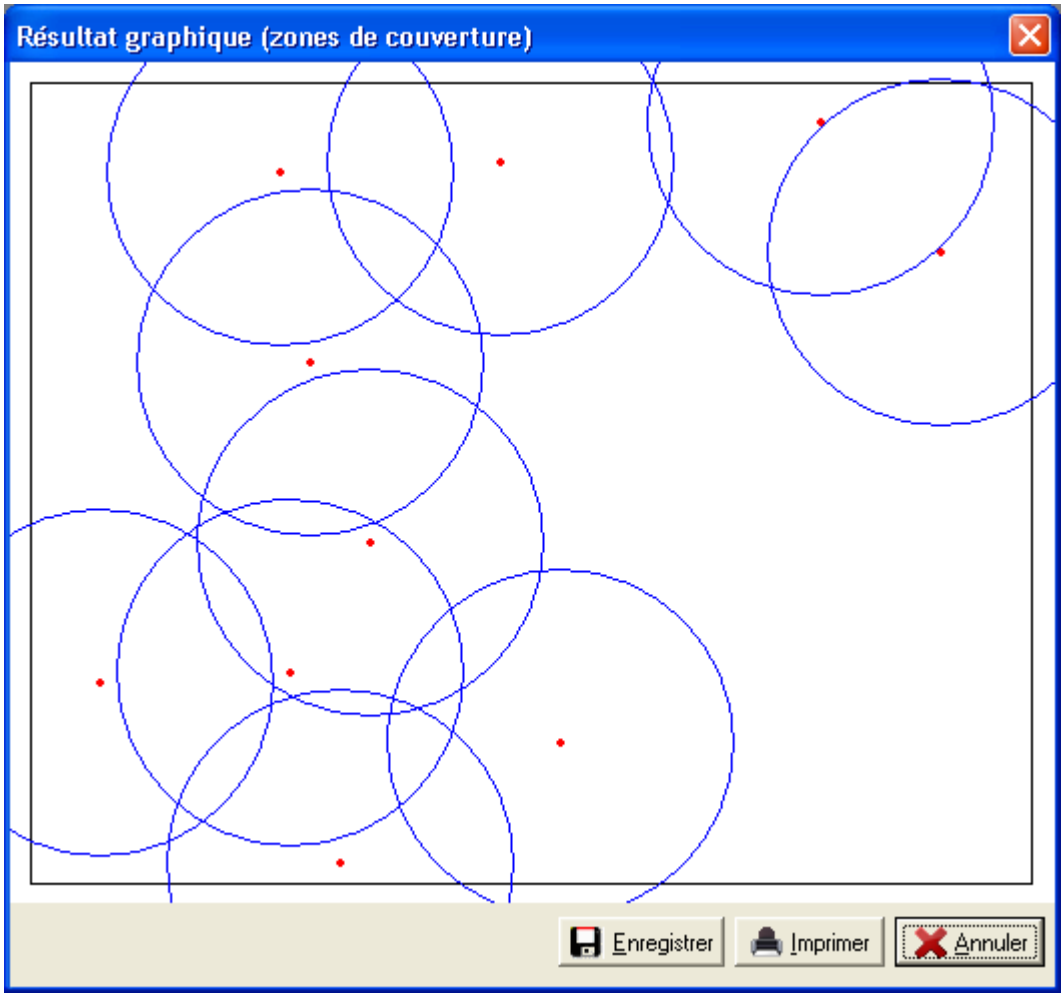


Figure 4.15 : zones de couverture représentées par des cercles.

- **Temps de calcul :**

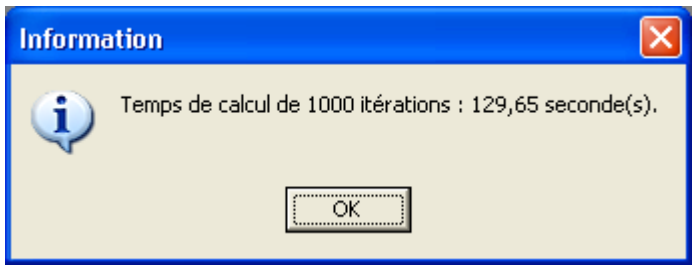


Figure 4.16 : temps de calcul après l'exécution.

Ce message nous indique le temps de calcul après que la phase d'exécution de notre système est terminée.

- **Fonction objectif :**

Sous forme d'un graphe, nous pouvons consulter l'évolution de la fonction objectif pendant l'exécution de notre système.

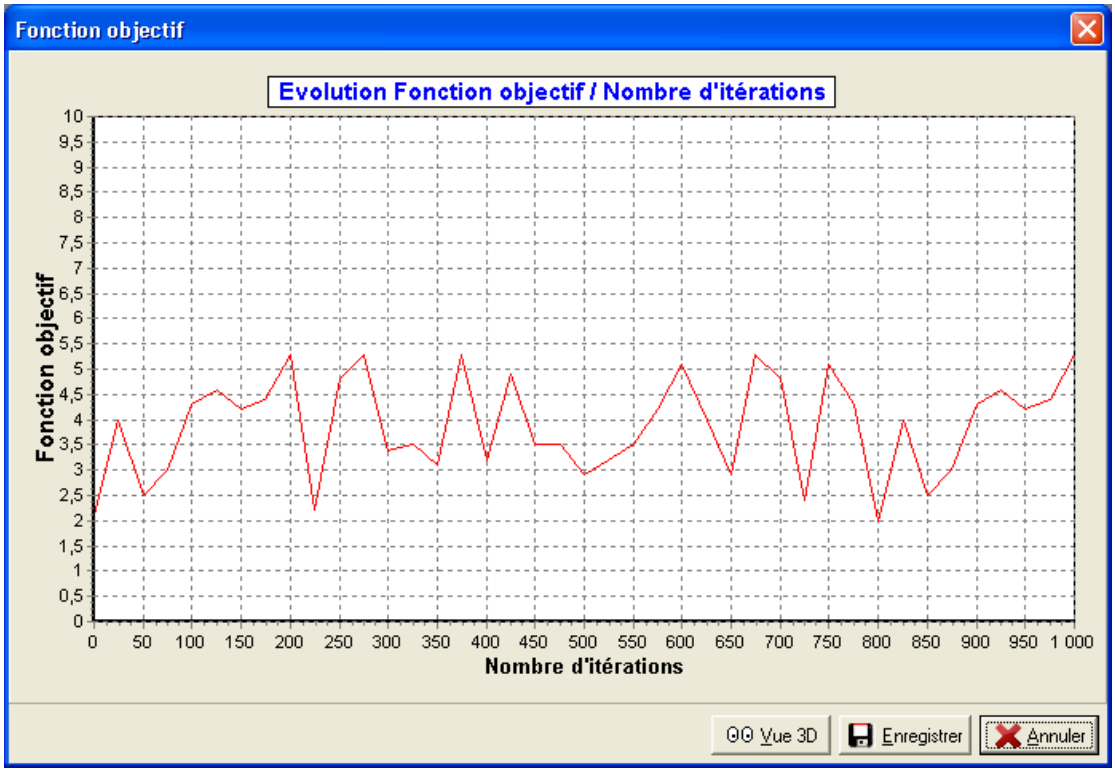


Figure 4.17 : évolution de la fonction objectif par rapport au nombre d'itérations.

- **Rapport des positions :**

Après l'exécution, les coordonnées des points d'accès en 3D sont classées dans un rapport :

Rapport des positions

Point d'accès	X	Y	Z
N° 1	47	8	2
N° 2	28	28	50
N° 3	7	60	25
N° 4	26	59	5
N° 5	91	17	44
N° 6	34	46	50
N° 7	31	78	23
N° 8	53	66	9
N° 9	79	4	24
N° 10	25	9	48

Annuler

Figure 4.18 : coordonnées des points d'accès après l'exécution.

- **Historique :**

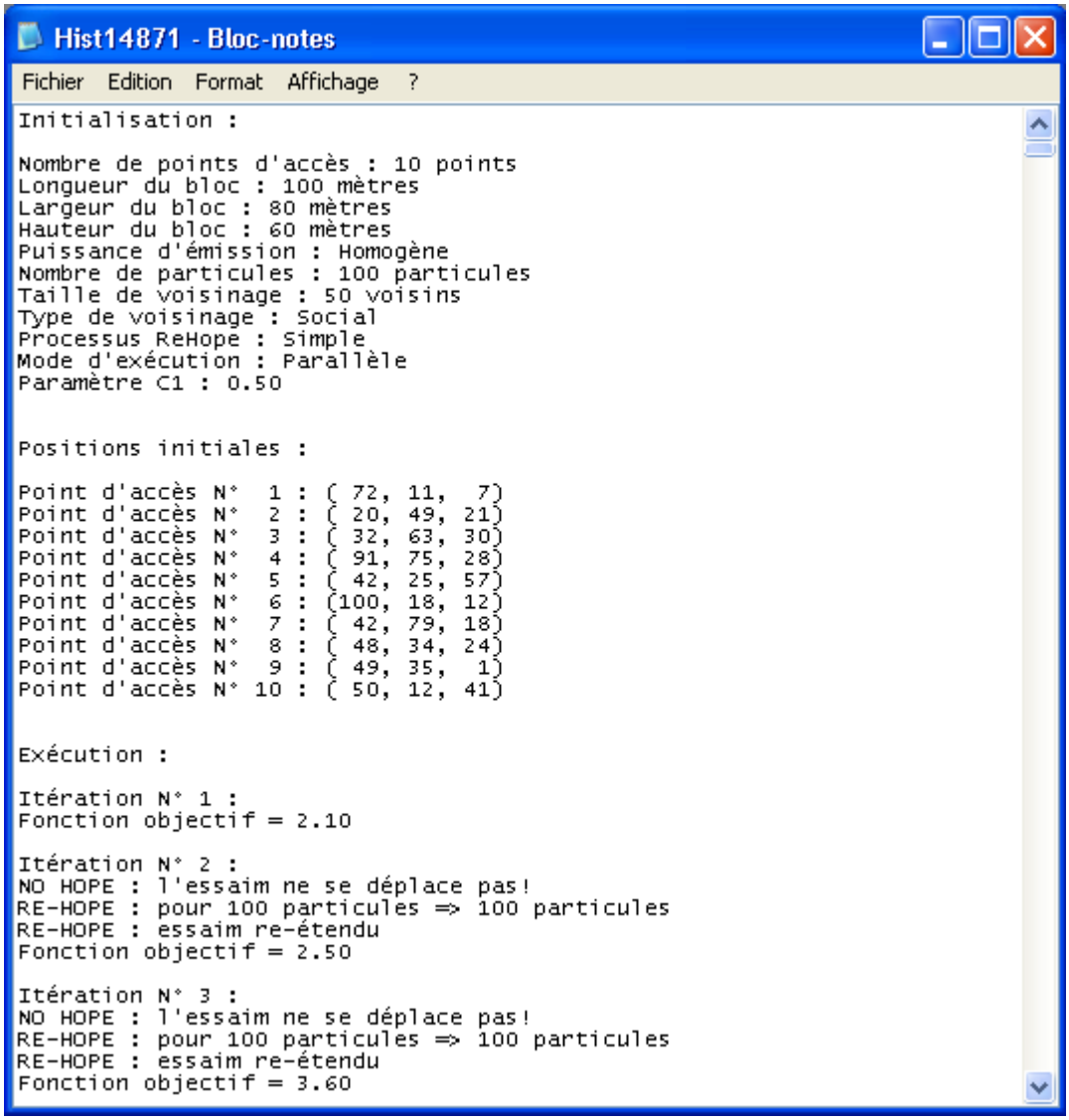


Figure 4.19 : historique du système après l'exécution.

IX. Étude comparative

Dans cette partie, nous voulons découvrir l'influence de chacun des paramètres (le nombre de particules ou bien la taille d'essaim, la taille de voisinage, le type de voisinage et les valeurs de paramètre c_1) sur les résultats de notre logiciel qui sont représentés surtout par la valeur de la fonction objectif et le temps de calcul.

Toutes nos expérimentations (**Annexe**) sont faites avec 10 points d'accès, un nombre d'itérations limité à 1000 et à chaque itération, les valeurs des paramètres c_2, c_3 sont prises aléatoirement dans l'intervalle]0,2].

- **Nombre de particules :**

Nombre de particules	Puissance homogène		Puissance non homogène	
	Valeur moyenne	Meilleure itération	Valeur moyenne	Meilleure itération
10	2,00	20	0,96	42
25	2,01	18	1,61	116
50	2,01	25	2,42	81
100	2,02	22	2,93	316
150	2,06	18	2,64	272

Tableau 4.1 : fonction objectif par rapport au nombre de particules.

Dans le cas d'une puissance d'émission homogène, nous remarquons que les valeurs de la fonction objectif portent une différence négligeable par rapport l'augmentation du nombre de particules car en général, la valeur de la fonction objectif est la moyenne des puissances d'émission de points d'accès comme nous trouvons que l'obtention de ces valeurs demande un nombre d'itérations minimale.

Dans le cas contraire ; c'est-à-dire d'une puissance d'émission non homogène, l'augmentation du nombre de particules augmente également la valeur de la fonction objectif :

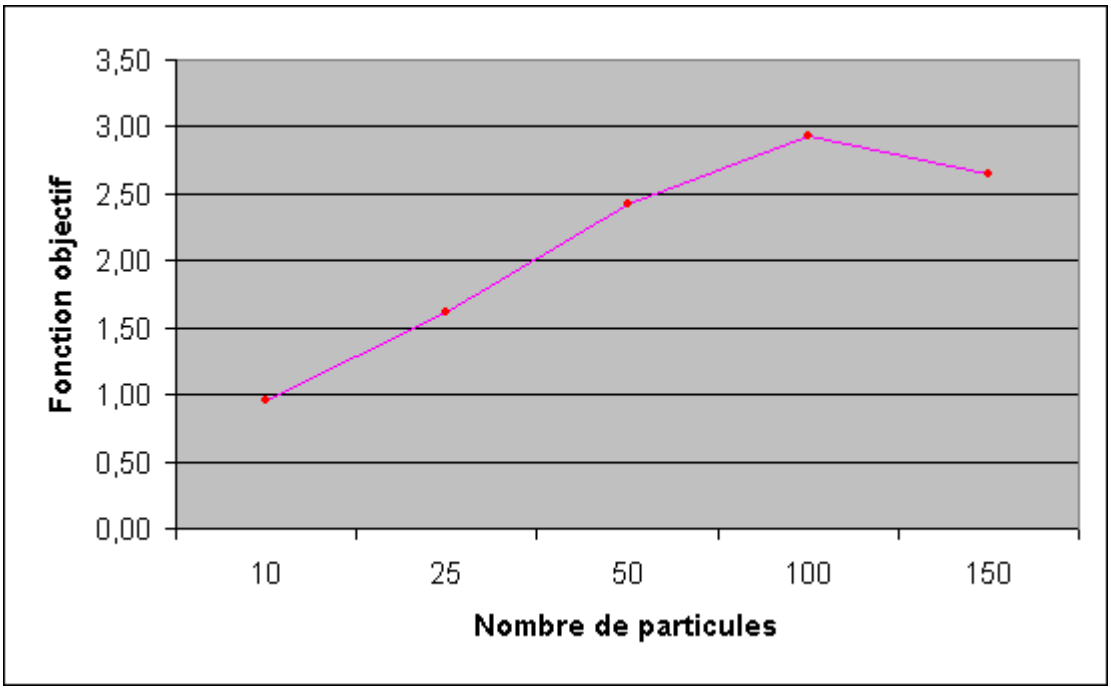


Figure 4.20 : fonction objectif par rapport à la taille d'essaim (puissance non homogène).

Nombre de particules	Puissance homogène	Puissance non homogène
	Temps de calcul moyen	Temps de calcul moyen
10	4,37	2,04
25	11,94	3,59
50	34,49	11,28

100	78,02	32,50
150	147,47	77,31

Tableau 4.2 : temps de calcul par rapport au nombre de particules.

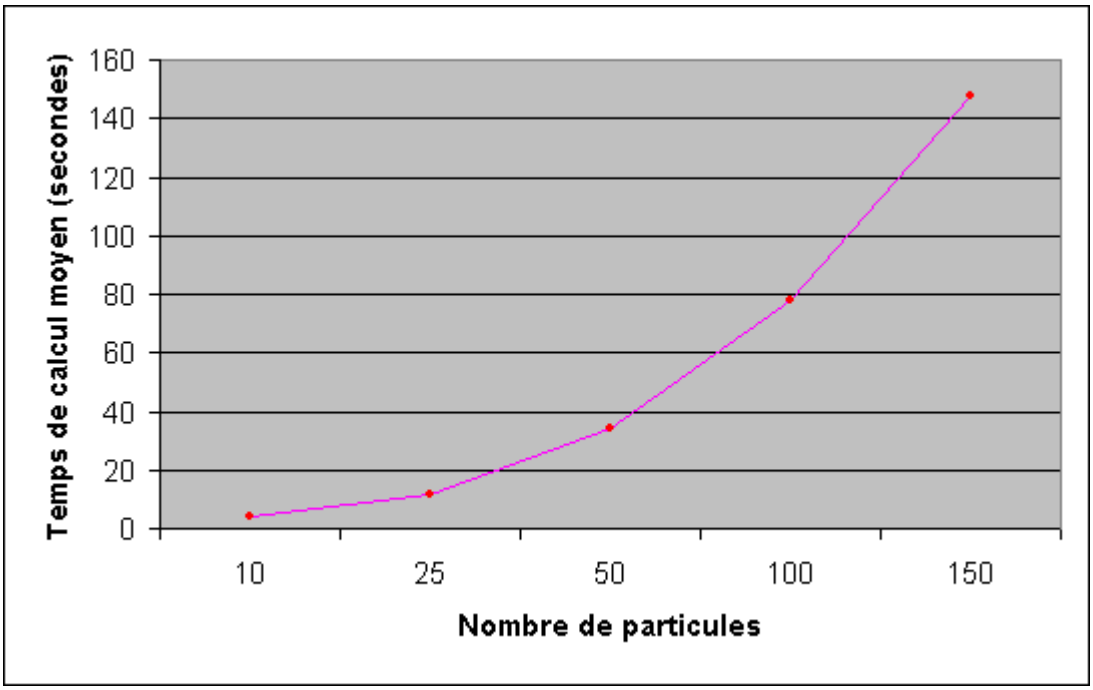


Figure 4.21 : temps de calcul par rapport à la taille d'essaim (puissance homogène).

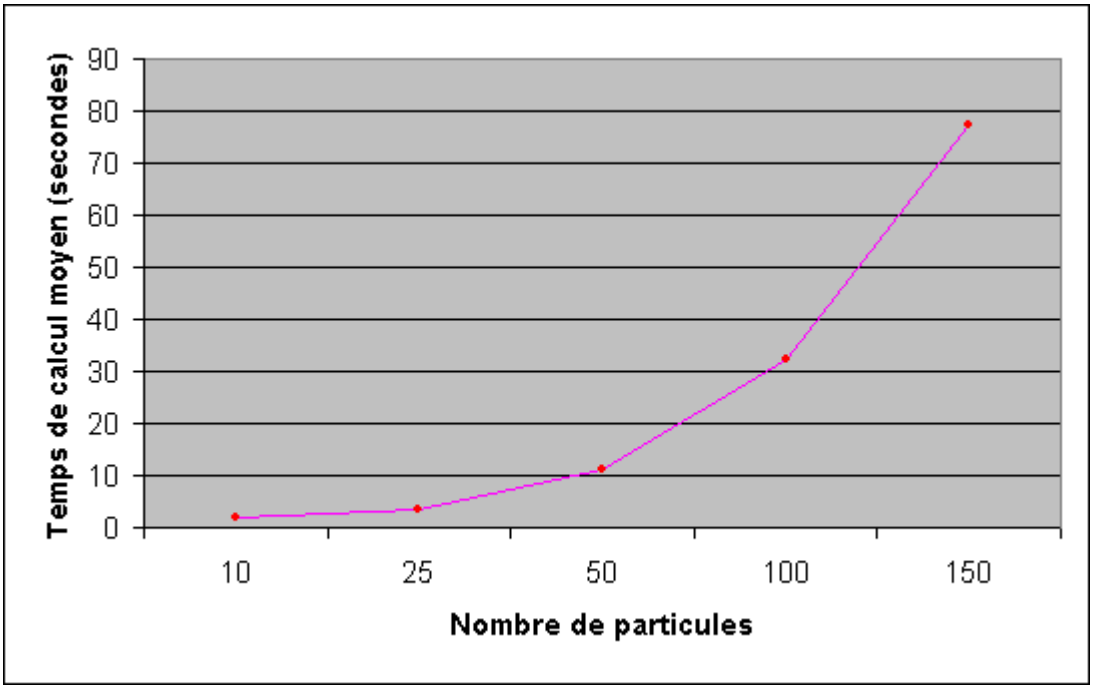


Figure 4.22 : temps de calcul par rapport à la taille d'essaim (puissance non homogène).

En terme de temps de calcul, l'augmentation du nombre de particules est traduite par une croissance dans les deux cas.

- **Taille de voisinage :**

Taille de voisinage	Puissance homogène		Puissance non homogène	
	Valeur moyenne	Meilleure itération	Valeur moyenne	Meilleure itération
10	2,02	24	2,25	157
25	2,03	23	2,32	228
50	2,03	15	2,53	194

Tableau 4.3 : fonction objectif par rapport à la taille de voisinage.

Dans la plupart des cas, la valeur de la fonction objectif est influencée par les changements de la taille de voisinage comme nous avons vu sur le **Tableau 4.3**.

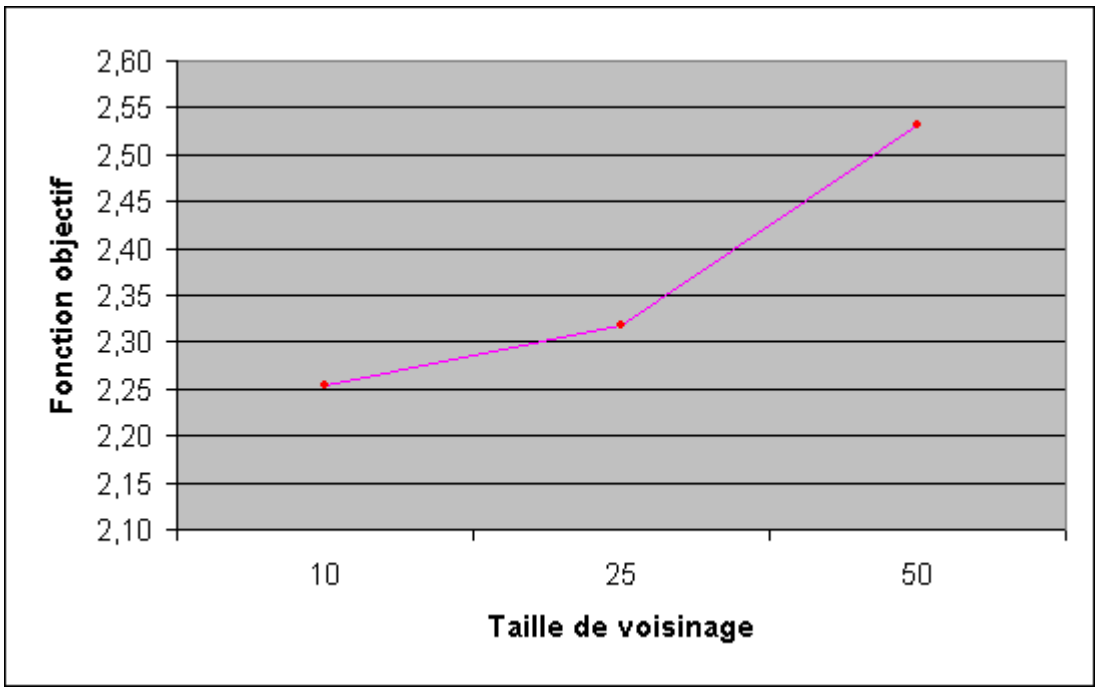


Figure 4.23 : fonction objectif par rapport à la taille de voisinage (puissance non homogène).

En ce qui concerne le temps de calcul, le fait d'augmenter la taille de voisinage augmente le temps de calcul :

Taille de voisinage	Puissance homogène	Puissance non homogène
	Temps de calcul moyen	Temps de calcul moyen
10	29,32	14,22
25	61,31	27,40
50	138,78	63,93

Tableau 4.4 : temps de calcul par rapport à la taille de voisinage.

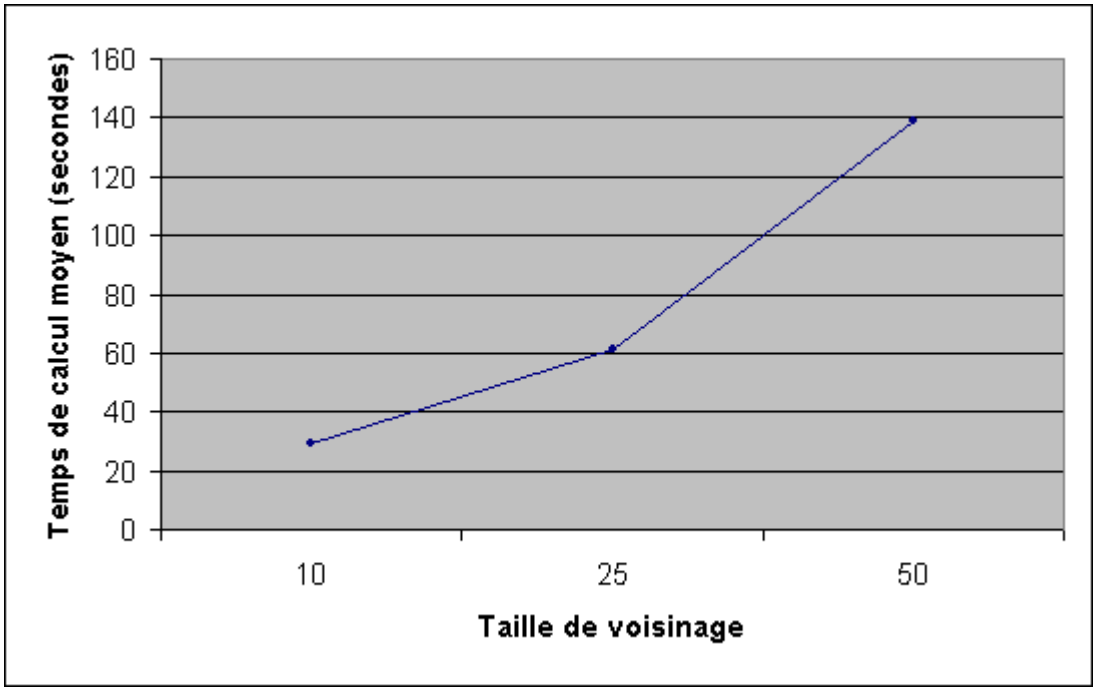


Figure 4.24 : temps de calcul par rapport à la taille de voisinage (puissance homogène).

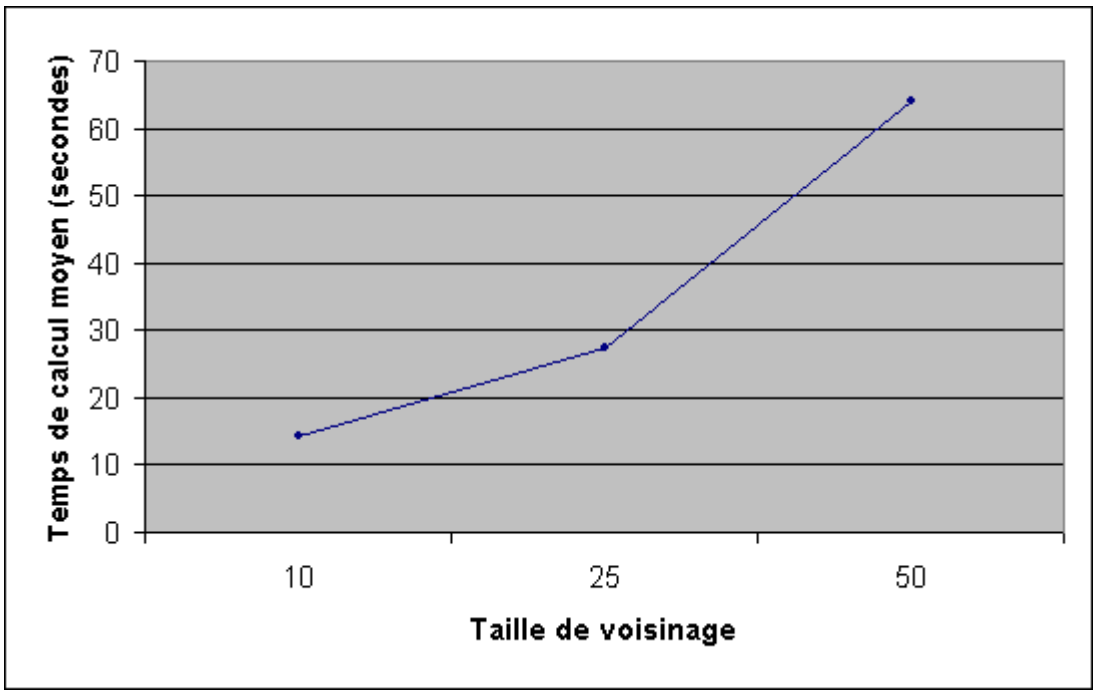


Figure 4.25 : temps de calcul par rapport à la taille de voisinage (puissance non homogène).

- **Type de voisinage :**

Sur le **Tableau 4.5**, nous avons trouvé que la valeur de la fonction objectif dans le cas d'un voisinage social est plus grande que dans le cas d'un voisinage physique malgré que le voisinage physique est plus coûteux en terme de temps de calcul que le voisinage social (**Tableau 4.6**) :

Type de voisinage	Puissance homogène		Puissance non homogène	
	Valeur moyenne	Meilleure itération	Valeur moyenne	Meilleure itération
Social	2,04	22	2,69	91
Physique	2,01	20	2,00	289

Tableau 4.5 : fonction objectif par rapport au type de voisinage.

Type de voisinage	Puissance homogène	Puissance non homogène
	Temps de calcul moyen	Temps de calcul moyen
Social	64,84	30,10
Physique	69,86	31,98

Tableau 4.6 : temps de calcul par rapport au type de voisinage.

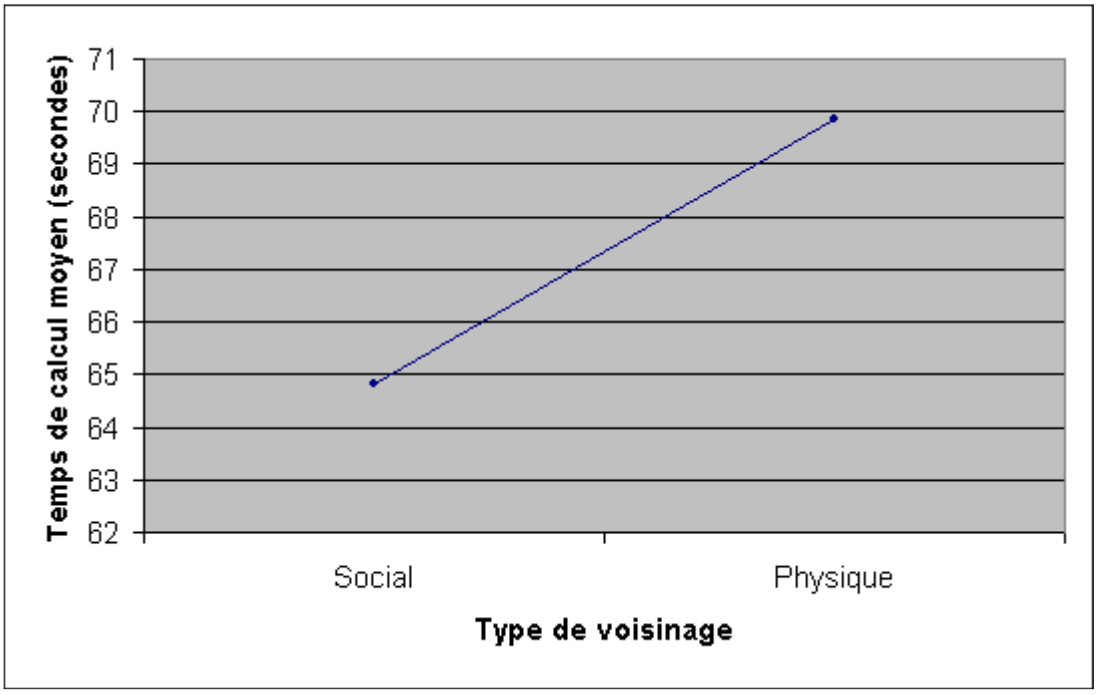


Figure 4.26 : temps de calcul par rapport au type de voisinage (puissance homogène).

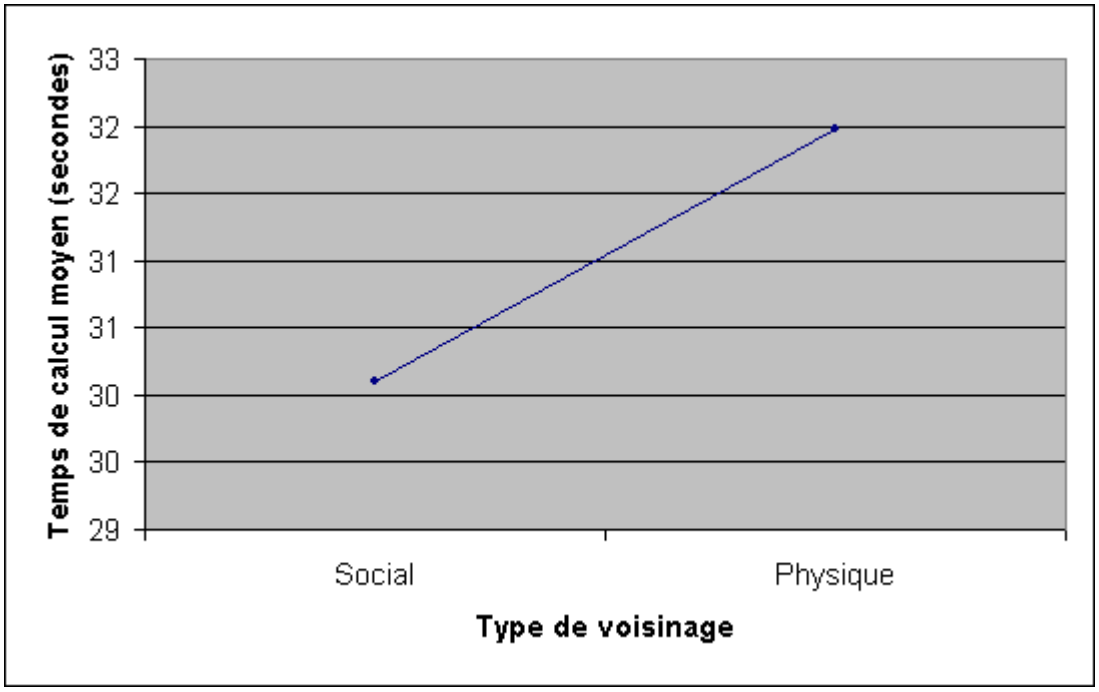


Figure 4.27 : temps de calcul par rapport au type de voisinage (puissance non homogène).

- **Paramètre c_1 :**

Paramètre c_1	Puissance homogène		Puissance non homogène	
	Valeur moyenne	Meilleure itération	Valeur moyenne	Meilleure itération
0,25	2,01	18	2,22	176
0,50	2,05	23	2,43	215
0,75	2,01	22	2,38	178

Tableau 4.7 : fonction objectif par rapport aux valeurs du paramètre c_1.

Si nous analysons le **Tableau 4.7**, nous trouvons que la minimisation de la fonction objectif a été réalisée par les deux valeurs 0,25 et 0,75 dans le cas d'une puissance d'émission homogène mais dans l'autre cas, la valeur du paramètre c_1 est 0,25.

Même pour le temps de calcul (**Tableau 4.8**), il est grand si le paramètre c_1 est égal à 0,50 soit la puissance d'émission est homogène ou non.

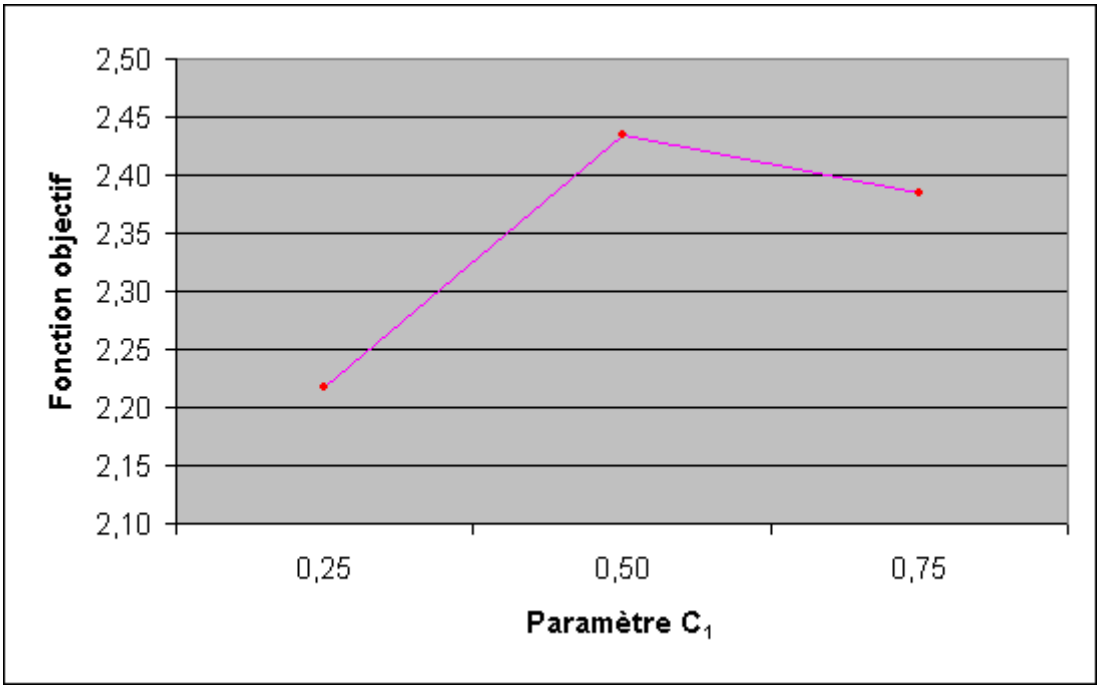


Figure 4.28 : fonction objectif par rapport aux valeurs du paramètre c_1 (puissance non homogène).

Paramètre c_1	Puissance homogène	Puissance non homogène
	Temps de calcul moyen	Temps de calcul moyen
0,25	66,79	29,68
0,50	68,66	31,97
0,75	66,60	31,47

Tableau 4.8 : temps de calcul par rapport aux valeurs du paramètre c_1.

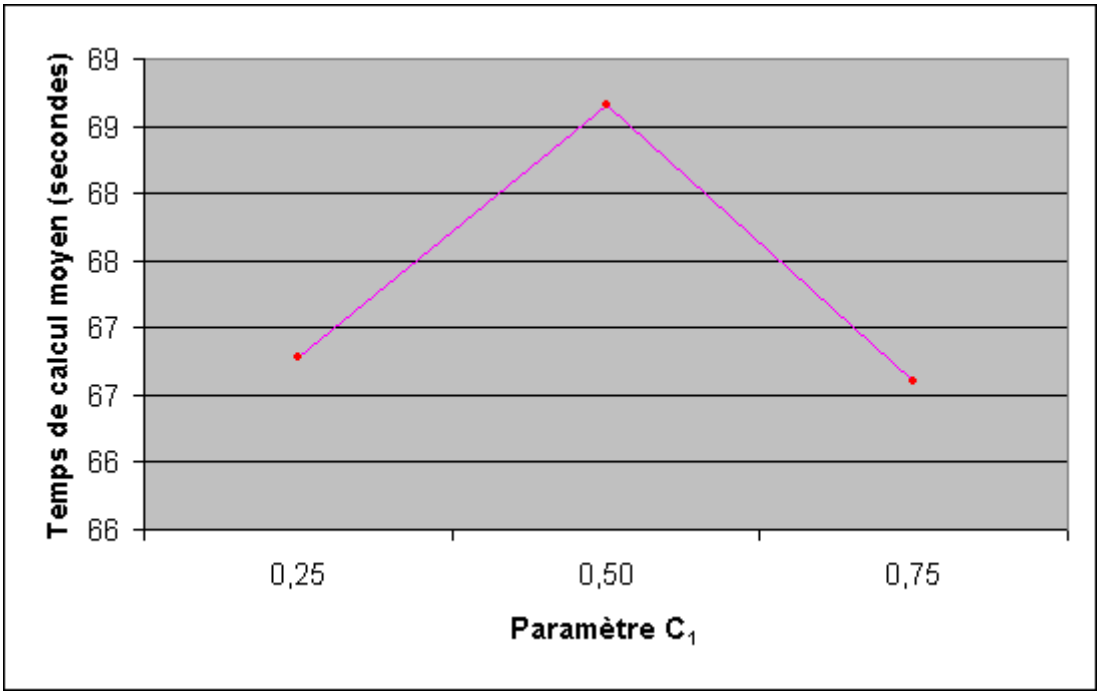


Figure 4.29 : temps de calcul par rapport aux valeurs du paramètre c_1 (puissance homogène).

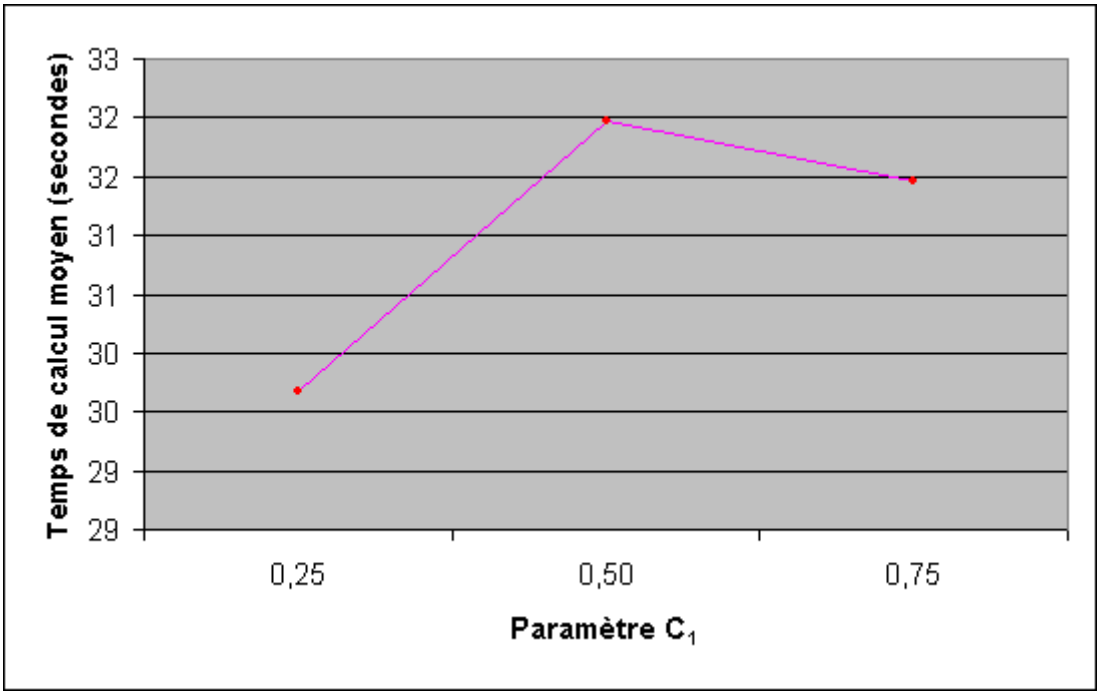


Figure 4.30 : temps de calcul par rapport aux valeurs du paramètre c_1 (puissance non homogène).

X. Conclusion

Dans ce chapitre, nous avons défini la problématique de la planification d'un réseau sans-fil et après une description de déploiement des réseaux sans-fil, nous avons dressé un état de l'art de toutes les caractéristiques du problème par l'analyse des variables et paramètres, les scénarios, les objectifs et les formulations et ensuite les métaheuristiques utilisés pour résoudre ce type de problèmes.

Dans ce chapitre, nous avons donné un aperçu général sur notre modèle d'implémentation qui se base sur le principe de la métaheuristique d'optimisation par essaim de particules.

Ensuite, nous avons présenté notre logiciel de planification de réseaux locaux sans-fil, ce dernier nous offre un outil très puissant en vue de la large gamme de configurations et de la quantité de résultats qu'on peut obtenir.

Après et afin d'examiner l'efficacité de notre application, nous avons fait des dizaines de tests avant de faire une étude comparative pour découvrir l'influence de quelques paramètres de notre approche tels que le nombre de particules, la taille de voisinage, le type de voisinage et les valeurs du paramètre c_1 sur la valeur résultante de la fonction objectif et sur le temps de calcul.

Enfin, nous sommes arrivés aux remarques suivantes :

- la valeur de la fonction objectif est minimisée si le nombre de particules est égal au nombre de points d'accès même pour la taille de voisinage ;

- le type de voisinage physique nous donne une possibilité de minimisation mieux que le voisinage social dans les deux cas de la puissance d'émission, homogène et non homogène, mais en terme de temps de calcul c'est l'inverse ;
- l'augmentation du nombre de particules et de la taille de voisinage augmente également le temps de calcul ;
- la chance de minimisation est grande lorsque la valeur du paramètre c_1 est différente de 0,50.

Conclusion générale

Ce projet avait pour objectif d'étudier l'adaptation des métaheuristiques aux problèmes d'optimisation à variables continues.

Dans la première partie, nous avons détaillé le principe de fonctionnement de principales métaheuristiques modernes telles que le recuit simulé, la recherche tabou, les algorithmes génétiques, les colonies de fourmis et l'essaim de particules.

Le recuit simulé, qui trouve ses origines dans la thermodynamique, est une métaheuristique inspirée d'un processus utilisé en métallurgie. Ce processus alterne des cycles de refroidissement lent et de réchauffage qui tendent à minimiser l'énergie du matériau. L'efficacité de cette méthode dépend du choix de principaux paramètres de contrôles qui sont la valeur initiale de la température, la fonction de décroissance de la température, le critère de changement de palier de température et les critères d'arrêt.

La recherche tabou est basée sur des idées simples, mais elle est néanmoins très efficace car la diversification et l'intensification sont des concepts complémentaires, qui enrichissent la méthode et la rendent plus robuste et plus efficace.

Les algorithmes génétiques s'inspirent du fonctionnement de l'évolution naturelle, notamment la sélection de Darwin, et la procréation selon les règles de Mendel. Ils ont relativement peu de fondements théoriques. Il n'existe aucune garantie que la méthode trouve la solution optimale.

Les algorithmes de colonie de fourmis s'inspirent du comportement des fourmis lorsque celles-ci sont à la recherche de nourriture mais les algorithmes d'essaim de particules s'inspirent à l'origine du monde du vivant. Cette dernière se base sur la collaboration des particules entre eux. Elle a d'ailleurs des similarités avec les algorithmes de colonie de fourmis, qui s'appuient eux aussi sur le concept d'auto-organisation. À cette idée, un groupe d'individus peu intelligents peut 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.

Ensuite, nous avons étudié le fonctionnement de chacune de ces métaheuristiques dans le cas d'optimisation difficile présenté par le problème le plus connu dans le monde de littérature qui est le voyageur de commerce ainsi que nous avons dressé un état de l'art de différents travaux d'adaptation de ces métaheuristiques pour des espaces en variables continues dans le deuxième et le troisième chapitre.

La description de la planification de réseaux locaux sans-fil dans le quatrième chapitre nous a permis de présenter les caractéristiques du problème

de planification wLAN par l'analyse des variables, des critères de planification, des formulations et des métaheuristiques de résolution.

Afin de valider notre travail, nous avons réalisé un logiciel de planification de réseaux locaux sans-fil dans le but d'adapter la méthode d'optimisation par essaim de particules aux problèmes d'optimisation à variables continues. Nous avons présenté l'environnement de développement choisi ainsi que nos choix techniques avant de terminer cette partie par une étude comparative des différents résultats obtenus.

D'après cette étude, nous avons constaté que :

- L'algorithme d'optimisation par essaim de particules est très simple à appliquer après avoir défini les différents éléments de base tels que l'espace de recherche, la position et la vitesse de chaque particule ;
- Il comprend plusieurs paramètres de réglage qui permettent d'agir sur le compromis exploration / exploitation ;
- L'optimisation par essaim de particules est une méthode très intéressante pour résoudre les différents problèmes d'optimisation discrète et continue ;
- Elle peut être appliquée avec succès pour résoudre des problèmes d'optimisation du monde réel, de l'ingénierie et de télécommunication.

De nombreuses perspectives peuvent être envisagées à ce travail à savoir :

- Développer la fonction objectif afin d'améliorer la qualité des résultats et de faire une étude comparative entre les deux types de puissance d'émission homogène et non homogène.
- Trouver un mécanisme pour prévenir l'algorithme de tomber dans des minimums locaux surtout dans le cas d'une puissance d'émission non homogène.
- Enrichir l'application par d'autres types de voisinage tels que la topologie en étoile, en anneau et en rayon ainsi qu'améliorer le processus *ReHope*.
- Expérimenter l'optimisation par essaim de particules sur d'autres problèmes d'optimisation.

Enfin, nous espérons que ce travail constituera une référence pour les intéressés par l'adaptation des métaheuristiques en général et par l'application de l'optimisation par essaim de particules aux problèmes à variables continues spécialement.

Annexe :

Résultats obtenus

Dans cet ensemble des tests, nous avons fixé le nombre de points d'accès à 10.

1. **Cas d'une puissance d'émission homogène :**

Nombre de particules	Taille de voisinage	Type de voisinage	Paramètre c_1	Fonction objectif	Meilleure itération	Temps de calcul
10	10	Social	0.25	2,00	48	4,56
10	10	Social	0.50	2,00	12	4,55
10	10	Social	0.75	2,00	44	4,59
10	10	Physique	0.25	2,00	02	4,22
10	10	Physique	0.50	2,00	02	4,15
10	10	Physique	0.75	2,00	11	4,15
25	10	Social	0.25	2,00	04	9,32
25	10	Social	0.50	2,00	36	9,13
25	10	Social	0.75	2,00	04	9,41
25	10	Physique	0.25	2,00	06	8,16
25	10	Physique	0.50	2,00	29	8,45
25	10	Physique	0.75	2,00	34	8,29
25	25	Social	0.25	2,00	27	15,03
25	25	Social	0.50	2,00	01	15,16
25	25	Social	0.75	2,00	09	15,12
25	25	Physique	0.25	2,00	01	14,86
25	25	Physique	0.50	2,00	01	15,18
25	25	Physique	0.75	2,10	68	15,13
50	10	Social	0.25	2,00	34	18,72
50	10	Social	0.50	2,10	31	18,20
50	10	Social	0.75	2,00	12	17,93
50	10	Physique	0.25	2,00	62	16,38
50	10	Physique	0.50	2,00	04	16,74
50	10	Physique	0.75	2,00	07	16,45
50	25	Social	0.25	2,00	30	30,33
50	25	Social	0.50	2,00	72	29,83
50	25	Social	0.75	2,00	39	30,09
50	25	Physique	0.25	2,00	02	31,16
50	25	Physique	0.50	2,00	16	31,30
50	25	Physique	0.75	2,00	15	31,50

Nombre de particules	Taille de voisinage	Type de voisinage	Paramètre c_1	Fonction objectif	Meilleure itération	Temps de calcul
50	50	Social	0.25	2,00	22	54,77
50	50	Social	0.50	2,00	26	55,48
50	50	Social	0.75	2,00	31	55,74
50	50	Physique	0.25	2,00	03	55,41
50	50	Physique	0.50	2,00	33	55,89
50	50	Physique	0.75	2,00	03	54,94
100	10	Social	0.25	2,00	14	39,15
100	10	Social	0.50	2,10	08	39,63
100	10	Social	0.75	2,00	34	38,65
100	10	Physique	0.25	2,00	23	39,60
100	10	Physique	0.50	2,10	57	41,24
100	10	Physique	0.75	2,00	10	41,24
100	25	Social	0.25	2,10	04	65,13
100	25	Social	0.50	2,10	10	70,47
100	25	Social	0.75	2,00	30	64,03
100	25	Physique	0.25	2,00	19	70,42
100	25	Physique	0.50	2,00	107	74,51
100	25	Physique	0.75	2,00	01	69,54
100	50	Social	0.25	2,00	06	120,87
100	50	Social	0.50	2,00	08	128,09
100	50	Social	0.75	2,00	35	132,80
100	50	Physique	0.25	2,00	28	121,64
100	50	Physique	0.50	2,00	01	125,12
100	50	Physique	0.75	2,00	06	122,25
150	10	Social	0.25	2,00	24	65,46
150	10	Social	0.50	2,00	43	70,66
150	10	Social	0.75	2,20	30	65,13
150	10	Physique	0.25	2,00	19	87,66
150	10	Physique	0.50	2,00	23	82,81
150	10	Physique	0.75	2,00	39	84,90
150	25	Social	0.25	2,20	27	131,38
150	25	Social	0.50	2,10	01	112,13
150	25	Social	0.75	2,00	34	114,04
150	25	Physique	0.25	2,00	11	133,09
150	25	Physique	0.50	2,00	01	150,91
150	25	Physique	0.75	2,00	19	141,13
150	50	Social	0.25	2,00	11	228,35
150	50	Social	0.50	2,50	03	233,99
150	50	Social	0.75	2,00	02	216,24
150	50	Physique	0.25	2,00	06	237,18

Nombre de particules	Taille de voisinage	Type de voisinage	Paramètre c_1	Fonction objectif	Meilleure itération	Temps de calcul
150	50	Physique	0.50	2,10	16	254,29
150	50	Physique	0.75	2,00	22	245,03

2. Cas d'une puissance d'émission non homogène :

Nombre de particules	Taille de voisinage	Type de voisinage	Paramètre c_1	Fonction objectif	Meilleure itération	Temps de calcul
10	10	Social	0.25	0,96	04	2,09
10	10	Social	0.50	1,08	10	2,04
10	10	Social	0.75	0,74	40	2,03
10	10	Physique	0.25	0,84	155	2,02
10	10	Physique	0.50	1,10	03	2,03
10	10	Physique	0.75	1,05	37	2,01
25	10	Social	0.25	1,90	01	2,10
25	10	Social	0.50	2,00	15	2,07
25	10	Social	0.75	2,72	01	2,16
25	10	Physique	0.25	1,26	86	3,42
25	10	Physique	0.50	1,40	468	2,22
25	10	Physique	0.75	1,70	75	3,35
25	25	Social	0.25	1,25	326	4,34
25	25	Social	0.50	1,36	162	4,43
25	25	Social	0.75	1,40	100	4,43
25	25	Physique	0.25	1,14	67	5,11
25	25	Physique	0.50	1,80	19	4,83
25	25	Physique	0.75	1,41	67	4,61
50	10	Social	0.25	2,99	36	4,83
50	10	Social	0.50	2,75	04	6,34
50	10	Social	0.75	3,36	20	5,20
50	10	Physique	0.25	2,75	108	5,04
50	10	Physique	0.50	2,05	70	7,20
50	10	Physique	0.75	1,95	613	5,39
50	25	Social	0.25	2,88	69	8,06
50	25	Social	0.50	1,67	28	8,85
50	25	Social	0.75	2,30	29	9,25
50	25	Physique	0.25	2,30	104	10,48
50	25	Physique	0.50	2,64	65	9,38
50	25	Physique	0.75	2,68	01	10,90
50	50	Social	0.25	2,65	101	18,70
50	50	Social	0.50	2,44	25	15,67
50	50	Social	0.75	2,00	68	17,95
50	50	Physique	0.25	1,75	69	18,89
50	50	Physique	0.50	2,45	07	19,63

Nombre de particules	Taille de voisinage	Type de voisinage	Paramètre c_1	Fonction objectif	Meilleure itération	Temps de calcul
50	50	Physique	0.75	1,88	39	21,34
100	10	Social	0.25	5,00	01	16,44
100	10	Social	0.50	3,75	02	15,01
100	10	Social	0.75	5,32	02	15,50
100	10	Physique	0.25	1,44	32	17,85
100	10	Physique	0.50	2,14	465	18,11
100	10	Physique	0.75	2,30	347	19,13
100	25	Social	0.25	3,20	01	32,68
100	25	Social	0.50	3,04	882	25,37
100	25	Social	0.75	3,60	08	34,09
100	25	Physique	0.25	1,70	748	28,37
100	25	Physique	0.50	2,60	954	27,32
100	25	Physique	0.75	1,82	771	31,52
100	50	Social	0.25	1,90	05	56,68
100	50	Social	0.50	4,10	11	46,25
100	50	Social	0.75	2,80	01	45,83
100	50	Physique	0.25	2,80	577	51,81
100	50	Physique	0.50	2,56	325	51,29
100	50	Physique	0.75	2,66	553	51,69
150	10	Social	0.25	3,30	238	31,85
150	10	Social	0.50	2,16	833	41,48
150	10	Social	0.75	3,00	01	43,11
150	10	Physique	0.25	1,30	689	48,29
150	10	Physique	0.50	2,72	44	48,33
150	10	Physique	0.75	2,60	324	49,86
150	25	Social	0.25	3,00	146	59,28
150	25	Social	0.50	4,06	01	75,96
150	25	Social	0.75	3,56	02	56,80
150	25	Physique	0.25	1,99	531	72,92
150	25	Physique	0.50	2,46	01	57,83
150	25	Physique	0.75	1,78	392	70,86
150	50	Social	0.25	2,62	47	101,26
150	50	Social	0.50	3,84	02	139,48
150	50	Social	0.75	2,13	47	126,01
150	50	Physique	0.25	2,30	93	109,81
150	50	Physique	0.50	2,25	768	136,16
150	50	Physique	0.75	2,45	745	122,24

Références bibliographiques

AIL03 Ait-Ahmed Madjid et Leborgne Fabien, « *Heuristique pour le problème du voyageur de commerce - Manuel utilisateur* », Décembre 2003.

BIA05 Charles-Edmond Bichot et Jean-Marc Alliot, « *Optimisation par colonies de fourmis appliquée au découpage de l'espace aérien européen en zones de qualification* », LOG (Laboratoire d'Optimisation Globale) CENA/ENAC, Toulouse, France, 2005.

BOU07 Mohamed Boudour, « *Application des métaheuristiques dans les systèmes d'énergie électrique* », Journée Nationale sur les Applications des Métaheuristiques (JNAM'07), Université des Sciences et de la Technologie Houari Boumediène, Alger, Algérie, 29 Mai 2007.

CAN99 Michel Van Caneghem, « *Le voyageur de commerce* », Octobre 1999.

CHE99 Rachid Chelouah, « *Chapitre I : Généralités sur les méthodes d'optimisation* », Université de Cergy-Pontoise, France, 15 Décembre 1999.

CLE00 Maurice Clerc, « *Discrete Particle Swarm Optimization Illustrated by the Travelling Salesman Problem* », 29 Février 2000.

CLE01 Maurice Clerc, « *Optimisation par essaim particulaire et coloriage de graphe - Étude en vue du traitement du problème de l'affectation de fréquences* », Rapport n° 3, Juin 2001.

CLE03 Maurice Clerc, « *L'optimisation par essaim particulaire* », Tutoriel pour OEP 2003 (partie I), Carré des Sciences, Paris, France, Octobre 2003.

CLE99 Maurice Clerc, « *The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization* », Congress on Evolutionary Computation, Washington D.C., IEEE, 1999.

CLS04 Maurice Clercc et Patrick Siarry, « *Une nouvelle métaheuristique pour l'optimisation difficile : la méthode des essaims particulaires* », 17 Septembre 2004.

DRE04 Johann Dréo, « *Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical* », Université de Paris 12, France, 13 Décembre 2004.

DRS03 Johann Dréo et Patrick Siarry, « *Diverses techniques d'optimisation inspirées de la théorie de l'auto-organisation dans les systèmes biologiques* », LERISS (Laboratoire d'Etude et de Recherche en Instrumentation, Signaux et Systèmes), Université de Paris XII Val-de-Marne, France, 24 Octobre 2003.

DUO00 Antoine Dutot et Damien Olivier, « *Optimisation par essaim de particules : Application au problème des n-Reines* », Laboratoire Informatique du Havre, Université du Havre, France.

FAR07 FAREH Abdelhak, « *Optimisation par essaims particulaires une métaheuristique pour l'optimisation difficile* », Journée Nationale sur les Applications des Métaheuristiques (JNAM'07), Université des Sciences et de la Technologie Houari Boumediène, Alger, Algérie, 29 Mai 2007.

LAP06 Lapetoule Kévin, « *Les algorithmes métaheuristiques* », 01 Juin 2006.

MAG03 Vincent Magnin, « *AG et voyageur de commerce (applet Java)* », *http://www.eudil.fr/~vmagnin/coursag/voyageur/voyageur.html*, 29 Janvier 2003.

MAG06 Vincent Magnin, « *L'optimisation* », *http://magnin.plil.net/spip.php?article47*, 7 Juillet 2006.

MCL03 Maurice Clerc, « *L'optimisation par essaim particulaire* », Tutoriel pour OEP 2003 (partie III), Carré des Sciences, Paris, France, Octobre 2003.

MDC96 V. Maniezzo, M. Dorigo et A. Colorni, « A*nt system : optimization by a colony of cooperating agents* », IEEE Transactions on Systems, Man, and Cybernetics - Part B, Volume 26, Numéro 2, Pages 29-41, 1996.

RAC99 Rachid Chelouah, « *Chapitre III : Élaboration d'un algorithme génétique continu* », Université de Cergy-Pontoise, France, 15 Décembre 1999.

RCH99 Rachid Chelouah, « *Chapitre II : Élaboration d'un algorithme tabou continu* », Université de Cergy-Pontoise, France, 15 Décembre 1999.

REN03 Jean-Philippe Rennard, « *Vie artificielle* », *http://www.rennard.org/iva/agapll.html*, 17 Janvier 2003.

RUN02 Katia Runser, « *La Planification de réseaux locaux sans fils : Modélisation Matricielle et Présentation d'un Algorithme Déterministe* », Institut National des Sciences Appliquées de Lyon, France, Année 2001-2002.

RUN05 Katia Runser, « *Méthodologies pour la planification de réseaux locaux sans-fil* », Institut National des Sciences Appliquées de Lyon, France, 27 Octobre 2005.

SGE05 Sghir Mohamed et Essabeur Brahim, « *WinMarechal Un logiciel pour maréchal-ferrant (rapport de projet)* », École Nationale Supérieure d'Ingénieurs de Caen, France, Année 2004-2005.

TOL03 Tollari Sabrina, « *Le problème du voyageur de commerce* », *http://sis.univ-tln.fr/~tollari/TER/AlgoGen1/node4.html*, 23 Mai 2003.

TOS03 Tollari Sabrina, « *Application* », *http://sis.univ-tln.fr/~tollari/TE R/Alg oGen1/node6.html*, 23 Mai 2003.

TRE03 Ioan Cristian Trelea, « *L'essaim de particules vu comme un système dynamique : convergence et choix des paramètres* », Séminaire sur L'optimisation par essaim de particules (OEP), Carré des Sciences, Paris, France, 2 Octobre 2003.

TSA03 Tollari Sabrina, « *Autres méthodes de résolution* », *http://sis.univ tln.fr/~tollari/TER/AlgoGen1/node8.html*, 23 Mai 2003.