MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS – SETIF
UFAS (ALGERIE)

MEMOIRE

Présenté à la Faculté des Sciences de l'Ingénieur
Département de d'informatique
Pour l'Obtention du Diplôme de

MAGISTER

ECOLE DOCTORALE D'INFORMATIQUE (STIC)

Option : Ingénierie des Systèmes Informatiques(ISI)

Par
Mr : BILAL BENMESSAHEL

THEME

L'optimisation par essaims particulaires : application à la fragmentation verticale

Soutenu le : 16/09/2009 devant un Jury composé de :

Moussaoui Abdelouahab	M.C à l'université de Sétif	Président
Touahria Mohamed	M.C à l'université de Sétif	Rapporteur
Khababa Abdallah	M.C à l'université de Sétif	Examinateur

Remerciements

Je remercie Mr Touahria Mohamed, Maître de Conférence à l'Université de Sétif qui a encadré mes travaux durant cette période de mémoire. Par ses conseils, son dévouement constant et ses nombreuses discussions, il a permis à ce travail d'être ce qu'il est aujourd'hui. Je le remercie également pour la liberté et la confiance qu'il m'a toujours accordée.

Je remercie Mr Moussaoui Abdelouahab, Maître de Conférence à l'Université de Sétif pour avoir accepter de présider le jury de ce mémoire.

Je remercie, Mr Khababa Abdallah, Maître de Conférence à l'université de Sétif pour avoir accepter de participer à ce jury et pour l'intérêt qu'il a manifesté envers ce travail.

Merci également à mes parents, pour l'encouragement constant et leur aide moral sans retenue tout au long de mes études.

Je remercie tous mes amis pour le soutien et ses encouragements pour la finalisation de ce travail.

Résumé

L'optimisation par essaim particulaire (OEP) est un nouveau paradigme d'Intelligence en Essaims. Les essaims de particules sont un outil précieux pour trouver des optima des fonctions dans un espace de recherche dans R^n, particulièrement utile lorsqu'il s'agit d'un grand nombre de dimensions et des problèmes où des informations spécifiques sur les fonctions est inexistantes. Sa convergence rapide et ses exigences de calculs très petits font un bon candidat pour résoudre les problèmes d'optimisation. Ce paradigme est inspiré par les concepts de la psychologie sociale et de la Vie Artificielle. Il permet de simuler les interactions sociales entre les individus, à savoir l'émergence de normes sociales, et la manière dont les individus à imiter les comportements des autres du même groupe qui semble plus efficace. Le but de cette simulation est de faire coopérer les particules pour trouver l'optimum global dans un espace de recherche.

Le travail présenté dans ce mémoire décrit une étude approfondie du métaheuristique d'optimisation par essaims particulaires et présente un travail d'adaptation et d'application de cette nouvelle métaheuristique pour la résolution d'un problème très connu dans le domaine des bases de données celui de la fragmentation verticale. Les résultats trouvés ici montrent que l'algorithme proposé dépasse les algorithmes génétiques en termes de rapidité et convergence.

Mots clés : Optimisation par essaims particulaires, métaheuristique, Fragmentation verticale, Bases de données,

Abstract

Particle Swarm Optimization (PSO) is a new paradigm of Swarm Intelligence. Particle swarms are a valuable tool to find optima in a fitness landscape in R^n, especially useful when dealing with a high number of dimensions and problems where problem specific information is non-existent. Its rapid convergence and small computational requirements make it a good candidate for solving optimization problems. This paradigm is inspired by concepts from Social Psychology and Artificial Life. It simulates social interactions among individuals, namely the emergence of social norms, and how individuals imitate behaviors of others of the same group that seem more successful. The goal of this simulation is to have the individuals cooperate to find the global optimum of a fitness landscape.

The work presented in this thesis describes a deep study on the particles swarms optimization and presents a work of adaptation and application of this new metaheuristics for solving a well known problem in the field of databases that is the vertical fragmentation. The results presented here demonstrate that the proposed algorithm outperform others approaches based on the genetic algorithms in terms of speed and convergence.

Keys words: Particle Swarm Optimization, metaheuristics, vertical fragmentation, databases

ملخـص

التحسين باستعمال الأسراب الجزيئية تعتبر طريقة جديدة مستوحاة من الذكاء الموجودة لدى الأسراب.تعتبر طريقة الأسراب الجزيئية أداة ثمينة لإيجاد الحلول للدوال في فضاء البحث R^n. وهي مفيدة خصوصا عندما يكون فضاء البحث ذو أبعاد كثيرة و كذلك بالنسبة للمشاكل التي لا توجد معلومات خاصة حول دوالها. تقاربها السريع و كذلك متطلباتها الحسابية القليلة تجعلها طريقة جد محببة لحل مشاكل التحسين. هذه الطريقة مستنبطة من مفاهيم علم الاجتماع و الحياة الصناعية. تسمح طريقة الأسراب الجزيئية بمحاكاة التفاعل الاجتماعي بين الأشخاص من أجل ظهور قواعد اجتماعية، و كذلك الطريقة التي يقلد بها الأشخاص بعضهم داخل نفس المجموعة الأشخاص ذو الكفاءة العالية. هدف هذه المحاكاة هي جعل الجزيئات تتعاون فيما بينها من أجل إيجاد الحل الأحسن في فضاء البحث.

العمل المقدم في هذه المذكرة يصف دراسة معمقة لطريقة التحسين باستعمال الأسراب الجزيئية و كذلك يقدم تعديل ثم تطبيق هذه الطريقة الجديدة من أجل حل لمشكلة معرفية في مجال قواعد البيانات وهي التجزئة العمودية. النتائج التي وجدت من خلال هذا البحث تبين أن الخوارزمي المقترح يفوق الطريقة التي تستعمل الخوارزميات الجينية و ذلك من ناحية السرعة و التقارب.

كلمات مفتاحية:التحسين باستعمال الأسراب الجزيئية، التجزئة العمودية، قواعد البيانات.

Table des Matières

Liste des Figures

Liste des Tableaux

Introduction

L'optimisation est omniprésente dans tous les domaines : dans l'industrie, dans l'économie, dans la recherche scientifique, dans l'aérospatiale… C'est la recherche des compromis entre un besoin et plusieurs contraintes : entre la résistance et le poids, entre les coûts, les ressources et les temps de production par exemple. Aussi elle occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation. Bien que les problèmes d'optimisation soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solutions algorithmiques efficaces valables pour toutes les données.

Étant donné l'importance de ces problèmes, de nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO) et en intelligence artificielle (IA). Ces méthodes peuvent être classées sommairement en deux grandes catégories : *les méthodes exactes* (complètes) qui garantissent la complétude de la résolution et *les méthodes approchées* (incomplètes) qui perdent la complétude pour gagner en efficacité.

Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des solutions de l'espace de recherche. Pour améliorer l'énumération des solutions, une telle méthode dispose de techniques pour détecter le plus tôt possible les échecs (calculs de bornes) et d'heuristiques spécifiques pour orienter les différents choix. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) comme les techniques de séparation et évaluation progressive (SEP) ou les algorithmes avec retour arrière. Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrès réalisés (notamment en matière de la programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés face aux applications de tailles importantes.

Les méthodes approchées constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. En effet, ces méthodes sont utilisées depuis longtemps par de nombreux praticiens. On peut citer les méthodes *gloutonnes* et *l'amélioration itérative* : Par exemple, la méthode de Lin et Kernighan

qui resta longtemps le champion des algorithmes pour le problème du voyageur de commerce [LIN et KERNIGHAN, 1973] .

Depuis une dizaine d'années, des progrès importants ont été réalisés avec l'apparition d'une nouvelle génération de méthodes approchées puissantes et générales, souvent appelées *métaheuristiques.* Une métaheuristique sont constitués d'un ensemble de concepts fondamentaux (par exemple, la liste taboue et les mécanismes d'intensification et de diversification pour le métaheuristique tabou), qui permettent d'aider à la conception de méthodes heuristiques pour un problème d'optimisation. Ainsi les métaheuristiques sont adaptables et applicables à une large classe de problèmes.

Les métaheuristiques sont représentées essentiellement par les méthodes de voisinage comme le *recuit simulé* et la *recherche tabou*, et les *algorithmes évolutifs* comme les *algorithmes génétiques* et les *stratégies d'évolution*. Grâce à ces métaheuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications qu'il était impossible de traiter auparavant. On constate, depuis ces dernières années, que l'intérêt porté aux métaheuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle.

Le paradigme d'optimisation par essaim particulaire (OEP) est un nouvel algorithme heuristique et de l'optimisation. Il provient de la psychologie sociale comme une simulation des processus socio-cognitifs pour tenter de modéliser des concepts abstraits comme l'intelligence et l'apprentissage. Fortement inspirés par le domaine de *l'intelligence en essaim* (SI) (Swarm Intelligence), Kennedy et Eberhart ont inventé le modèle en 1995 [EBERHART ET KENNEDY, 1995], comme une simulation du comportement social de l'homme construite autour de la thèse selon laquelle l'apprentissage et l'intelligence humaine proviennent des *interactions sociales*. L'algorithme est basé sur la notion que les individus raffinent leurs connaissances sur l'espace de recherche grâce à l'interaction sociale. Les normes sociales émergent parce que les individus ont tendance à *imiter* leurs meilleurs pairs. Chaque individu jongle deux concepts antagonistes : *l'individualisme* qui remonte à la meilleure solution il a trouvé à ce jour et le *conformisme*, l'imitation de la meilleure solution dans son voisinage.

L'optimisation par essaim particulaire (OEP) est une technique d'optimisation dans Le domaine de l'intelligence en essaim qui présente plusieurs avantages par rapport à d'autres techniques d'optimisation :

i. Elle est facile à comprendre ;
ii. Elle est simple à mettre en œuvre ;
iii. Elle a peu de paramètres à ajuster ;

iv. Elle utilise une population relativement petite ;
v. Elle a besoin d'un nombre relativement faible d'évaluations de la fonction objectif pour converger ;
vi. Elle est rapide.

Ces avantages ont donné à OEP une popularité croissante dans le domaine de d'optimisation numérique depuis sa création en 1995. Elle peut être appliquée, comme d'autres algorithmes dans le domaine du calcul évolutionnaire, dans le domaine de la conception des systèmes, l'optimisation multi-objectif, la classification, reconnaissance des formes, la modélisation des systèmes, l'ordonnancement, la planification, le traitement de signal, les applications robotiques, la prise de décision, datamining, la simulation et d'identification.

Notre contribution principale est le développement d'un algorithme discret d'optimisation par essaim particulaire, puis l'application de cet algorithme sur problème NP-difficile très connu dans le domaine des bases de données, celui de la *fragmentation verticale*. Au cours de notre étude on présentera une étude détaillée de la métaheuristique d'optimisation par essaim particulaire.

La fragmentation verticale (Attribute partition en anglais) est une technique importante dans la conception des bases de données volumineuses (VLDB Very Large Databases), et pour les bases de données distribuées, elle est utilisée pour améliorer les performances, c'est-à-dire minimisé le temps d'exécution des requêtes des utilisateurs de la base de données.

La fragmentation des données est un processus de conception dans laquelle une relation est partitionnée de sorte que la plupart des transactions accèdent à un sous-ensemble de fragments, et, par conséquent, la performance de la base de données est augmentée.

Dans ce mémoire on va résoudre le problème de la fragmentation verticale en utilisant une approche basée sur l'algorithme d'optimisation par essaim particulaire, puis on va essayer de comparer les résultats trouvés avec une autre approche développée récemment par Jun duo et al [Jun du et al, 2006] basée sur les algorithmes génétiques.

Ce mémoire est organisé autour de cinq chapitres principaux :

Le chapitre 1 commence avec une introduction de la théorie d'optimisation suivie par un résumé des concepts théoriques et les différents métaheuristiques les plus répondus dans le domaine d'optimisation difficile.

Le chapitre 2 présente une étude détaillée de l'intelligence en essaim et ces nouveaux apports au domaine de l'optimisation.

Le chapitre 3 donne un état d'art sur l'optimisation par essaim particulaire, il présente une description détaillée pour l'optimisation par essaim particulaire et ses évolutions. Il inclut aussi une présentation des différentes modifications publiées des extensions faites pour algorithme original d'optimisation par essaim particulaire.

Le chapitre 4 présente le problème traité dans ce mémoire, celui de la fragmentation verticale, et il donne aussi une description détaillée des techniques de fragmentation dans les bases de données.

Le chapitre 5 présente un nouvel algorithme discret à base d'optimisation par essaim particulaire, et il présente l'application de ce nouvel algorithme sur le problème de la fragmentation verticale dans les bases de données, et conclu par une comparaison des résultats obtenus avec un autre algorithme développé récemment à base dès les'algorithmiques génétiques.

Chapitre 1

Métaheuristiques et Optimisation

Table des matières

1.1 INTRODUCTION

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 recherche opérationnelle, la conception de systèmes mécaniques, le traitement des images, et tout particulièrement en électronique (C.A.O. de circuits électroniques, placement et routage de composants, amélioration des performances ou du rendement de fabrication de circuits, apprentissage de bases de réseaux neurones…). Le problème à résoudre peut souvent s'exprimer comme un *problème d'optimisation :* on définit une fonction objectif, ou fonction de coût (voir 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* : tous les paramètres des solutions retenues doivent respecter ces contraintes, faute de quoi ces solutions ne sont pas réalisables. Nous présentons dans ce chapitre un groupe de méthodes, dénommées *métaheuristiques* ou *métaheuristiques*, comprenant notamment la méthode du recuit simulé, les algorithmes évolutionnaires, la méthode de recherche avec tabous, les algorithmes de colonies de fourmis, l'optimisation par essaims particulaires. Crées avec une ambition commune : résoudre au mieux les problèmes dits d'*optimisation difficile*.

Ce chapitre décrit tout d'abord le cadre de l'optimisation difficile et des métaheuristiques dans lequel nous nous plaçons dans ce travail, puis présente une brève description sur quelques métaheuristiques utilisées pour résoudre les problèmes d'optimisation difficile.

1.2 METAHEURISTIQUES POUR L'OPTIMISATION DIFFICILE

1.2.1 Optimisation difficile

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 x^* possédant la meilleure qualité $f(x^*)$ (par la suite, on peut chercher à minimiser ou à maximiser $f(x)$).

La minimisation de problème peut être définie comme suit :

Étant donné $f: S \to R$ où $S \subseteq \mathrm{R}^N$ et N est la dimension de l'espace de recherche S

Trouver x^* tels que $f(x^*) \leq f(x), \forall\, x \in S$ (1.1)

La variable x^* est appelée le minimiseur global (ou tout simplement le minimiseur) et la valeur de $f(x^*)$ est appelée le minimum global (ou tout simplement le minimum) de *f*.

- x^* n'est pas obligatoirement unique ;
- f est la fonction objectif est également appelée fonction de coût ;
- S est également appelé espace de recherche.

Cela peut être illustré comme dans la figure 1.1, où x^* est un Minimiseur global de f. Le processus pour trouver la solution optimale est connu sous le nom *optimisation globale* [Gray et al. 1997].

Un vrai algorithme optimisation globale trouvera x^* indépendamment de la sélection de point de départ $x_0 \in S$ [Van den Bergh, 2002]

Dans la figure 1.1, x_B^* est appelé le Minimiseur local de la région ***B*** parce que $f(x_B^*)$ est la plus petite valeur dans la région ***B***, Mathématiquement parlant, la variable x_B^* est un Minimiseur local de la région ***B***, si.

$$f(x_B^*) \leq f(x), \forall\, x \in B \qquad (1.2)$$

Où B ⊂ S. Chaque Minimiseur global est un Minimiseur local, mais un Minimiseur local n'est pas nécessairement, un Minimiseur global.

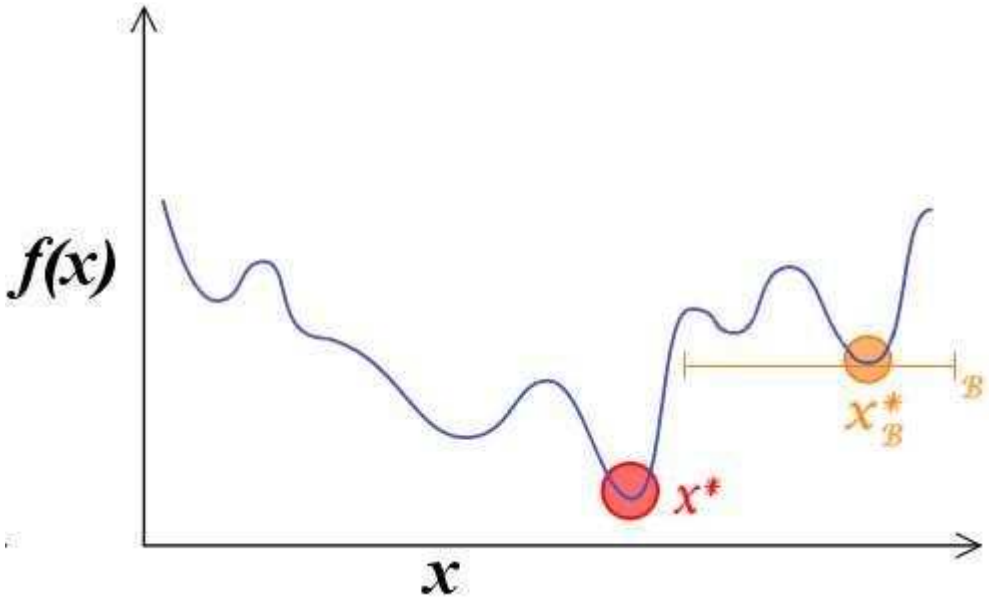


Figure 1.1 : Exemple d'un Minimiseur global x^* ainsi que d'un Minimiseur local x_B^*

Un problème d'optimisation peut présenter des contraintes d'égalité (ou d'inégalité) sur *S*, être dynamique si maximiser $f(x)$ 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.

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 certains 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 présentent à la fois des variables discrètes et des variables continues.

1.2.2 Algorithmes d'optimisation approchée

Heuristiques

Une heuristique d'optimisation est une méthode approchée se voulant simple, rapide et adaptée à un problème donné. Sa capacité à optimiser un problème avec un minimum d'informations est contrebalancée par le fait qu'elle n'offre aucune garantie quant à l'optimalité de la meilleure solution trouvée.

Du point de vue de la recherche opérationnelle, ce défaut n'est pas toujours un problème, tout spécialement quand seule une approximation de la solution optimale est recherchée.

Métaheuristiques

Parmi les heuristiques, certaines sont adaptables à un grand nombre de problèmes différents sans changements majeurs dans l'algorithme, on parle alors de métaheuristiques.

La plupart des heuristiques et des métaheuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire. En plus de cette base stochastique, les métaheuristiques sont généralement itératives, c'est-à-dire qu'un même schéma de recherche est appliqué plusieurs fois au cours de l'optimisation, et directes, c'est-à-dire qu'elles n'utilisent pas l'information du gradient de la fonction objectif. Elles tirent en particulier leur intérêt de leur capacité à éviter les optima locaux, soit en acceptant une dégradation de la fonction objectif au cours de leur progression, soit en utilisant une population de points comme méthode de recherche (se démarquant ainsi des heuristiques de *descente locale*).

Souvent inspirées d'analogies avec la réalité (physique, biologie, éthologie, . . ., Elles sont généralement conçues au départ pour des problèmes discrets, mais peuvent faire l'objet d'adaptations pour des problèmes continus.

Les métaheuristiques, du fait de leur capacité à être utilisées sur un grand nombre de problèmes divergents, se prêtent facilement à des extensions. Pour illustrer cette caractéristique, citons notamment :

- l'optimisation *multiobjectif* (dites aussi multicritère) où il faut optimiser plusieurs objectifs contradictoires. La recherche vise alors non pas à trouver un optimum global, mais un ensemble d'optima "au sens de Pareto" formant la "surface de compromis" du problème.

- l'optimisation *multimodale*, où l'on cherche un ensemble des meilleurs optima globaux et/ou locaux.
- l'optimisation de *problèmes bruités*, où il existe une incertitude sur le calcul de la fonction objectif. Incertitude dont il faut alors tenir compte dans la recherche de l'optimum.
- l'optimisation *dynamique*, où la fonction objectif varie dans le temps. Il faut alors approcher au mieux l'optimum à chaque pas de temps.
- la *parallélisation*, où l'on cherche à accélérer la vitesse de l'optimisation en répartissant la charge de calcul sur des unités fonctionnant de concert. Le problème revient alors à adapter les métaheuristiques pour qu'elles soient distribuées.
- l'*hybridation*, qui vise à tirer parti des avantages respectifs de métaheuristiques différentes en les combinants.

Enfin, la grande vitalité de ce domaine de recherche ne doit pas faire oublier qu'un des intérêts majeurs des métaheuristiques est leur facilité d'utilisation dans des problèmes concrets. L'utilisateur est généralement demandeur de méthodes efficaces permettant d'atteindre un optimum avec une précision acceptable dans un temps raisonnable. Un des enjeux de la conception des métaheuristiques est donc de faciliter le choix d'une méthode et de simplifier son réglage pour l'adapter à un problème donné.

Du fait du foisonnement de la recherche, un grand nombre de classes de métaheuristiques existent, certaines seront présentées plus en détail dans la suite, comme le recuit simulé, les algorithmes évolutionnaires ou la recherche avec tabous.

Organisation générale

D'une manière générale, les métaheuristiques s'articulent autour de trois notions [Jin-Kao Hao et al, 1999]:

- ***Diversification /exploration*** : désigne les processus visant à récolter de l'information sur le problème optimisé.
- ***L'intensification/exploitation*** : vise à utiliser l'information déjà récoltée pour définir et parcourir les zones intéressantes de l'espace de recherche.
- ***La mémoire*** : est le support de l'apprentissage, qui permet à l'algorithme de ne tenir compte que des zones ou l'optimum global est susceptible de se trouver, évitant ainsi les optimums locaux.

Les métaheuristiques progressent de façon itérative, en alternant des phases d'intensification, de diversification et d'apprentissage. L'état de départ est souvent choisi aléatoirement, l'algorithme se déroulant ensuite jusqu'à ce qu'un critère d'arrêt soit atteint.

1.3 PRINCIPES DES METAHEURISTIQUES LES PLUS REPONDUES :

Les métasheuristiques sont souvent inspirées par des systèmes naturels, qu'ils soient pris en physique (les méthodes de voisinage comme le recuit simulé et la recherche tabou), en biologie de l'évolution (les algorithmes évolutifs comme les algorithmes génétiques et les stratégies d'évolution) ou encore en éthologie (les algorithmes de colonies de fourmis, les essaims de particules).

1.3.1 La méthode du recuit simulé (Simulated annealing) :

La méthode de recuit simulé s'inspire du processus de recuit physique [J.Dreo et al, 2003]. Ce processus utilisé en métallurgie pour améliorer la qualité d'un solide cherche un état d'énergie minimale qui correspond à une structure stable du solide. Les origines du recuit simulé remontent aux expériences réalisées par Metropolis et al dans les années 50 pour simuler l'évolution d'un tel processus de recuit physique. Metropolis et al utilisent une méthode stochastique pour générer une suite d'états successifs du système en partant d'un état initial donné. Tout nouvel état est obtenu en faisant subir un déplacement (une perturbation) aléatoire à un atome quelconque.

L'utilisation d'un tel processus du recuit simulé pour résoudre des problèmes d'optimisation combinatoire a été reportée dans le recuit simulé peut être vue comme une version étendue de la méthode de descente. Le processus du recuit simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui dégradent la fonction de coût. A chaque nouvelle itération, un voisin de la configuration courante est généré de manière aléatoire. Selon les cas, ce voisin sera soit retenu pour remplacer celle-ci, soit rejeté. Si ce voisin est de performance supérieure ou égale à celle de la configuration courante, il est systématiquement retenu.

Dans le cas contraire, il est accepté avec une probabilité qui dépend de deux facteurs : d'une part l'importance de la dégradation (les dégradations plus faibles sont plus facilement acceptées), d'autre part un paramètre de contrôle, la température (une température élevée correspond à une probabilité plus grande d'accepter des dégradations).

La température est contrôlée par une fonction décroissante qui définit un schéma de refroidissement. Les deux paramètres de la méthode définissent la longueur des paliers et la fonction permettant de calculer la suite décroissante des températures. En pratique, l'algorithme s'arrête et retourne la meilleure configuration trouvée lorsque aucune configuration voisine n'a été acceptée pendant un certain nombre d'itérations à une température ou lorsque la température atteint la valeur zéro.

La performance du recuit simulé dépend largement du schéma de refroidissement utilisé. De nombreux schémas théoriques et pratiques ont été proposés. De manière générale, les schémas de refroidissement connus peuvent être classés en trois catégories :

- *réduction par paliers* : chaque température est maintenue égale pendant un certain nombre d'itérations, et décroît ainsi par paliers.
- *Réduction continue* : la température est modifiée à chaque itération.
- *réduction non-monotone*: la température décroît à chaque itération avec des augmentations occasionnelles.

Il existe des schémas qui garantissent la convergence asymptotique du recuit simulé. En pratique, on utilise des schémas relativement simples même s'ils ne garantissent pas la convergence de l'algorithme vers une solution optimale.

Le recuit simulé constitue, parmi les méthodes de voisinage, l'une des plus anciennes et des plus populaires. Il a acquis son succès essentiellement grâce à des résultats pratiques obtenus sur de nombreux problèmes NP- difficiles. La preuve de convergence a également contribué à cette popularité, bien que cette preuve n'ait pas de portée en pratique.

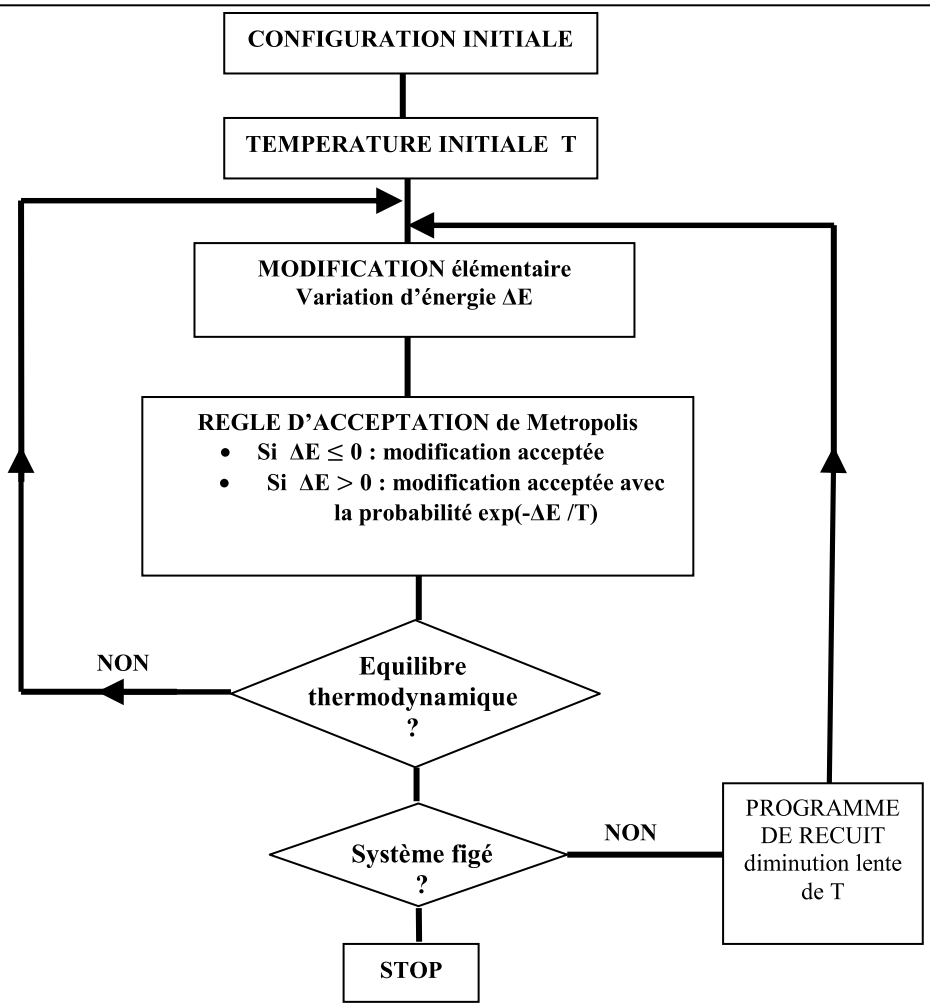


Figure. 1.2 Organigramme de l'algorithme du recuit simulé

1.3.2 La méthode de recherche avec tabous (Tabu Search)

La méthode de recherche avec tabous, ou simplement recherche tabou ou méthode tabou, a été formalisée en 1986 par F.Glover [Glover, 1986]. Sa principale particularité tient dans la mise en œuvre de mécanismes inspirés de la mémoire humaine. La méthode tabou prend, sur ce plan, le contre-pied du recuit simulé, totalement dépourvu de mémoire, et donc incapable de tire les leçons du passé. En revanche, la modélisation de la mémoire introduit de multiples degrés de liberté, qui s'opposent à toute analyse mathématique rigoureuse de la méthode tabou. Le principe de base de la méthode tabou est simple : comme le recuit simulé, la méthode tabou fonctionne avec une seule « configuration courante » à la fois (au départ, une solution quelconque), qui est actualisée au cours des « itérations » successives. À chaque itération, le mécanisme de passage d'une configuration, soit *s*, à la suivante, soit *t*, qui comporte deux étapes :

- On construit l'ensemble des voisins de *s*, c'est-à-dire l'ensemble des configurations accessibles en un seul mouvement élémentaire à partir de *s* (si cet ensemble est trop vaste, on applique une technique de réduction de sa taille : par exemple, on a recours à une liste de candidats, ou on extrait aléatoirement un sous-ensemble de voisins de taille fixée) ; soit *V(s)* l'ensemble (ou le sous-ensemble) de ces voisins ;
- On évalue la fonction objectif f du problème en chacune des configurations appartenant à *V(s)*. La configuration *t*, qui succède à s dans la suite de solutions construite par la méthode tabou, est la configuration de *V(s)* en laquelle f prend la valeur minimale. Notons que cette configuration *t* est adoptée même si elle est moins bonne que *s*, c'est-à-dire. si $f(t) > f(s)$: c'est grâce à cette particularité que la méthode tabou permet d'éviter le piégeage dans les minimums locaux de f.

Telle quelle, la procédure précédente est inopérante, car il ya un risque important de retourner à une configuration déjà retenue lors d'une itération précédente, ce qui engendre un cycle. Pour éviter ce phénomène, on tient à jour et on exploite, à chaque itération, une liste de mouvements interdits, la « liste tabou » : cette liste – qui a donné son nom à la méthode – contient m mouvement $(t \rightarrow s)$, qui sont les inverses des m derniers mouvements $(s \rightarrow t)$ effectués. L'organigramme de cet algorithme dit « tabou simple » est illustré par la figure 1.3.

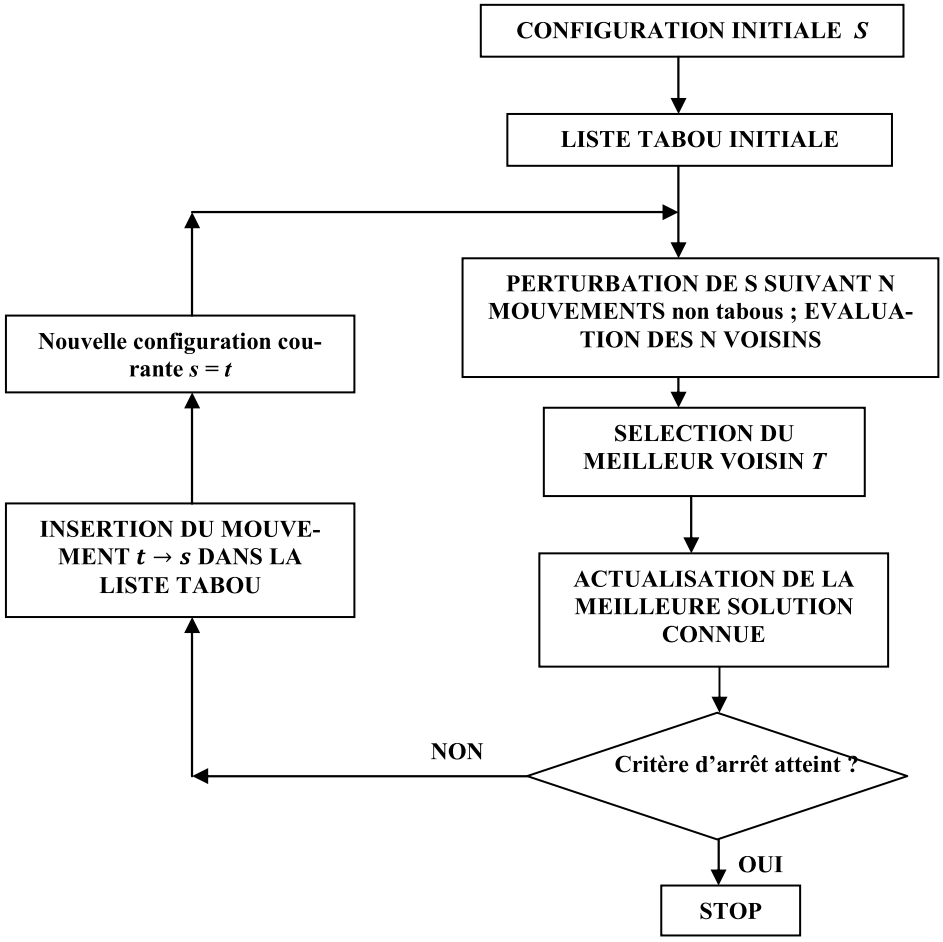


Figure. 1.3 – Organigramme de l'algorithme tabou simple.

L'algorithme modélise ainsi une forme rudimentaire de mémoire, la mémoire à court terme des solutions visitées récemment. Deux mécanismes supplémentaires, nommés intensification et diversification, sont souvent mis en œuvre pour doter aussi l'algorithme d'une mémoire à long terme. Ces processus n'exploitent plus la proximité dans le temps d'événements particuliers, mais plutôt la fréquence de leur occurrence, sur une période plus longue.

L'intensification consiste à approfondir l'exploration de certaines régions de l'espace des solutions, identifiées comme particulièrement prometteuses. La diversification est au contraire la réorientation périodique de la recherche d'un optimum vers des régions trop rarement visitées jusqu'ici.

Pour certains problèmes d'optimisation, la méthode tabou a donné d'excellents résultats ; En outre, sous sa forme de base, la méthode comprenait moins de paramètres de réglage que le recuit simulé, ce qui la rend plus simple à l'emploi. Cependant, les divers mécanismes annexes, comme l'intensification et la diversification, apportent une notable complexité.

1.3.3 Algorithmes évolutionnaires

Les algorithmes évolutionnaires (AE) sont des techniques de recherche inspirées par l'évolution biologique des espèces, apparues à la fin des années 1950 [Fraser, 1957]. Parmi plusieurs approches [Holland, 1962], [Fogel et al., 1966], [Rechenberg, 1973], les algorithmes génétiques (AG) en constituent certainement l'exemple le plus connu, à la suite de la parution en 1989 du célèbre livre de D. E. Goldberg [Goldberg, 1989] : Genetic Algorithms in Search, Optimization and Machine Learning (voir en [Goldberg, 1994] la traduction française).

Le principe d'un algorithme évolutionnaire se décrit simplement (cf. figure 1.4). Un ensemble de *N* points dans un espace de recherche, choisis a priori au hasard, constituent la population initiale ; chaque individu *x* de la population possède une certaine performance, qui mesure son degré d'adaptation à l'objectif visé : dans le cas de la minimisation d'une fonction objectif *f*, *x* est d'autant plus performant que *f(x)* est plus petit. Un AE consiste à faire évoluer progressivement, par générations successives, la composition de la population, en maintenant sa taille constante. Au cours des générations,

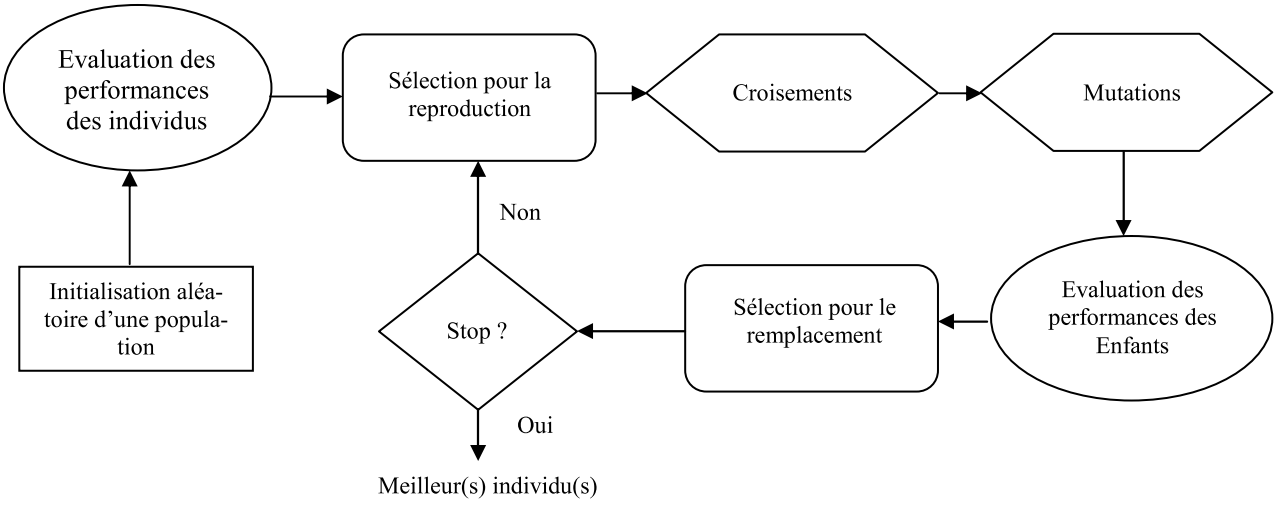


Figure. 1.4 – Principe d'un algorithme évolutionnaire standard

L'objectif est d'améliorer globalement la performance des individus ; on s'efforce d'obtenir un tel résultat en mimant les deux principaux mécanismes qui régissent l'évolution des êtres vivants, selon la théorie de C. Darwin :

— *la sélection*, qui favorise la reproduction et la survie des individus les plus performants,

— et *la reproduction*, qui permet le brassage, la recombinaison et les variations des caractères héréditaires des parents, pour former des descendants aux potentialités nouvelles.

En pratique, une représentation doit être choisie pour les individus d'une population.

Classiquement, un individu pourra être une liste d'entiers pour des problèmes combinatoires, un vecteur de nombres réels pour des problèmes numériques dans des espaces continus, une chaîne de nombres binaires pour des problèmes booléens, ou pourra même, au besoin, combiner ces représentations dans des structures complexes. Le passage d'une génération à la suivante se déroule en quatre phases : une *phase de sélection*, une *phase de reproduction* (ou de variation), une phase *d'évaluation* des performances et une *phase de remplacement*. La phase de sélection désigne les individus qui participent à la reproduction. Ils sont choisis, éventuellement à plusieurs reprises, a priori d'autant plus souvent qu'ils sont performants. Les individus sélectionnés sont ensuite disponibles pour la phase de reproduction. Celle-ci consiste à appliquer des opérateurs de variation sur des copies des individus sélectionnés, pour en engendrer de nouveaux ; les opérateurs les plus utilisés sont le croisement (ou recombinaison), qui produit un ou deux descendants à partir de deux parents, et la mutation, qui produit un nouvel individu à partir d'un seul individu.

La structure des opérateurs de variation dépend étroitement de la représentation choisie pour les individus. Les performances des nouveaux individus sont ensuite mesurées, durant la phase d'évaluation, à partir des objectifs fixés. Enfin, la phase de remplacement consiste à choisir les membres de la nouvelle génération : on peut, par exemple, remplacer les individus les moins performants de la population par les meilleurs individus produits, en nombre égal. L'algorithme est interrompu après un certain nombre de générations, selon un critère d'arrêt à préciser.

Dans cette famille de métaheuristiques, les rétroactions sont parfois difficiles à cerner, tant les variantes sont nombreuses. D'une façon générale, les rétroactions positives sont implémentées sous la forme d'opérateurs de type sélection, alors que les rétroactions négatives sont typiquement mises en place par des opérateurs de mutation. La mémoire est située au niveau local, l´évolution de chaque individu d'une itération à l'autre étant liée à l'évolution des individus voisins.

1.3.4 Métaheuristiques à colonies de fourmis (Ant Colony)

Cette métaheuristique s'inspire des comportements collectifs des fourmis dans leurs découvertes de nouvelles sources de nourriture: en effet ces insectes utilisent des phéromones afin de marquer les informations qu'ils ont recueillies sur leur environnement. On appelle cela la *stigmergie*. L'utilisation de ces phéromones leur permet (entre autres) de repérer le plus court chemin entre une source de nourriture et leur nid. Car malgré leurs capacités cognitives individuelles limitées, elles sont collectivement capables de résoudre des problèmes complexes.

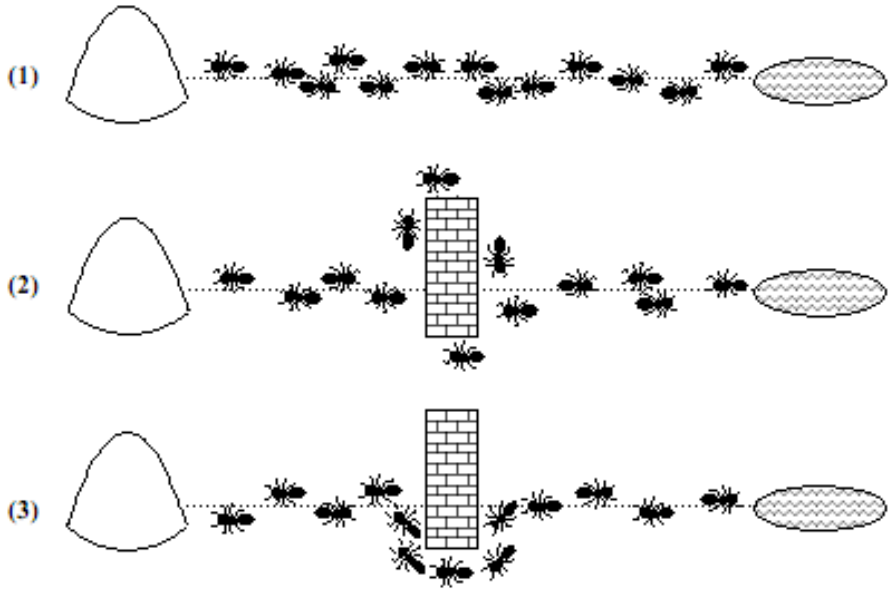


Figure. 1.5-- Faculté d'une colonie de fourmis de retrouver le plus coût chemin, observé fortuitement par un obstacle.

Initialement les fourmis connaissent une route pour aller de leur fourmilière à une source de nourriture. On y place alors un obstacle.

Dans un premier temps autant de fourmis vont choisir le chemin long et le chemin court. Noté *z* que chaque fourmi dépose la même quantité de phéromone sur son passage. Les fourmis qui prennent le chemin le plus long ne mettront plus de temps pour le parcourir que les fourmis ayant optées pour le chemin court.

Conséquences : le chemin long sera petit à petit le moins marqué pour ensuite être totalement délaissé au profit du chemin court. Finalement la quantité de phéromone sur le chemin le plus court sera tellement importante que la plupart des fourmis le choisiront.

Le fonctionnement de ce système est basé sur le principe des rétroactions positives car le dépôt de phéromones sur une piste attire d'autres fourmis qui vont à leur tour la renforcer. Négatives car l'évaporation (dû à la non-utilisation) provoque l'effet inverse.

Ce système est à la limite un système chaotique car en théorie si toutes les pistes ont la même quantité de phéromone le système n'évolue plus, il est en un point instable : Aucune piste n'est plus marquée qu'une autre et les fourmis emprunte autant l'une que l'autre. Mais les rétroactions permettent d'atteindre un point stable en amplifiant la moindre variation positive ou négative de phéromones.

Comme l'illustre le schéma ci-dessus l'intérêt principal de cette métaheuristique est sa grande adaptativité. On peut introduire un nouvel élément au problème pendant son exécution sans en bloquer la convergence.

Ci-dessous un algorithme de colonies de fourmis permettant de résoudre le problème du voyageur de commerce :

Pour t = 1,...,t{max}

Pour chaque fourmi k = 1,...,m

Choisir une ville au hasard vh

Pour chaque ville non visitée vi

Choisir une ville vj, dans la liste des villes restantes, selon

La règle aléatoire de transition

Fin Pour

Déposer une piste sur le trajet entre vh et vj

Fin Pour

Évaporer les pistes

Fin Pour

La règle aléatoire de transition est une heuristique spécialisée dont le seul et unique but et d'obtenir de meilleures diversification et intensification des trajets. Cette heuristique peut être extrêmement simple et, par exemple, prendre la ville la plus proche.

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.

1.3.5 Métaheuristiques à essaim de particules (Particle Swarm Optimiser)

Les algorithmes d'optimisation par essaim de particules (PSO) ont été introduits en 1995 par Kennedy et Eberhart [Kennedy et Eberhart, 1995] comme une alternative aux algorithmes génétiques standards. Ces algorithmes sont inspirés des essaims d'insectes(ou banc de poissons ou nuées d'oiseaux) et de leurs mouvement coordonnés. En effet, tout comme ces animaux se déplacent en groupe pour trouver de la nourriture ou éviter les prédateurs, les algorithmes à essaim de particules recherchent des solutions pour un problème d'optimisation. Les individus de l'algorithme sont appelés particules et la population est appelée essaim.

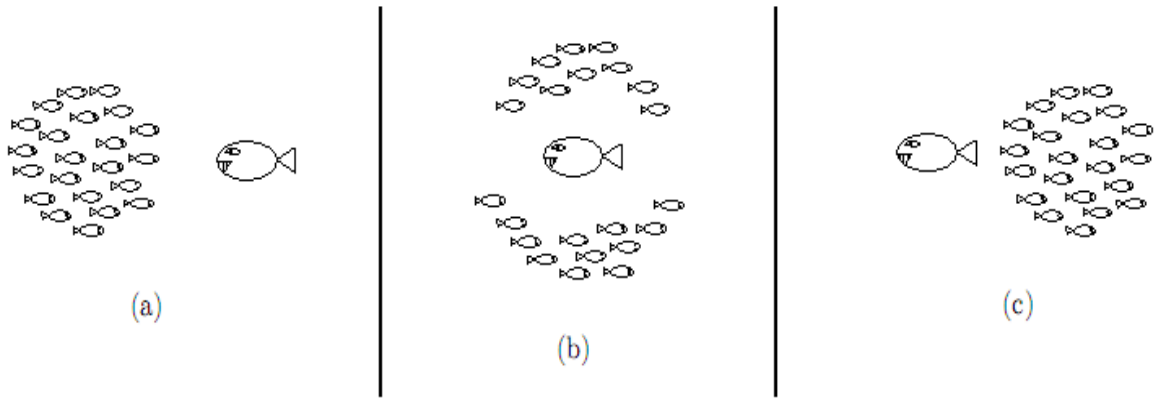


Figure. 1.6 Schéma de l'évitement d'un prédateur par un banc de poissons. (a) le banc forme un seul groupe, (b) les individus évitent le prédateur en formant une structure en "fontaine", (c) le banc se reforme.

Dans cet algorithme, une particule décide de son prochain mouvement en fonction de sa propre expérience, qui est dans ce cas la mémoire de la meilleure position qu'elle a rencontrée, et en fonction de son meilleur voisin. Ce voisinage peut être défini spatialement en prenant par exemple la distance euclidienne entre la position de deux particules ou socio métriquement (position dans l'essaim de l'individu). Les nouvelles vitesses et direction de la particule seront définies en fonction de trois tendances : la *propension* à suivre son propre chemin, *sa tendance à revenir* vers sa meilleure position atteinte et sa tendance à aller vers son meilleur voisin. Les algorithmes à essaim de particules peuvent s'appliquer aussi bien à des données discrètes qu'à des données continues. (cf. §3).

1.4 METAHEURISTIQUES ET ETHOLOGIE

Les métaheuristiques sont la plupart du temps issues de métaphores provenant de la nature, et notamment de la biologie [Dreo and Siarry, 2003a]. Les algorithmes de colonies de fourmis sont inspirés par le comportement d'insectes sociaux, mais ce ne sont pas les seuls algorithmes à être issus de l'étude du comportement animal (éthologie). En effet, l'optimisation par essaim particulaire ("Particle Swarm Optimization" [Eberhart et al., 2001],) est issue d'une analogie avec les comportements collectifs de déplacements d'animaux, tels qu'on peut les observer dans les bancs de poissons ou les vols d'oiseaux ; il existe également des algorithmes inspirés des comportements des abeilles [Choo, 2000, Panta, 2002]. On trouve encore des algorithmes s'inspirant de certains aspects du comportement des insectes sociaux, bien que ne faisant pas usage des caractéristiques classiques des algorithmes de colonies de fourmis (voir par exemple [De Wolf et al, 2002, Nouyan, 2002]).

1.5 PLACE DES METAHEURISTIQUES DANS UNE CLASSIFICATION DES METHODES D'OPTIMISATION

Pour tenter de récapituler les considérations précédentes, nous proposons (figure 1.7) une classification générale des méthodes d'optimisation mono-objectif, déjà publiée dans [Collette et al. 2002] ce graphe montre les principales distinctions opérées plus hauts :

- On différencie, en premier lieu, l'optimisation combinatoire de l'optimisation continue ;
- Pour l'optimisation combinatoire, on a recours aux méthodes approchées, lorsqu'on est confronté à un problème difficile ; dans ce cas, le choix est parfois possible entre

une heuristique « spécialisée », entièrement dédiée au problème considéré, et une métaheuristique ;

- Pour l'optimisation continue, on sépare sommairement le cas linéaire (qui relève notamment de la programmation linéaire) du cas non-linéaire, où l'on trouve le cadre de l'optimisation difficile ; dans ce cas, une solution pragmatique peut être de recourir à l'application répétée d'une méthode locale qui exploite, ou non, les gradients de la fonction objectif. Si le nombre de minimums locaux est très élevé, le recours à une méthode globale s'impose : on retrouve alors les métaheuristiques, qui offrent une alternative aux méthodes classiques d'optimisation globale, celles-ci requérant des propriétés mathématiques restrictives de la fonction objectif.
- Parmi les métaheuristiques, on peut différencier les métaheuristiques « de voisinage », qui font progresser une seule solution à la fois (recuit simulé, recherche taboue…), et les métaheuristiques « distribuées », qui manipulent en parallèle toute une population de solutions (algorithmes génétiques…) ;
- Enfin, les méthodes hybrides associent souvent une métaheuristique et une méthode locale. Cette coopération peut prendre la simple forme d'un passage de relais entre la métaheuristique et technique locale, chargée d'affirmer la solution. Mais les deux approches peuvent aussi être entremêlées de manière plus complexe.

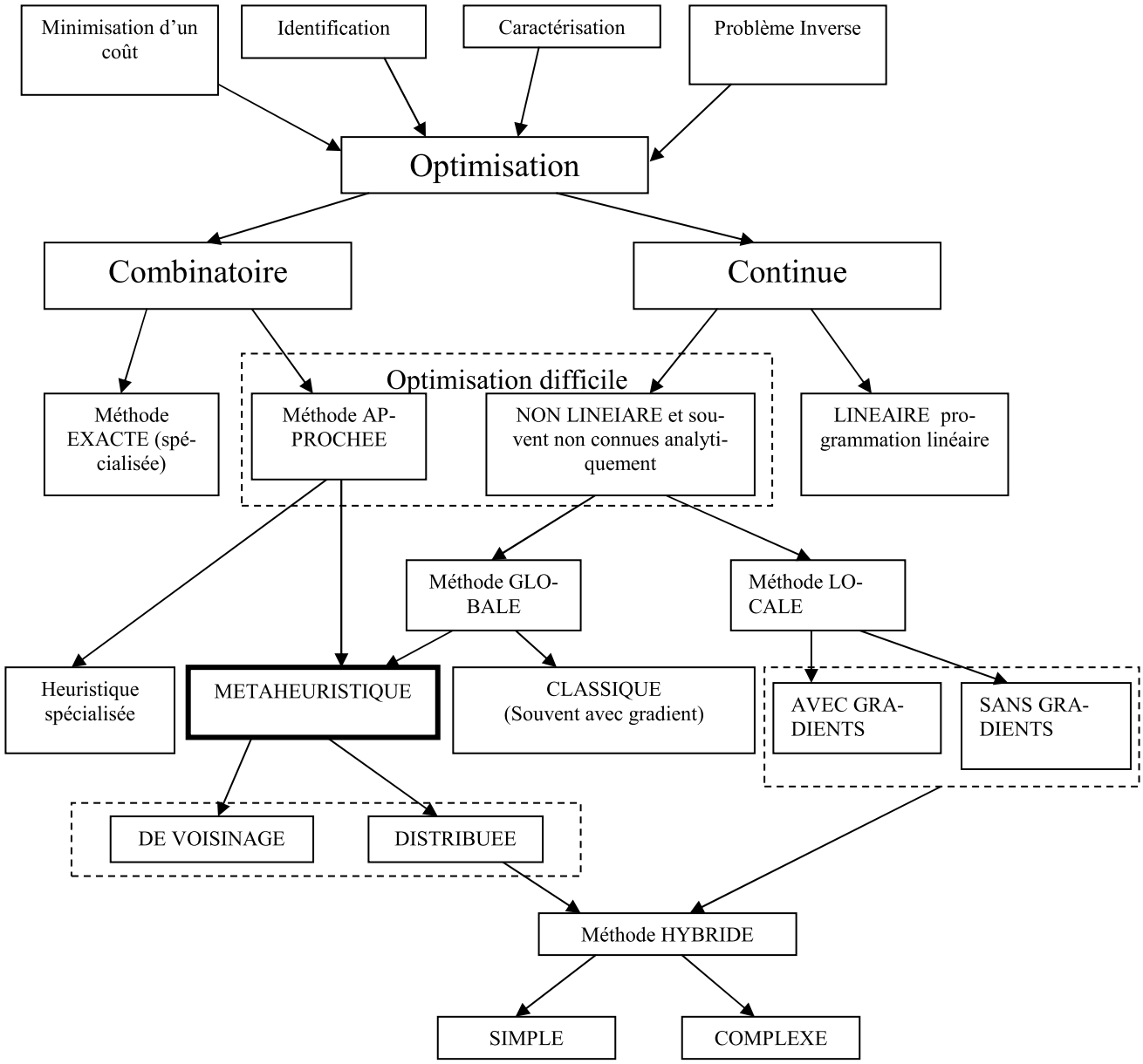


Figure. 1.7 Classification générales des méthodes d'optimisation mono-objectif.

1.6 LE CHOIX D'UNE METAHEURISTIQUE

La principale difficulté à laquelle est confronté l'ingénieur, en présence d'un problème d'optimisation concret : celui du choix d'une méthode « efficace », capable de produire une solution « optimale », ou de qualité acceptable au prix d'un temps de calcule « «raisonnable ». Face à ce souci pragmatique de l'utilisateur, la théorie n'est pas encore d'un grand secours, car les théorèmes de convergence sont souvent inexistants, ou applicables sous des hypothèses très restrictives. En outre, le réglage « optimal » des divers paramètres d'une métaheuristique, qui peut être préconisé par la théorie, est souvent inapplicable en pratique, car il induit un coût de calcul prohibitif. En conséquence, le choix d'une « bonne » méthode, et le réglage des

paramètres de celle-ci, font généralement appel au savoir-faire et à l' « expérience » de l'utilisateur, plutôt qu'à l'application fidèle de règles bien établies.

1.7 CONCLUSION

Dans ce chapitre nous avons présenté les problèmes d'optimisation difficile et une telle métaheuristique utilisée pour résoudre ces problèmes. Les méthodes de résolution sont extrêmement nombreuses, elles sont basées sur des principes totalement différents, chacune explore et exploite l'espace de recherche selon des techniques qui lui sont propres.

Comparer ces méthodes entre elles n'est pas une chose facile. Toutefois, il n'existe pas de méthodes de recherche qui soit véritablement plus performante qu'une autre sur l'ensemble des problèmes.

Pour notre étude, nous avons retenu l'optimisation par essaim particulaire parce qu'elle est récente, et elle peu appliquée dans le domaine de l'optimisation discrète, aussi parce qu'elle et extrêmement performante. C'est une méthode très efficace lorsqu'il s'agit d'un espace de recherche de grandes dimensions. D'autre part, elle n'a pas été appliquée sur le problème posé.

Chapitre 2

Intelligence en Essaim (Swarm Intelligence)

Table des matières

2.1 INTRODUCTION

À ses débuts, l'Intelligence Artificielle a puisé son inspiration dans le comportement individuel de l'être humain, en cherchant à reproduire son raisonnement. Elle s'est donc focalisée sur la manière de représenter les connaissances d'un expert et de modéliser son processus de décision, pour construire des systèmes dont le résultat pouvait être qualifié d'« intelligent ». [Chevrier, 2005]

Mais dans la nature, on observe bien des formes d'intelligence. Pourquoi ne pas prendre en compte une intelligence qui ne serait plus individuelle, mais collective ? Les chercheurs ont ainsi conçu des systèmes « multi-agents » en s'inspirant du fonctionnement d'un groupe d'experts humains.

Une autre source d'inspiration est celle des sociétés d'insectes. Ainsi est née l'« *intelligence en essaim* », qui trouve des applications dans le domaine de la simulation et au-delà, par exemple en optimisation, en robotique collective ou pour les réseaux.
Un groupe d'experts humains qui doit résoudre un problème, ou encore une société d'insectes, peuvent devenir des sources d'inspiration pour concevoir un système informatique dont l'« intelligence » provient d'un ensemble d'entités - des « agents » - en interaction.
Dans le premier cas, lorsqu'on emploie la métaphore d'un collectif d'humains, chaque entité est supposée douer d' « intelligence ». Les agents communiquent directement en s'envoyant des messages, ils possèdent des représentations du problème à résoudre et sont capables de raisonnements élaborés. La complexité des modèles utilisés autorise à faire appel à des facultés cognitives avancées : représentation explicite et raisonnement sur les autres, raisonnement sur l'avancement de la résolution, persistance d'intentions, notion d'engagement, etc.
Dans le second cas, au contraire, la métaphore est d'ordre biologique : les agents sont dotés de capacités restreintes de représentation et de raisonnement ; ils sont situés dans un environnement au travers duquel ils interagissent de manière indirecte en y déposant des marques. Alors que dans le premier cas, l'environnement est souvent absent, ici il joue un rôle primordial puisqu'il sert de support aux interactions entre les individus.

Une intelligence collective

On a longtemps cru (à tort) que les insectes sociaux étaient plus intelligents que les insectes solitaires, au vu des tâches complexes qu'ils accomplissaient. En effet, bien que ne pouvant être individuellement qualifiés d'intelligents, les membres de ces sociétés sont collectivement capables de réaliser des constructions sophistiquées, de s'adapter à des environnements changeants, de trouver le plus court chemin à une source de nourriture. Autant d'activités collectives dont la sophistication va bien au-delà des simples capacités de chacun des individus : on parle alors d'intelligence collective.

Avantage au collectif

De tels systèmes se caractérisent par leur adaptabilité et leur robustesse. En effet, du fait d'un contrôle décentralisé, chaque agent réagit en fonction de ses propres perceptions aux modifications de son contexte et il est capable de s'adapter continuellement aux variations de celui-ci. De plus, le nombre des agents, leur caractère interchangeable, l'absence d'entité centralisatrice rendent un tel système tolérant à la défaillance d'un de ses membres.

Ces deux propriétés permettent à un tel système de changer de comportement en cours de fonctionnement pour qu'il s'adapte aux évolutions de son environnement et d'avoir une dégradation progressive du fonctionnement collectif plutôt qu'un effondrement brutal.

Pour concevoir de tels systèmes, la difficulté principale qui se pose est de déterminer les comportements individuels, leur environnement et la dynamique qui vont régir le fonctionnement du système afin qu'il produise la réponse collective souhaitée.

Cependant, les caractéristiques de ces modèles font qu'il est souvent difficile voire impossible de prédire a priori le comportement collectif à partir des comportements individuels. C'est pourquoi la mise au point de ces systèmes passe généralement par une phase expérimentale où l'on évalue le comportement global en fonction de différentes valeurs des paramètres du modèle.

2.2 EXEMPLE DEMONSTRATIF

Supposons que vous et un groupe d'amis vous êtes dans une mission de recherche d'un trésor (cf. figure 2.1). Le trésor se trouve au fond de la mer. Chacun dans le groupe a un détecteur de métaux et il peut communiquer le signal et la position actuelle à *n* plus proches voisins. Chaque personne sait donc si l'un de ses voisins est plus proche au trésor que lui. Si tel est le cas, vous pouvez vous déplacer vers le voisin le plus proche au trésor. Ce faisant, vos chances sont améliorées pour trouver le trésor. En outre, le trésor peut être trouvé plus rapidement que si vous suivez seulement votre propre expérience. (Exemple tiré du livre [Andries. P Engelbrecht, 2002]

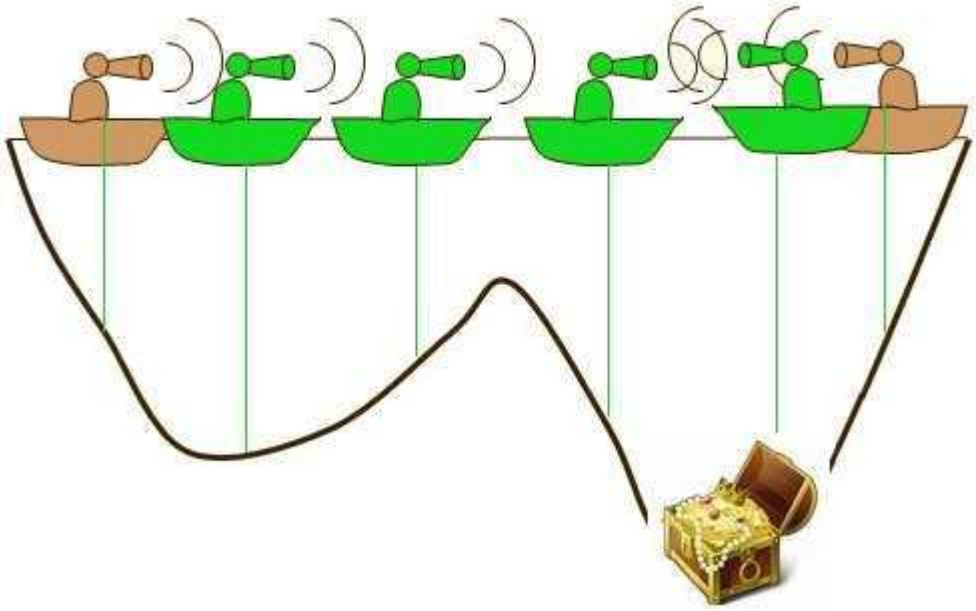


Figure 2.1 Recherche d'un trésor en utilisant l'intelligence en essaim

Il s'agit là d'une illustration extrêmement simple du comportement d'un essaim, où les individus au sein d'essaim interagissent pour résoudre un objectif global d'une manière plus efficace que celui qu'une seule personne pourrait.

Un essaim peut être défini comme une collection structurée d'organismes en interaction (ou agents). Dans les études informatiques de l'intelligence en essaim, les organismes ont inclu les fourmis, abeilles, guêpes, les termites, les poissons (dans les Bancs) et les oiseaux (dans les vols).

Dans le cadre de ces essaims, les individus sont relativement simples en structure, mais leur comportement collectif peut devenir un problème assez complexe. Par exemple, dans une colonie de fourmis, les individus se spécialisent dans l'un des ensembles des tâches simples. Collectivement, les actions et les comportements des fourmis assurent la construction des structures optimales du nid, la protection de la reine et larve, le nettoyage des nids, trouver les meilleurs sources de nourriture, d'optimiser les stratégies d'attaque, etc.

Le comportement global d'un essaim d'organismes sociaux émerge donc dans une manière non-linéaire par rapport au comportement des individus dans cet essaim : Ainsi, il existe un couplage étroit entre les comportements individuels et le comportement de l'ensemble de l'essaim. Le

comportement collectif des individus forme et dicte le comportement de l'essaim. Sur d'autre part, le comportement de l'essaim détermine les conditions dans lesquelles un individu effectue des actions. Ces actions peuvent changer l'environnement, et ainsi, les comportements de cet individu et de ses pairs peuvent également changer. Les conditions tel que déterminé par le comportement de l'essaim incluent des formes spatiales ou temporelles.

Le comportement d'un essaim n'est pas déterminé seulement par le comportement des individus, indépendamment des autres individus. Mais, l'interaction entre les individus joue un rôle vital dans la formation du comportement de l'essaim. L'interaction entre les individus aide

Dans le raffinage des connaissances des expériences sur l'environnement, et renforce lncement de l'essaim vers l'optimalité. L'interaction, ou la coopération, entre les individus est déterminée génétiquement ou par l'intermédiaire de l'interaction sociale.

Les différences anatomiques peuvent, par exemple, imposer les tâches effectuées par les individus. À titre d'exemple, dans une espèce de fourmis, les fourmis mineures nourrissent le couvain et nettoient le nid, alors que les grandes fourmis coupent les grandes proies et protègent le nid. Les fourmis mineures sont plus petites morphologiquement et différentes des grandes fourmis. L'interaction sociale peut être directe ou indirecte. Exemples d'interaction directe sont par le biais du contact visuel, audio ou chimique. L'interaction sociale indirecte se produit quand un individu change l'environnement et les autres individus répondent à ce nouvel environnement. Ce type d'interaction est appelé ***Stigmergie***.

La structure du réseau social d'un essaim forme donc une partie intégrante de l'existence de cet essaim. Elle fournit des canaux de communication par lesquels les connaissances d'expérience sont échangées entre les individus. Une conséquence étonnante des structures du réseau social d'essaims est leur capacité à s'**auto-organisé** pour former des structures optimales de nid, la répartition du travail, la collecte de nourriture, etc.

La modélisation informatique des essais a donné lieu à de nombreuses applications couronnées de succès, par exemple, *l'optimisation des fonctions*, la *recherche des itinéraires optimaux*, *l'ordonnancement*, *optimisation structurelle*, et *l'analyse de l'image et de données*. Différentes applications d'origine d'étude des différents essaims. De ce nombre, plus notable est le travail sur les colonies de fourmis et les vols d'oiseaux (bird flocks).

2.3 DEFINITION D'INTELLIGENCE EN ESSAIM

L'intelligence en essaim est la discipline qui traite les systèmes naturels et artificiels, composé de nombreux individus qui coordonnent leurs tâches en utilisant un contrôle décentralisé et l'auto-organisation. En particulier, la discipline met l'accent sur les comportements collectifs qui résultent de l'interaction des individus entre eux et avec leur environnement. Exemples de systèmes étudiés par l'intelligence en essaim les colonies de fourmis et de termites, des bancs de poissons (schools of fish), de vols d'oiseaux (flocks of birds), des troupeaux d'animaux terrestres. Certains humains artificiels tombent aussi dans le domaine de l'intelligence en essaim, en particulier certains systèmes multi-robots, et aussi certains programmes informatiques qui sont écrits pour s'attaquer à l'optimisation et les problèmes d'analyse des données.

2.4 LES PROPRIETES D'UN SYSTEME D'INTELLIGENCE EN ESSAIM

Un système typique d'intelligence en essaim a les propriétés suivantes :

- Il est composé de plusieurs individus ;
- Les individus sont relativement homogènes (c'est-à-dire, qu'ils sont tous identiques ou qu'ils appartiennent à un petit nombre de typologies) ;
- Les interactions entre les individus sont fondées sur de simples règles de comportement qui exploitent seulement l'information locale que les individus l'échangent d'une façon directe ou via l'environnement (*stigmergie*) ;
- Le comportement global du système résulte de l'interaction des individus entre eux et avec leur environnement, c'est le comportement d'*auto-organisation* du groupe.

La propriété caractéristique d'un système d'intelligence en essaim est sa capacité à agir de façon coordonnée, sans la présence d'un coordonnateur ou d'un contrôleur externe. De nombreux exemples peuvent être observés dans la nature des essaims qui exécutent certains comportements collectifs, sans un contrôle global qui contrôle le groupe. Malgré l'absence de membres en charge du groupe, l'essaim dans son ensemble peut montrer (émerger) un comportement intelligent. Ceci est le résultat de l'interaction des individus voisins en espace qui agissent sur la base de règles simples.

Généralement, le comportement de chaque individu de l'essaim est décrit en termes probabilistes : Chaque individu a un comportement stochastique qui dépend de sa perception locale du voisinage.

2.5 TAXONOMIE D'INTELLIGENCE EN ESSAIM

L'intelligence d'essaim a un caractère pluridisciplinaire marqué puisque les systèmes avec les caractéristiques mentionnées ci-dessus peuvent être observés dans une variété de domaines.

La recherche dans l'intelligence en essaim peut être classifiée selon des critères différents.

- **Naturel Vs. Artificiel** : Il est usuel de diviser la recherche en intelligence en essaim en deux secteurs selon la nature des systèmes analysés. Nous parlons donc de recherche d'intelligence en essaim *naturelle*, où des systèmes biologiques sont étudiés ; Et d'intelligence en essaim *artificielle*, où des artefacts humains sont étudiées.
- **Scientifique Vs Ingénierie** : On peut donner une alternative et une classification plus informative de recherche en intelligence en essaim basée sur les buts qui sont poursuivis : Nous pouvons identifier un courant scientifique et un courant d'ingénierie. Le but du courant scientifique est de modéliser les systèmes d'intelligence en essaim et de choisir et comprendre les mécanismes qui permettent à un système dans son ensemble de se comporter d'une façon coordonnée suite à l'interaction d'individu avec un autre individu et l'interaction entre l'individu et l'environnement. D'autre part, le but du courant d'ingénierie est d'exploiter la compréhension développée par le courant scientifique pour concevoir des systèmes qui sont capables de résoudre les problèmes de pertinence pratique.

2.6 INTELLIGENCE EN ESSAIM POUR REPRESENTER ET SIMULER DES SYSTEMES

Cette approche sert tout d'abord à représenter des comportements naturels observés, en biologie du comportement animal, en agronomie ou en sociologie.

Elle se base sur la première expérience qui a permis de mettre en évidence l'intelligence collective développée par les fourmis en quête de nourriture. C'est cette expérience, menée dans les années 1980 par Jean-Louis Deneubourg, professeur à l'Université Libre de Bruxelles, qui est à l'origine du développement du domaine de recherche de l'intelligence en essaim. [Chevrier, 2005].

Deux chemins de longueurs différentes permettent aux fourmis d'accéder de leur nid à une source de nourriture. Une fourmi seule prendra indifféremment l'un ou l'autre chemin, alors qu'une colonie de fourmis choisira de manière privilégiée le chemin le plus court.

L'explication biologique de ce phénomène est assez simple. En effet, en se déplaçant, chaque fourmi marque son chemin en déposant une certaine quantité de phéromones. Au départ, les fourmis empruntent au hasard l'un des chemins. Mais celles qui ont pris le chemin le plus court retournent plus rapidement au nid, déposant donc plus rapidement une nouvelle quantité de phéromones sur le chemin du retour. Les autres fourmis sont ensuite guidées par les traces de phéromones les plus importantes. On observe donc qu'après un temps de convergence, le plus court chemin est emprunté collectivement par une majorité d'individus.

Mais ce type d'approche ne se limite pas à la représentation de comportements naturels, l'intelligence en essaim est aussi appliquée à la gestion de trafics routiers urbains, à la simulation du mouvement de foule. Ainsi, le logiciel MASSIVE (Multiple Agent Simulation System In Virtual Environnement) utilise ces principes pour créer des scènes d'animations de foules en 3D, dans le cadre du film « Le seigneur des anneaux ». Chaque agent d'une scène est défini par ses caractéristiques et ses réactions à son environnement. Les interactions entre centaines ou milliers d'agents produisent des mouvements de foule réalistes.

2.7 INTELLIGENCE EN ESSAIM POUR RESOUDRE DES PROBLEMES

Par ailleurs, cette approche peut être utilisée pour concevoir des systèmes de résolution collective de problèmes. Il s'agit alors de définir des comportements individuels en interaction pour que collectivement se produise une réponse qui sera interprétée comme solution d'un problème. Dans la construction de tels systèmes, les collectifs naturels servent de sources d'inspiration pour concevoir un algorithme de résolution collective de problème. Parmi tous les modèles qu'offre la biologie, les sociétés d'insectes figurent en bonne place.

L'un des premiers algorithmes s'inspirant du comportement d'une colonie de fourmis est dû à l'Italien Marco Dorigo, de l'Université Libre de Bruxelles. Cet algorithme, inspiré de la recherche de nourriture par les fourmis, s'applique au fameux problème du voyageur de commerce : quel est le chemin le plus court entre n villes qui passe une seule fois par chaque ville ?

Cet algorithme a servi de base pour mettre au point diverses applications, qui concernent par exemple les réseaux de communication, comme le routage d'informations dans un réseau. Les

performances de ce type d'algorithmes sont comparables à celles d'approches plus conventionnelles.

Ces divers exemples montrent que l'approche d'intelligence en essaim est adaptée pour la modélisation de systèmes faisant intervenir un grand nombre d'entités différentes, en interaction, situées dans un environnement dynamique et fonctionnant de manière décentralisée.
C'est pourquoi elle s'applique bien à la simulation de phénomènes collectifs dont les propriétés globales ne découlent pas directement des propriétés de ses composants, en y apportant une dimension explicative.
Cependant, un grand nombre d'autres domaines de recherche actuelle sont caractérisés par les éléments précédents (distribution, absence de point de vue global, environnement dynamique) et sont donc des champs potentiels d'application de cette approche, ne serait-ce qu'en robotique collective ou dans le domaine des réseaux (où la transposition du comportement collectif des fourmis a permis l'élaboration d'un algorithme efficace de routage).
Grâce aux évolutions technologiques, les systèmes informatiques sont de plus en plus omniprésents (on les retrouve dans les objets de la vie quotidienne), de plus en plus interconnectés, et toujours en fonctionnement. Cela préfigure donc des systèmes distribués, décentralisés, plongés dans un environnement dynamique et dont le fonctionnement global sera le résultat d'interactions entre les composants : un futur champ d'application pour l'intelligence en essaim.

2.8 PHENOMENES ETUDIES DANS LE COMPORTEMENT DES ESSAIMS

2.8.1 Le phénomène d'émergence dans les essaims

Dans une approche basée sur l'intelligence en essaim, les entités formant l'essaim ont un comportement relativement simple, que l'on ne peut pas qualifier d'intelligent. Cependant ces comportements individuels peuvent faire *émerger* grâce à des interactions locales inter-individus et avec l'environnement des règles locales un comportement complexe, adaptatif sans aucun contrôle central. « Dans les sociétés d'insectes, le «projet» global n'est pas programmé explicitement chez les individus, mais émerge de l'enchaînement d'un grand nombre d'interactions élémentaires entre individus, ou entre individus et environnement. Il y a en fait intelligence collective construite à partir de nombreuses simplicités individuelles.»[Deneubourg,1991].

L'intelligence en essaim tente alors de simuler les mécanismes produisant ce type de comportements, afin de proposer de nouvelles techniques pour la résolution de problèmes.

D'une manière intuitive, la notion d'émergence peut être définie comme une propriété macroscopique d'un système qui ne peut pas être déduite à partir de son fonctionnement microscopique. On parle d'émergence quand il y a apparition de structures, et de comportements plus complexes que ceux des entités qui forment le système. Ces comportements sont non programmés explicitement et donc non prévisibles. Cette propriété initialement dans les domaines de la biologie, de la thermodynamique est reprise depuis quelques années dans le domaine de l'informatique et a été exploitée pour la conception des systèmes artificiels.

2.8.2 L'auto-organisation dans les essaims

Depuis quelques années l'utilisation du concept d'émergence pour la conception de systèmes complexes s'est très vite répondue. En effet, de tels systèmes ne permettent pas d'appliquer l'approche cartésienne classique pour laquelle la tâche globale est décomposée en sous-tâches car les étapes de résolution ne peuvent pas être programmées explicitement au départ. L'approche émergentiste se révèle donc comme un moyen de passage entre l'activité du "micro-niveau" (les interactions locales entre les composants du système et celui du "macro-niveau" (le comportement global). Tenant compte des définitions présentées dans le paragraphe précédent, il apparaît que l'auto-organisation est un élément essentiel pour l'obtention d'un phénomène émergent. Dans ce qui suit, nous allons définir le terme « auto-organisation » ainsi que son principe.

Définitions

Depuis son apparition dans les domaines de biologie, de chimie et de physique, l'utilisation du terme « auto-organisation » s'est largement répandue ces dernières années pour la conception des systèmes informatiques artificiels. Plusieurs définitions du concept d'auto-organisation existent dans la littérature. Nous pouvons en citer les suivantes :

- Définition 1 : « Un système auto-organisateur est un système qui change sa structure de base en fonction de son expérience et de son environnement. » [Ünsal ,1993].

- Définition 2 : «L'auto-organisation est un processus où l'organisation (contrainte, redondante) d'un système croît de manière spontanée, i.e. Sans que cet accroissement soit contrôlé par l'environnement ou ce qui l'entoure ou encore un système externe. » [Krippendorff, 1997].

- Définition 3 : « L'auto-organisation est définie par un ensemble de mécanismes dynamiques, permettant à des structures d'apparaître au niveau global d'un système (niveau macro) à partir des interactions de ses composants de plus bas-niveau (niveau micro). Les règles spécifiant les interactions entre les composants du système, sont exécutées sur la base d'informations purement locales (niveau micro), et ne font en aucun cas référence à la configuration globale (du niveau macro). Cette configuration globale est une propriété émergente du système plutôt qu'une propriété imposée au système, sous l'influence d'une commande extérieure » [Bonabeau, 1999; Bonabeau, 1997].

Toutes ses définitions font référence les mêmes concepts : structuration, organisation, interaction, autonomie et enfin émergence d'un comportement global à partir de plusieurs comportements locaux [Georgé, 2004]. L'auto-organisation peut être alors définie comme un moyen permettant à un système de se structurer et de se maintenir sans aucune intervention de l'extérieur. Chaque composant du système réagit aux stimulus par des règles locales simples et modifie ainsi son environnement et donc le comportement des autres composants (par exemple le dépôt de traces de phéromones chez les fourmis). De ce processus émerge une intelligence collective qui permet au système de réaliser des tâches difficiles voire complexes non explicites. On voit là que le concept d'émergence et fortement lié à celui de l'auto-organisation.

Les mécanismes de l'auto-organisation

Devant la complexité des structures obtenues par le phénomène d'auto-organisation, on se demande comment ces structures émerges-t-elles sans un plan prédéfini. En fait, leur apparition est due à de nombreuses interactions entre les éléments du système ainsi qu'avec l'environnement externe. Nous pouvons distinguer en particulier les principaux facteurs suivants :

- ***La rétroaction*** : pendant son exécution, les composants du système interagissent ensemble ou avec leur environnement. On considère une forme particulière d'interaction est la rétroaction (ou feed-back) qui est obtenue lorsque les résultats (obtenus par transformation des données d'entrée) sont retransmis au système sous la forme de nouvelles données d'entrée. Deux types de rétroaction peuvent alors être observés :
- **La rétroaction positive** : appelé aussi « auto-catalyse », c'est un mécanisme qui permet de renforcer une action impliquant une convergence très rapide. Il est observé

quand les résultats agissent de la même manière que les données d'entrée c'est-à-dire renforcent le processus de transformation. Ce phénomène est observé chez les fourmis lors de la recherche de nourriture. Le cumul de traces de phéromones déposées par chaque fourmi sur le chemin qu'elle emprunte, permet de renforcer ultérieurement le choix de ce chemin par les autres fourmis de la même colonie. [Deneubourg ,1987].

- **La rétroaction négative** : c'est un mécanisme de maintien d'équilibre qui va agir dans le sens inverse de l'amplification des fluctuations du système. Une rétroaction positive peut conduire à une stagnation prématurée et à une explosion du système alors qu'une rétroaction négative le stabilise. Ce phénomène est aussi observé dans le comportement de fourragement des fourmis avec le mécanisme d'évaporation des traces de phéromones. Les chemins les moins choisis par les fourmis seront de moins au moins imprégnés de phéromones [Deneubourg, 1987].
- **La gestion des flux** : ce sont des moyens de communication entre les composants du système et avec leur environnement. La communication peut être directe par messages ou signaux ou bien indirecte par le biais de modifications de l'environnement. Cette deuxième possibilité de communication a été appelée par le chercheur Pierre-Paul Grassé « stigmergie » [Grassé, 1959] à partir des racines stigma qui signifie piqûre, et ergon. qui veut dire travail ou œuvre.

En effet Grassé a montré vers la fin des années 1959 que chez les termites, la construction d'une battisse est guidée par la construction elle même et ne dépendait pas seulement des termites bâtisseuses : l'insecte ne dirige pas son travail mais il est guidé par lui [Grassé 1959]. Ainsi, toute nouvelle forme construite, devient un nouveau point de départ matériel pour les autres termites produisant ainsi une nouvelle forme stimulante, qui peut orienter et déclencher en retour une nouvelle activité bâtisseuse chez les autres membres de la colonie. Grâce à la stigmergie, les termites arrivent à s'auto-organiser pour réaliser un travail collectif sans aucune coordination directe.

Dans un système auto-organisé, la stigmergie utilise l'environnement comme une sorte de mémoire collective à travers lequel les composants du système interagissent. L'environnement est aussi utilisé comme support d'inscription des effets de leurs actions qui ont pour conséquence la modification de l'environnement. Chaque composant effectue une action individuelle en fonction de sa perception locale de l'environnement. Cette action va engendrer des modifications qui vont modifier cet environnement d'une manière pertinente et donc influencer son futur comportement ainsi que celui de tous les autres composants du système. L'auto-organisation ainsi obtenue sans aucune supervision des différentes actions des

composants est alors observable de l'extérieur du système sans être au préalable programmé ni explicitée.

2.8.3 Stigmergie

La stigmergie est un des concepts à la base de la création des métaheuristiques de colonies de fourmis. Elle est précisément définie comme une "forme de communication passant par le biais de modifications de l'environnement", mais on peut rencontrer le terme "interactions sociales indirectes" pour décrire le même phénomène. La spécificité de la stigmergie est que les individus échangent des informations par le biais du travail en cours, de l'état d'avancement de la tache globale à accomplir.

2.8.4 Contrôle décentralisé

Dans un système auto-organisé, il n'y a pas de prise de décision à un niveau donné, suivie d'ordres et d'actions prédéterminées. En effet, dans un système décentralisé, chaque individu dispose d'une vision locale de son environnement, et ne connaît donc pas le problème dans son ensemble. La littérature des systèmes multi-agents (voir [Ferber, 1995] pour une première approche) emploie souvent ce terme ou celui "d'intelligence artificielle distribuée" [Jennings, 1996], bien que, d'une manière générale, l'étude des systèmes multi-agents tende à utiliser des modèles de comportement plus complexes, fondés notamment sur les sciences de la cognition. Les avantages d'un contrôle décentralisé sont notamment la robustesse et la flexibilité [Bonabeau, 1999] : systèmes robustes, car capables de continuer à fonctionner en cas de panne d'une de leurs composantes ; flexibles, car efficaces sur des problèmes dynamiques.

2.9 ETUDES ET APPLICATIONS DE L'INTELLIGENCE EN ESSAIM

Cette section présente quelques exemples des études scientifiques et d'ingénierie sur l'intelligence en essaim.

A. **Comportement de groupement (Clustering) chez les fourmis :** Les fourmis construisent les cimetières par la collecte des cadavres dans un lieu unique dans le nid. Ils organisent également la disposition spatiale des larves dans Clusters (groupes) avec les jeunes, les plus petites larves dans le centre de cluster et les plus âgées, à sa périphérie. Ce comportement de groupement qui a motivé un certain nombre d'études scientifiques. Les scientifiques ont construit des modèles probabilistes simples de ces comportements et les ont testés en simulation [Bonabeau et al. 1999]. Les modèles de base

mentionnent qu'une fourmi déchargée a une probabilité de récupérer un cadavre ou d'une larve qui est inversement proportionnelle à leur densité perçus localement, tandis que la probabilité que la fourmi chargée est de supprimer le lieu du point est proportionnelle à la densité locale de même items. Ce modèle a été validé avec des données expérimentales obtenues avec des fourmis réelles. Dans la taxonomie c'est un exemple naturel et scientifique d'intelligence en essaim.

B. **Comportement des guêpes (Wasps) et de termites dans la construction de nid :** Les guêpes construisent des nids d'une structure interne très complexe qui est bien au-delà des capacités cognitives d'une guêpe. Les termites construisent des nids, dont les dimensions (ils peuvent atteindre plusieurs mètres de diamètre et de hauteur) sont énormes par rapport à un seul individu, qui peut mesurer aussi peu que quelques millimètres. Les chercheurs ont étudié les mécanismes de coordination qui permettent la construction de ces structures et ont proposé des modèles probabilistes qui exploitent la communication stigmergique pour expliquer le comportement des insectes. Certains de ces modèles ont été mis en œuvre dans les programmes d'ordinateur et utilisé pour produire des simulations des structures qui rappellent la morphologie des nids réels [Bonabeau et al. 1999]. Dans la taxonomie c'est un exemple de naturel et scientifique d'intelligence en essaim.

C. **Les vols d'oiseaux et les bancs de poissons :** Le déplacement des oiseaux et de poissons sont des exemples de comportements de groupe hautement coordonnée. Les chercheurs ont montré que ces comportements élégants d'essaim peuvent être compris comme le résultat d'un processus auto-organisés en l'absence d'un contrôleur global, et chaque individu fonde ses décisions uniquement sur la circulation des informations disponibles au niveau local : *la distance*, *la vitesse perçue*, et de *la direction de circulation des voisins*. Ces études ont inspiré un certain nombre de simulations par ordinateur (« Reynolds' Boids simulation program ») qui sont maintenant utilisés dans le secteur de l'infographie pour la reproduction réaliste de mouvement dans les films et les jeux d'ordinateur.

D. **Optimisation par colonies de fourmis :** Optimisation par colonie de fourmis est une métaheuristique utilisée pour la recherche d'une solution approchée à un problème d'optimisation, inspirée des colonies de fourmis. (cf. §1).

E. **Optimisation par essaim particulaire :** Optimisation par essaim particulaire est une technique d'optimisation à population stochastiques. Elle est inspirée par des comportements sociaux dans les vols d'oiseaux et de bancs de poissons. (cf. §3).

F. **Le comportement coopératif dans les essaims de robots :** Il existe un certain nombre de comportements d'essaim observé dans les systèmes naturels qui ont inspiré de nouvelles façons de résoudre les problèmes en utilisant des essaims de robots. C'est ce qu'on appelle la robotique en essaim. En d'autres termes, la robotique en essaim est l'application des principes de l'intelligence en essaim à la maîtrise des essaims de robots. Comme avec les systèmes d'intelligence en essaim, en général, les systèmes de la robotique en essaim peuvent avoir soit un but scientifique ou d'ingénierie. Le groupement (clustring) par un essaim de robots a été mentionné ci-dessus, comme un exemple d'artificiel / scientifique système.

2.10 CONCLUSION

Dans ce chapitre nous avons présenté l'intelligence en essaim, et le phénomène d'émergence. Intuitivement, Le phénomène d'émergence peut-être définit comme étant l'apparition d'une propriété au niveau macroscopique d'un système sans qu'elle soit préalablement programmée d'une manière explicite ni qu'elle puisse déduire à partir des propriétés des niveaux microscopiques. Un moyen de mise en œuvre de l'émergence est l'auto-organisation qui fait référence à un processus au cours duquel le système se restructure, se maintient sans nécessiter une contrainte explicite qui provient de l'extérieur du système.

Avec la complexité croissante des systèmes informatiques et le besoin d'avoir des systèmes adaptatifs et dynamiques, les méthodes classiques de résolution de problèmes sont devenues inefficaces. Ce qui a conduit les chercheurs à explorer de nouvelles voies et de nouveaux outils de développement. D'où l'utilisation de la notion d'émergence pour la conception de nouveaux types de systèmes artificiels. L'objectif est de concevoir des systèmes complexes constitués de petites entités en interaction entre elles et avec leur environnement et dont le comportement global est émergent et peut être qualifié d'intelligent.

Ces systèmes s'inspirent en large partie de l'observation des systèmes naturels et en particulier du comportement de groupes et d'animaux sociaux. Parmi les systèmes artificiels collectifs à fonctionnalité émergente, on trouve les algorithmes évolutionnaires et leurs variantes, les réseaux de neurones formels ainsi que les algorithmes inspirés des comportements collec-

tifs des animaux sociaux tels que les algorithmes d'optimisation par essaim particulaires qui simulent les comportements d'essaim observés dans les bancs de poissons ou les vols d'oiseaux et les algorithmes de fourmis artificielles qui miment les comportements des fourmis pour la résolution des problèmes complexes et à s'auto-organiser par stigmergie.

Le chapitre suivant présentera en détail les différents algorithmes de l'optimisation par essaim particulaires dans la littérature ainsi que leurs domaines d'application.

Chapitre3

Optimisation par Essaims Particulaires (OEP): état de l'art

Table des matières

3. L'OPTIMISATION PAR ESSAIMS PARTICULAIRES

3.1 ORIGINES

L'optimisation par essaims particulaires (OEP) est une méthode née en 1995 aux Etats-Unis sous le nom de Particle Swarm Optimization (PSO). Initialement, ses deux concepteurs, Russel Eberhart et James Kennedy, 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. Comme nous allons le voir, le fonctionnement de l'OEP fait qu'elle peut être rangée dans les méthodes itératives (on approche peu à peu de la solution) et stochastiques (on fait appel au hasard). Sous ce terme un peu technique, on retrouve un comportement qui est aussi vieux que la vie elle-même : améliorer sa situation en se déplaçant partiellement au hasard et partiellement selon des règles prédéfinies.

3.2 DESCRIPTION INFORMELLE

La version historique (canonique) 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 donnés 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 (cf. Figure 3.1) :

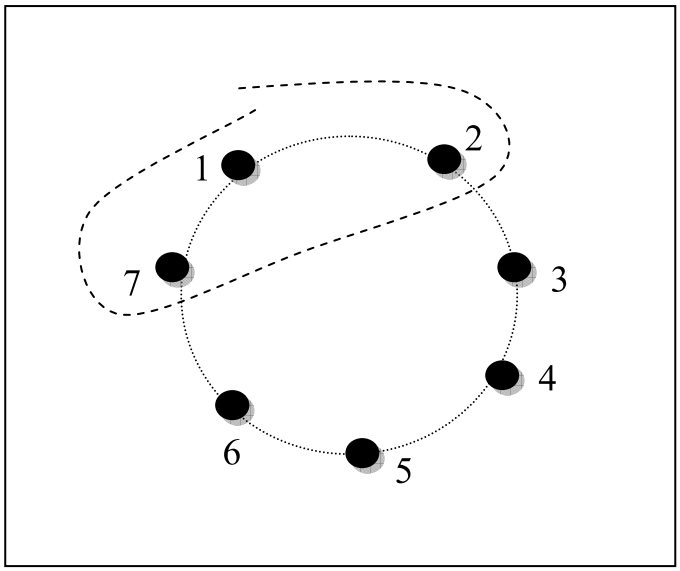


Figure. 3.1 Le cercle virtuel pour un essaim de sept particules. Le groupe d'informatrices de taille trois de la particule 1 est composé des particules 1, 2 et 7.

On suppose toutes les particules 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 détectée, la modification de la vitesse est une simple combinaison linéaire de trois tendances, à l'aide de coefficients de confiance :

- La tendance *aventureuse*, consistant à continuer selon la vitesse actuelle,
- La tendance *conservatrice*, ramenant plus ou moins vers la meilleure position déjà trouvée,
- La tendance *panurgienne*, 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. La Figure 3.2 présente un schéma de principe résumant les explications ci-dessus. Naturellement, pour pouvoir être programmé, tout ceci est formalisé dans des équations de mouvement. Un point intéressant est que, contrairement à bien d'autres heuristiques qui restent purement expérimentales, il existe une analyse mathématique précisant les conditions de convergence et le choix des paramètres.

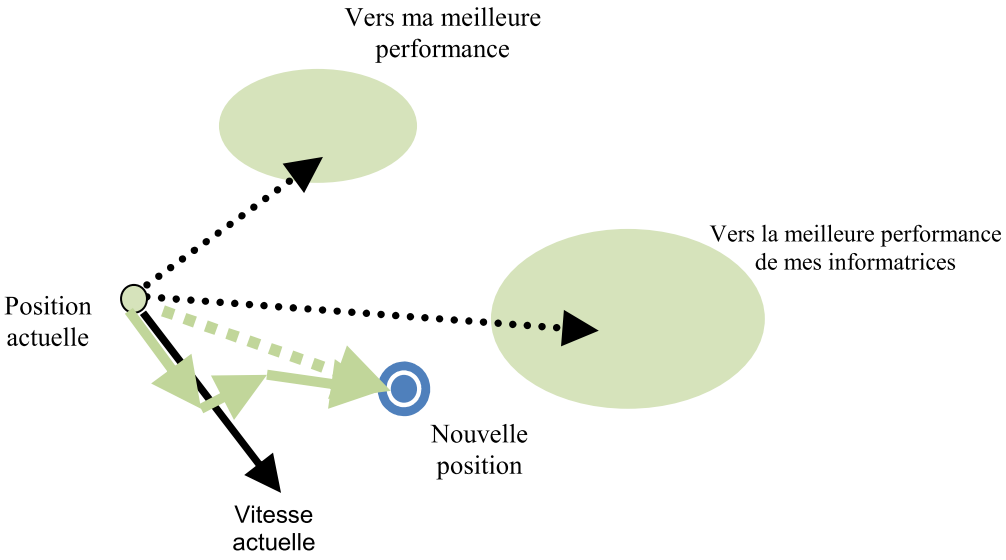


Figure 3.2. Schéma de principe du déplacement d'une particule. Pour réaliser son prochain mouvement, chaque particule combine trois tendances : suivre sa vitesse propre, revenir vers sa meilleure performance, aller vers la meilleure performance de ses informatrices.

3.3 PRINCIPALES CARACTERISTIQUES

L'optimisation par essaims particulaires présente quelques propriétés intéressantes, qui en font un bon outil pour de nombreux problèmes d'optimisation, particulièrement les problèmes fortement non linéaires, continus ou mixtes (certaines variables étant réelles et d'autres entières) :

- Il est facile à programmer, quelques lignes de code suffisent dans n'importe quel langage évolué,
- Il est robuste (de mauvais choix de paramètres dégradent les performances, mais n'empêchent pas d'obtenir une solution).

3.4 L'ALGORITHME D'OPTIMISATION PAR ESSAIMS PARTICULAIRES

Dans un système d'essaim de particules, un essaim d'individus (appelées particules) volent à travers l'espace de recherche. Chaque particule représente une solution potentielle au problème d'optimisation. La position d'une particule est influencée par la meilleure position visitée par elle-même (c'est-à-dire sa propre expérience) et la position de la meilleure particule dans son voisinage (c'est-à-dire là l'expérience des particules voisines). Lorsque le voisinage d'une particule est l'ensemble de l'essaim, la meilleure position dans le voisinage est considérée comme la meilleure particule globale, et l'algorithme est appelé un *Gbest PSO* (pour Global Best PSO). Quand seulement les voisins les plus proches sont utilisés, l'algorithme est généralement considéré comme un *Lbest PSO* (pour local Best) [Shi et Eberhart, Parameter 1998]. La performance de chaque particule est mesurée au moyen d'une fonction objectif qui varie en fonction du problème d'optimisation.

Chaque particule dans l'essaim est représentée par les caractéristiques suivantes :

x_i: La position actuelle de la particule ;

v_i: La vitesse courante de la particule ;

y_i: La meilleure position personnelle de la particule.

$\hat{y}_i$: La meilleure position dans le voisinage de la particule.

La meilleure position personnelle de la particule *i* est la meilleure position (c'est-à-dire celle qui résulte de la meilleure valeur de la fonction objectif) visité par la particule *i* jusqu'à ici. Soit f désigne la fonction objectif. Alors La meilleure position personnelle d'une particule au temps *t* est mise à jour comme suite :

$$y_i(t+1) = \begin{cases} y_i(t) & Si\ f\big(x_i(t+1)\big) \geq f(y_i(t)) \\ x_i(t+1) & Si\ f\big(x_i(t+1)\big) < f(y_i(t)) \end{cases} \qquad (3.1)$$

Pour le modèle Gbest, la meilleure particule est déterminée à partir de l'ensemble de l'essaim par la sélection de la meilleure position personnelle. Si la position de la meilleure particule globale est dénotée par le vecteur $\hat{y}$, alors

$$\hat{y}(t) \in \{y_0, y_1, \dots, y_s\} = \min\{f(y_0(t)), f(y_1(t)), \dots, f(y_s(t))\} \qquad (3.2)$$

Où *S* représente la taille de l'essaim.

L'étape de mise à jour de la vitesse est indiquée pour chaque dimension $\in 1,2, \dots, N_d$, par conséquent, v_{ij} représente l'élément *j* du vecteur de vitesse de la particule *i*. Ainsi, la vitesse de particule *i* est mise à jour selon l'équation suivante :

$$v_{i,j}(t+1) = \omega v_{i,j}(t) + c_1 r_{1,j}(t)\left(y_{i,j}(t) - x_{i,j}(t)\right) + c_2 r_{2,j}(t)\left(\hat{y}_j(t) - x_{i,j}(t)\right) \quad (3.3)$$

Où ω est l''inertie, c_1 et c_2 sont des constantes de l'accélération, et $r_{1,j}(t)$, $r_{2,j}(t) \sim U(0,1)$.

L'équation (3.3) se compose de trois éléments, à savoir

- Le terme d'inertie ω, qui a été introduit pour la première fois par Shi et Eberhart [A modified 1998]. Ce terme sert comme une mémoire des vitesses précédentes. L''inertie contrôle l'impact des vitesses précédentes : Une grande inertie favorise l'exploration, tandis que une petite inertie favorise l'exploitation [Shi et Eberhart, Parameter 1998].
- La composante cognitive; $y_i(t) - x_i$, Qui représente l'expérience personnelle de la particule quant à l'endroit de la meilleure solution.
- La composante sociale, $\hat{y}(t) - x_i(t)$, Qui représente la croyance de l'ensemble de l'essaim quant à l'endroit de la meilleure solution.

Selon Van den Bergh [Van den Bergh, 2002], le rapport entre l'inertie et les constantes d'accélération doit satisfaire l'équation suivante afin d'avoir une convergence garantie :

$$\frac{c_1 + c_2}{2} - 1 < \omega \qquad (3.4)$$

Sinon, les particules peuvent avoir un comportement cyclique divergent. Pour un examen approfondi de l'étude de la relation entre l'inertie et les constantes d'accélération, les lecteurs sont priés de se référer à [Ozcan et Mohan,1998], [Clerc et Kennedy,2001], [Van den Bergh, 2002], [Zheng et al, 2003], [Yasuda et al,2003] et [Trelea, 2003].

La valeur de mise à jour de la vitesse peut également être bornée avec une vitesse maximale définis par l'utilisateur; V_{max}, qui les empêcheraient de l'explosion, provoquant ainsi une convergence prématurée [Eberhart et al. 1996].

La position de la particule *i*, x_i, est ensuite mise à jour selon l'équation suivante:

$$x_i(t+1) = x_i(t) + v_i(t+1) \qquad (3.5)$$

L'algorithme d'essaims particulaires met à jour les particules dans l'essaim en utilisant les équations (3.3) et (3.5). Ce processus est répété jusqu'à ce qu'un certain nombre d'itérations est dépassé, ou la vitesse de mises à jour est proche de zéro. La qualité des particules est mesurée au moyen d'une fonction objectif qui reflète l'optimalité d'une solution particulière.
La figure 3.3 résume la base de l'algorithme d'OEP.

Algorithme 3.1 de L'OEP

Pour chaque particule $i \in 1, \dots, s$ *Faire*
  Initialiser aléatoirement x_i
  Initialiser aléatoirement v_i *(ou bien mettre* v_i *à zéro)*
  Mettre $y_i = x_i$
Fin pour
Répéter
  Pour chaque particule $i \in 1, \dots, s$ *Faire*
    Evaluer la fitness de la particule i, $f(x_i)$
    Mettre à jour y_i *en utilisant l'équation (3.1)*
    Mettre à jour $\hat{y}$ *en utilisant l'équation (3.2)*
    Pour chaque dimension $j \in 1, \dots, N_d$ *Faire*
      Appliquer la mise à jour de la vitesse en utilisant l'équation (3.3)
    Fin pour
    Appliquer la mise à jour de la position en utilisant l'équation (3.5)
  Fin pour
Jusqu'à Satisfaction d'un critère de convergence

Figure 3.3 pseudo code général pour l'OEP modèle Gbest

3.4.1 Le Modèle Gbest

Le modèle Gbest (Global Best model) converge rapidement au détriment de la robustesse [Eberhart et al, 1996]. Ce modèle maintient seulement une seule meilleure solution, appelée la meilleure particule globale, parmi toutes les particules dans l'essaim. Cette particule globale agit comme un attracteur, tirant toutes les particules à son égard. Eventuellement toutes les particules seront convergées vers cette position, si elle n'est pas mise à jour régulièrement, l'essaim peut converger prématurément.

Équations de la mise à jour pour $\hat{y}$ et v_i sont celles qui sont présentées ci-dessus, repretées ici par souci d'exhaustivité.

$$\begin{aligned}\hat{\mathrm{y}}(t) \in \{y_0(t), y_1(t), \dots, y_s(t)\} \\ = \min\{f(y_0(t)), f(y_1(t)), \dots, f(y_s(t))\}\end{aligned} \tag{3.6}$$

$$\begin{aligned}v_{i,j}(t+1) = \omega v_{i,j}(t) + c_1 r_{1,j}(t)\left(y_{i,j}(t) - x_{i,j}(t)\right) \\ + c_2 r_{2,j}(t)\left(\hat{\mathrm{y}}_j(t) - x_{i,j}(t)\right)\end{aligned} \tag{3.7}$$

Notez que $\hat{y}$ est la meilleure position globale, et elle appartient à la particule référencée comme la meilleure particule globale.

3.4.2 Le Modèle Lbest

Le modèle Lbest (Local Best model) cherche à éviter la convergence prématurée par le maintien de plusieurs attracteurs. Un sous-ensemble de particules est défini pour chaque particule à partir de laquelle la meilleure particule locale, $\hat{\mathrm{y}}_i$, est ensuite sélectionnée, le symbole $\hat{\mathrm{y}}_i$ est appelé la meilleure position locale, ou la meilleure position de voisinage.

En supposant que les indices des particules sont autour de S (S est taille de l'essaim), les équations de mise à jour du modèle Lbest pour le voisinage de taille L sont les suivantes :

$$\begin{aligned}N_i = \{y_{i-L}(t), y_{i-L+1}(t), \dots, y_{i-1}(t), y_i(t), \\ y_{i+1}(t), \dots, y_{i+L-1}(t), y_{i+L}(t)\}\end{aligned} \tag{3.8}$$

$$\hat{\mathrm{y}}_i(t+1) \in N_i \mid \quad f\left(\hat{\mathrm{y}}_i(t+1)\right) = \min\{f(a)\},\ \forall\, a \in N_i \tag{3.9}$$

$$v_{i,j}(t+1) = \omega v_{i,j}(t) + c_1 r_{1,j}(t)\left(y_{i,j}(t) - x_{i,j}(t)\right)$$

$$+c_2 r_{2,j}(t)\left(\hat{y}_{i,j}(t) - x_{i,j}(t)\right) \quad (3.10)$$

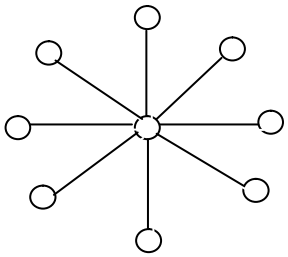
Notez que les particules sélectionnées pour être dans le sous-ensemble de voisinage N_i n'ont aucun lien les uns aux autres dans le domaine d'espace de recherche, la sélection est basée uniquement sur le numéro d'index des particules. Cela est fait pour deux raisons principales : elle est peu coûteuse en temps de calcul, étant donné que le regroupement (Clustring) ne doit pas être effectué, et elle contribue à promouvoir la diffusion des informations concernant les bonnes solutions à toutes les particules, indépendamment de leur emplacement actuel dans l'espace de recherche.

Enfin, notons que le modèle Gbest est en fait un cas particulier du modèle Lbest avec L = S. Expériences avec L = 1 ont montré que l'algorithme Lbest a convergé un peu plus lentement que la version Gbest, mais il est moins susceptible d'être trouvé coincé dans un minimum local.

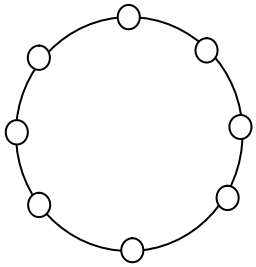
3.5 TOPOLOGIE DU VOISINAGE

Différentes topologies de voisinage ont été étudiées [Kennedy, 1999 ; Kennedy et Medes, 2002]. Deux topologies communes de voisinage est *l'étoile* et *anneau*. Pour la topologie en étoile une particule est sélectionnée comme un centre de l'étoile, qui est reliée à toutes les autres particules dans l'essaim. Cependant, toutes les autres particules ne sont reliées qu'au centre de l'étoile (modèle Gbest). Pour la topologie en anneau, les particules sont disposées en anneau. Chaque particule a un certain nombre de particules à sa droite et à gauche de son voisinage (modèle Lbest). Récemment, Kennedy et Mendes [2002] ont proposé un nouveau modèle d'OEP en utilisant un modèle de topologie de Von Neumann. Pour la topologie de Von Neumann, les particules sont connectées au moyen d'un réseau en grille (2 dimensions en treillis), où chaque particule est connectée à ses quatre voisins (ci-dessus, ci-dessous, à droite et à gauche). Figure 3.4 illustre les différentes topologies de voisinage.

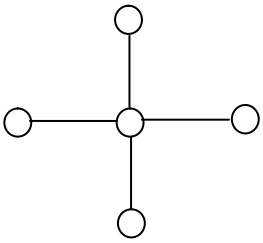
La topologie du voisinage défini avec qui chacune des particules va pouvoir communiquer. Il existe de nombreuses combinaisons dont les suivantes sont les plus utilisées :



(a) Topologie en étoile



(b) Topologie en anneau



(c) Topologie de Von Neumann

Figure 3. 4 les différentes topologies de voisinages de l'OEP

Le voisinage géographique auquel nous sommes amenés à penser en premier lieu n'est pas nécessairement pertinent car, d'une part, il s'agirait d'un voisinage trop local, et d'autre part car la sociabilisassion des particules tendent à rendre tout voisinage social en voisinage géographique.

Enfin, le voisinage géographique très lourd en terme de calculs car nécessitant de recalculer le voisinage de chaque particule à chaque itération.

Le choix de la topologie du voisinage a un profond effet sur la propagation de la meilleure solution trouvée par l'essaim. En utilisant le modèle Gbest la propagation est très rapide (c'est-à-dire toutes les particules dans l'essaim seront touchées par la meilleure solution trouvée dans itération *t*, immédiatement à l'itération *t+1*). Toutefois, en utilisant la topologie en anneau et la topologie de Von Neumann pourraient ralentir le taux de convergence parce que la meilleure solution a trouvée se propager dans plusieurs voisinages avant d'affecter tous les particules dans l'essaim. Cette lente propagation permettra aux particules d'explorer plus domaines dans l'espace de recherche et donc diminue les chances de la convergence prématurée.

3.6 MODIFICATIONS DE L'OEP

De nombreuses améliorations à l'algorithme d'optimisation par essaim de particules sont proposées. Les améliorations listées ci-dessous sont regroupées en fonction de la lacune qu'œP vise à régler. Avant que les améliorations soient discutées, une section décrivant une version binaire de d'OEP est présentée.

3.6.1 L'optimisation par essaims particulaires binaires :

Kennedy et Eberhart [Kennedy et Eberhart, 1997] ont adapté l'OEP à la recherche dans les espaces binaire. Pour l'OEP binaire, les valeurs possibles pour x_i, y_i, et $\hat{y}$ sont limités à l'ensemble $\{0, 1\}$. La vitesse, v_i, est interprétée comme une probabilité de changement entre 0 à 1, ou de 1 à 0 pendant la mise à jour de la position des particules. Par conséquent, le vecteur de vitesse reste prendre des valeurs continues. Comme chaque $v_{ij} \in \Re$, une correspondance doit être défini de v_{ij} à une probabilité dans le'ntervalle [0, 1]. Ceci est fait en utilisant une fonction sigmoïde qui donne des valeurs de vitesse dans un intervalle [0, 1]. La fonction sigmoïde est définie comme suite :

$$sig(v) = \frac{1}{1 + e^{-v}} \qquad (3.11)$$

L'équation de mise à jour de positions (équation (3.5)) est alors remplacée par l'équation probabiliste de mise à jour [Kennedy et Eberhart 1997]:

$$x_{i,j}(t+1) = \begin{cases} 0 & Si \quad r_{3,j}(t) \geq sig(v_{i,j}(t+1)) \\ 1 & Si \quad r_{3,j}(t) < sig(v_{i,j}(t+1)) \end{cases} \qquad (3.12)$$

Où $r_{3,j}(t) \sim U(0,1)$

Il peut être observé à partir de l'équation (3.12) que si $sig(v_{i,j}) = 0$ alors $x_{i,j} = 0$. Cette situation se produit lorsque $v_{i,j} < -10$. En outre, $sig(v_{i,j})$ va saturer lorsque $v_{i,j} > 10$ [Van den Bergh, 2002]. Pour éviter ce problème, il est suggéré de mettre $v_{i,j} \in$ [-4,4] et à limiter la vitesse maximale, Vmax = 4 [Kennedy et Eberhart, 2001].

OEP a également été étendu pour l'optimisation des problèmes discrets avec une représentation arbitraire [B. Jarboui et al, 2007], [Yoshida et al. 1999]; [Fukuyama et Yoshida 2001]; [Venter et Sobieszczanski Sobieski, 2002]; [Al-Kazemi et Mohan 2000]; [Mohan et Al-Kazemi, 2001]. Ces les extensions sont généralement réalisés par des arrondis de $x_{i,j}$ vers sa

valeur discrete la plus proche après l'application de l'équation de mise à jour de position (3.5) [Venter et Sobieszczanski-Sobieski, 2002].

3.6.2 L'optimisation par essaims particulaires adaptatives : TRIBES

TRIBES est un algorithme OEP créé par M. Clerc dans [CLERC, 2003]. Cet algorithme permet d'avoir un paramétrage automatique de l'OEP, l'utilisateur doit seulement définir la fonction objectif et le critère d'arrêt. Cependant, il est à signaler que TRIBES ne peut pas résoudre tous les problèmes avec exactitude. De plus, ses résultats sont probabilistes à cause de son caractère stochastique. Le but de TRIBES, d'après son auteur, est d'être efficace dans la plupart des cas et de permettre à ses utilisateurs de gagner du temps, en évitant l'étape de réglage de la métaheuristique.

La thèse de Yann Cooren [COOREN, 2008] s'attache particulièrement à l'algorithme TRIBES. Il le décrit comme un algorithme d'OEP mono-objectif sans aucun paramètre de contrôle. Cet algorithme fonctionne comme une boîte noire, pour laquelle l'utilisateur n'a qu'à définir le problème à traiter et le critère d'arrêt de l'algorithme. Dans TRIBES, l'essaim particulaire est divisé en plusieurs sous-essaims appelés tribus. Les tribus sont de tailles différentes, qui évoluent au cours du temps. Le but est d'explorer simultanément plusieurs régions de l'espace de recherche, généralement des optima locaux, avant de prendre une décision globale. Dans le but de prendre une telle décision, les tribus échangent leurs résultats tout au long du traitement. Deux types de communications sont donc à définir : la communication intra-tribu et la communication inter-tribu. Chaque tribu étant composée d'un nombre variable de particules.
TRIBES est un algorithme compétitif qui permet de trouver rapidement des optima locaux (très utile pour l'optimisation dynamique). Cependant les particules ont tendance à rester dans ces optima locaux et ont du mal à en sortir.

3.6.3 L'optimisation par essaims particulaires à facteur de construction

Clerc [1999] et Clerc et Kennedy [2001], ont proposé l'utilisation d'un facteur de constriction pour assurer la convergence. Le facteur de constriction peut être utilisé pour choisir les valeurs de ω, $c1$ et $c2$ pour assurer la convergence de l'OEP. L'équation modifiée de la vitesse de mise à jour est définie comme suit:

$$v_{i,j}(t+1) = \chi(v_{i,j}(t) + c_1 r_{1,j}(t)\left(y_{i,j}(t) - x_{i,j}(t)\right)$$

$$+c_2 r_{2,j}(t)\left(\hat{y}_j(t) - x_{i,j}(t)\right)) \qquad (3.13)$$

Où χ est le facteur de constriction définis comme suit:

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}$$

Et $\varphi = c1 + c2,\ \varphi > 4$

Eberhart et Shi [Eberhart et Shi ,2000] ont montré empiriquement que en utilisant à la fois le facteur de constriction et de la limitation de la vitesse maximale, en général améliore les performances et le taux de convergence de l'OEP.

3.6.4 L'optimisation par essaims particulaires à convergence garantie (GCPSO)

Les versions originales de "OEP" comme indiqué dans la section 3.3, soufre du problème de convergence prématurée quand $x_i = y_i = \hat{y}$, Car l'équation de mise à jour de la vitesse dépendra uniquement sur le terme $\omega v_i(t)$ [Van den Bergh et Engelbrecht 2000, Van den Bergh, 2002]. Pour surmonter ce problème, une nouvelle version d'OEP avec une convergence locale garantie a été présentée par Van den Bergh [2002], appelée GCPSO. Dans GCPSO, la meilleure particule globale avec l'index τ est mise à jour en utilisant une autre équation de mise à jour de la vitesse :

$$v_{\tau,j}(t+1) = -x_{\tau,j}(t) + \hat{y}_j(t) + \quad \omega v_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}(t)) \qquad (3.14)$$

Qui se traduit par l'équation de mise à jour de position de

$$x_{\tau,j}(t+1) = \quad \hat{y}_j(t) + \quad \omega v_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}(t)) \qquad (3.15)$$

Van den Bergh propose de répéter l'algorithme jusqu'à ce que ρ est suffisamment petites, ou jusqu'à ce que les critères d'arrêt sont remplies. L'arrêt de l'algorithme, une fois ρ atteint une limite inférieure n'est pas conseillé, car il ne signifie pas nécessairement que toutes les particules ont convergé - d'autres particules peut rester encore explorer les différentes parties de l'espace de recherche.

Il est important de noter que, pour l'algorithme GCPSO, toutes les particules sauf la meilleure particule globale suivent les équations (3.3) et (3.5). Seule la meilleure particule globale suit les nouvelles équations de vitesse et de mise à jour de la position.

Selon Van den Bergh [2002] et Peer et al. [2003], GCPSO est généralement plus performant que l'OEP lorsqu'il est appliqué aux problèmes de Benchmark. Cette amélioration de la performance est particulièrement notable lorsqu'OEP et GCPSO sont appliquées sur des fonctions uni-modales, mais les performances des deux algorithmes ont été généralement comparables pour les fonctions multimodales [Van den Bergh, 2002]. En outre, en raison de son taux de convergence rapide, GCPSO est légèrement plus susceptibles d'être pris au piège dans les optima locaux [Van den Bergh, 2002]. Toutefois, il a une convergence locale garantie et que l'OEP original ne l'est pas.

3.6.5 L'optimisation par essaims particulaires Multi-start (MPSO)

Van den Bergh [2002], a proposé MPSO qui est une extension de GCPSO afin de le rendre un algorithme de recherche globale. MPSO fonctionne comme suit:

1. initialiser aléatoirement toutes les particules dans l'essaim.
2. Appliquer GCPSO jusqu'à ce que la convergence atteigne un optimum local. Et enregistrer la position de cet optimum local.
3. Répétez les étapes 1 et 2 jusqu'à ce que certains critères d'arrêt soient remplis.

A l'étape 2, GCPSO peut être remplacé par l'OEP original. Plusieurs versions de MPSO ont été proposées par Van den Bergh [2002] basé sur la manière utilisée pour déterminer la convergence de GCPSO. Une bonne approche consiste à mesurer le taux de variation de la fonction objectif comme suit:

$$f_{ratio} = \frac{f(\hat{y}(t)) - f(\hat{y}(t-1))}{f(\hat{y}(t))}$$

Si f_{ratio} est moins d'un seuil spécifié par l'utilisateur, un compteur est incrémenté. L'essaim est supposé avoir convergé si le compteur atteint un certain seuil [Van den Bergh, 2002]. Selon Van den Bergh [2002], MPSO généralement donne des meilleurs résultats que GCPSO dans la plupart des cas testés. Toutefois, la performance de MPSO se dégrade de manière significative lorsque le nombre de dimensions de la fonction objectif augmentent. [Van den Bergh, 2002].

3.6.6 L'optimisation par essaims particulaires Attractives et Répulsives (ARPSO)

ARPSO [Riget et Vesterstrøm 2002] alterne entre deux phases : L'attraction et la répulsion basant sur une mesure de diversité. Dans la phase d'attraction, ARPSO utilise œP pour permettre une circulation rapide de l'information, en tant que telle particule attirer les uns les autres et ainsi de la diversité est réduites. Il a été constaté que 95 % des améliorations de la fonction objectif ont été réalisées au cours de cette phase. Cette observation montre l'importance de la faible diversité de peaufiner la solution. Dans la phase de répulsion, les particules sont poussées loin de la meilleure solution trouvée jusqu'à ici, augmentant ainsi la diversité. Sur la base des expériences menées par Riget et Vesterstrøm [Riget et Vesterstrøm 2002] ARPSO dépasse OEP et AG dans la plupart des cas testés.

3.6.7 L'optimisation par essaims particulaires à Sélection

Une approche hybride combinant l'OEP avec une méthode de sélection par tournoi a été proposée par Angeline [Angeline, Using Selection 1998]. Chaque particule est classée sur la base de ses performances par rapport à un groupe sélectionné de façon aléatoire des particules. À cet effet, une particule est récompensée par un point pour chacun des adversaires dans le tournoi pour lequel la particule a un meilleur fitness. La population est ensuite triée en ordre descendant selon les points accumulés. La moitié inférieure de la population est alors remplacée par la moitié supérieure. Ce pas réduit la diversité de la population. Les résultats ont démontré que l'approche hybride donne des meilleurs résultats que l'OEP (sans et χ) pour les fonctions uni-modales. Toutefois, l'approche hybride donne des mauvais résultats que l'OEP pour les fonctions ayant beaucoup d'optima locaux. Par conséquent, il peut être conclu que bien que l'usage d'une méthode de sélection améliore la capacité d'exploitation de l'OEP, il réduit sa capacité d'exploration [Ven den Bergh, 2002]. Par conséquent, en utilisant une méthode de sélection avec OEP peut aboutir à la convergence prématurée.

3.6.8 L'optimisation par essaims particulaires à Reproduction (Breeding

Løvberg et al. [2001] ont proposé une modification aux OEP en utilisant un opérateur arithmétique de croisement, considéré comme un opérateur de reproduction (Breeding operator), afin d'améliorer le taux de convergence de l'OEP. Pour chaque particule dans l'essaim est attribuée une probabilité de reproduction définie par l'utilisateur. Sur la base de ces probabilités, deux particules parents sont choisis au hasard afin de créer des descendants en utilisant un opérateur arithmétique de croisement. Les descendants (Offspring) remplacent les parti-

cules parents. La meilleure position personnelle de chaque particule descendant est initialisé à sa position actuelle (c'est-à-dire $x_i = y_i$), et sa vitesse est défini comme la somme des vitesses des deux parents normalisée à la longueur initiale de vitesse de chaque parent. Le processus est répété jusqu'à ce qu'un nouvel essaim de même taille a été généré. OEP avec reproduction en général donne des meilleurs résultats que l'OEP lorsqu'il est appliqué sur les fonctions multimodales [Løvberg et al. 2001].

3.6.9 L'optimisation par essaims particulaires à mutation

Récemment, Higashi et Iba [2003], ont proposé un OEP hybride avec une mutation gaussienne. De même, Esquivel et Coello Coello [2003], ont proposé hybridation Lbest-et-Gbest OEP avec un puissant mécanisme de maintenance de la diversité, appelé un opérateur de mutation non-uniforme pour résoudre le problème de convergence prématurée de l'OEP. Selon Esquivel et Coello Coello [2003] l'approche hybride de lbest de l'OEP et l'opérateur de mutation non-uniforme dépasse OEP et GCPSO dans toutes les expériences menées.

3.6.10 L'optimisation par essaims particulaires multi-essaim (Multi Swarm PSO)

L'idée d'utiliser plusieurs essaims au lieu d'un seul essaim a été appliquée à l'OEP par Løvberg et al. [2001] et Van den Bergh et Engelbrecht [2001]. L'approche proposée par Løvberg et al. [2001] est une extension de l'OEP avec un opérateur croisement discuté ci-dessus. L'idée consiste à diviser l'essaim en plusieurs essaims. Chaque essaim dispose de ses propres meilleures particules globales. La seule interaction entre les essaims se produit lors l'opération de croisement sélectionne deux particules à partir des essaims différents pour le croisement. Les résultats dans Løvberg et al. [2001] ont montré que cette approche n'a améliore pas les performances de OEP. Les raisons pour se la [Jensen et Kristensen, 2002]:

- Les auteurs ont divisé un essaim de 20 particules dans six essaims. Par conséquent, chaque essaim contient peu de particules. Les essaims avec peu de particules manquent de diversité et donc peu de pouvoir d'exploration.
- Aucune action n'a été prise pour empêcher les essaims d'être trop similaires

Les problèmes ci-dessus ont été traités par Jensen et Kristensen [2002]. L'approche modifiée fonctionne à l'aide de deux essaims (chacun avec une taille de 20 particules) et le maintien les éloigner les uns des autres de façon aléatoire, par la dispersion de l'essaim (avec le pire per-

formance) sur l'espace de recherche, ou en ajoutant une petite mutation de la position de les particules dans cet essaim. L'approche à l'aide de la technique de mutation donne généralement de meilleurs résultats que l'OEP lorsqu'elle est appliquée sur les problèmes de références [Jensen et Kristensen, 2002].

3.6.11 L'optimisation par essaims particulaires Coopératives

Les algorithmes de PSI discutés jusqu'à présent permettent de résoudre des problèmes d'optimisation par des particules à J-dimensions, où J ai le nombre de paramètres a optimisé, c'est-à-dire le nombre de composants de la solution finale. Un essaim a pour tâche de trouver les valeurs optimales pour tous les J paramètres. Une approche coopérative a été développée dans [Van den Bergh et Engelbrecht 2000, Van den Bergh et Engelbrecht 2001, Van den Bergh, 2002]. Pour l'OEP coopérative (CPSO), les J paramètres à optimiser peut être divisé en J essaims, où chaque essaim optimise un seul paramètre du problème. Le processus d'optimisation au sein de chacune des J essaims se produisaient en utilisant l'un des algorithmes d'EOP discuté précédemment.

La difficulté avec l'algorithme d'OEP coopérative CPSO est comment évaluer la fonction objectif de ces particules unidimensionnelles dans chaque essaim. L'optimalité de chaque particule de l'essaim S_j ne peut pas être calculée dans l'isolement des autres essaims, puisqu'une chaque particule dans un essaim représente seulement un paramètre de la solution complète J-dimensionnel.
Il est important de noter que l'algorithme CPSO est surtout applicable aux problèmes où les paramètres à être optimisés sont indépendants l'un de l'autre.

3.7 OPTIMISATION PAR ESSAIMS PARTICULAIRES & ALGORITHMES GENETIQUES

L'algorithme d'optimisation par essaim de particule est un algorithme intrinsèquement continu alors que l'algorithme génétique est un algorithme discret [Venter et Sobieszczanski-Sobieski, 2002]. Expériences menées par [Veeramachaneni et al, 2003] ont montrés qu'une OEP donne des meilleurs résultats que les AGs appliquée sur certains problèmes optimisation. En outre, selon à [Robinson et al, 2002], une OEP a fait mieux que les AGs lorsqu'elle est appliquée à la conception d'un problème difficile d'ingénierie. En plus, un algorithme d'OEP binaire a été comparé avec un AG par [Eberhart et Shi, Comparaison 1998] et [Kennedy et Spears, 1998]. Les résultats montrés que les OEPs binaires sont généralement plus rapides,

plus robustes et plus performants que les Algorithmes génétiques binaires, en particulier lorsque la dimension d'un problème augmente.

Les approches hybrides combinant l'algorithme d'OEP et AG ont été proposés par [Veeramachaneni et al, 2003] afin d'optimiser « the profiled corrugated horn antenna ». Les travaux d'hybridation, en prenant la population d'un algorithme quand il n'a pas fait d'amélioration de la fonction objectif et en l'utilisant comme une population de départ pour l'autre algorithme. Deux versions ont été proposées : GA-PSO et PSO-GA. GA-PSO, la population de l'algorithme génétique AG est utilisée pour initialiser la population de l'algorithme d'optimisation particulaire OEP. Pour PSO-GA, la population l'OEP est utilisée pour initialiser la population d'AG. Selon [Veeramachaneni et al, 2003], PSO-GA effectué un peu mieux que PSO. Les deux OEP et PSO-GA donnent des résultats mieux que les deux GA et GA-OEP.

Certaines des premières applications d'OEP ont été appliquées pour entrainer des réseaux de neurones (NNs). Les résultats ont démontré que l'OEP est mieux que l'AG et d'autres algorithmes d'entraînement [Eberhart et Shi, Evolving, 1998] ; [Van den Bergh et Engelbrecht 2000]; [Ismail et Engelbrecht, 2000].

Selon [Eberhart et Shi, 1998], la performance de l'algorithme d'optimisation par essaim de particule est insensible à la taille de la population (toutefois, la taille de la population ne devrait pas être trop petite). Cette observation a été vérifiée par [Løvberg, 2002] et [Krink et al, 2002]. En conséquence, OEP avec un essaim de petite taille donne des résultats comparables à des résultats des algorithmes génétiques avec des populations plus importantes. En outre, Shi et Eberhart ont observé que l'OEP est efficace pour les problèmes de grandes dimensions. Cette observation a été vérifiée par [Løvberg, 2002].

3.8 INCONVENIENTS DE L'OPTIMISATION PAR ESSAIMS PARTICULAIRES

L'optimisation par essaims particulaires et autres algorithmes de recherche stochastiques ont deux inconvénients majeurs [Løvberg 2002]. Le premier inconvénient de l'OEP, et les autres algorithmes de recherche stochastiques, est la *convergence prématurée* (ou *stagnation*).

La convergence prématurée arrive quand quelques individus pauvres attirent la population, à cause d'une mauvaise initialisation ou optimum locale, qui provoque la limitation de l'exploration des nouvelles zones de l'espace de recherche [Dorigo et al. 1999]. Une des

causes de ce problème est qu'une particule attire tout l'essaim vers elle, ce qui provoque que toutes les particules deviennent trop semblables (c'est-à-dire la population perd la diversité).

Selon Angeline [Angeline, 1998], malgré OEP trouve de bonnes solutions beaucoup plus rapidement que d'autres algorithmes évolutionnaires, elle ne peut pas améliorer la qualité des solutions que le nombre d'itérations augmente. OEP souffre généralement de la convergence prématurée pour l'optimisation des problèmes multi-modaux. La raison derrière ce problème est que, pour la version Gbest d'OEP, les particules convergent vers un point unique, qui est sur la ligne entre la meilleure position globale et la meilleure position personnelle. Ce point n'est pas garanti d'être même un optimum local. Les preuves peuvent être trouvées dans [Van den Bergh, 2002]. Une autre raison de ce problème est le rythme rapide des flux d'information entre particules, résultant en la création de particules similaires (avec une perte de diversité), qui augmente la possibilité d'être pris au piège dans les optima locaux [Riget et Vesterstrøm, 2002]. Plusieurs modifications de l'OEP ont été proposées pour remédier à ce problème. Deux de ces modifications ont déjà été examinés, à savoir, le facteur d'inertie et le modèle Lbest. D'autres modifications sont examinées dans la section suivante.

Le deuxième inconvénient est que les approches stochastiques ont des performances dépendantes du problème à optimisé. Cette dépendance se traduit généralement par la configuration des paramètres de chaque algorithme. Ainsi, en utilisant différents paramètres pour un algorithme de recherche stochastique aboutir à des écarts de haute performance. En général, aucun paramétrage n'existe qui peut être appliqué à tous les problèmes. Ce problème est amplifié dans le cas d'OEP où la modification d'un paramètre d'OEP peut causer un effet grand proportionnellement sur le résultat [Løvberg 2002]. Par exemple, l'augmentation de la valeur du facteur d'inertie, ω, augmentera la vitesse des particules résultant en plus d'exploration (recherche globale) et moins exploitation (recherche locale). D'autre part, la diminution de la valeur de ω va diminuer la vitesse de la particule résultant en plus de l'exploitation et de moins d'exploration. Donc trouver la meilleure valeur pour ω n'est pas une tâche facile et elle peut se différé d'un problème à un autre. Par conséquent, il peut être conclu que les performances OEP sont dépendantes au problème à optimisé.

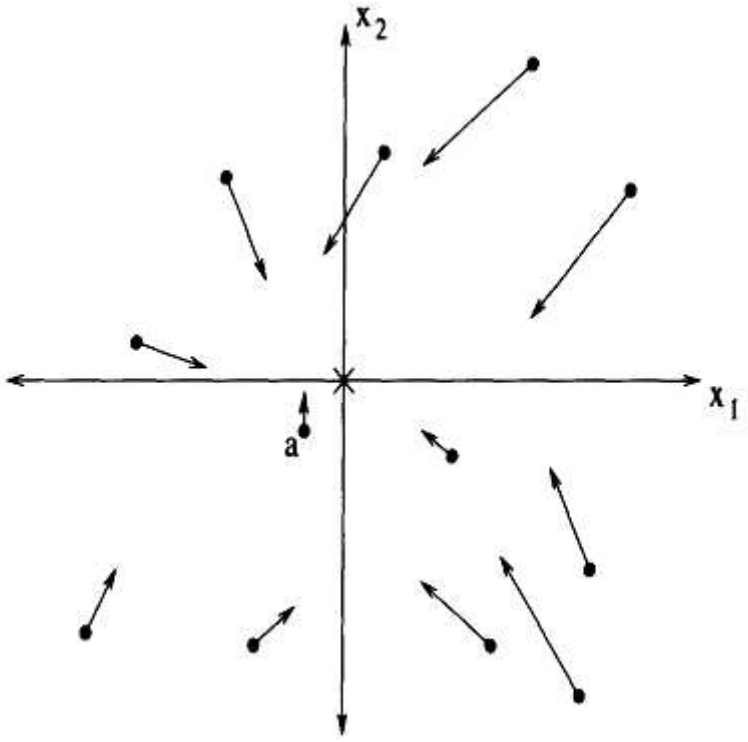
Une solution au problème dépendance des performances OEP au problème à optimiser est d'utiliser l'auto-adaptation des paramètres. Dans l'auto-adaptation, les paramètres de l'algorithme sont ajustés en fonction du retour d'information (Feedback) du processus de recherche [Løvberg 2002]. [Bäck, 1992] a appliquée avec succès l'auto-adaptation au AG. Auto-adaptation a été appliquée aux OEP par [Clerc, 2003], [Shi et Eberhart, 2001], [Hu et Eber-

hart Adaptive, 2002], [Ratnaweera et al, 2003] et [Tsou et Macnish, 2003], [Yasuda et al, 2003] et [Zhang et al, 2003].

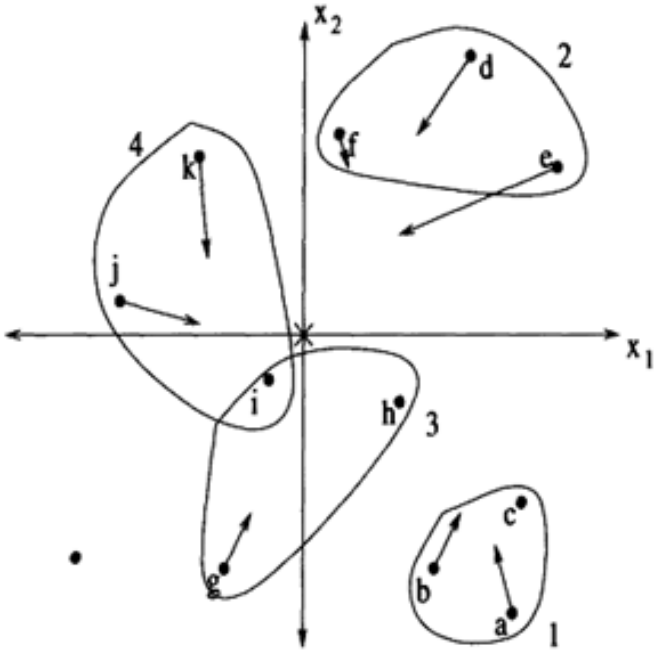
Le problème de dépendance de la performance au problème à optimisé peut être résolu par le biais de l'hybridation. L'hybridation se réfère à combiner différentes approches et de bénéficier des avantages de chaque approche [Løvberg 2002]. L'hybridation a été appliqué aux OEP par [Angeline, 1998], [Løvberg, 2002], [Krink et Løvbjerg, 2002], [Veeramachaneni et al, 2003].

3.9 EXEMPLE DEMONSTRATIF

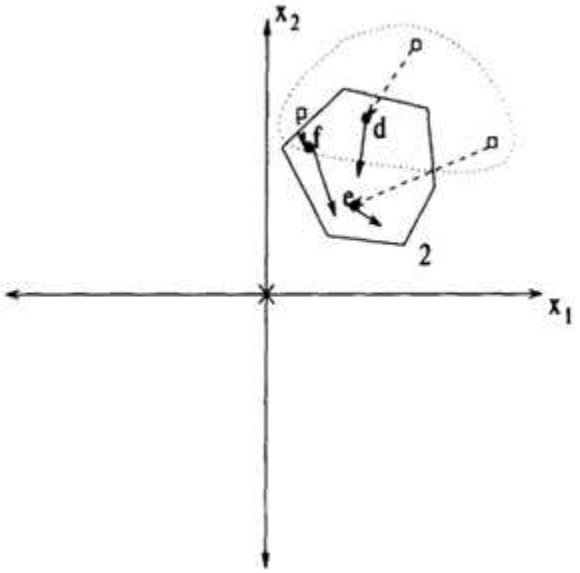
Cette section illustre l'application de l'OEP pour trouver le minimum de la fonction $f(x_1, x_2) = x_1^2 + x_2^2$, pour qui le minimum est x1 = 0, x2 = 0. Les particules ont volé dans un espace en deux dimensions. Les positions initiales de l'essaim de particules sont représentées par les points à la figure 3.4. Le meilleur point est indiqué par une croix. La Figure 3.4 (a) illustre la version *Gbest* de l'OEP. La Particule *a* est la meilleure solution globale. Initialement, le *Pbest* (meilleure position personnelle) de chaque individu est son point actuel. Par conséquent, seulement la particule *a* influencé le mouvement de toutes les particules. Les flèches indiquent la direction et de l'amplitude du changement dans les positions. Toutes les particules sont ajustées vers la particule *a*. La version *Lbest*, comme l'illustre la figure 3.4 (b), montre comment les particules sont influencées par leurs voisins immédiats. Pour garder le graphique lisible, seuls certains des mouvements sont illustrés. Dans le voisinage 1, les deux particules *a* et *b* se déplacent vers *c*, qui est la meilleure solution dans ce voisinage. Considérant le voisinage 2, la particule *d* se déplace vers *f*, il en va de même *e*. Pour la prochaine itération, *e* sera la meilleure solution pour le voisinage 2. Maintenant, *d* et *f* se déplacent vers *e* comme l'illustre à la figure 3.4 (c) (seulement une partie de l'espace de solution est illustrée).Les blocs représentent les positions antérieures. Notez que *e* reste la meilleure solution pour le voisinage 2. Aussi évident le mouvement se dirige vers le minimum, bien que plus lent que pour la version Gbest.



(a) Illustration pour le modèle Gbest d'OEP



(b) Illustration pour le modèle Lbest d'OEP (essaim initial)



(b) Illustration pour le modèle Lbest d'OEP (deuxième essaim)

Figure. 3.5 illustration des modèles Gbest et Lbest d'OEP

3.10 APPLICATIONS DE L'OEP

L'optimisation par essaims particulaires a été principalement utilisée pour trouver les minima et maxima de fonctions non linéaires [Shi et Eberhart, 1999]. OEP a également été utilisée avec succès pour entériner les réseaux de neurones NNS [Eberhart et al. 1996, Kennedy et Eberhart 1999, et Ismail Engelbrecht 1999, Van den Bergh 1999, Van den Bergh et Engelbrecht, 2001]. L'OEP a également été été utilisé avec succès pour l'analyse de tremblement humaine pour prévoir la Maladie de Parkinson [Shi et Eberhart 1999].

3.11 CONCLUSION

L'OEP a déjà montré qu'elle est une métaheuristique d'optimisation efficace et robuste, en considérant la simplicité de l'algorithme. Cette méthode d'optimisation se base sur la collaboration des individus entre eux. Elle a d'ailleurs des similarités avec les algorithmes de colonies de fourmis, qui s'appuient eux aussi sur le concept d'auto-organisation. Cette idée veut qu'un groupe d'individus peu intelligents puisse posséder une organisation globale complexe. L'OEP a donnée des meilleurs résultats par rapport avec d'autres métaheuristiques, comme les algorithmes génétiques, l'optimisation par colonie de fourmis,… etc. ces résultats nous ont

motivés pour essayer d'appliquer l'OEP pour résoudre le problème de la fragmentation verticale des relations dans les bases de données distribuées.

A ce jour, environ 250 articles sur le sujet ont été publiés, mais très peu en français. Un excellent point d'entrée est le site Particle Swarm Central, avec une bibliographie très complète, des liens vers des documents et des programmes à télécharger, ainsi qu'une liste de chercheurs travaillant dans ce domaine. Le livre Swarm Intelligence, écrit par les deux concepteurs de la méthode, y consacre quelques chapitres et donne des aperçus plus généraux sur les questions d'intelligence collective.

Chapitre 4

Problème traité : Fragmentation Verticale

Table des matières

4.1 INTRODUCTION

La technologie des Bases de données distribuées est prévue d'avoir un impact significatif sur le traitement des données dans les années à venir. L'introduction de nombreux produits commerciaux et de la poursuite intensité de l'intérêt dans les systèmes de bases de données distribuées dans la communauté de recherche et industrielle, indiquent que les systèmes de bases de données distribuées deviendront de plus en plus populaires et, éventuellement, ils vont remplacer les systèmes des bases de données centralisées dans l'avenir. En plus, la dis-

ponibilité des réseaux de communication à grande vitesse et, en particulier, de plus en plus la popularité de l'Internet et des intranets peut accélérer le processus de transition. Les systèmes de base de données distribuée ont de nombreux avantages potentiels prometteurs, tels que l'amélioration de la fiabilité et la disponibilité, l'amélioration de la performance, partageabilité, l'évolutivité, l'augmentation de la robustesse, et de l'autonomie locale, entre d'autres. Pour concrétiser pleinement le potentiel de bénéfices des bases de données distribuées toutefois, une série de problèmes techniques doivent être résolue de façon satisfaisante. La conception des bases de données distribuées est certainement l'un des plus importants et des questions fondamentales à résoudre.

La fragmentation des données et l'allocation sont deux des aspects essentiels de la conception des bases de données distribuées. Les problèmes de la fragmentation des données et l'allocation dans la conception de bases de données distribuées sont NP-difficile dans la plupart des cas et notoirement difficiles à résoudre, ce qui rend développer de bonnes méthodes de solution une priorité élevée. L'Allocation de données est généralement traitée indépendamment de la fragmentation. Ils sont, toutefois, des processus hautement interdépendants. Il est plus raisonnable d'étendre la méthodologie en deux étapes afin que l'interdépendance de la fragmentation et les décisions d'allocation soient correctement prises en compte. Bien que la méthodologie intégrée puisse être très compliquée, depuis la fragmentation et l'allocation sont des problèmes de conception, il est souvent abordable ou utile de développer et d'utiliser certaines méthodes sophistiquées de solution. Dans ce chapitre, nous tentons d'expliquer en détail l'intérêt de la fragmentation et nous présentons quelques travaux qui sont déjà faits pour résoudre le problème de la fragmentation verticale.

4.2 LA FRAGMENTATION

Dans la littérature, deux termes sont utilisés pour la fragmentation d'une relation : *partitionnement* et *fragmentation*. Le partitionnement d'une relation se définit par la division de cette dernière en plusieurs partitions disjointes. La fragmentation consiste en la division en plusieurs fragments (sous-ensembles de la relation) qui peuvent être non disjoints. Cependant, la plupart des chercheurs utilisent les deux termes indifféremment [S.B. Navathe et al, 1995], [Ozsu et Valduriez, 1999], dans ce mémoire, nous ne ferons pas de différence entre les deux concepts.

Dès le début des années 80, de nombreuses études ont été menées sur la fragmentation dans les bases de données relationnelles [S. Ceri et al,1982], [S. Ceri et G. Belagatti, 1984],

[D. Cornell and P.S. Yu, 1987],[S.B. Navathe et al, 1984], , [Cornell & Yu, 1990], [Jun du et al, 2006]. Avec l'apparition des systèmes de bases de données objets, de nouvelles études ont vu le jour [K. Karlapalem et al, 1994], [K. Karlapalem et al, 1995], [L. Bellatreche et al, 1999].

Définition : Un schéma de fragmentation [K. Karlapalem et al, 1994] *est le résultat du processus de la fragmentation.*

Donc La fragmentation est le processus de décomposition d'une base de données en un ensemble de sous-bases de données. Cette décomposition doit être sans perte d'information.
La fragmentation peut être coûteuse s'il existe des applications qui possèdent des besoins opposés.

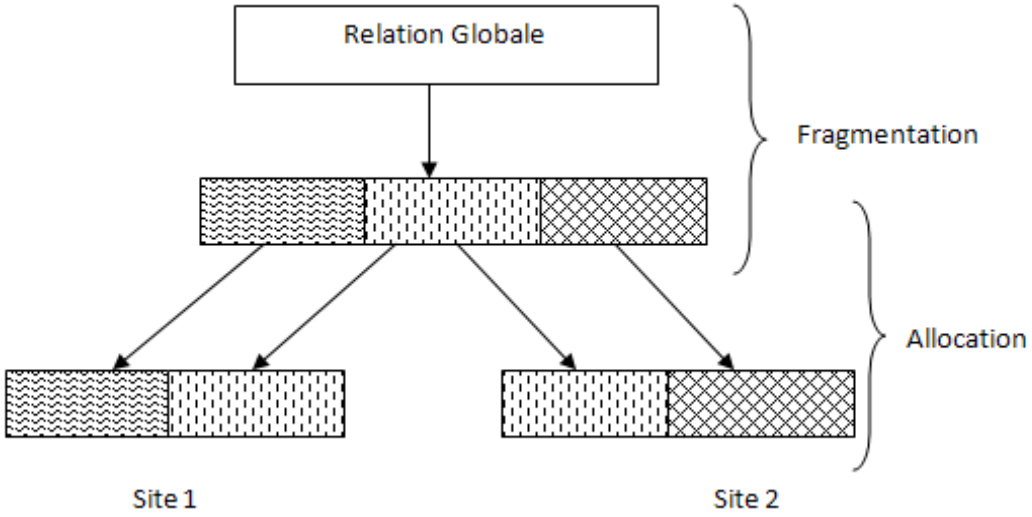


Figure. 4.1 Fragmentation et allocation d'une base de données

Les règles de la fragmentation sont les suivantes :

1. **La complétude** : pour toute donnée d'une relation R, il existe un fragment R_i de la relation R qui possède cette donnée.
2. **La reconstruction** : pour toute relation décomposée en un ensemble de fragments R_i, il existe une opération de reconstruction.
3. **Disjonction** : chaque élément de R ne doit pas être dupliqué

4.3 INTERET DE LA FRAGMENTATION

Dans un contexte distribué, la fragmentation consiste à partitionner les tables du schéma conceptuel global de la base de données en plusieurs fragments. Cette fragmentation est faite en fonction des besoins de chaque site du système distribué.

Les fragments issus du processus de partitionnement seront ensuit alloués de façon optimale sur les sites du système réparti (cf. Figure 4.2). L'optimisation est guidée par une fonction objectif (dans la plupart des cas, on cherche à minimiser les coûts de transfert des données lors de l'exécution d'un ensemble de requêtes. Une fois les fragments alloués aux sites, des optimisations locales peuvent être réalisées au niveau de chaque site (comme par exemple, la création d'index propres aux fragments de chaque site).

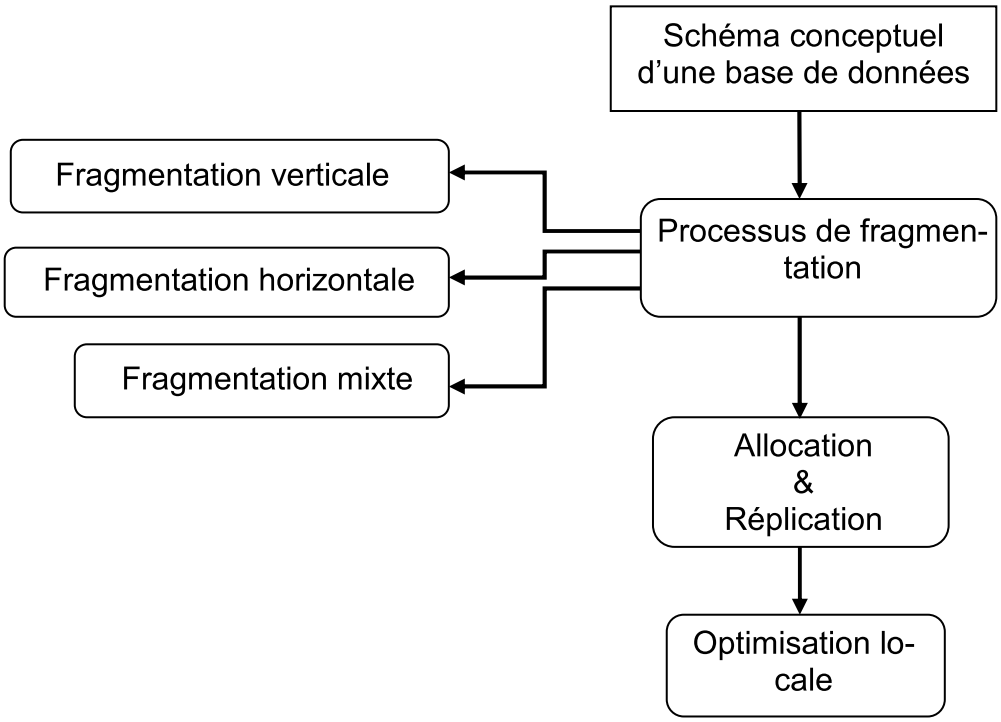


Figure. 4.2 Processus de fragmenation

4.4 LES TYPES DE FRAGMENTATION

Deux sortes de fragmentation sont possibles : la fragmentation horizontale et la fragmentation verticale en plus la combinaison des deux types qui est la fragmentation mixte ou hybride.

4.4.1 La fragmentation verticale

Dans la fragmentation verticale, une relation est divisée en sous-relations appelées *fragments verticaux* qui sont des projections appliquées à la relation.

Soit une relation $R(K\,;\,A_1, A_2, \dots, A_n)$, avec n attributs $A_1, A_2, \dots, A_n$ et une clé K. Chaque attribut A_i a un domaine de valeurs que nous appelons $dom(A_i)$. La fragmentation verticale de la relation R est donnée par :

$V^1(K, A_1^1, A_2^1, \dots, A_{k1}^1)$, $V^2(K, A_1^2, A_2^2, \dots, A_{k2}^2)$,… $V^p(K, A_1^p, A_2^p, \dots, A_{kp}^p)$ où chaque attribut $A_j^i \in \{A_1, A_2, \dots, A_n\}$. Les fragments verticaux $V_1, V_2, \dots, V_p$ sont disjoints si chaque attribut A_i ($1 \leq i \leq n$) de la relation R appartient à un et un seul fragment vertical. La clé K est dupliquée dans chaque fragment afin de facilité la reconstruction de la relation R à partir de ces fragments qui est réalisée par la jointure de p fragment verticaux.

La fragmentation verticale favorise naturellement le traitement des requêtes de projection portant sur les attributs utilisés dans le processus de la fragmentation, en limitant le nombre de fragments à accéder. Son inconvénient est qu'elle requiert des jointures supplémentaires lorsqu'une requête accède à plusieurs fragments.

Exemple Fragmentation Verticale :

1. Fragments définis par projection

   Exemple Commandes(NC, NoClient, Produit, Qté)

   Relation Commandes

NC	***NoClient***	***Produit***	***Qté***
CO1	C1	P1	10
CO2	C2	P2	20
CO3	C3	P3	100
CO4	C4	P4	30

   CommandeA=$\pi_{NC,NoClient}(Commandes)$

   CommandeB=$\pi_{NC,Produit,Qté}(Commandes)$

   Relation CommandeA

NC	***NoClient***
CO1	C1

CO2	C2
CO3	C3
CO4	C4

Relation CommandeB

NC	***CO1***	***CO2***
CO3	CO4	Produit
P1	P2	P3
P4	Qté	10
20	100	30

2. Reconstruction par jointure

   Commandes = CommandeA * CommandeB

4.4.2 La fragmentation horizontale

Comme pour la fragmentation verticale, la fragmentation horizontale a été étudiée dans le cadre des bases de données relationnelles. Elle consiste à diviser une relation R en sous ensemble de n-uplets appelés *fragments horizontaux*, chacun étant défini par une opération de restriction appliquée à la relation. Les n-uplets de chaque fragment horizontal satisfait une clause de prédicats[1]. Le schéma de fragmentation de la relation R est donné par : $H_1 = \sigma_{cl1}(R)$, $H_2 = = \sigma_{cl2}(R)$, …, $H_q = \sigma_{clq}(R)$, où cl_i est une clause de prédicats. La reconstruction de la relation R à partir de ces fragments horizontaux est obtenue par l'opération d'union de ces fragments.

La fragmentation horizontale se décline en deux versions la fragmentation *primaire* et *dérivée* (ou indirecte) La fragmentation primaire d'une relation est effectuée grâce à des prédicats de sélection définis sur la relation. La fragmentation horizontale dérivée s'effectue avec des prédicats de sélection définis sur une autre relation.

La fragmentation horizontale primaire favorise le traitement des requêtes de restriction portant sur les attributs utilisés dans le processus de la fragmentation.

[1] Une clause de prédicats est une combinaison de prédicats avec les opérateurs logiques ∧ et ∨

La fragmentation horizontale dérivée est utile pour le traitement des requêtes de jointure.

Exemple Fragmentation Horizontale :

Soit la relation : Clients (NoClient, Nom, Ville)

NoClient	***Nom***	***Ville***
C1	Ali	BBA
C2	Mohamed	Sétif
C3	Khaled	Alger
C4	Samir	Sétif

1. Fragments définis par sélection

   Client1=$\sigma_{Ville=Setif}(Clients)$
   Client2 =$\sigma_{Ville\neq Setif}(Clients)$

   La relation Client1

NoClient	***Nom***	***Ville***
C2	Mohamed	Sétif
C4	Samir	Sétif

   La relation Client2

NoClient	***Nom***	***Ville***
C1	Ali	BBA
C3	Khaled	Alger

2. Reconstruction par Union des fragments
   Clients = Client1 ∪ Client2

4.4.3 La fragmentation hybride ou mixte

Dans la plupart des cas, un partitionnement horizontal ou vertical simple d'une base de données ne satisfait pas les demandes des applications. Dans ce cas, un partitionnement horizontal peut être suivi par un partitionnement vertical ou vice versa.

La fragmentation mixte combine les deux types de fragmentation : horizontale et verticale. Elle consiste à partitionner une relation en sous-ensemble de sous relation, ces dernières étant définies par la fragmentation verticale et les sous-ensembles par la fragmentation horizontale.

4.5 LES CONTEXTES DE LA FRAGMENTATION

L'idée sous-jacente à la fragmentation est de pouvoir constituer des ensembles de données partielles dont les attributs (pour la fragmentation verticale) et les n-uplets (pour la frag-

mentation horizontale) ont des propriétés géographiques communes. Le mot géographique peut, comme nous allons le voir, être perçu à une échelle différente suivant le contexte centralisé, réparti ou parallèle.

***Définition* : Base de Données Répartie** est l'ensemble de bases de données coopérantes, chacune résident sur un site différent, cet ensemble étant vu et manipulé par l'utilisateur comme une seule base de données centralisée.

***Définition* : Base de Données Parallèle e**st une base de données repartie homogène dont les sites sont les nœuds d'une machine parallèle (multi-processeurs), les nœuds communiquent par messages.

4.5.1 Le contexte centralisé

La fragmentation verticale dans un contexte centralisé favorise naturellement le traitement des requêtes de projection portant sur les attributs utilisés dans la définition des fragments, en limitant le nombre de fragments à accéder.

La fragmentation horizontale permet de limiter le nombre de pages accédées, ou en d'autres termes, à réduire le nombre de n-uplets accédés inutilement.

4.5.2 Le contexte réparti

Le principe général reste bien sûr le même, seule, l'interprétation de la notion de propriétés géographique, communes entre attributs, change. Dans le contexte réparti, la fragmentation consiste à découper une relation par rapport à un ensemble de sites. Il faut donc une certaine adéquation entre sites et l'utilisation des données. Dans un tel contexte, deux phases se distinguent lorsqu'on parle de la fragmentation : la phase de partitionnement en elle-même, et la phase d'allocation des fragments aux sites.

Les premiers travaux importants sur la fragmentation ont été réalisés dans le contexte de la conception d'une base de données repartie (distributed data base design)

4.5.3 Le contexte parallèle

Dans ce contexte, la fragmentation consiste à partitionner les données d'une relation afin de pouvoir les traiter en parallèle et ainsi diminuer le temps de réponse. Il n'est donc plus question de rassembler les données utilisées simultanément, mais au contraire de les disséminer pour obtenir un taux important de parallélisme pour l'exécution d'une même opération.

Les fragmentations verticale et horizontale permettent respectivement l'augmentation du parallélisme inter-requêtes et intra-requêtes.

4.6 LES AVANTAGES ET LES INCONVENIENTS DE LA FRAGMENTATION

Les objectifs de la fragmentation de données sont multiples :

- Limiter le nombre d'accès inutiles aux données (attributs ou n-uplets).
- Limiter le transfert de données (en nombre et en volume) en les répartissant où elles sont le plus utilisées dans le cas d'une base de données repartie (par exemple dans les agences d'une banque nationale, ou encore aux guichets de réservation d'une compagnie aérienne).
- Accroître les performances par la répartition de la charge de travail en plusieurs unités de traitement opérant en parallèle.
- Augmenter la fiabilité en faisant effectuer le même traitement par plusieurs ordinateurs. Par exemple, dupliquer les données sur différents sites, lorsque la base de données concerne un domaine sensible où la moindre erreur peut avoir des conséquences catastrophiques (comptes bancaires, centrales nucléaires, lancement de fusées spatiales).
- Etendre la disponibilité des informations, en les dupliquant sur plusieurs sites. L'exemple-type est l'annuaire téléphonique national dont chaque ville importante possède un exemplaire.

La fragmentation a plusieurs avantages, mais elle a aussi des inconvénients :

- Certains problèmes, tels que la sécurité, la synchronisation ou la coordination, sont difficiles à résoudre dans les systèmes répartis.
- Le manque de méthode de conception d'une base de données répartie. De nos jours, il n'existe pas ou peu de méthode ou outils pour convertir une base de données centralisée en une base de données répartie.
- Qui dit fragmentation des données, dit surplus d'informations de maintenance tels que les dictionnaires globaux et locaux de données, les fonctions de partitionnement, les index locaux et globaux, etc. la gestion de ces informations et de ces mécanismes est parfois complexe et nécessite une charge supplémentaire.

L'administrateur d'une base de données fragmentée doit assurer la transparence aux utilisateurs.

- La principale difficulté de la fragmentation horizontale réside dans le fait qu'une opération de mise à jour dans une relation de schéma global se traduit par plusieurs sous-opérations de mises à jour dans différents fragments. Il faut donc identifier les fragments concernés, puis décomposer l'opération initiale en un ensemble d'opération sur ces fragments. Après l'opération de mise à jour certains n-uplets devront changer de fragments (migration de n-uplets). Cette opération de migration peut être coûteuse.

La fragmentation semble être une bonne technique pour l'amélioration des performances dans les entrepôts de données connus pour leur taille volumineuse. Oracle dans sa version Oracle8-i-offre une option de fragmentation (partitioning option) permettant de fragmenter un schéma d'un entrepôt de données, ses vues matérialisées, et ses index.

4.7 LES ALGORITHMES DE LA FRAGMENTATION VERTICALE

Le problème de la fragmentation verticale est de déterminer comment partitionner une relation (ou une classe) en fragments, dans le but de maximiser la performance du système. Dès le années 70, de nombreux travaux ont été proposés pour définir les fragments verticaux d'un fichier [M. J. Eisner, 1976], d'une relation [Navathe et al, 1984] , [S. Ceri et al, 1981] [Hammer & Niamir, 1979] [Navathe & Ra, 1989], d'une classe [C.I. Ezeifeet al, 1995] , [L. Bellatreche et al, 1996] ou de bases de données parallèles définies avec le modèle NF2 (Non First Normal Form) [J-C. Nicolas, 1991].

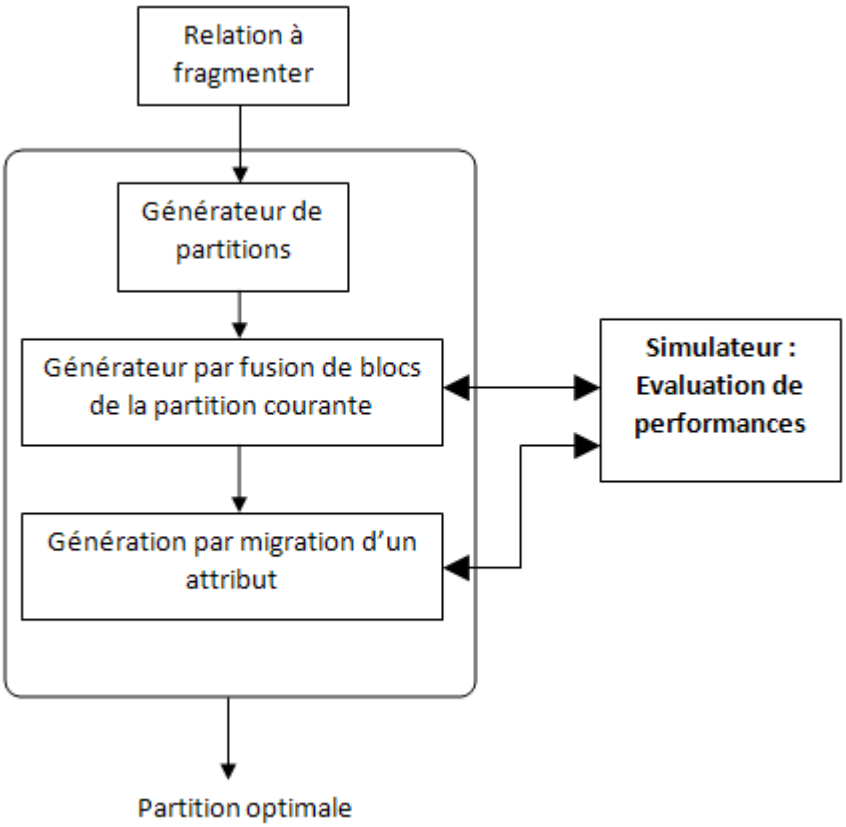


Figure. 4.3 Principe générale de la méthode de Hammer et Niamir

Sélectionner les fragments optimaux d'une relation est un *problème difficile* : en effet, une relation de *m* attributs peut être fragmentée en *B(m)* différentes manières [Ozsu et Valduriez, 1999] *B(m)* étant le nombre de Bell. Pour *m* grand, *B(m)* peut atteindre m^m .

Dans cette section, nous présentons deux travaux que nous considérons de base dans la littérature de la fragmentation verticale : **les travaux de Hammer et AL.** [Hammer & Niamir, 1979]et ceux de **Navathe et al.** [Navathe et al, 1984]

Travaux de Hammer et Niamir : les auteurs ont proposé une solution basée sur un modèle de simulation pour l'évaluation des performances présentée par le schéma après une fragmentation verticale. Les deux composante principale de cette méthodes sont : (1) **un générateur de partitions** et (2) **un évaluateur de performances** qui attribue une valeur de mérite à chaque partition (cf. Figure 4.3).

Le générateur de partitions fonctionne aussi selon un principe heuristique qui comprend deux étapes itératives. A chaque itération, le générateur construit un ensemble de variation autour de la fragmentation courante. Les variations sont ensuite soumises à l'évaluateur, et la frag-

mentation qui représente le plus faible coût est sélectionnée comme fragmentation courante de l'itération suivante.

Les différences entre les deux étapes concernent le choix de la fragmentation initiale et la génération des variations. Dans la première étape, la fragmentation initiale est constituée de fragments à un seul attribut. La fragmentation initiale de la deuxième étape est celle obtenue à l'issue de la deuxième étape. Dans la première étape, les variations sont générées en considérant toutes les fusions (deux à deux) possibles des blocs de la fragmentation courante. Dans la seconde, les variations sont obtenues par mouvement d'un seul attribut à la fois de son bloc actuel vers tous les autres blocs.

L'évaluation des performances se faits grâce à un estimateur de coûts de transactions qui simule le nombre de pages de la mémoire secondaire transférées pendant l'exécution des transactions. L'algorithme s'arrête dès qu'aucune opération n'est possible ou qu'aucune réduction du coût n'est constatée.

Les travaux de Navathe et al. Ces travaux ont approfondi ceux de Hoffer et al. [Hoffer & Severance, 1975] et Hammer et al. [Hammer & Niamir, 1979] Deux approches ont été proposées par Navathe et al. : le partitionnement binaire [Navathe et al, 1984] et le partitionnement graphique [Navathe & Ra, 1989].

La première approche suppose une relation ayant *m* attributs à fragmenter et un ensemble de *n* requêtes les plus fréquentes. Elle s'exécute en deux phases. Pendant la première phase, trois matrices sont construites : (i) la matrice d'usage des attributs, (ii) la matrice des affinités d'attributs, et (iii) la matrice d'affinité ordonnée.

- Dans ***la matrice d'usage des attributs*** (AUM pour Attribut Usage Matrix), la valeur de l'élément de la ligne *i* et la colonne *j* a pour valeur 1 si l'attribut *j* est accéder par la requête *i*, sinon cette valeur est nulle. Cette matrice est complétée par une valeur de fréquence d'accès pour chacune des requêtes représentée dans une colonne supplémentaire.

$$U(q_i, A_j) = \begin{cases} 1 & Si\ q_i\ utilise\ A_j \\ 0 & Sinon \end{cases}$$

Exemple : Une matrice d'usage d'attribut de 4 attributs et 4 requêtes

	A_1	A_2	A_3	A_4	$Fréquence$
q_1	1	0	1	0	75
q_2	0	1	1	0	50
q_3	0	0	0	1	25
q_4	0	1	0	1	25

- ***La matrice des affinités d'attributs*** a *m* lignes et *m* colonnes correspondant aux attributs de la relation à fragmenter. La valeur de l'élément de la ligne *i* et la colonne *j* reporte les valeurs d'affinité définies entre ces attributs. Cette affinité correspond à la somme des fréquences d'accès des requêtes accédant simultanément aux deux attributs.

  Exemple :

- ***Matrice d'accès*** $m \times l$: acc_{ik} : représente la fréquence d'accès de la requête *i* sur le site *k*
- ***Matrice de référence*** m× l : ref_{ik} : représente le nombre d'accès aux attributs pour chaque exécution de la requête q_i sur le site *k*

  Mesure d'affinité entre deux attributs A_i et A_j (aff_{ik}) d'une relation *R* est définie comme:

$$aff_{ij} = \sum \{k | U_{ki} = 1 \wedge U_{kj} = 1\} \sum_{l} acc_{lk}$$

$$U = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad acc = \begin{bmatrix} 15 & 20 & 10 \\ 5 & 0 & 0 \\ 25 & 25 & 25 \\ 3 & 0 & 0 \end{bmatrix} \qquad aff = \begin{bmatrix} 45 & 0 & 45 & 0 \\ 0 & 80 & 5 & 75 \\ 45 & 5 & 53 & 3 \\ 0 & 75 & 3 & 78 \end{bmatrix}$$

- La matrice d'affinité ordonnée est obtenue après l'application de l'algorithme de B.E.A. [W.T. McCormicket al, 1972] sur la matrice des affinités d'attributs. Par permutation de lignes et de colonnes, l'algorithme B.E.A. regroupe les attributs qui sont utilisés simultanément en fournissant une matrice sous la forme d'un semi-bloc diagonal.

La deuxième phase est la recherche des fragments verticaux qui optimisent une fonction objectif. Elle consiste à effectuer un partitionnement binaire récursif de la matrice d'affinité ordonnée. Cette étape revient à définir des sous-matrices sur la diagonale principale. Chaque sous-matrice regroupe un ensemble de valeurs proches et donc d'attributs fréquemment accédés simultanément. Chaque sous-matrice contient donc les attributs d'un fragment vertical.

Pour la seconde approche, Navathe et Ra [Navathe & Ra, 1989] ont présenté une méthode de regroupement des attributs basée sur les graphes. Cet algorithme part de la matrice des affinités d'attributs considérée comme un graphe complet, appelé graphe d'affinité. Les nœuds de ce graphe représentent les attributs de la relation à fragmenter, et les valeurs des arêtes représentent les valeurs d'affinités. Cet algorithme cherche à générer des cycles dont les arêtes pos-

sèdent des grandes valeurs d'affinités. Ces cycles sont appelés cycles d'affinités qui sont en fait des fragments potentiels.

Karlapalem et al. [K. Karlapalem et al, 1994] ont été les premiers à étudier les aspects de la fragmentation dans les modèles objets. Ces derniers se sont essentiellement concentrés sur les besoins des algorithmes de fragmentation et non sur les algorithmes en eux-mêmes. Ils définissent les concepts suivants : attribut simple, attribut complexe, méthode simple et méthode complexe.

Dans [K. Karlapalem et al, 1995], les autres définissent les concepts de fragments verticaux et horizontaux de classes. Un fragment-classe verticale d'une classe C est un sous-ensemble non vide de l'ensemble d'attributs de C et son ensemble d'objets est identique à C. Un fragment-classe horizontal d'une classe C possède une définition structurelle identique à celle de C et son ensemble d'objets représente un sous-ensemble non vide de C. Une représentation objet est proposée pour implémenter les fragments verticaux et horizontaux. Cette représentation facilite le support de la transparence de la fragmentation. Les algorithmes de partitionnement définis pour le modèle relationnel [Navathe & Ra, 1984], [Navathe & Ra, 1989] peuvent être adaptés pour les bases de données objet.

Toutefois aucune précision n'est donnée à ce sujet.

Ezeife et Barker [C.I. Ezeifeet al, 1995] ont suggéré des algorithmes pour les fragmentations verticale et horizontale. Se basant sur les concepts de Karlapalem et al. [K. Karlapalem et al, 1994]ils définissent les attributs simples (le domaine de l'attribut est atomique) et complexes (le domaine de l'attribut est une autre classe) ainsi que les notions méthodes simples (elles n'invoquent pas d'autres méthodes) et complexes (elles invoquent d'autres méthodes). Les algorithmes proposés sont basés sur les affinités entre les méthodes définies dans les requêtes.

La construction des fragments suit la démarche suivante :

Les méthodes sont réparties en groupes en utilisant le concept d'affinité entre méthodes et en reprenant les algorithmes développés dans [Navathe & Ra, 1984], Cette première étape permet de construire la matrice d'usage pour chaque classe. Les lignes et les colonnes de cette matrice représentent les requêtes et les méthodes spécifiques à la classe C, respectivement.

Les groupes de méthodes sont complétés par les attributs manipulés par ceux-ci. Si un même attribut est manipulé par plusieurs groupes, il est placé dans le groupe ave lequel il a le plus d'affinité.

Un ensemble d'algorithmes pour la fragmentation horizontale pour quatre modèles de classe est proposé : (1) classes d'attributs simples et méthode simples, (2) classes d'attributs complexes et méthodes simples, (3) classes d'attributs complexes. Ils reprennent les principes de

l'algorithme de Ozsu et al. [Ozsu et Valduriez, 1991] avec la même notion de prédicats simples. Leur complexité est exponentielle en nombre de prédicats simples.

4.8 LA FRAGMENTATION DANS LES ENTREPOTS DE DONNEES

Peu de travaux ont apporté une solution à la fragmentation dans les entrepôts de données, bien que ces derniers contiennent de très grandes tables.

Wu et al. [M-C. Wu et al, 1997] dans leur article « research isues in data warehousing » ont recommandé l'utilisation de la fragmentation verticale et horizontale dans les entrepôts de données pour améliorer l'évaluation des requêtes en évitant le balayage des grandes tables. Il a précisé que les algorithmes développés dans les bases de données réparties doivent être adaptés et réévalués pour les entrepôts de données.

Deux cas d'utilisation de la fragmentation dans les entrepôts sont considérés :

– Une table des faits peut être partitionnée d'une manière horizontale en fonction d'une ou plusieurs table(s) de dimension. En d'autres termes, la table des faits peut être fragmentée en utilisant la fragmentation dérivée.

– Une table des faits peut également être fragmentée verticalement ; toutes les clés étrangères de la table des faits y sont partitionnées. Cependant la reconstruction des tables fragmentées verticalement nécessite l'opération de jointure, qui est très coûteuse.

Les auteurs [M-C. Wu et al, 1997] n'ont pas proposé d'algorithme de fragmentation.

Le concept de fragmentation verticale a été introduit dans la définition des index de projection dans les entrepôts par O'Neil et al. [P. O'Neil et al, 1997] Les index de projection ressemblent à un fragment vertical d'une relation.

Golfarelli et al. [M. Golfarelli, 1999] ont utilisé la fragmentation verticale pour partitionner les vues matérialisées. Pour eux, le terme de fragmentation verticale signifie deux choses : d'une part le partitionnement des attributs d'une vue en plusieurs fragments et d'autre part l'unification en une seule vue de deux ou plusieurs vues ayant une clé commune. L'unification correspond à la règle de reconstruction d'une table fragmentée à partir de ses fragments verticaux [Ozsu et Valduriez, 1991].

Pour ce qui est de la fragmentation horizontale, peu de travaux ont été réalisés, excepté celui de Noaman et al. [A. Y. Noaman et al, 1999]. Ces auteurs ont proposé une technique de construction d'un entrepôt réparti en utilisant la stratégie descendante [G. Gardarin et al, 1990] Cette stratégie est couramment utilisée pour la conception de bases de données réparties. Elle part du schéma conceptuel global d'un entrepôt, qu'elle répartit pour construire les schémas

conceptuels locaux. Cette répartition se fait en deux étapes essentielles, à savoir, la fragmentation et l'allocation, suivies éventuellement d'une optimisation locale [S. Chakravarthy et al, 1994]. Les auteurs [A. Y. Noaman et al, 1999] se sont inspirés de ce qui avait été fait pour les bases de données relationnelles [Ozsu et Valduriez, 1991] Ils ont proposé une architecture pour entrepôt distribué, basée sur l'architecture ANSI/SPARK [D. Tsichritzis, 1978] et suggéré un algorithme adapté de Ozsu et al. [Ozsu et Valduriez, 1991] pour fragmenter la table des faits.

4.9 LA TRANSPARENCE DE LA FRAGMENTATION

Lorsqu'une relation est fragmentée en plusieurs fragments, le système doit assurer la transparence aux utilisateurs, c'est-à-dire cacher aux utilisateurs le fait que les données sont fragmentées. L'utilisateur ne manipule que des relations conceptuelles non partitionnées. Dès que l'utilisateur pose une requête, ces requêtes doivent être automatiquement transposées sur les fragments, en identifiant les fragments valides. Le même processus d'identification est réalisé lors des opérations de mise à jour. Ce problème a jusqu'à présent été peu pris en compte par les chercheurs.

4.10 LE GROUPEMENT (CLUSTRING)

Le groupement de données (Clustring dans la terminologie anglo-saxonne) consiste à stocker les données (n-uplets ou objets) logiquement liées (c'est-à-dire, susceptibles d'être utilisées fréquemment ensemble) proches les unes des autres en mémoire secondaire. L'avantage de cette technique est que lorsqu'un objet est chargé depuis le disque, tous les objets liés sont chargés en même temps en mémoire primaire. En conséquence, un groupement judicieux des objets minimise en fait non seulement le nombre de pages disques à lire, mais aussi le nombre d'objets non pertinents chargés en mémoire primaire.

Dans les systèmes relationnels, de telles optimisations sont principalement destinées à accélérer les opérations de jointure, grâce à la création de clusters [J. Darmont, 1999], Dans le cadre des bases de données objet, le groupement est plus complexe compte tenu des caractéristiques de modèles objet (lien de composition, hiérarchie d'héritage, etc.) [J. Darmont, 1999].

Dans les entrepôts, peu de travaux ont été réalisés sur le groupement. Un seul article traite le problème [H. Jagadish et al, 1999]. Il s'agit de Jagadish et al. qui ont considéré le problème de regrouper les enregistrements de la table des faits, afin de minimiser le nombre d'entrées-

sorties d'exécution d'un ensemble de requêtes. Sélectionner les clusters optimaux est un problème combinatoire compte tenu des hiérarchies définies sur les attributs des tables. Les auteurs ont proposé des heuristiques de groupement basés sur les courbes Z et les courbes de Hilbert.

Le problème de regroupement a été largement étudié dans le data mining. Le but de cette technologie est d'extraire des données pour permettre à une entreprise d'améliorer ses fonctions de soutien au marketing, aux ventes et au service client à travers une meilleure compréhension de ses clients. Parmi les techniques les plus utilisées dans le data mining nous trouvons le groupement, qui consiste à segmenter une population hétérogène en un certain nombre de sous-groupes plus homogènes, ou clusters. Les enregistrements sont, en fait, regroupés en fonction d'une similitude mutuelle. L'analyse des clusters est souvent préalable à une autre forme d'exploitation des données. Par exemple, pour étude de segmentation de marché, au lieu de chercher une règle comme « à quel type de promotion les clients répondent le mieux ? », il est possible de diviser la base de données client en clusters de personnes ayant les mêmes habitudes de consommation avant de poser la question pour chaque cluster.

Après avoir défini le concept de groupement, nous avons quelques remarques à présenter :

- La fragmentation et le groupement sont orthogonaux, le premier est réalisé pendant la conception logique de l'entrepôt et le deuxième pendant la conception physique.
- Les deux concepts ont le même objectif, à savoir regrouper les objets afin de minimiser le nombre des entrées-sorties.
- Les fragments peuvent être considérés comme des unités de groupement, si tous le n-uplets d'un fragment sont stockés de façon contigüe sur le disque.

4.11 CONCLUSION

Dans ce chapitre, nous avons présenté le problème de la fragmentation verticale. Cette technique de conception des bases de données à pour but l'optimisation des requêtes ou transactions effectuées sur la base de données, c'est-à-dire la minimisation des entrés-sorties dans l'exécution des requêtes et par conséquence l'amélioration des performances de la base de données. La technique de la fragmentation verticale consiste à fragmenter ou diviser une relation globale à un ensemble de fragments ou sous-relations qui correspondent de près aux exigences des transactions de l'utilisateur, les temps d'accès pour les opérations seront réduits de manière significative.

Le problème de la fragmentation verticale est un problème difficile, est un problème NP-difficile dans la plupart des cas. Donc il n'existe pas un algorithme efficace pour le résoudre d'une manière exacte, le recoure à une métaheuristique est inévitable.

Dans le chapitre suivant on présentera une application d'un algorithme à base d'optimisation par essaim particulaire comme approche de résolution du problème de la fragmentation verticale.

Chapitre 5

Un nouvel Algorithme à base d'essaims particulaires pour la fragmentation verticale

Table des matières

5.1 INTRODUCTION

La fragmentation verticale (Attribute partition en anglais) est une technique importante dans la conception des bases de données volumineuses (exemple : les entrepôts de données), et pour les bases de données distribuées, elle est utilisée pour améliorer les performances, c'est-à-dire minimisé le temps d'exécution des requêtes des utilisateurs de la base de données. La fragmentation des données est un processus de conception dans laquelle une relation est partitionnée de sorte que la plupart des transactions accèdent à un sous-ensemble de fragments, et, par conséquent, la performance de la base de données est augmentée.

Trois types de fragmentation existent :

a) *la fragmentation verticale*, qui traite avec l'affectation d'attributs à plusieurs fragments,
b) *la fragmentation horizontale*, qui traite avec le partitionnement des tuples de la table qui répondent à une condition sur les valeurs d'attributs, et
c) *la fragmentation mixte*, combinant à la fois fragmentations verticale et horizontale.

Les algorithmes de la fragmentation verticale contiennent deux parties essentielles :

1) *La méthode d'optimisation* et
2) *La fonction objectif*.

Ozsu et Valduriez [Ozsu et Valduriez, 1999] argumentent que la recherche du meilleur plan de partitionnement pour une relation avec *m* attributs par recherche exhaustive doit comparer au moins le M^{eme} *nombre de Bell* de fragments, ce qui signifie qu'un tel algorithme a une complexité de O (m^m). Ainsi, il est plus logique de chercher des heuristiques pour trouver les solutions optimales.

D'autre part, la fragmentation dans les bases de données vise à renforcer les opérations de traitement dans la base de données. *La fonction objectif* évalue si un tel objectif est atteint. Presque tous les algorithmes précédents ont ces deux éléments discernables c'est-à-dire méthode d'optimisation et fonction objectif. Toutefois, l'approche graphique développée par et

Navathe et al [Navathe et al, 1989] est une exception, elle ne possède pas une fonction objectif explicite.

La plupart des algorithmes précédents utilisent plusieurs itérations de la fragmentation binaire pour s'approcher à une fragmentation de plusieurs fragments. Navathe et al [Navathe et al, 1984] proposent un Algorithme récursif de partition binaire (RBPA), qui étend le travail de Hoffer et Severance [Hoffer & Severance, 1975] en automatisant le processus de sélection de la fragmentation verticale, ils proposent quelques fonctions objectives empiriques. Cornell et Yu [Cornell & Yu 1990] adoptent la même approche, mais ils ont remplacé les fonctions objectifs empiriques avec une fonction construite sur un modèle de base de données. Chu et Ieong [Chu & Ieong, 1992] adoptent les transactions dans leur algorithme, mais il reste un partitionnement binaire. Des efforts ont également été faits pour l'utilisation d'autres techniques d'optimisation pour la fragmentation verticale. Hammer et Niamir [Hammer & Niamir, 1979] propose une méthode qui utilise *HillClimbing* algorithme qui alternativement groupe et regroupe les attributs et les fragments pour atteindre solution sous optimale. Gorla et Song [Song & Gorla 2000] utilisent les algorithmes génétiques. Toutefois, chaque exécution de leur algorithme est seulement une partition binaire. Enfin, Jun du et Reda Alhajj et Ken Barker [Jun du et al, 2006] utilisent aussi un algorithme génétique GRGS-GA'Group oriented Restricted Growth String GA) pour résoudre lnt problème de la fragmentation verticale, ils propssent un ensemble de opérateurs génétiques orientés groupe' ces opérateurs dédies spécialement pour travailler avec les problèmes de groupent et une solution pure d'algorithme g-énétique, c'est-à-dire le résultat direct de l'exécution de l'algorithme génétique est une fragmentation à plusieurs fragments (m-way partition) non une fragmentation binaire.

Dans ce chapitre on propose une nouvelle approche pour la résolution du problème de la fragmentation verticale en utilisant l'optimisation par essaim particulaire pour le problème de la fragmentation verticale.

L'optimisation par essaims particulaires est une métaheuristique récente d'optimisation, inventée par Russel Eberhart et James Kennedy en 1995. À cause de ses capacités d'exploration globale et locale, sa simplicité d'implémentation et ses performances. OEP est largement utilisée. OEP est originalement proposée pour l'optimisation continue, beaucoup de chercheurs ont essayé de l'utiliser pour l'optimisation combinatoire. Le travail décrit dans le présent chapitre considère le problème de la fragmentation verticale et report à une application d'une version combinatoire de l'algorithme d'optimisation par essaim de particule qui donne une solution directement sous forme d'une fragmentation à plusieurs fragments (m--

way partition) non une fragmentation binaire comme les algorithmes proposés précédemment et en utilisant restrictif grth string [Ruskey, 1993] qui permet d'éliminer la redondance de codage dans la représentation de la solution. Ce chapitre est structuré comme suit. Dans la section 2 on introduit l'évaluateur de partition (PE) qui a été mise au point par Chakra-varthy, Muthuraj, Varadarjan, et Navathe [Chakravarthy et al, 1992], cet évaluateur sera utilisé comme fonction objectif pour l'algorithme utilisé. Dans la section 3 on commence avec un bref aperçu général sur RG chaînes, dans la section 4 on donne une brève présentation de l'optimisation par essaim de particule. Dans la section 5 on présente l'application de l'algorithme d'optimisation par essaim de particule pour le problème de la fragmentation verticale. Section 6 démontre que l'OEP trouve la solution optimale dans tous les cas basant sur des expérimentations et deux études de cas. Enfin on introduit un résumé et une conclusion dans la section 7.

5.2 FONCTIONS OBJECTIFS POUR LA FRAGMENTATION VERTICALE :

Il existe deux types de fonctions objectifs utilisées dans les algorithmes de la fragmentation verticale :

1) Fonctions utilisant un modèle de coût fondées sur l'analyse de la méthode d'accès de la transaction pour un modèle de SGBD donné.
2) Les fonctions fondées sur une hypothèse empirique.

Le premier type est spécifique à un SGBD spécifique alors que le second est plus général et intuitif. En plus de la matrice d'usage d'attributs (AUM) utilisée comme entrée pour les deux types de fonctions objectifs, les fonctions basées sur un modèle de coût prend en compte le plan d'accès choisi par l'optimiseur de la requête, par exemple, la méthode de jointure et le type de balayage sur la relation de chaque type de transaction. Sans ce complément d'information, la fonction empirique montre l'évolution du coût affecté par le processus de fragmentation. Toutefois, la fonction empirique est utile pour la conception logique d'une base de données lorsque les informations sur les paramètres physiques ne sont pas disponibles. Bien que moins précis que les fonctions basées sur un modèle de coût, ils peuvent être très efficaces dans la comparaison de différentes techniques d'optimisation utilisée par les algorithmes.

Dans l'approche proposée, nous utilisons une fonction objectif empirique, une version modifiée de l'évaluateur de partitions (PE) proposée par Chakravarthy et al. [Chakravarthy et

al, 1994] Cet évaluateur de partition utilise le critère d'erreur carrée (Squar Error) communément appliqué dans les stratégies de groupement (Clustring). Nous avons donc appelé le Square-Erreur partition Evaluator (SEPE).

Le SEPE se compose de deux facteurs principaux de coûts : Coût d'accès *des attributs locaux non intéressants* (*irrelevant local attribute access cost*) et le coût d'accès des *attributs distants intéressants* (relevant remote attribute access cost). Ils représentent le coût supplémentaire nécessaire, autres que le coût minimal idéal. En outre, le coût idéal est le coût lorsque les transactions accèdent seulement à un seul fragment et n'ont pas à des instances d'attributs non intéressants dans ce fragment. Les deux coûts sont calculés en utilisant l'erreur carrée résultant, ils sont désignés par E_M^2 et E_R^2, respectivement. Pour plus de détails sur SEPE, y compris la formule, peut être trouvée dans Chakravarthy et al. [Chakravarthy et al, 1994].

5.3 PRINCIPES DE BASE DES CHAÎNES DE CARACTERES A CROISSANCE RESTREINTE (RESTRICTED GROWTH STRINGS).

Le codage sous forme de chaînes de caractères à croissance restreinte (RGS) représente une solution de groupement comme tableau d'entiers, noté par *a[n]*, où *n* est le nombre d'attributs dans la relation. Les éléments du tableau peuvent être des valeurs entières allant dans l'intervalle de 1 à *n*. les solutions de groupement pour être une chaîne RG, elles doivent satisfaire la Définition 1. En plus de la définition formelle des chaînes de caractères RG, d'autres définitions concernant les chaînes RG présentées :

- **Définition 1 :** une chaîne RG *r* est une séquence d'entiers représentés comme un tableau, qui remplit l'inégalité suivante :

$$\boldsymbol{r[i] \leq (max(r(0), r[1], \dots, r[i-1]) + 1), 0 < i < n, r[0] = 1}$$

Par exemple, {1 1 2 3 1 1 2 4} est une chaîne RG, mais {4 4 2 3 4 4 2 1} n'est pas une chaîne RG, par ce qu'elle représente la même solution dans un système de codage aléatoire.

- **Définition 2 :** Le degré d'une chaîne RG r est la plus grande valeur dans r, noté d (r). Par exemple considérant r = {11123221} alors d(r) = 3.
- **Définition 3 :** le i^{eme} préfixe de la chaîne RG r, noté p_r^i , est la sous-chaîne qui inclut les i premières valeurs de r .
  Par exemple considérant r = {11123221} alors p_r^4 = {1112}.

5.4 LE RECTIFICATEUR

Le rectificateur est une question clé dans l'approche proposée pour que chaque particule soit une chaîne RG. Toutefois, l'initialisation aléatoire des particules ne garantit pas que chaque particule soit une chaîne RG. Aussi les deux opérations de croisement et de mutation dans l'algorithme génétique et l'opération de mise à jour des positions des particules peuvent changer la représentation d'un chromosome ou de particule de la population d'une manière qui viole la contrainte de chaîne RG comme illustré dans l'exemple 1. Pour traiter de tels cas, nous introduisons une fonction de rectification pour garantir que chaque particule soit représentée comme une chaîne RG. Pour les particules qui violent la contrainte de la chaîne RG, le rectificateur simplement scan une particule et la convertit en chaîne RG, en adaptant les positions de ses attributs.

Exemple 1 Considérons les deux chaînes RG r1 = {11123213} et r2 = {12134231}; et supposant un point de croisement se produit après la position 4. Les deux chromosomes sont changé à (11124231) et (12133213), mais la première n'est pas une chaîne RG. De même, une mutation à un seul point sur R1 à la position 4 peut produire le chromosome (11133213), qui n'est pas une chaîne RG.

Pour éviter de casser la contrainte, on peut ajouter des règles de sorte que la mutation du r1 à la position 4 dans l'exemple 1 est rejetée parce que changer l'élément à tout autres nombre que 2 entraînerait une violation de la contrainte de chaîne RG. Cependant, l'obligation d'utilisation des chaînes RG limite la diversité dans la population. La solution la plus simple qui s'avère être la plus puissante : Après chaque opération, nous utilisons le rectificateur pour veiller à ce que chaque particule soit représentée comme une chaîne RG. Le rectificateur employé implémente l'algorithme suivant à l'aide d'une table de hachage dans le processus pour stocker les valeurs à utiliser dans l'échange de gènes.

Algorithme Rectifier :

Entrée : Une chaîne *a* représenté dans un tableau de *n* valeurs entières.

Résultats : La chaîne *a*, avec ses valeurs réorganisées pour faire tourner une la chaîne *a* en une chaîne RG.

```
int [] rectifier(int a[]){
    int degree=0;
    Integer b[]=new Integer[a.length];
    for(int i=0;i<a.length;i++){
```

```
        b[i]=Integer.valueOf(a[i]);
        }
Hashtable hashtable = new Hashtable();
for(int i=0;i<a.length;i++){

        if(hashtable.containsKey(b[i])){
        Integerj=(Integer) hashtable.get(b[i]);
                a[i]= j.intValue();
                }
        else{
                degree++;
                Integerh= Integer.valueOf(degree);
                hashtable.put(b[i] ,h);
                a[i]=degree;
                }
        }
return a;
}
```

5.5 UN CONSTRUCTEUR DENSE ET UN CONSTRUCTEUR CLAIR-SEME

Nous avons conçu deux constructeurs de particules différents pour le processus d'initialisation dans L'algorithme ICPSO basé sur les chaînes RG. Ils sont le *Constructeur Clairsemé* (SC) et le *Constructeur dense* (DC). Le SC génère une chaîne de caractères aléatoirement, cette chaîne est convertie en chaîne de caractère RG avec le rectificateur. Alors que le DC génère des particules où la contrainte des chaînes RG est appliquée dès le début donc, pas de processus de rectification dans DC, parce que chaque gène est créé comme un entier aléatoire entre 1 et le *degré potentiel* élevé de sa position, conforme à la contrainte des chaînes RG.
Ces deux constructeurs prennent ses noms du fait que SC devrait normalement Être en mesure de créer une partition qui a plus de fragments que celle créé par DC. Les deux constructeurs utilisent une fonction aléatoire pour générer un entier. Toutefois, dans SC, chaque élément peut varier de 1 à n, où n est fixe et égal au nombre maximal de

Fragments prévus par l'utilisateur. Dans DC, le premier élément est fixé à 1 et la limite supérieure de chaque élément augmente progressivement, et ceci atteint rarement n-1.

Algorithme du Constructeur Dense (DC)
Début
Création d'un essaim de *n* particules
Pour (chaque particule dans l'essaim) *faire*
Pour (tout attribut dans le vecteur de position de chaque particule) *faire*
Essaim. Particule(*i*).posittion.attribut(*j*) = un nombre aléatoire entre (1 et *m*)
// *m* est le nombre de maximal de fragments
Fin pour
Fin pour
Fin Début

Algorithme du Constructeur Clairsemé (SC)
Début
Création d'un essaim de *n* particules
Variable *max* =1
Pour (chaque particule dans l'essaim) *faire*
Attribut(0)=1
Pour (tout attribut dans le vecteur de position de chaque particule) *faire*
Attribut(j) = un nombre aléatoire entre (1 et max+1)
Si (l'attribut(j)>max) *alors*
Max =Attribut(j)
Fin Si
Fin pour
Fin pour

Fin Début

5.6 OPTIMISATION PAR ESSAIM PARTICULAIRE OEP

L'OEP présenté par Kennedy et Eberhart [Kennedy et Eberhart, 1995] est une des plus récentes métaheuristiques, qui a été inspirée par le comportement d'essaimage des animaux et des comportements sociaux des humains. Les chercheurs ont constaté que la synchronie du comportement animal a été démontrée par le maintien de la distance optimale entre les différents membres et de leurs voisins. Ainsi, la vitesse joue le rôle important de l'adaptation de chaque membre à la distance optimale. En plus, les chercheurs ont simulé le scénario dans lequel les oiseaux recherchent l'alimentation et ils ont observé leur comportement social. Ils ont perçu que, pour trouver de la nourriture chaque membre de l'essaim détermine sa vitesse en fonction de deux facteurs, sa propre expérience et la meilleure expérience de tous les autres membres. Il est similaire au comportement humain pour la prise d'une décision, où les gens

prennent la décision en fonction de leurs propres expériences et les meilleures expériences des autres personnes autour d'eux.

5.7 L'ALGORITHME D'OEP

Les principes généraux de l'algorithme d'OEP sont les suivants :

- D'une façon similaire à la technique des algorithmes évolutionnaire, OEP maintient une population de particules, où chaque particule représente une solution potentielle à un problème optimisation.
- Supposant que *m* est la taille de l'essaim. Chaque particule *i* peut être représentée comme un objet avec plusieurs caractéristiques (position, vitesse,..). Et Supposons que l'espace de recherche est un espace à n-dimensions, alors l'ième particule peut être représentée par un vecteur à n-dimension; $X_i = \{x_{i1}, x_{i2}, \dots x_{in}\}$, et la vitesse , $V_i = \{v_{i1}, v_{i2}, \dots v_{in}\}$, Où i = 1, 2, ..., m.
- Dans OEP, chaque particule se souvient de la meilleure position qu'elle a visité jusqu'à ici, dénommée $P_i = \{p_{i1}, p_{i2}, \dots p_{in}\}$, et la meilleure position de la meilleure particule dans l'essaim, dénommée $G_i = \{G_{i1}, G_{i2}, \dots, G_{in}\}$.
- OEP est similaire à aux algorithmes évolutionnaires et, à chaque génération *t*, la particule *i* ajuste sa vitesse v_{ij}^t et sa position x_{ij}^t pour chaque dimension j, avec des multiplicateurs aléatoires qui sont : *la meilleure position personnelle* p_{ij}^{t-1} et *la meilleure position dans l'essaim* G_{ij}^{t-1}, En utilisant les Equations. (1) et (2), comme suit:

$$v_{ij}^t = v_{ij}^{t-1} + c1r1\left(p_{ij}^{t-1} - x_{ij}^{t-1}\right) + c2r2\left(G_{ij}^{t-1} - x_{ij}^{t-1}\right) \quad (1)$$

Et

$$x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t \quad (2)$$

Où c1 et c2 sont les constantes d'accélération et r1 et r2 sont des nombres réels aléatoires tirés de l'intervalle [0, 1]. Donc la particule vole dans l'espace des solutions possibles vers p_i^t et G_i^t tout en continuant à explorer de nouvelles régions dans l'espace de recherche. Ce mécanisme stochastique peut permettre d'éviter les optima locaux. Comme il n'y avait pas de véritable mécanisme de contrôle de la vitesse d'une particule, il est nécessaire d'imposer une valeur maximale Vmax sur elle. Si la vitesse dépasse ce seuil, elle a été fixée à Vmax, qui contrôle la distance maximale de voyage à chaque itération, afin d'éviter qu'une particule ne dé-

passe les bonnes solutions. L'algorithme d'OEP est terminé avec un nombre maximal de générations ou la meilleure position des particules de l'ensemble d'essaim ne peut pas encore être améliorée après un nombre suffisamment important de générations.

Le problème ci-dessus a été dressé en intégrant un paramètre de poids dans la vitesse de la particule. Ainsi, dans les dernières versions de l'OEP, les équations. (2) et (3) sont modifiées comme suit :

$$v_{ij}^{t} = \chi(\omega v_{ij}^{t-1} + c1r1(p_{ij}^{t-1} - x_{ij}^{t-1}) + c2r2(G_{ij}^{t-1} - x_{ij}^{t-1})) \quad (3)$$

Et

$$x_{ij}^{t} = x_{ij}^{t-1} + v_{ij}^{t} \quad (4)$$

ω est appelé l'inertie est utilisée pour contrôler l'impact de l'historique des vitesses sur la vitesse actuelle. En conséquence, le paramètre ω réglemente la relation entre les capacités d'exploration globales et locales de l'essaim. Une grande inertie facilite l'exploration globale, tandis qu'une petite valeur d'inertie facilite l'exploration locale. Une valeur appropriée pour la valeur d'inertie ω généralement donne un équilibre entre les capacités d'explorations globales et locales et, par conséquent, entraîne une réduction du nombre d'itérations nécessaires pour localiser la solution optimale. χ est un facteur de construction utilisé pour limiter la vitesse.

L'algorithme d'OEP a montré sa robustesse et l'efficacité pour résoudre des problèmes d'optimisation dans l'espace des nombres réels. Seul un petit nombre de recherches ont été menées pour étendre l'OEP à résoudre des problèmes d'optimisation combinatoire.

5.8 L'OEP POUR L'OPTIMISATION COMBINATOIRE

Dans ce travail, nous proposons une extension de l'algorithme d'OEP visant à résoudre un problème d'optimisation combinatoire. L'OEP combinatoire diffère essentiellement de l'OEP original (ou continue) dans certaines caractéristiques.

5.8.1 Définition d'une Particule

On associe à la solution $X_i^t = \{x_{i1}^t, x_{i2}^t, \dots, x_{in}^t\}$ un vecteur à n-dimensions noté par $Y_i^t = \{y_{i1}^t, y_{i2}^t, \dots, y_{in}^t\}$ qui prend des valeurs dans {-1, 0, 1} en fonction de l'état de solution de la particule *i* à l'itération *t*.

Y_i^t est une variable fictive utilisée pour permettre le passage de l'état combinatoire (discret) à l'état continu et vice versa.

$$Y_{ij}^{t} = \begin{cases} 1 & Si\ x_{ij}^{t} = G_j^{t} \\ -1 & Si\ x_{ij}^{t} = p_{ij}^{t} \\ -1\ Ou\ 1 & Si\ (x_{ij}^{t} = G_j^{t} = p_{ij}^{t}) \\ 0 & autrement \end{cases} \quad (5)$$

5.8.2 La vitesse de particule

Supposant $d1 = -1 - y_{ij}^{t-1}$ est la distance entre x_{ij}^{t-1} et la meilleure solution obtenue par la ième particule.

Supposant $d2 = 1 - y_{ij}^{t-1}$ est la distance entre x_{ij}^{t-1} et la meilleure solution obtenue dans l'essaim.

L'équation de mise à jour de la vitesse utilisée dans ICPSO est alors :

$$v_{ij}^{t} = w.v_{ij}^{t-1} + r1.c1.d1 + r2.c2.d2 \quad (6)$$

$$v_{ij}^{t} = w.v_{ij}^{t-1} + r1.c1.\left(-1 - y_{ij}^{t-1}\right) + r2.c2.(1 - y_{ij}^{t-1}) \quad (7)$$

Avec cette fonction, le changement de la vitesse v_{ij}^{t} dépend du résultat de y_{ij}^{t-1}.

Si $x_{ij}^{t-1} = G_j^{t-1}$, alors $y_{ij}^{t-1} = 1$. Par la suite d2 se transforme en « 0 » et d1 prend «-2», imposant ainsi à la vitesse de changer dans le sens négatif.

Si $x_{ij}^{t-1} = p_j^{t-1}$, alors $y_{ij}^{t-1} = -1$. Par la suite d2 se transforme en « 2 » et d1 prend « -0 », imposant ainsi à la vitesse de changer dans le sens positif.

Le cas où $x_{ij}^{t-1} \neq G_j^{t-1}$ et $x_{ij}^{t-1} \neq p_j^{t-1}$, y_{ij}^{t-1} tourne à « 0 », d2 est égale « 1 » et d1 est égale « -1 », par la suite les paramètres r1, r2, c1 et c2 permettront de déterminer le sens de la variation de la vitesse

Le cas où $x_{ij}^{t-1} = G_j^{t-1}$ et $x_{ij}^{t-1} = p_j^{t-1}$, y_{ij}^{t-1} prend une valeur dans {-1,1} imposant ainsi à la vitesse de changer dans le sens inverse du signe de y_{ij}^{t}.

5.8.3 La construction de la solution d'une particule

La mise à jour de la solution est calculée en y_{ij}^{t}:

$$\lambda_{ij}^{t} = y_{ij}^{t-1} + v_{ij}^{t} \quad (8)$$

La valeur de y_{ij}^{t} est ajustée selon la fonction suivante:

$$y_{ij}^{t} = \begin{cases} 1 & si\ \lambda_{ij}^{t} > \alpha \\ -1 & si\ \lambda_{ij}^{t} < -\alpha \\ 0 & autrement \end{cases} \quad (9)$$

La nouvelle solution est :

$$x_{ij}^{t} = \begin{cases} G_j^{t-1} & Si\ y_{ij}^{t} = 1 \\ p_{ij}^{t-1} & Si\ y_{ij}^{t} = -1 \\ un\ nombre\ aléatoire & Autrement \end{cases} \quad (10)$$

Le choix précédemment obtenu pour l'affectation d'une valeur aléatoire dans {1, -1} pour y_{ij}^{t-1} dans le cas d'égalité entre x_{ij}^{t-1}, p_{ij}^{t-1} et G_j^{t-1} permet d'assurer que la variable y_{ij}^{t} prend une valeur 0, et de permettre un changement dans la valeur de la variable x_{ij}^{t} . Nous définissons un paramètre α pour le montage intensification et la diversification. Pour une petite valeur de α, x_{ij}^{t} prend l'une des deux valeurs p_{ij}^{t-1} ou G_j^{t-1}ou (intensification). Dans le cas contraire, nous imposons à l'algorithme d'attribuer une valeur nulle à y_{ij}^{t} , induisant ainsi pour x_{ij}^{t} une valeur différente de p_{ij}^{t-1} et G_j^{t-1} (diversification). Les paramètres c1 et c2 sont deux paramètres liés à l'importance des solutions p_{ij}^{t-1} et G_j^{t-1} pour la génération de nouvelle solution X_i^t, ils ont également un rôle dans l'intensification de la recherche.

5.9 UN ALGORITHME DISCRET A BASE D'OEP POUR LE PROBLEME LA FRAGMENTATION VERTICALE

Dans cette section, nous présentons une application de l'algorithme proposé nommé ICPSO (Improved Combinatorial Particle Swarm Optimization) pour résoudre le problème de la fragmentation verticale.

5.9.1 La représentation de la solution

Dans ce paragraphe, nous décrivons la formulation de l'algorithme ICPSO pour le problème de la fragmentation verticale.

Une des questions les plus importantes lors de la conception des algorithmes d'OEP repose sur la représentation de sa solution. Dans la solution proposée l'espace de recherche est de n-dimension où chaque dimension représente un attribut de la relation à fragmenter et une particule $X_i^t = \{x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t\}$ correspond à l'affectation des n attributs à des fragments, tels que $x_{ij}^t \in \{1, 2, 3, \ldots, k\}$ où k est le nombre de fragments.

Le schéma de la Figure. 5.1 illustre un exemple de la représentation de la solution de particule X_i^t de l'algorithme ICPSO.

Supposant on une relation à n attributs et n=6 attributs

Et k= 3 fragments

On a utilisé la représentation de la solution sous forme d'une chaîne à croissante restreinte RG pour éliminer la représentation redondante par exemple, r1 ={1 1 2 3 1 1 2 4} est une chaîne RG, mais r2 ={4 4 2 3 4 4 2 1} n'est pas une chaîne RG, par ce qu'elle représente la même solution dans un système de codage aléatoire. Donc r2 est une représentation redondante de r1. L'utilisation de la représentation sous forme de chaîne RG va minimiser l'espace de recherche par l'élimination des solutions redondantes.

	X_{t1}^t	X_{i2}^t	X_{i3}^t	X_{i4}^t	X_{i5}^t	X_{i6}^t
X_i^t	1	2	1	3	3	2

Figure 5.1 Un exemple de représentation d'une solution

p_i^{t-1}	2	1	3	2	3	1
G^{t-1}	3	2	3	1	2	2
X_i^{t-1}	1	1	1	2	3	2

Nous obtenons le vecteur valeur y_i^{t-1} en utilisant l'équation (5):

y_i^{t-1}	0	1	0	-1	-1	1

Le vecteur de vitesse est calculé avec l'équation (6):

V_i^t	-1.2	-0.9	0.5	0.7	0.9	0.3

La nouvelle valeur de $\lambda_{ij}^t = \gamma_{ij}^{t-1} + \gamma_{ij}^t$

λ_i^t	-1.2	0.1	0.5	-0.3	-0.1	1.3

Supposant que $\alpha = 0.35$ γ_{ij}^t est déterminé en utilisant l'équation (9)

λ_i^t	-1	0	1	0	0	1

En utilisant l'équation (10), le vecteur de la nouvelle solution peut être définie comme suit :

X_i^t	2	3	3	1	1	2

Figure 5.2 Création d'une nouvelle Solution

5.9.2 La population initiale

Un essaim de particules est construit en utilisant le Constructeur Dense (DC) où le Constructeur Clairsemé (SC), Le SC génère une chaîne de caractères aléatoirement, cette chaîne est convertie en chaîne de caractère RG avec le rectificateur. Alors que le DC génère des particules où la contrainte des chaînes RG est appliquée dès le début, donc pas de processus de rectification dans DC, parce que chaque gène est créé comme un entier aléatoire entre 1 et le degré potentiel élevé de sa position, conforme à la contrainte des chaînes RG.

Ces deux constructeurs prennent ses noms du fait que SC devrait normalement être en mesure de créer une partition qui a plus de fragments que celle créé par DC. Les deux constructeurs utilisent une fonction aléatoire pour générer un entier. Toutefois, dans SC, chaque élément peut varier de 1 à n, où n est fixe et égale au nombre maximal de Fragments prévus par l'utilisateur. Dans DC, le premier élément est fixé à 1 et la limite supérieure de chaque élément augmente progressivement, et ceci atteint rarement n-1.

5.9.3 Création d'une nouvelle solution

Avec l'exemple de la Figure 5.2, nous illustrons une création de la nouvelle solution fondée sur la solution X_i^{t-1}, Une meilleure solution pour la i^{ème} particules p_i^{t-1}, La meilleure solution dans la population G^{t-1}, Et la vitesse V_i^{t-1}.

5.10 IMPLEMENTATION ET RESULTATS EXPERIMENTAUX

Dans cette section nous appliquons l'algorithme d'optimisation par essaim de particule ICPSO développé dans les sections précédentes sur cinq cas d'expérimentation, les deux premiers cas sont largement utilisés dans la littérature sur le problème de la fragmentation verticale. Le premier cas d'expérimentation est une matrice d'usage de 10 attributs et le deuxième cas avec une matrice d'usage de 20 attributs largement discuté dans la littérature sur le problème de la fragmentation verticale. Les trois derniers cas sont des matrices de grande taille, et elles sont générées aléatoirement par le générateur pseudo aléatoire des matrices d'usage d'attributs.

Pour comparer l'algorithme ICPSO nous avons implémenté deux algorithmes génétiques, un s'appelle GRGS-GA (Group oriented Restricted Growth String GA) développé par Jun dû et al [Jun du et al, 2006], et un autre algorithme génétique traductionnel que nous avons l'appelé SGA pour Simple Genetic Algorithm. SGA utilise les opérateurs génétiques traditionnels. Croisement à un seul point, mutation uniforme et élitisme par tournoi.

Les algorithmes (ICPSO, GRGS-GA et SGA) ont été implémentés en Java. Toutes les expériences avec ICPSO et AG ont été exécutées sous Windows XP sur un PC de bureau avec un processeur Intel Pentium4, processeurs 3,6 GHZ.

Dans les algorithmes génétiques AG utilisés, chaque solution est codée comme une chaîne de caractères (appelé chromosome) de longueur n (où n est le nombre d'attributs de la relation à fragmenter) $i^{ème}$ élément de la chaîne indique le numéro de groupe attribué à l'attribut A_i. Chaque chaîne ou chromosome est considéré comme un individu. Une collection de ces *P* individus est appelée une population. À chaque itération, une nouvelle population de la même taille est générée à partir de la population actuelle au moyen de deux opérations de base sur les individus. Ces opérations sont *la sélection* et *la reproduction.* La reproduction consiste en croisement, le croisement entre deux chaînes, comme indiqué ci-dessus, est effectué en une seule position. C'est ce qu'on appelle comme un croisement à un seul point. Une opération de mutation introduit une nouvelle chaîne de caractères dans la population.

L'algorithme génétique Group oriented Restricted Growth String GA (GRGS-GA) développé par Jun du et al [Jun du et al, 2006] utilise des opérateurs génétiques orienté groupes, ces opérateurs crées spécialement pour résoudre les problèmes de groupement comme c'est le cas du problème de fragmentation verticale.

5.10.1 Configuration des paramètres

Une des clés de succès des métaheuristiques est la configuration optimale des paramètres. Pour que les algorithmes ICPSO, GRGS-GA et SGA donnent des bonnes performances de recherche est n'est possible que si l'on prend soin de configurer les algorithmes de façon appropriée. Malheureusement la théorie des métaheuristiques donne peu d'indications pour bien choisir les paramètres de configuration.

La configuration utilisée dans les études de cas est basée sur nos expériences et de la recommandation communément trouvée dans la littérature. Ces paramètres ne sont pas nécessairement les meilleures pour tous les exemples examinés ci-après, mais pour faciliter la comparaison entre les approches (OEP, GRGS-GA et SGA). Le tableau suivant présente les paramètres utilisés dans les expériences :

ICPSO (modèle GBest)	GRGS-GA	SGA
Taille de l'essaim = 200 Particules	Taille de la population =200 Chromosomes	Taille de la population =200 Chromosomes
L'inertie w=0.85	Type d'opérateur de croisement : group oriented crossover	Type d'opérateur de croisement : croisement un seul point.
	Taux de croisement =0.8	Taux de croisement =0.8
c1=0.3 et c2 =1.2	Type d'opérateur de mutation : merge mutation et jump mutation alternativement avec un taux de 1 :2	Type d'opérateur de mutation : Mutation uniforme
	Taux de mutation =0.01	Taux de mutation =0.01
	Mécanisme de sélection = sélection par tournoi avec 2-concurrents par tournoi	Mécanisme de sélection = sélection par tournoi avec 2-concurrents par tournoi
	Elitisme : copies d'élitisme = 8	Elitisme: copies d'élitisme = 8

Tableau 5.1 Configuration des paramètres des trois algorithmes

5.10.2. Cas 1: exemple de matrice de 10 attributs et 8 transactions

Dans ce cas d'expérimentation nous avons utilisé une matrice d'usage d'attributs (AUM) de 10 attributs et de 8 transactions comme montré dans le tableau ci-dessous. Cette matrice d'usage a été utilisée par Cornell and Yu, 1990, Navathe et al 1984, Suk-kyu Song et Narasimhaiah Gorla, 2000, J. Muthuraj et al 1993. J. Muthuraj et al ont trouvé que pour cette matrice d'usage la solution optimale de la fonction objectif est égale à 5820, et qui donne une fragmentation verticale optimale de 4 fragments {1 5 7} {2 3 8 9} {4 6 10}.

$T\backslash A$	1	2	3	4	5	6	7	8	9	10
$T1$	25	0	0	0	25	0	25	0	0	0
$T2$	0	50	50	0	0	0	0	50	50	0
$T3$	0	0	0	25	0	25	0	0	0	25
$T4$	0	35	0	0	0	0	35	35	0	0
$T5$	25	25	25	0	25	0	25	25	25	0
$T6$	25	0	0	0	25	0	0	0	0	0
$T7$	0	0	25	0	0	0	0	0	25	0
$T8$	0	0	15	15	0	15	0	0	15	15

Tableau 5.2 Matrice 10 attributs et 8 transactions

Dans la figure 5.3 montre l'évolution de la valeur de PE c'est-à-dire la fonction objectif, nous remarquant que la solution optimale est obtenue dans l'itération numéro 11 et qui correspond une valeur de fonction objectif égale à 5820.

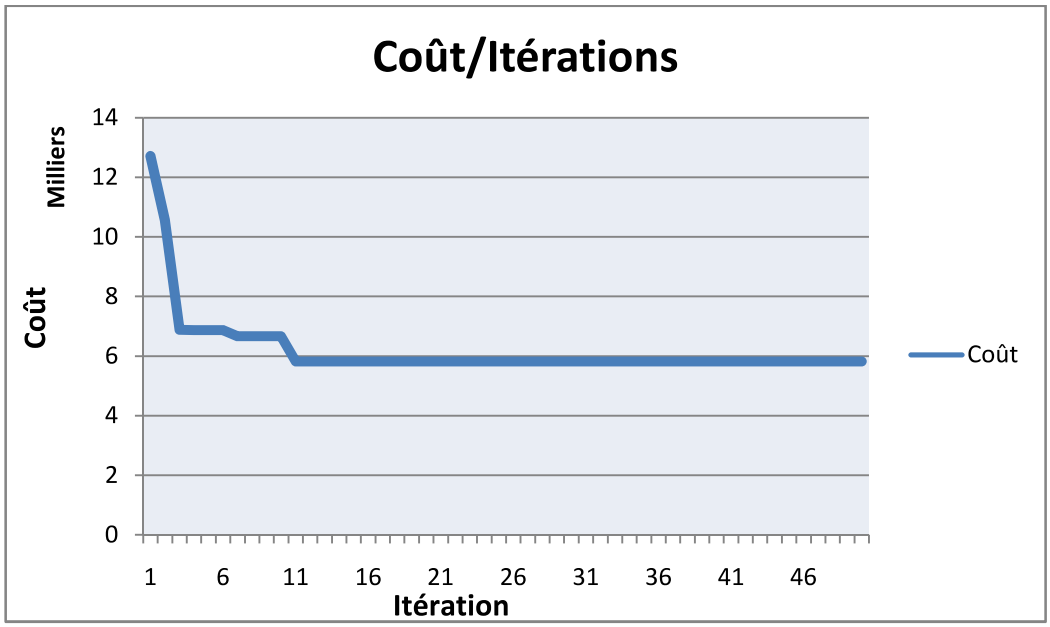


Figure 5.3 Graphe qui montre l'évolution de la solution optimale pour le Cas N°1

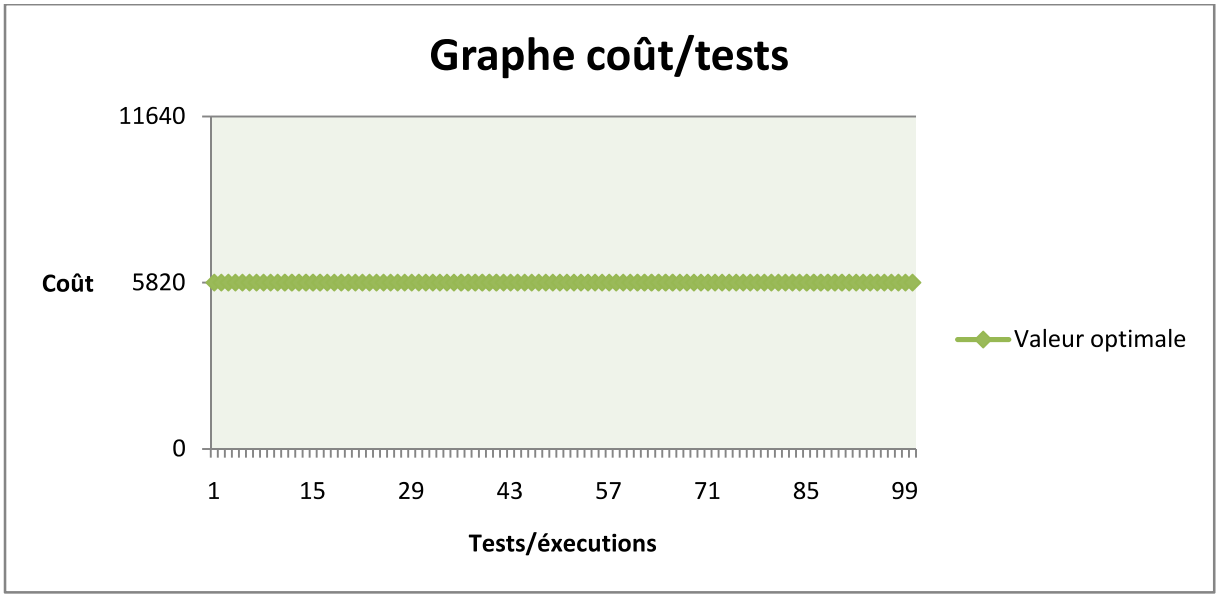


Figure 5.4 Graphe test-coût optimale pour l'exemple à 10 attributs

Pour tester la fiabilité de l'algorithme ICPSO à trouver la solution optimale, nous avons exécuté l'algorithme ICPSO 100 fois, comme le montre la figure 5.4. Dans tous les cas de test ICPSO trouve la solution optimale qui est égale à 5820 pour l'exemple de 10 attributs.

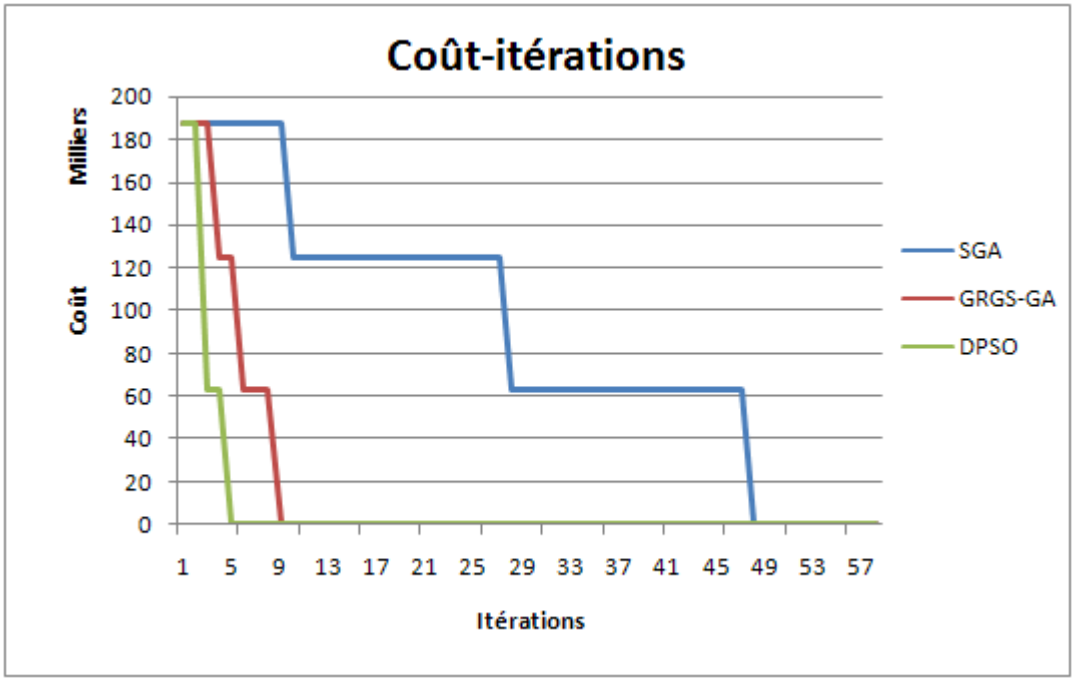


Figure 5.5 Comparaison de la convergence des trois algorithmes pour l'exemple à 10 attributs

La figure 5.5 montre que l'algorithme ICPSO a trouvé la solution optimale qui est zéro pour l'exemple utilisé dans l'itération numéro 4 et GRGS-GA a trouvé la solution optimale dans l'itération numéro 9 alors que SGA trouve la solution optimale dans l'itération 48. Donc ICP-SO converge plus rapidement que les deux autres algorithmes GRGS-GA et SGA.

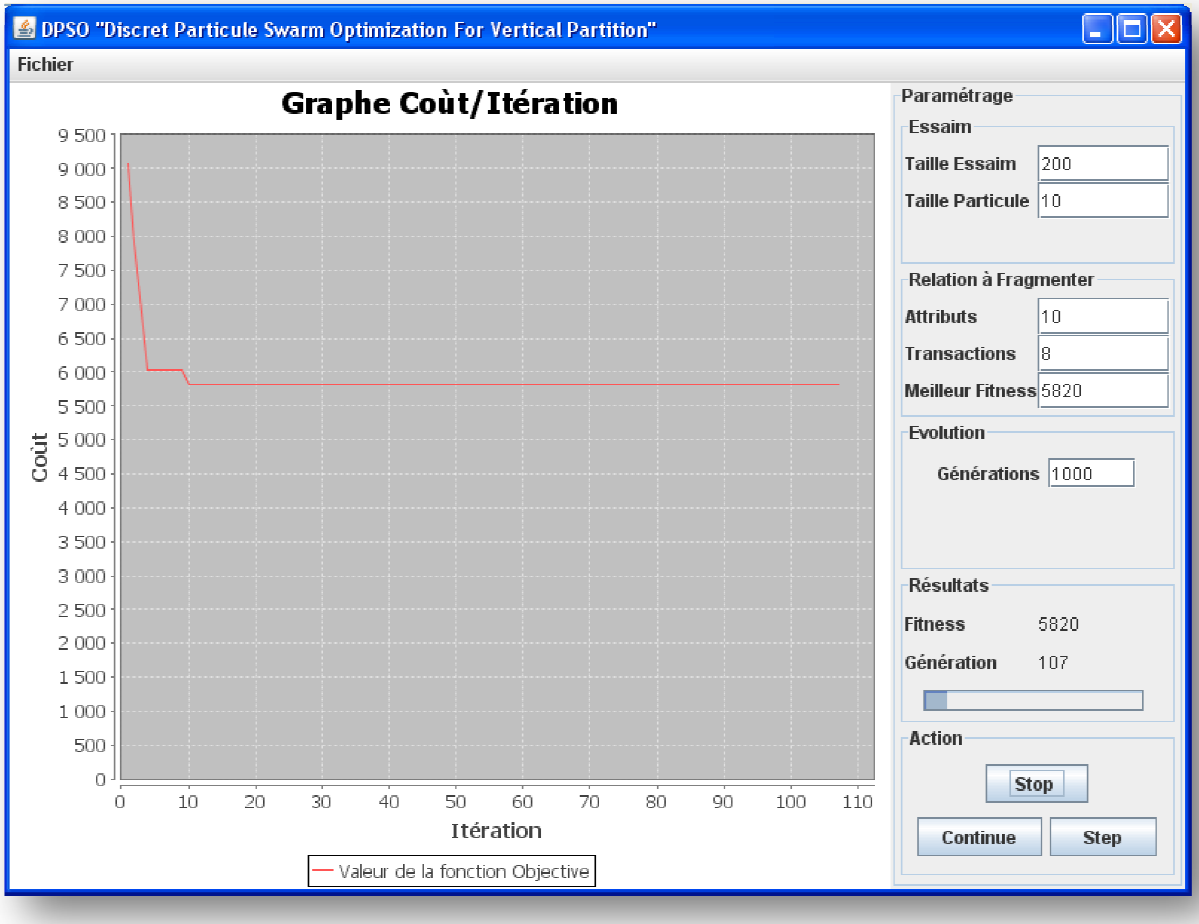


Figure 5.6 Interface du programme développé pour l'algorithme ICPSO exemple Cas N° 1

5.10.3. Cas 2: exemple de 20 attributs et 15 transactions

L'exemple de 20 attributs est déjà été utilisé par d'autres chercheurs tels qu'il est décrit dans Jun Du et al. (2006), Chakravarthy et al. (1992), Navathe et al. (1984), et Navathe et Ra (1989). La matrice d'usage utilisée dans cet exemple décrit 15 transactions qui accèdent à 20 attributs. Les méthodes proposées dans Navathe et al. (1984) et Chakravarthy et al. (1992) les deux trouvent la même fragmentation optimale qui groupe les 20 attributs en quatre fragments. En particulier, Chakravarthy et al. (1992) décide sur la base de la valeur de PE que cette fragmentation de quatre partitions est meilleure que celle de cinq partitions trouvée dans Navathe et Ra (1989).

T/A	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20
T1	50	0	0	50	50	50	0	50	0	0	0	0	0	0	0	0	0	0	0	0
T2	0	50	0	0	0	0	0	0	50	0	0	50	50	50	0	0	0	0	0	0
T3	0	0	50	0	0	0	50	0	0	50	50	0	0	0	0	0	50	50	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	50	0	0	50	50
T5	15	0	0	0	15	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0
T6	15	0	0	0	0	0	0	15	0	15	15	0	0	0	0	0	0	0	0	0
T7	0	0	15	0	0	0	15	0	0	15	15	0	0	0	0	0	15	0	0	0
T8	0	15	0	0	0	0	0	0	0	0	0	15	15	15	15	15	0	15	0	15
T9	0	10	0	0	10	0	0	0	0	0	10	0	0	10	0	0	0	0	10	0
T10	10	0	0	0	0	0	0	0	10	0	0	0	0	0	0	10	0	10	0	0
T11	10	10	10	0	10	10	0	0	10	0	0	10	10	0	0	0	0	0	0	0
T12	0	0	0	10	0	0	10	0	0	10	0	0	0	10	0	0	0	0	10	10
T13	0	0	0	0	0	0	0	10	0	0	10	0	0	0	10	10	10	10	0	0
T14	5	5	5	5	5	5	5	5	5	0	0	0	0	0	0	0	0	0	0	0
T15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	5	5	5	5	5

Tableau 5.3 Matrice 20 attributs et 15 transactions

Dans la figure 5.7 montre l'évolution de la valeur de PE c'est-à-dire la fonction objectif, nous remarquant que la solution optimale est obtenue dans l'itération numéro 51 et qui correspond une valeur de fonction objectif égale à 4644.

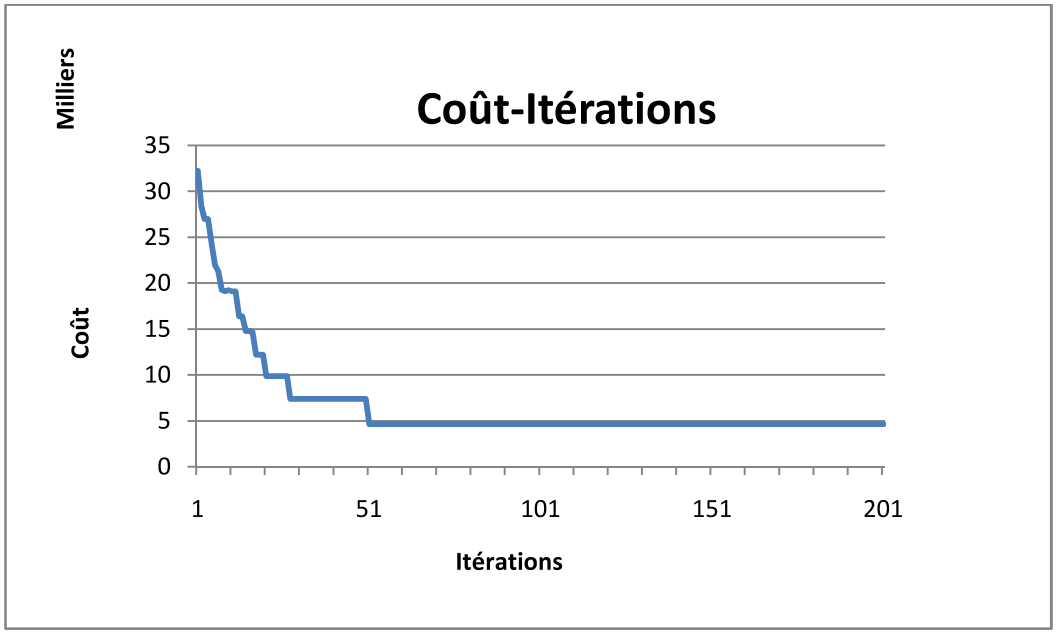


Figure. 5.7 Graphe qui montre l'évolution de la solution optimale pour le Cas N°2

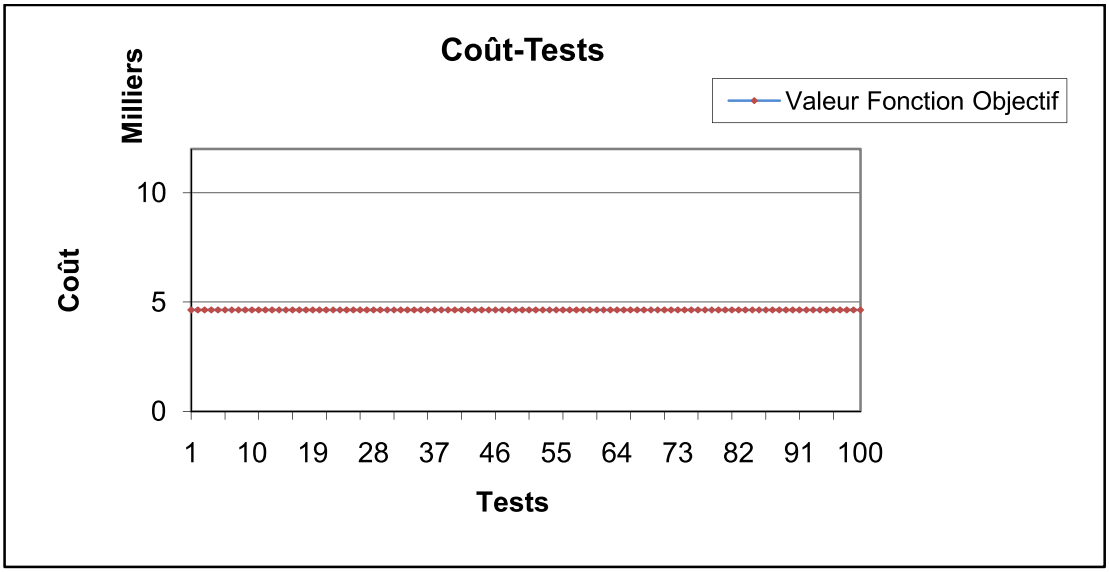


Figure. 5.8 Graphe test-coût optimale pour l'exemple à 20 attributs

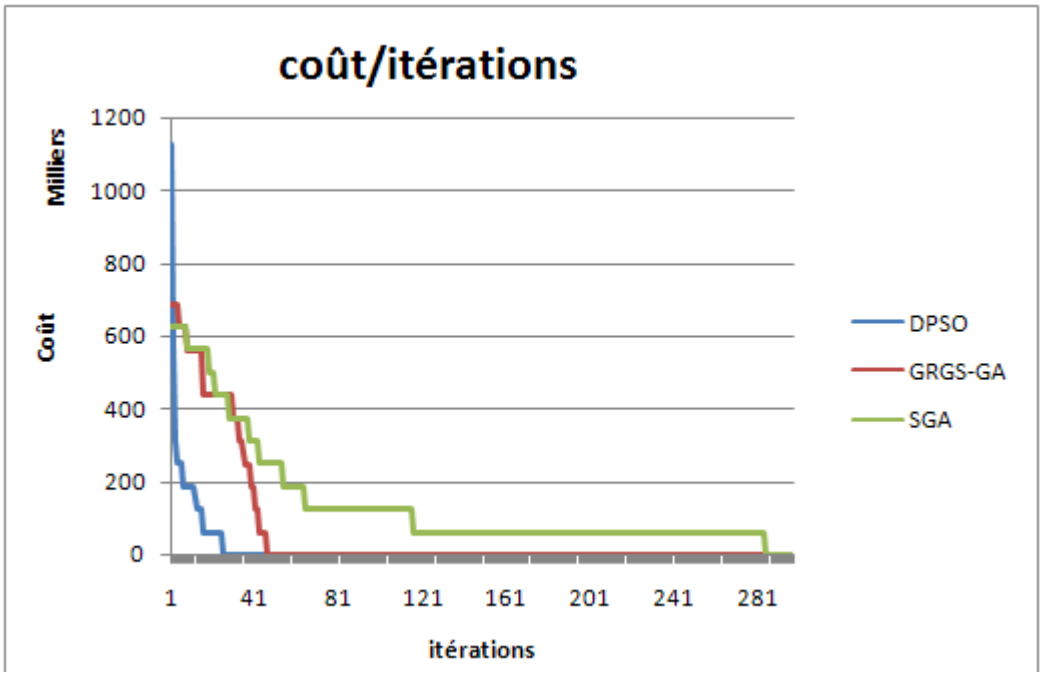


Figure. 5.9 Comparaison de la convergence des trois algorithmes ICPSO, GRGS-GA et SGA pour l'exemple à 20 attributs

La figure 5.9 montre que l'algorithme ICPSO a trouvé la solution optimale qui est zéro pour l'exemple utilisé dans l'itération numéro 31 et GRGS-GA a trouvé la solution optimale dans l'itération numéro 45 alors que SGA trouve la solution optimale dans l'itération 281. Donc ICPSO converge plus rapidement que les deux autres algorithmes GRGS-GA et SGA.

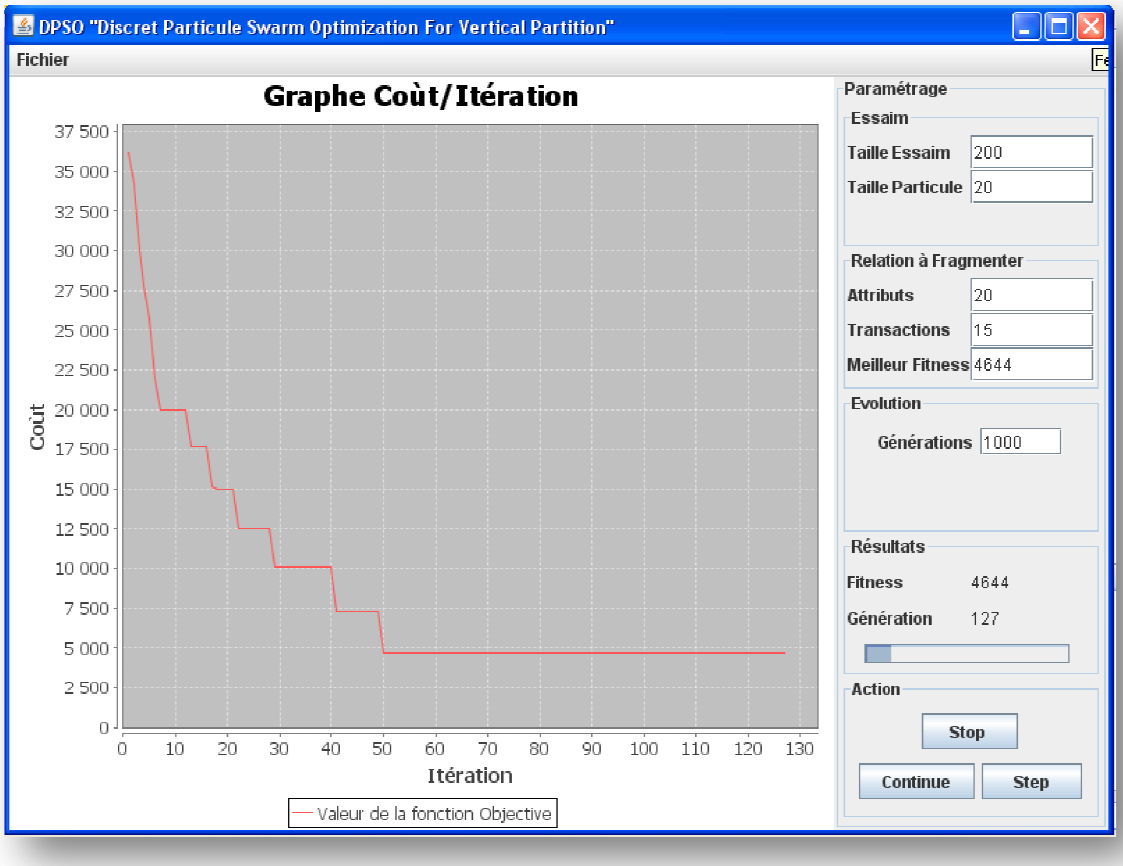


Figure. 5.10 Interface du programme développé pour l'algorithme ICPSO exemple Cas N° 2

5.10.4 Un générateur pseudo aléatoire des matrices d'usage d'attributs :

Pour tester l'algorithme ICPSO sur des cas de grande taille nous avons conçu un générateur pseudo aléatoire des matrices d'usage d'attributs AUM. Les paramètres de ce générateur sont : la taille de la matrice d'usage d'attributs, c'est-à-dire le nombre d'attributs et le nombre de transaction impliqués dans la conception des partitions ; et le nombre optimal des fragments nécessaires.

Dans notre matrice d'usage pseudo aléatoire, il doit y avoir *n* lignes décrivant *n* transactions spéciales. En outre, chaque transaction accède aux attributs d'un seul fragment et ne doivent pas accéder au même attribut accédé par une autre transaction spéciale. Aussi, les transactions doivent avoir des fréquences d'exécution dominantes pour s'assurer que la fragmentation optimale correspond au nombre de fragments prédéfinit dans le paramétrage du générateur.

L'AUM montrée dans le tableau 5.3 est une matrice usage de 10 attributs produite par le générateur pseudo aléatoire, il existe trois groupes.

Il est trivial de vérifier que la meilleure fragmentation pour les 10 attributs est {(1, 5, 7), (2, 4,6), (3, 8, 9, 10)}

Ref	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Freq
T1	1	0	0	0	1	0	1	0	0	0	75
T2	0	1	0	1	0	1	0	0	0	0	75
T3	0	0	1	0	0	0	0	1	1	1	75
T4	1	0	0	1	1	1	1	1	0	1	10
T5	0	0	0	1	0	1	0	1	0	0	10
T6	0	0	0	0	0	1	1	0	1	1	15
T7	1	0	1	1	0	0	0	0	1	0	5
T8	0	1	0	0	0	1	0	1	1	0	10

Tableau 5.4 Matrice 10 attributs et 8 transactions créée par le générateur pseudo aléatoire

5.10.5. Cas 3: exemple de 30 attributs et 30 transactions

Dans ce cas, nous essayons d'appliquer l'algorithme ICPSO sur une matrice d'usage d'attributs de grande taille générée par le générateur pseudo aléatoire. Cette matrice a 30 attributs et 30 transactions, le nombre optimal de fragments choisi est 30 fragments qui correspond à une valeur de la fonction objectif de 0. La figure 5.11 montre la convergence de l'algorithme ICPSO.

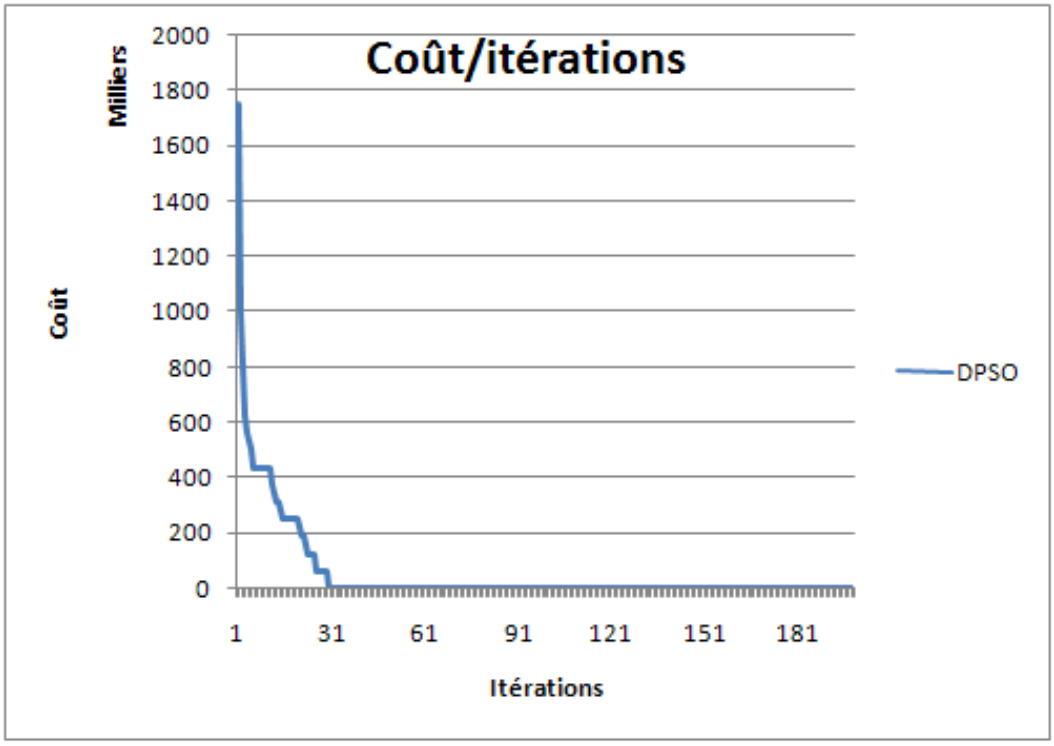


Figure. 5.11 Graphe qui montre l'évolution de la solution optimale pour le Cas N°3

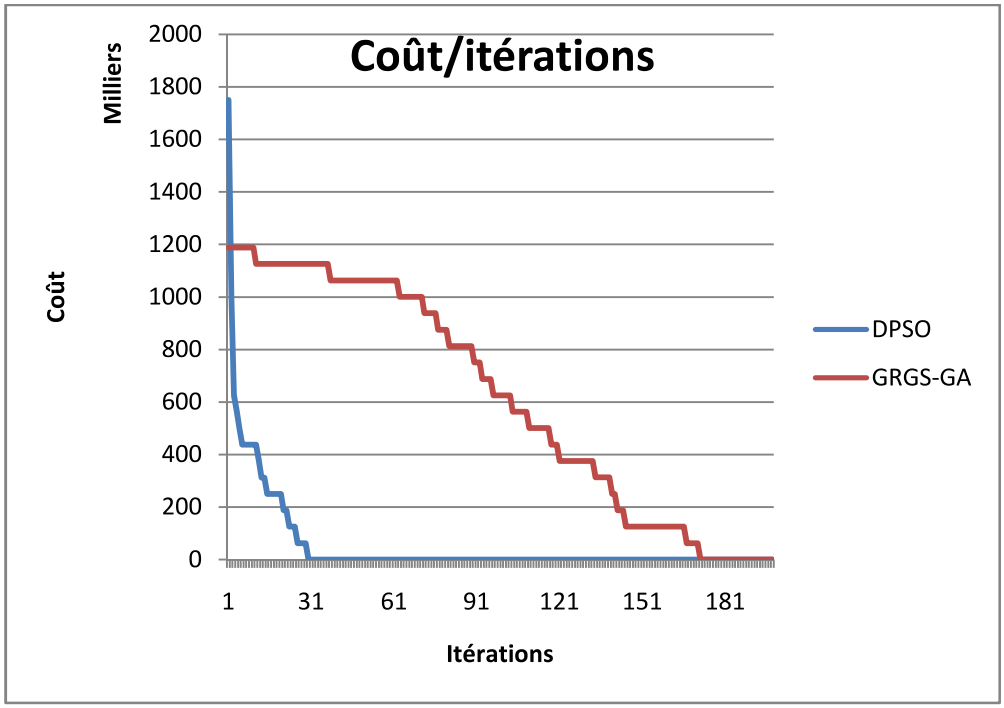


Figure. 5.12 Comparaison de la convergence des trois algorithmes ICPSO, GRGS-GA et SGA pour cas N°3

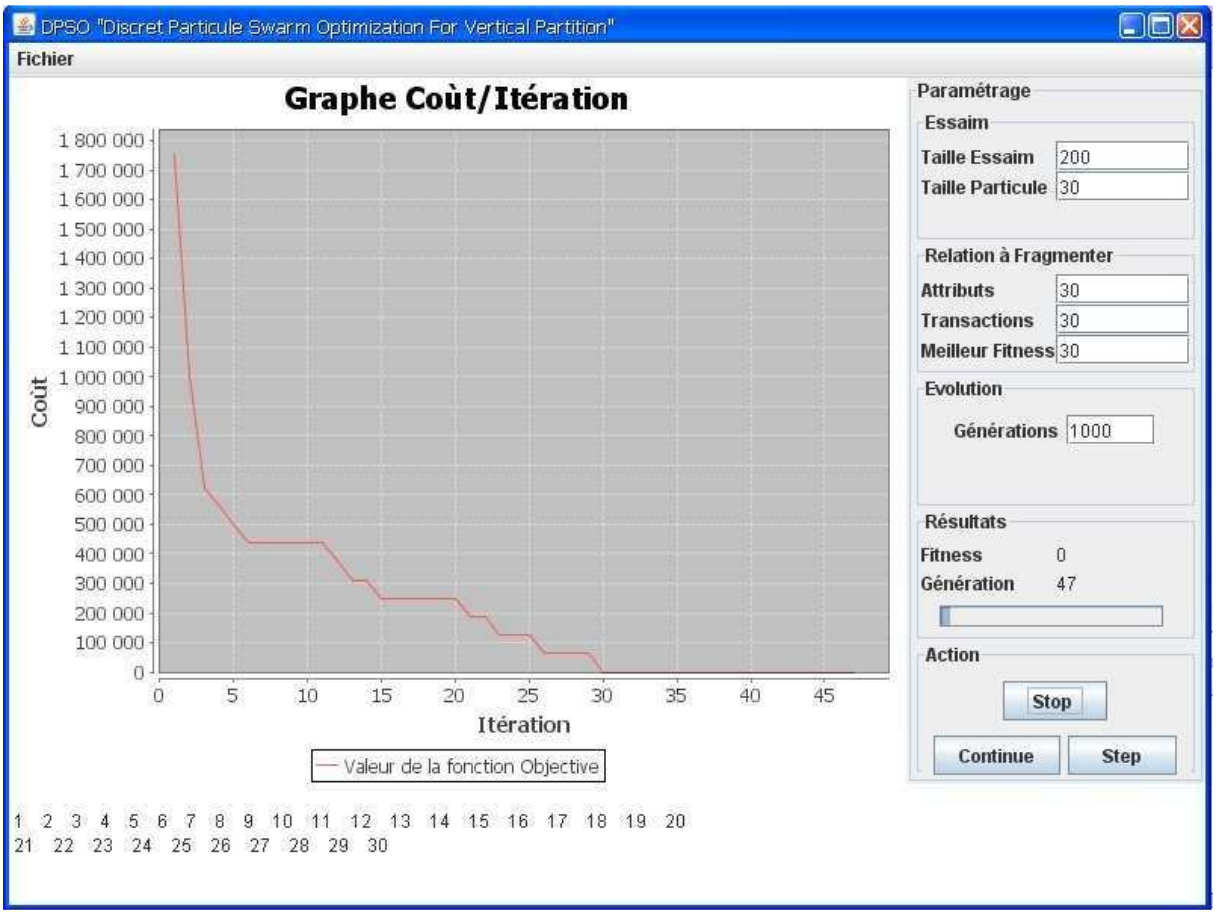


Figure. 5.13 Interface du programme développé pour l'algorithme ICPSO exemple Cas N° 3

5.10.6. Cas 4: exemple de 40 attributs et 40 transactions

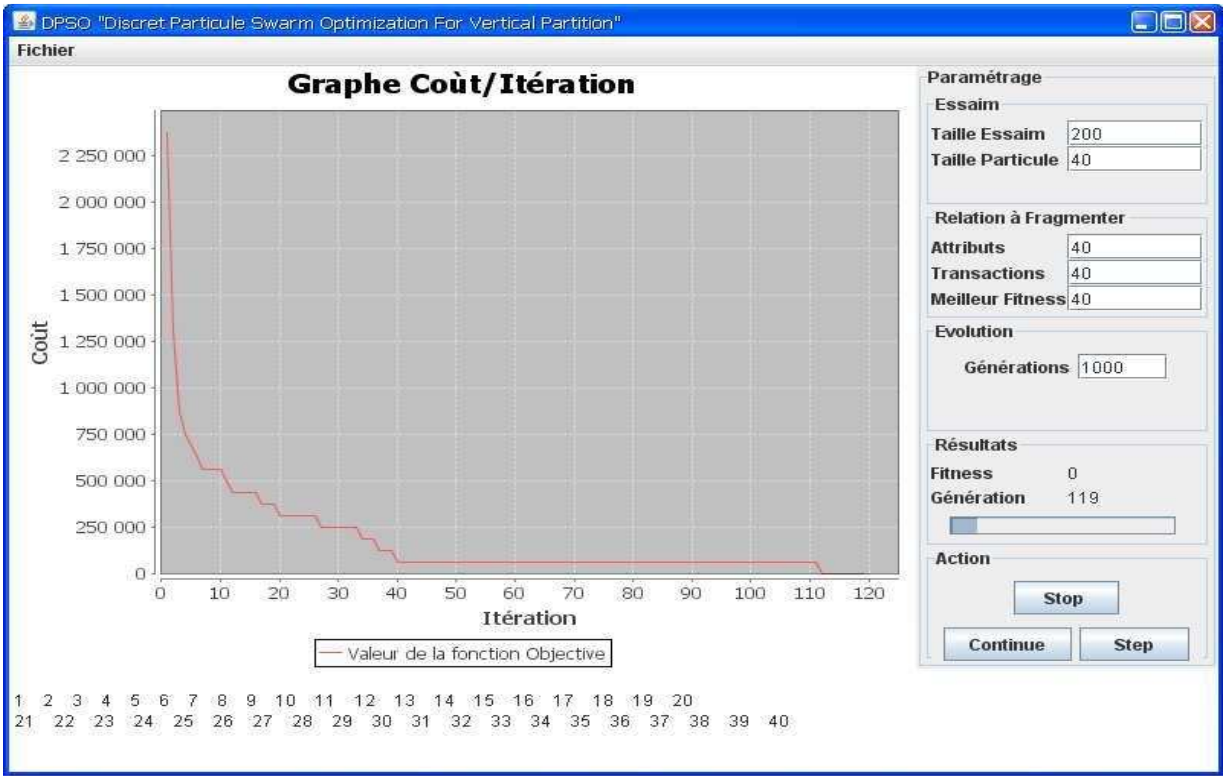


Figure. 5.14 Interface du programme développé pour l'algorithme ICPSO exemple Cas N° 4

5.10.7. Cas 5: exemple de 50 attributs et 50 transactions

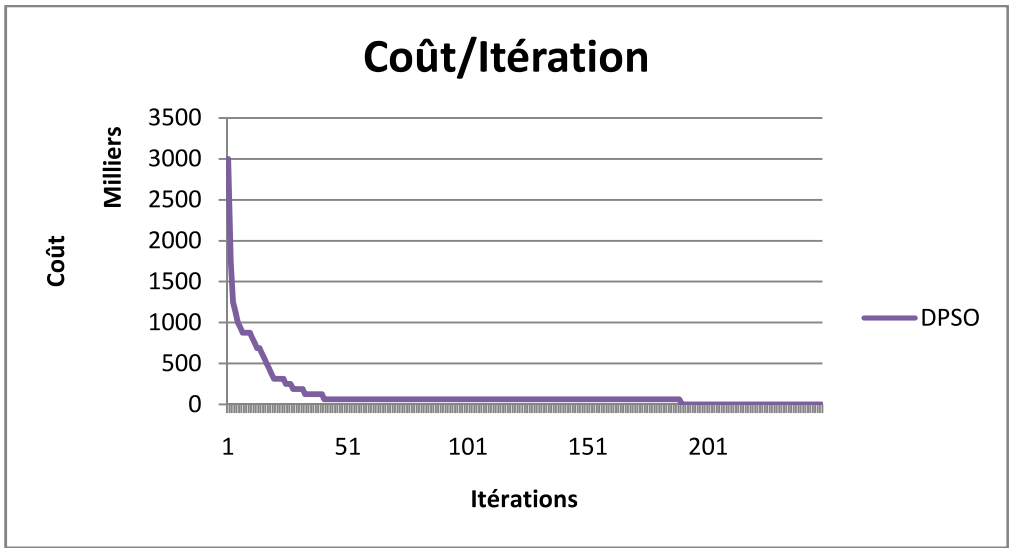


Figure. 5.15 Graphe qui montre l'évolution de la solution optimale pour le Cas N°5

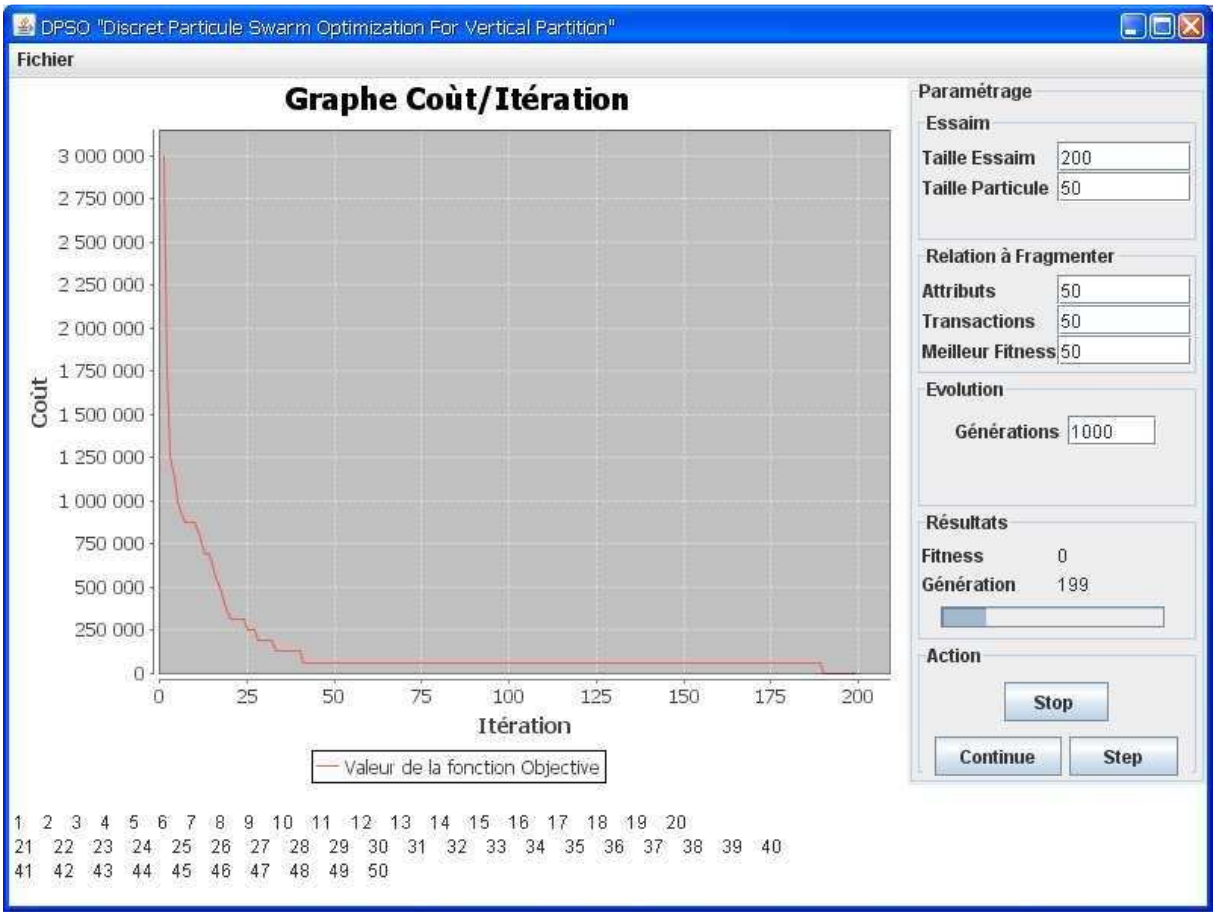


Figure. 5.16 Interface du programme développé pour l'algorithme ICPSO exemple Cas N° 5

5.11. Analyse des résultats obtenus

Dans cette section on va essayer d'analyser et de commenter les résultats obtenus en termes de convergence et des paramètres des algorithmes.

5.11.1 Convergence de l'algorithme ICPSO :

Dans le cas 1,2 et 3 ICPSO converge très rapidement par rapport aux GRGS-GA et SGA, et comme le nombre d'attributs augmente, il devient plus difficile pour ICPSO à converger. Nous remarquons dans le cas 4 et 5 que l'algorithme ICPSO converge rapidement mais lorsqu'il atteint l'itération 40 il trouve des difficultés pour converger. A cause de la complexité du problème de fragmentation.

5.11.2 Effet du nombre de particules dans l'essaim :

Dans la plupart des cas, le fait d'augmenter le nombre des particules dans l'essaim a diminué le nombre d'itérations nécessaires pour atteindre l'objectif. Cela est normal, puisque plus de particules échantillonnent mieux l'espace de recherche. Toutefois, plus de particules

demandent plus d'évaluations de la fonction objectif. Or, dans les applications réelles, le coût de l'évaluation de la fonction objectif est en général dominant. Le critère de performance retenu pour l'algorithme a été le nombre attendu d'évaluations de la fonction objectif, car il prend en compte à la fois le nombre de particules, le nombre d'itérations et le taux de succès.

5.11.3 Comparaison entre OEP et AG

D'après les résultats obtenus on peut dire l'algorithme ICPSO donne des meilleurs résultats par rapport aux deux algorithmes génétiques utilisés dans notre étude. De point de vue de convergence et de temps d'exécution. Le tableau 5.4 présente une petite comparaison entre la méthode d'optimisation par essaims particulaires et les algorithmes génétiques.

OEP	AG
Méthode d'optimisation à population	Méthode d'optimisation à population
L'algorithme original développé pour l'optimisation continue	Développé pour l'optimisation combinatoire
Pas d'opérateurs génétiques	Utilise les opérateurs génétiques (croisement, mutation, sélection) dans l'évolution de la population.
Implémentation très simple le code source de l'algorithme ne dépasse pas 100 lignes de code	Implémentation simple mais selon les opérateurs utilisés.
Il y a peu de paramètres à ajuster	Il y a beaucoup de paramètres à ajuster
Une seule particule (la meilleure particule) qui partage l'information aux autres.	L'information est partagée d'un chromosome à un autre

Tableau 5.4 comparaison entre OEP et AG

Pour comparer aussi les trois algorithmes ICPSO, GRGS-GA et SGA. 100 fois On a exécuté les trois algorithmes, la figure 5.17 illustre le nombre moyen d'itérations nécessaires pour obtenir la solution optimale, qui est pour l'exemple de 10 attributs égale 0 et ce qui correspond à une fragmentation de 10 fragments.

Le nombre moyen de génération nécessaires pour atteindre la solution optimale dans les trois algorithmes est 5,6 et 7.9 et 75.5 itérations pour ICPSO, GRGS-GA et SGA, respectivement. Apparemment, ICPSO à une vitesse de convergence plus rapide que GRGS-GA et SGA.

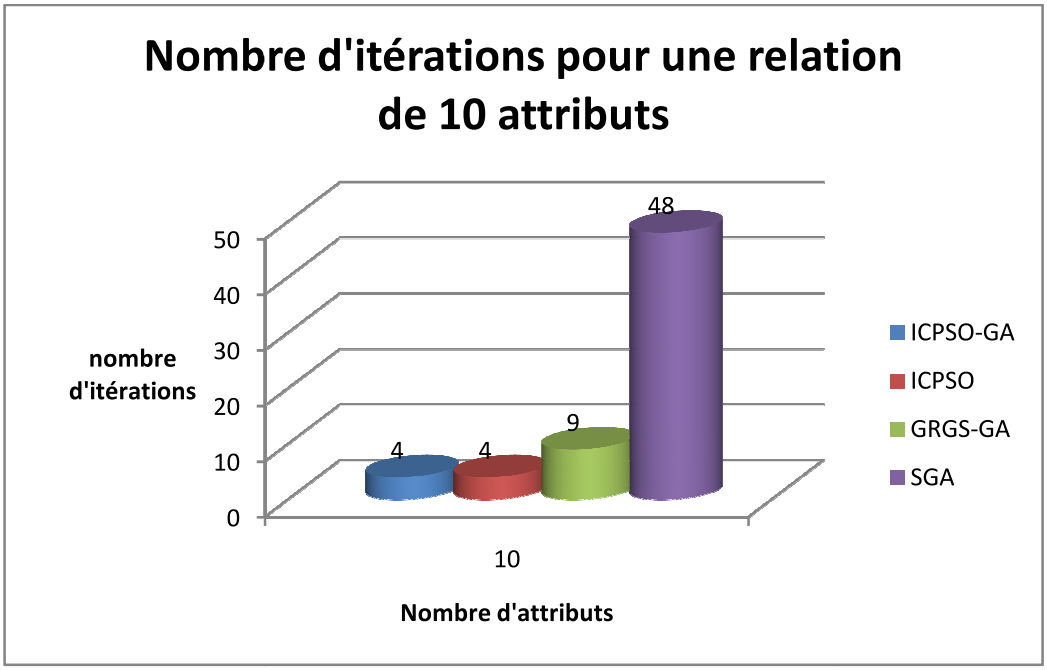


Figure 5.17 Nombre moyen d'itérations nécessaires pour trouver la solution optimale de l'exemple de 10 attributs généré aléatoirement.

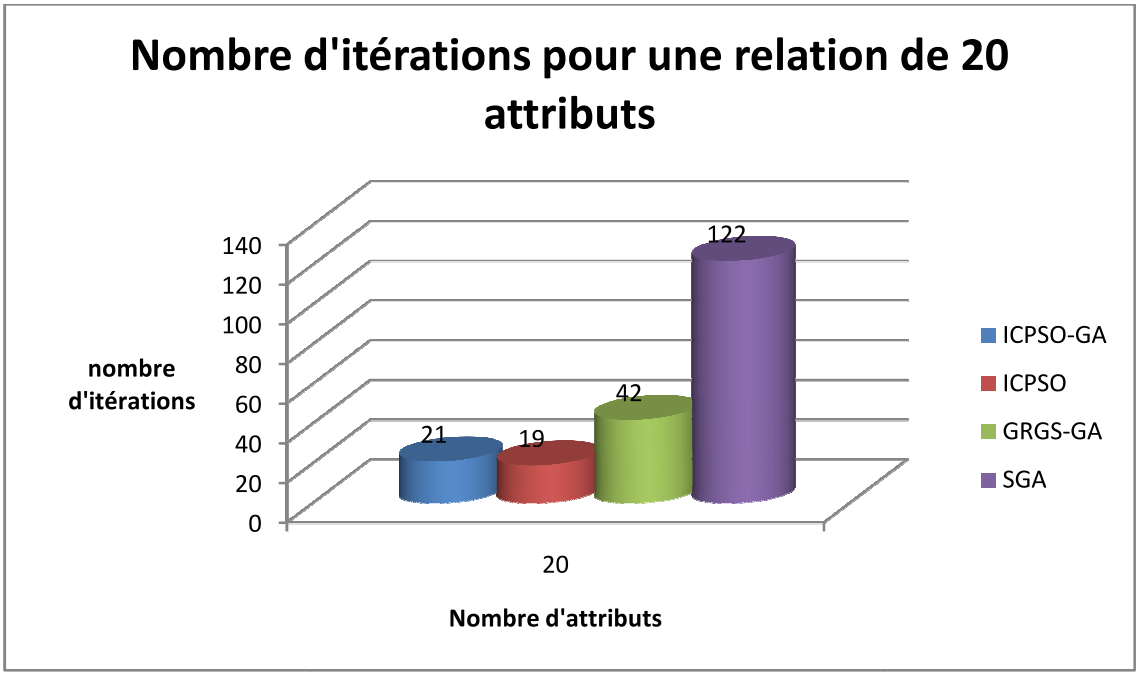


Figure 5.18 Nombre moyen d'itérations nécessaires pour trouver la solution optimale de l'exemple de 20 attributs généré aléatoirement.

10.12 Conclusion

La fragmentation verticale est un problème important pour améliorer les performances des transactions dans les bases de données. Certains chercheurs ont utilisé les techniques d'optimisations pour améliorer la fragmentation verticale (Jun du et al, 2006 ; Niamir & Hammer, 1979; Song & Gorla, 2000). Dans ce chapitre, nous avons proposé une solution basée sur l'optimisation par essaim particulaire (OEP). En particulier, cette solution dispose de deux nouvelles tentatives: Tout d'abord, l'utilisation de la contrainte des chaînes à croissance restreinte (Restricted Growth String) pour représenter les solutions et éliminé la redondance dans l'encodage trouvé dans les solutions précédentes pour le problème de la fragmentation verticale. Deuxième chose, nous avons comparé l'algorithme d'optimisation par essaim particulaire avec deux algorithmes génétiques GRGS-GA développé par [Jun Du et al, 2006], et SGA. Les résultats trouvés montrent que l'algorithme proposé dépasse les deux autres algorithmes et présentes une nouvelle approche pour résoudre le problème de la fragmentation verticale

Conclusion générale

Dans ce travail de mémoire, nous avons présenté un travail d'adaptation et d'application d'une méthode efficace et récente d'optimisation, appelée l'optimisation par essaims particulaires. Les essaims de particules présentent, en effet, des comportements auto-organisés, où des interactions simples au niveau local permettent l'émergence d'un comportement global complexe. Nous avons appliqué cette nouvelle métaheuristique pour la résolution d'un problème très connu dans le domaine des bases de données celui de la fragmentation verticale.

L'algorithme original d'optimisation par essaim particulaire est un algorithme pour les problèmes d'optimisation continue, et le problème de la fragmentation verticale est un problème combinatoire. Nous avons adapté l'algorithme d'optimisation par essaim de particule au cas combinatoire, pour cela on a proposé une approche pour étendre l'algorithme au cas combinatoire.

L'approche proposée utilise le codage sous forme de chaînes de caractères à croissance restreinte (Restricted Growth Strings) pour la représentation des solutions, l'avantage de cette représentation est l'élimination des redondances dans l'encodage des solutions, qui a un grand effet sur l'efficacité de l'algorithme proposé. L'utilisation du codage sous forme de chaîne de caractères à croissance restreinte nous a obligé de proposer un algorithme de rectification pour garantir que la mise à jour des solutions soit toujours sous formes de chaînes à croissance restreinte, et aussi on a développé des nouvelles équations de mise à jours des positions et des vitesses des particules.

Nous avons appliqué l'algorithme développé sur le problème de la fragmentation verticale, ce problème qui a une grande importance dans l'optimisation des performances des bases de données. Pour cela nous avons utilisé une fonction objectif pour l'évaluation de la fragmentation une version modifiée d'évaluateur de partition développé par Chakravarthy et al [Chakravarthy et al, 1992].

Pour tester l'algorithme développé nous avons appliqué l'algorithme sur deux cas d'expérimentation largement discuté dans la littérature sur la fragmentation verticale, puis pour tester l'algorithme sur des cas de grande taille nous avons développé un générateur pseudo aléatoire qui permet de générer des cas d'expérimentation de différentes tailles.

Pour comparer les résultats obtenus avec les résultats des travaux précédents, nous avons reprogrammé deux approches basées sur les algorithmes génétiques appelées GRGS-

GA (Group oriented Restricted Growth String GA) développé par Jun Du et al [Jun du et al, 2006] et SGA (Simple Genetic Algorithm) pour résoudre le même problème celui de la fragmentation verticale. Notre algorithme a donné des bon résultats et compétitifs par rapport à l'approche basée sur les algorithmes génétiques.

D'après les résultats présentés dans ce mémoire, il peut être conclu que l'OEP est un algorithme d'optimisation efficace pour les problèmes d'optimisation difficile, comme le cas du problème de la fragmentation verticale. Ces problèmes sont considérés comme difficiles, car ils sont NP-Difficile et ils sont des problèmes combinatoires.

Ce travail de mémoire se positionne comme une contribution au développement d'un algorithme combinatoire d'optimisation par essaim particulaire et leur application pour la résolution du problème de la fragmentation verticale.

Perspectives

De nombreuses perspectives peuvent être envisagées. Dans cette section nous présentons succinctement celles qui nous parassent êtres les plus intéressantes.

La fonction objectif utilisée est très complexe, et elle nécessite un beaucoup de temps de calcule, qui ne permet pas d'appliquer l'algorithme sur des cas de grandes tailles. Le développement d'une nouvelle fonction objectif pour la fragmentation verticale est primordial.

La fragmentation dynamique : la plupart des algorithmes de fragmentation existants sont de nature statique : ils fragmentent une table en se basant sur un ensemble de requêtes définies préalablement. Or les applications exploitant une base de données peuvent évoluer. Il serait donc intéressant de développer ou d'aménager des algorithmes pour tenir compte de l'évolution des requêtes tant dans leurs structures que dans leurs fréquences.

ANNEXE

Les codes sources jointés dans cette annexe sont donnés sous formes de fonctions implémentées en langage JAVA en utilisant le JDK version 1.6.

1. Le code source de la *fonction objectif* (évaluateur de partition en anglais PE pour Partition Evaluator) utilisée dans l'algorithme de l'optimisation par essaim particulaire DPSO.

CODE SOURCE DE LA FONCTION OBJECTIF

```
PUBLIC  INT PE(INT PARTITION[][]){

        FOR (I=0;I<M;I++)
            FOR(J=0;J<N;J++) V[I][J]=0;
         //INT V[][]=NEW INT[M][N];
                    I=0;
                    J=0;
         WHILE((I<N)&&(PARTITION[I][J]!=-2) ){
                  FOR(INT K=0;K<M;K++){
                  WHILE((J<N) && (PARTITION[I][J]!=-1)){
                                  V[K][I]+=AUM[K][PARTITION[I][J]-1];
                                     J++;
                              }

                        // DOUBLE    DF = (DOUBLE) V[K][I]/J;
                        DF = (DOUBLE) V[K][I]/J;
                         V[K][I]=(INT) MATH.ROUND(DF);
                           J=0;
               }
                         I++;
                         J=0;
                         NBF++;
        }
        //INT EM=0;
         EM=0;
          I=0; J=0;
         WHILE((I<N)&&(PARTITION[I][J]!=-2)   ){
                  FOR( KM=0;KM<M;KM++){
                  WHILE((J<N) && (PARTITION[I][J]!=-1) ){
                                 EM+=(AUM[KM][PARTITION[I][J]-1]-
                              V[KM][I])*(AUM[KM][PARTITION[I][J]-1]-V[KM][I]);
                                  J++;
                              }
                                 J=0;
               }
                         I++;
                         J=0;
         }

        //CALCULE DE ER
```

```
ER=0;
CANRUN=0;
 FI=0; //FREQUENCE
 SOM=0;
 MINT=0;
FOR( T=1; T<=M;T++){
          CANRUN=0;
          FI=0; //FREQUENCE
          SOM=0;
FOR( M=0;M<NBF;M++){
   KR=0;
     WHILE((CANRUN==0)&&(KR<N)&&(M<N) &&(PARTITION[M][KR]!=-1) ){
           IF((PARTITION[M][KR]>0)&&(AUM[T-1][PARTITION[M][KR]-
            1]!=0)){
                        CANRUN=1;
                        FI=AUM[T-1][PARTITION[M][KR]-1];
                        }
                KR++;
                }
            IF(CANRUN==1){
                FOR(  P=0;P<NBF;P++){
                        IF(P!=M){
                          N2=0;
                          R1=0;
                FOR(DF1=0;((DF1<N)&&(P<N)&&(PARTITION[P][DF1]!=-
                        1));DF1++)     N2++;// REMARQUE DF<N
                FOR(DF1=0;((DF1<N)&&(P<N)&&(PARTITION[P][DF1]!=-
                1));DF1++){  // REMARQUE DF<N

          IF((PARTITION[P][DF1]>0)&&(AUM[T-1][PARTITION[P][DF1]-
                        1]!=0)){R1++;}
                        }
           IF (N2!=0){
                        DOUBLE DS = (DOUBLE) FI*FI*R1*R1/N2;
                      SOM+=(INT) MATH.ROUND(DS)     ;
                     }
             }
   }
       IF(MINT==0)MINT=SOM;
       IF((SOM<=MINT)&& (SOM!=0)) {MINT=SOM;}
                        som=0;
                        canrun=0;
              }
     }//
Er+=Mint;Mint=0;
}

return Em+Er;
}
```

FIN DU CODE SOURCE DE LA FUNCTION OBJECTIF

2. Le code source du générateur pseudo Aléatoire des matrices d'usage d'attributs.

CODE SOURCE DU GENERATEUR PSEUDO ALEATOIRE DES MATRICES D'USAGE D'ATTRIBUTS

```
INT NBATTRIBUTS;// NOMBRE D'ATTRIBUTS DE LA RELATION A FRAGMENTEE
INT NBTRANSACTIONS; //NOMBRE DE TRANSACTIONS OU REQUETES
INT NBFRAGMENTS; // NOMBRE DE FRAGMENTS
INT FREQUENCEMAX; // FREQUENCE MAXIMALE
PUBLIC OBJECT AUM[][]; // MATRICE D'USAGE D'ATTRIBUTS
RANDOM ALEATOIRE; // GENERATEUR DE NOMBRE ALEATOIRE

PUBLIC VOID AUM_GENERATOR(INT NBA,INT NBT,INT NBF){
                NBATTRIBUTS=NBA;
                NBTRANSACTIONS=NBT;
                NBFRAGMENTS=NBF;
                STRING[] COLUMNNAMES =NEW STRING[NBATTRIBUTS];
     AUM = NEW OBJECT[NBTRANSACTIONS][NBATTRIBUTS];
     ALEATOIRE =NEW RANDOM();
     FOR(INT I=1;I<=NBATTRIBUTS;I++) {
         COLUMNNAMES[I-1]= "A"+I;
          }
    INT S ;
      FOR(S=0;S<NBFRAGMENTS;S++) {
         FOR(INT M=0;M<NBFRAGMENTS;M++){
                AUM[M][S]=NEW INTEGER(0);
              IF(M==S)          AUM[M][S]=NEW INTEGER(FREQUENCEMAX);
                }
      }
      FOR(INT I=S;I<NBATTRIBUTS;I++){
          FOR(INT J=0;J<NBFRAGMENTS;J++){
                        AUM[J][I]=NEW INTEGER(0);

                }
                INT R=ALEATOIRE.NEXTINT(NBFRAGMENTS);
                AUM[R][I]=NEW INTEGER(FREQUENCEMAX);
        }
       FOR(INT J=NBFRAGMENTS;J<NBTRANSACTIONS;J++){
                INT R=ALEATOIRE.NEXTINT(10)+1;
                R*=5;
        FOR(INT I=0;I<NBATTRIBUTS;I++){
                        INT R1=ALEATOIRE.NEXTINT(2);
                        IF(R1==1)
                                AUM[J][I]=NEW INTEGER(R);
                        ELSE    AUM[J][I]=NEW INTEGER(0);

                }
        }
    /*     FOR(INT I=0;I<NBFRAGMENTS;I++)
             { INT NB=0;
                FOR(INT J=0;J<NBATTRIBUTS;J++){
```

```
                    IF(AUM[I][J].EQUALS(NEW INTEGER(FREQUENCEMAX)))NB++;
                }
            IF(NB==0) SYSTEM.OUT.PRINTLN("ERREUR ERREUR ERREUR ERREUR ERREUR ERREUR
                    ERREUR ERREUR ");
            SYSTEM.OUT.PRINTLN("FRAGMENT "+I+"= "+NB);

        }*/
         JTABLE TABLE = NEW JTABLE(AUM, COLUMNNAMES);
        //ADD THE SCROLL PANE TO THIS PANEL.
        JSCROLLPANE PANNEAU = NEW JSCROLLPANE(TABLE,
        JSCROLLPANE.VERTICAL_SCROLLBAR_AS_NEEDED,
        JSCROLLPANE.HORIZONTAL_SCROLLBAR_AS_NEEDED); //METTRE LA TABLE DANS ASCENCEUR
        TABLE.SETAUTORESIZEMODE(JTABLE.AUTO_RESIZE_OFF);
            ADD(PANNEAU);
             SETSIZE(500,500);
             SETVISIBLE(TRUE);

        }
```

FIN DU CODE SOURCE DU GENERATEUR PSEUDO ALEATOIRE

3. Le code source de l'algorithme de rectification

CODE SOURCE DU RECTUFICATEUR

```
INT [] RECTIFIER(INT A[]){
        INT DEGREE=0;
        INTEGER B[]=NEW INTEGER[A.LENGTH];
        FOR(INT I=0;I<A.LENGTH;I++){
            B[I]=INTEGER.VALUEOF(A[I]);
            }
        HASHTABLE HASHTABLE = NEW HASHTABLE();
        FOR(INT I=0;I<A.LENGTH;I++){

            IF(HASHTABLE.CONTAINSKEY(B[I])){
            INTEGER      J=(INTEGER) HASHTABLE.GET(B[I]);
                A[I]= J.INTVALUE();
                }
            ELSE{
                DEGREE++;
                INTEGER      H= INTEGER.VALUEOF(DEGREE);
                HASHTABLE.PUT(B[I] ,H);
                A[I]=DEGREE;
                }
                }
        RETURN A;
        }
```

FIN DU CODE SOURCE DU RECTUFICATEUR

4. Le code source de l'algorithme d'optimisation par essaim particulaire appliqué au problème de la fragmentation verticale développé dans ce mémoire.

CODE SOURCE DE L'ALGORITHME D'OPTIMISATION PAR ESSAIM PARTICULAIRE DPSO

```
PUBLIC  VOID NOUVELLEGENERATION(){

        // UPDATE THE LOCAL BESTS AND THEIR FITNESS
        ITERATOR IT=PARTICULES.ITERATOR();
        PARTICULE P;
        WHILE(IT.HASNEXT()) {
                    P=(PARTICULE)(IT.NEXT());
                    P.EVALFITNESS();
                   //  PERSONNEL BEST
                     IF(P.FITNESS<P.F_PBEST){
                    P.F_PBEST=P.FITNESS;
                    FOR(INT J=0;J<P.TAILLE;J++){
                              P.X_PBEST[J]=P.GENES[J];
                   }
         }

         // GLOBAL BEST
         IF(P.FITNESS<BESTPARTICULE.FITNESS){
                   BESTPARTICULE.FITNESS = P.FITNESS;
                              FOR(INT J=0;J<P.TAILLE;J++){
                              //         GBEST.X[J]=ESSAIM[I].X[J];
                             BESTPARTICULE.GENES[J]=P.GENES[J];
                                        }
                   }

        }
    /*   FOR(INT I=0;I<TAILLEESSAIM;I++){
         PARTICULES..F= PE( N, M,LISTPART(ESSAIM[I].X,N));
          IF(ESSAIM[I].F<ESSAIM[I].F_PBEST){
                   ESSAIM[I].F_PBEST=ESSAIM[I].F;
                              FOR(INT J=0;J<N;J++){
                                        ESSAIM[I].X_PBEST[J]=ESSAIM[I].X[J];
                                        }
                   }

          IF(ESSAIM[I].F<GBEST.F){
                   GBEST.F=ESSAIM[I].F;
                              FOR(INT J=0;J<N;J++){
                                        GBEST.X[J]=ESSAIM[I].X[J];
                                        }
                   }
          }*/
          RANDOM R=NEW RANDOM();
           ITERATOR ITT=PARTICULES.ITERATOR();

           WHILE(ITT.HASNEXT()) {
```

```
                P=(PARTICULE)(ITT.NEXT());
                FOR(INT J=0;J<P.TAILLE;J++){
                        IF(     P.GENES[J]==P.X_PBEST[J]){
                        IF(     P.GENES[J]==BESTPARTICULE.GENES[J]){
                                IF(R.NEXTINT(2)==1)
                                P.Y[J]=1;
                                ELSE    P.Y[J]=-1;
                                }
                        ELSE
                                P.Y[J]=-1;
                        }
                        ELSE {IF(P.GENES[J]==BESTPARTICULE.GENES[J]){
                                P.Y[J]=1;

                        } ELSE  P.Y[J]=0;}

                    P.VP[J]  =W* P.VP[J]+R1*C1*(-1-P.Y[J])+R2*C2*(1-P.Y[J]);
                    P.Y[J]+=  P.VP[J];
                 IF(P.Y[J]>ALPHA){
                        P.GENES[J]=BESTPARTICULE.GENES[J];
                } ELSE
                IF(P.Y[J]<-ALPHA){
                        P.GENES[J]=P.X_PBEST[J];
                        }
                        ELSE  P.GENES[J]=R.NEXTINT(P.TAILLE);

                }
          P.GENES=P.RECTIFIER(P.GENES);
         }
}
```

FIN DU CODE SOURCE DE L'ALGORITHME D'OPTIMISATION PAR ESSAIM PARTICULAIRE DPSO

RO : Recherche Opérationnelle

IA : Intelligence Artificielle

OEP : L'optimisation par essaim particulaire

SI : intelligence en essaim ou Swarm Intelligence en anglais

PSO: Particle Swarm Optimization

ACO: Ant Colony Optimization

DE : Différentiel Evolution

EA : Evolutionary Algorithms

EC : Evolutionary Computation

EP: Evolutionary Programming

EPSO: Evolutionary Particle Swarm Optimization

ES : Evolution Strategies

FEP: Fast Evolutionary Programming

GA: Genetic Algorithms

GO: Global Optimization

GP: Genetic Programming

MO: Multiobjective Optimization

C.A.O : Conception Assistée par Ordinateur

NP : La classe NP des problèmes Non-déterministes Polynomiaux

AIS : Artificial Immune System ou système immunitaire artificiel

SMA : Systèmes multi-agents

DPSO : discret particle swarm optimization

CPSO : Combinatorial particle swarm optimization

MASSIVE : Multiple Agent Simulation System In Virtual Environnement

AUM : Matrice d'usage d'attribut

IAD: Intelligence Artificielle Distribuée

Gbest : Global best PSO

Lbest : Local best PSO

GCPSO : L'optimisation par essaim particulaire à convergence garantie

MPSO : L'optimisation par essaim particulaire Multi-start

ARPSO : L'optimisation par essaim particulaire Attractives et Répulsives

GA-PSO : Genetics algorithms and particle swarm optimisation

NNS : réseaux de neurones

Bibliographie

- **[B. Benmessahel et M. Touahria, 2009]** Bilal Benmessahel, Mohamed Touahria. *Appling A Discrete Particle Swarm Optimization Algorithm to Database Vertical Partition*. In CIIA'09, Conférence internationale sur l'informatique et ses applications 2009, Saida. Algérie
- **[Das et al , 2009]** Swagatam Das, Ajith Abraham, Amit Konar, *Metaheuristic Clustering,* (Studies in computational intelligence, Vol. 178), Springer; 1 edition (Mar 25 2009)
- **[COOREN, 2008]** COOREN, Y. (2008). *Perfectionnement d'un algorithme adaptatif d'Optimisation par Essaim Particulaire. Applications en génie médical et en électronique*. PhD thesis, Université Paris 12.
- **[Van den Bergh, 2002]** F. Van den Bergh. *An Analysis of Particle Swarm Optimizers*, PhD Thesis. Department of Computer Science, University of Pretoria, South Africa, 2002.
- **[B. Jarboui et al, 2007]** B. Jarboui, M. Cheikh, P. Siarry, A. Rebai: *Combinatorial particle swarm optimization (CPSO) for partitional clustering problem*. Applied Mathematics and Computation 192(2): 337-345 (2007)
- **[CLERC, 2003]** CLERC, M. (2003). *Tribes - un exemple d'optimisation par essaim particulaire sans paramètres de contrôle*. In Optimisation par Essaim Particulaire (OEP 2003).
- **[Ozsu et Valduriez, 1999]** Ozsu, M. T., & Valduriez, P. (1999). *Principles of distributed database systems*. Prentice Hall.
- **[Jun du et al, 2006]** Jun du, Reda Alhajj, Ken Barker « *Genetic algorithms based approach to database vertical partition* » Journal of Intelligent Information Systems Volume 26 , Issue 2 (March 2006) Pages: 167 - 183 Year of Publication: 2006
- **[Jin-Kao Hao et al, 1999]** Jin-Kao Hao,Philippe Galinier, Michel Habib, *Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes*, Revue d'Intelligence Artificielle, Vol : No. 1999.
- **[J.Dreo et al, 2003]**J. Dreo, A. Petrowski, P. Siarry et E. Taillard*, Métaheuristiques pour l'optimisation difficile*, Eyrolls, 2003.
- **[Glover, 1986]** Glover, F. (1986). *Future paths for integer programming and links to artificial intelligence*. Computers and Operations Research, 13:533–549.
- **[Glover, 1989]** Glover, F. (1989). *Tabu search — part I*. ORSA Journal on Computing, 1:190–206.
- **[Fraser, 1957]** Fraser, A. S. (1957). *Simulation of genetic systems by automatic digital computers*. Australian Journal of Biological Sciences, 10:484–491.
- **[Holland, 1962]** Holland, J. H. (1962). *Outline for logical theory of adaptive systems*. J. Assoc. Comput. Mach., 3:297–314.
- **[Fogel et al., 1966]** Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artifical Intelligence through Simulated Evolution*. Wiley.
- **[Rechenberg, 1973]** Rechenberg, I. (1973). *Evolutions strategie: Optimierung technis-cher Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog,Stuttgart.
- **[Goldberg, 1989]** Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine learning*. Addison-Wesley.
- **[Goldberg, 1994]** Goldberg, D. E. (1994). *Algorithmes génétiques. Exploration, optimisation et apprentissage automatique*. Addison-Wesley France.
- **[De Castro and Von Zuben, 2000]** De Castro, L. and Von Zuben, F. (2000). *Artificial Immune Systems : Part II - A Survey of Applications*. Technical Report DCA-RT 02/00, Department of

Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil.

- **[Dréo and Siarry, 2003a]** Dréo, J. and Siarry, P. (2003a). *Colonies de fourmis et optimisation continue. De l'utilité de l'optimisation en général et des colonies de fourmis en particulier, quand l'éthologie et l'informatique se croisent*. In Séminaire de recherche de l'ISBS-Paris, Créteil. Université Paris 12.
- **[Eberhart et al., 2001]** Eberhart, R., Kennedy, J., and Shi, Y. (2001). *Swarm Intelligence. Evolutionary Computation*. Morgan Kaufmann.
- **[Choo, 2000]** Choo, S.-Y. (2000). *Genetic Algorithms and Genetic Programming* at Stanford 2000, chapter Emergence of a Division of Labour in a Bee Colony, pages 98–107.Stanford Bookstore, Stanford, California.
- **[Panta, 2002]** Panta, L. (2002). *Modeling Transportation Problems Using Concepts of Swarm Intelligence and Soft Computing.* PhD thesis, Virginia Tech, Blacksburg, Virginia.
- **[De Wolf et al., 2002]** De Wolf, T., Liesbeth, J., Holvoet, T., and Steegmans, E. (2002). *A Nested Layered Threshold Model for Dynamic Task Allocation*. In Dorigo, M., Di Caro, G., and Sampels, M., editors, Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002), volume 2463 of Lecture Notes in Computer Science, pages 290–291, Brussels, Belgium. Springer Verlag.
- **[Nouyan, 2002]** Nouyan, S. (2002). *Agent-Based Approach to Dynamic task Allocation*. In Dorigo, M., Di Caro, G., and Sampels, M., editors, Proceedings of the Third International workshop on Ant Algorithms (ANTS'2002), volume 2463 of Lecture Notes in Computer Science, pages 28–39, Brussels, Belgium. Springer Verlag.
- **[P. Gray et al, 1997]**P. Gray, W. Hart, L. Painton, C. Phillips, M. Trahan and John Wagner. *A Survey of Global Optimization Methods*, Sandia National Laboratories, 1997, *http://www.cs.sandia.gov/opt/survey (visited 10 September 2008).*
- **[Andries. P Engelbrecht, 2002]** Andries. P Engelbrecht, *Computational Intelligence : An Introduction*, John Wiley & Sons Ltd, 2002.
- **[Ali, 1997]** S-M.Ali, et R.M. Zimmer, *The question concerning emergence*, in CASYS'97, Abstract Book, First International Conference on Computing Anticipatory Systems, CHAOS asdl, 1997
- **[Georgé, 2004]** J-P Georgé *Résolution de problèmes par émergence*, *Etude d'un Environnement de Programmation Emergente*, Thèse de l'Université Paul Sabatier, Toulouse, Juillet 2004.
- **[Holland, 1975]** J. Holland *Adaptation in Natural and Artificial Systems*, MIT Press, 1975.
- **[Holland, 1999]** O.E Holland and Melhuish.C. *Stigmergy, self-organization, and sorting in collective robotics*. Artificial Life, 5 :173–202, 1999.
- **[Gleizes, 2004]** M-P Gleizes *Vers la résolution de problèmes par émergence* Habilitation à diriger des recherches de l'Université Paul Sabatier Spécialité : Informatique, 2004.
- **[Van de Vijver, 1997]** G. Van De Vijver, *Emergence et explication, Intellectica : Emergence and explanation*, 1997/2 n°25, ISSN n°0984-0028 185-194, 1997.
- **[Atlan, 2000]** H. Atlan, La finalité, Hors série Science&Avenir, 2000.
- **[Goldstein, 1999]** J. Goldstein, *émergence as a Construct : History and issues, Emergence* Volume 1, Issue 1, pp 49-71, 1999.
- **[Langton, 1991]** C. Langton. *Life at the Edge of Chaos, in Artificial Life II* : SFI Studies in the Sciences of Complexity Vol. 10, C. Langton Ed., Addison-Weasley, 1991, pages 41-91.

- **[Odell, 2001]** J. Odell, H.V Parunak and B. Bauer *Representing Agent Interaction Protocols in UML*, Oriented Software Engineering, P. Ciancarini and M. Wooldridge Editors, Springer Verlag, Berlin, pages 121-140, 2001
- **[Prigogine 1977]** I. Prigogine, G. Nicolis, *Self Organization in Non-Equilibrium Systems*,(Chaps. III and IV), J. Wiley and Sons, New York, 1977.
- **[Varela, 1988]** F. Varela *Autonomie et connaissance : essai sur le vivant* Editions du seuil – 1988.
- **[Ünsal, 1993]** C. Ünsal **Self-organization in large populations of mobile robots** Master of Sciences in Electrical Engineering, May 1993, Blacksburg, Virginia. http://armyant.ee.vt.edu/unsalWWW/cemthesis.html.
- **[Krippendorff, 1997]** K. Krippendorff *Dictionnary of Cybernetics*" 27 janvier 2008. http://pespmc1.vub.ac.be/SELFORG.html
- **[Bonabeau, 1997]** E Bonabeau et G. Theraulaz G., *Auto-organisation et comportements collectifs : la modélisation des sociétés d'insectes, Auto-organisation et comportement*, Editions Hermès, 1997.
- **[Bonabeau, 1999]** E. Bonabeau, M. Dorigo, G.Theraulaz, *Swarm Intelligence : From Natural to Artificiel Systems*, NEW York, Oxford University Press,1999.
- **[Camazine, 2002]** S. Camazine, J-L. Deneubourg , N.R. Franks , J. Sneyd , G. Theraulaz et E. Bonabeau?*Self-Organization in Biological Systems*, Princeton University Press, 2002.
- [Theraulaz, 1997] G. Theraulaz, F. Spitz, Auto-organisation et comportement, Hermès, Paris, 1997.
- **[Deneubourg, 1987]** J.L Deneubourg, S. Goss, J. Pasteels D. Fresneau et J.P. Lachaud J. P., *Self-Organization Mechanisms in Ants Societies (II): Learning in Foraging and Division of Labour, in From Individual to Collective Behaviour in Social Insects*, Birkhäuser Verlag, Basel, pp. 177-196, 1987.
- **[Deneubourg, 1991]** J.L Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, et L. Chretien,. *The dynamic of collective sorting robot-like ants and ant-like robots*, in J. A. Meyer and S. W. Wilson (eds), SAB 90 - 1st Conference On Simulation of Adaptive Behavior: From Animals to Animats, MIT Press. 1991.
- **[Grassé, 1959]** P.P Grassé. *La reconstruction du nid et les interactions inter-individuelles chez les bellicoitermes natalenis et cubitermes, la théorie de la stigmergie - essai d'interprétation des termites constructeurs*. Insectes Sociaux, 6 :41–81, 1959.
- **[Ferber, 1995]** Ferber Jacques, *Les Systèmes Multi-Agents* : *vers une Intelligence Collective*, Inter Editions, 1995.
- **[Jennings, 1996]** Jennings, N. R. (1996*). Coordination Techniques for Distributed Artificial Intelligence*. In G. M. P. O'Hare and N. R. Jennings, editor, Foundations of Distributed Artificial Intelligence, pages 187–210. John Wiley & Sons.
- **[Wilson and Holldobler, 1988]** Wilson, E. and Holldobler, B. (1988). *Dense Heterarchy and mass communication as the basis of organization in ant colonies*. Trend in Ecology and Evolution, 3 :65–68.
- **[Taillard, 1998]** Taillard, E. D. (1998). *Programmation à mémoire adaptative et algorithmes pseudo-gloutons : nouvelles perspectives pour les méta-heuristiques*. Thèse d'habilitation ` a diriger les recherches, Université de Versailles Saint Quentin en Yvelines, France.
- **[Taillard et al., 1998]** Taillard, E. D., Gambardella, L. M., Gendreau, M., and Potvin, J.-Y. (1998). *Adaptive Memory Programming : A Unified View of Meta-Heuristics*.European Journal of Operational Research, 135(1) :1–16.

- **[Talbi, 2002]** Talbi, E.-G. (2002). *A Taxonomy of Hybrid Metaheuristics*. Journal of Heuristics, 8(5) :541–564.
- **[Nelder and Mead, 1965]** Nelder, J. A. and Mead, R. (1965). *A simplex method for function minimization*. Computer Journal, 7 :308–313.
- **[Resende, 2000]** Resende, M. (2000). *Greedy randomized adaptive search procedures (GRASP*). Technical Report TR 98.41.1, AT&T Labs-Research.
- **[Larranaga and Lozano, 2002]** Larranaga, P. and Lozano, J. (2002). *Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation*. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers.
- **[Chevrier, 2005]** Vincent Chevrier et Aurélien Saint-Dizier , « *l'intellegonce en essaim ou comment faire complexe avec du simple* » , http://interstices.info/jcms/c_7083/l-intelligence-en-essaim-ou-comment-faire-complexe-avec-du-simple page consulté le 10/08/2008.
- **[Shi et Eberhart, Parameter 1998]** Y. Shi and R. Eberhart. *Parameter Selection in Particle Swarm Optimization*. Evolutionary Programming VII: Proceedings of EP 98, pp. 591-600, 1998.
- **[Shi et Eberhart, A Modified 1998]** Y. Shi and R. Eberhart. *A Modified Particle Swarm Optimizer*. In Proceedings of the IEEE International Conference on Evolutionary Computation, Piscataway, New Jersey, pp. 69-73, 1998.
- **[Ozcan et Mohan, 1998]** E. Ozcan and C. Mohan. *Analysis of a Simple Particle Swarm Optimization System*. Intelligent Engineering Systems Through Artificial Neural Networks, vol. 8, pp. 253-258, 1998.
- **[Clerc et Kennedy,2001]**, M. Clerc and J. Kennedy. *The Particle Swarm: Explosion, Stability and Convergence in a Multi-Dimensional Complex Space*. IEEE Transactions on Evolutionary Computation, vol. 6, pp. 58-73, 2001.
- **[Zheng et al, 2003]** Y. Zheng, L. Ma, L. Zhang and J. Qian. *Robust PID Controller Design using Particle Swarm Optimizer*. In Proceedings of IEEE International Symposium on Intelligence Control, pp. 974-979, 2003.
- **[Yasuda et al,2003]** K. Yasuda, A. Ide and N. Iwasaki. *Adaptive Particle Swarm Optimization*. In Proceedings of IEEE International Conference on Systems, Man and Cybernetics, pp. 1554-1559, 2003.
- **[Trelea, 2003]** I. Trelea. *The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection*. Information Processing Letters, vol. 85, no. 6, pp. 317-325, 2003.
- **[Eberhart et al. 1996]** R. Eberhart, P. Simpson and R. Dobbins. *Computational Intelligence PC Tools*. Morgan Kaufmann, 1996.
- **[Kennedy, 1999]** J. Kennedy. *Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance*. In Proceedings of the Congress on Evolutionary Computation, pp. 1931-1938, 1999.
- **[Kennedy et Medes, 2002]**J. Kennedy and R. Medes. *Population Structures and Particle Swarm Performance*. In Proceedings of the IEEE Congress on Evolutionary Computation, Hawaii, USA, 2002.
- **[Løvberg, 2002]** M. Løvberg. *Improving Particle Swarm Optimization by Hybridization of Stochastic Search Heuristics and Self Organized Critically*, Master's Thesis. Department of Computer Science, University of Aarhus, Denmark, 2002.
- **[Dorigo et al. 1999]** M. Dorigo and G. Di Caro. *The Ant Colony Optimization Meta-Heuristic. New Methods in Optimization*, D. Corne, M. Dorigo and F. Glover, Eds., McGraw-Hill, 1999.

- **[Angeline, 1998]** P. Angeline. *Evolutionary Optimization versus Particle Swarm Optimization: Philosophy and Performance Difference*. In Proceedings of the Seventh Annual Conference on Evolutionary Programming, pp. 601-610, 1998.
- **[Riget et Vesterstrøm, 2002]** J. Riget and J. Vesterstrøm. *A Diversity-Guided Particle Swarm Optimizer – The ARPSO*. EVALife Technical Report no. 2002-2, 2002.
- **[Bäck, 1992]** T. Bäck. *Self-Adaptation in Genetic Algorithms*. In Proceedings of the First European Conference on Artificial Life, pp. 227-235, MIT Press, 1992.
- **[Clerc, 1999]** M. Clerc. *The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization*. In Proceedings of the Congress on Evolutionary Computation, Washington DC, USA, vol. 3, pp. 1951-1957, IEEE Press, 1999.
- **[Shi et Eberhart, 2001]** Y. Shi and R. Eberhart. *Fuzzy Adaptive Particle Swarm Optimization*. In Proceedings Congress on Evolutionary Computation, Seoul, S. Korea, 2001.
- **[Angeline,Using Selection 1998]** P. Angeline*. Using Selection to Improve Particle Swarm Optimization.* In International Conference on Evolutionary Computation, Piscataway, New Jersey, USA, pp. 84-89, IEEE Service Center, 1998.
- **[Hu et Eberhart, 2002]** X. Hu and R. Eberhart. *Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems*. In Proceedings of congress on Evolutionary Computation, Hawaii, USA, pp. 1666-1670, 2002.
- **[X. Hun, 2004]**. *Particle Swarm Optimization*: Bibliography, 2004. http://www.swarmintelligence.org/bibliography.php (visited 8 September 2008).
- **[Ratnaweera et al, 2003]** A. Ratnaweera, S. Halgamuge and H. Watson. *Particle Swarm Optimization with Self-adaptive Acceleration Coefficients*. In Proceedings of the 1st International Conference on Fuzzy Systems and Knowledge Discovery 2002 (FSKD 2002), pp. 264-268, 2003.
- **[Tsou et Macnish, 2003]** D. Tsou and C. MacNish. *Adaptive Particle Swarm Optimisation for High-dimensional Highly Convex Search Spaces*. In Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003), Canbella, Australia. pp. 783-789, 2003.
- **[Zhang et al, 2003]** W. Zhang, Y. Liu and M. Clerc. *An Adaptive PSO Algorithm for Reactive Power Optimization*. In Advances in Power System Control Operation and Management, Hongkong. 2003.
- **[Krink et Løvbjerg, 2002]** T. Krink, and M. Løvbjerg. *The LifeCycle model: Combining Particle Swarm Optimisation, Genetic Algorithms and HillClimbers*. In Proceedings of Parallel Problem Solving from Nature VII, pp. 621-630, 2002.
- **[Veeramachaneni et al, 2003]**K. Veeramachaneni, T. Peram, C. Mohan and L. Osadciw. *Optimization Using Particle Swarm with Near Neighbor Interactions*. Lecture Notes Computer Science, vol. 2723, Springer Verlag, 2003.
- **[Venter et Sobieszczanski-Sobieski, 2002]** G. Venter and J. Sobieszczanski-Sobieski. *Particle Swarm Optimization*. In the 43rd AIAA/ASME/ASCE/AHA/ASC Structures, Structural Dynamics and Materials Conference, Denver, Colorado, USA, 2002.
- **[Veeramachaneni et al, 2003]** K. Veeramachaneni, T. Peram, C. Mohan and L. Osadciw. *Optimization Using Particle Swarm with Near Neighbor Interactions*. Lecture Notes Computer Science, vol. 2723, Springer Verlag, 2003.
- **[Robinson et al, 2002]** J. Robinson, S. Sinton and Y. Rahmat-Samii. *Particle Swarm, Genetic Algorithm, and their Hybrids: Optimization of a Profiled Corrugated Horn Antenna*. In IEEE International Symposium on Antennas & Propagation. San Antonio, Texas, USA, 2002.

- **[Eberhart et Shi, Comparaison 1998]** R. Eberhart and Y. Shi. *Comparison between Genetic Algorithms and Particle Swarm optimization*. In Proceedings of the Seventh Annual Conference on Evolutionary Programming, pp. 611-619. Springer-Verlag, 1998.
- **[Kennedy et Spears, 1998]** J. Kennedy and W. Spears. *Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator*. In IEEE International Conference on Evolutionary Computation, Achorage, Alaska, USA, 1998.
- **[Eberhart et Shi, évolution 1998]**R. Eberhart and Y. Shi. *Evolving Artificial Neural Networks.* In Proceedings of the International Conference on Neural Networks and Brain, Beijing, China, PL5-PL13, 1998.
- **[Van den Bergh et Engelbrecht 2000]** F. Van den Bergh and A. Engelbrecht. *Cooperative Learning in Neural Networks using Particle Swarm Optimizers*. South African Computer Journal, vol. 26, pp. 84-90, 2000.
- **[Ismail et Engelbrecht, 2000]** A. Ismail and A. Engelbrecht. *Global Optimization Algorithms for Training Product Unit Neural Networks*. In IEEE International Conference on Neural Networks, Como, Italy,2000.
- **[Kennedy et Eberhart, 1997]** J. Kennedy and R. C. Eberhart. *A Binary Version of the Particle Swarm Algorithm*. In Proceeding of on the conference on Systems, Man, and Cybernetics, pages 4104-4109, 1997
- **[Yoshida et al. 1999]** H. Yoshida, K. Kawata, Y. Fukuyama and Y. Nakanishi. *A Particle Swarm Optimization for Reactive Power and Voltage Control Considering Voltage Stability*. In Proceedings of the International Conference on Intelligent System Application to Power Systems, Rio de Janeiro, Brazil, pp. 117–121, 1999.
- **[Fukuyama et Yoshida 2001]** Y. Fukuyama and H. Yoshida. A Particle Swarm Optimization for Reactive Power and Voltage Control in Electric Power Systems. In Proceedings of the IEEE Congress on Evolutionary Computation, Seoul, S. Korea, pp. 87-93, 2001.
- **[Venter et Sobieszczanski Sobieski, 2002]** G. Venter and J. Sobieszczanski-Sobieski. *Particle Swarm Optimization*. In the 43rd AIAA/ASME/ASCE/AHA/ASC Structures, Structural Dynamics and Materials Conference, Denver, Colorado, USA, 2002.
- **[Al-Kazemi et Mohan 2000]** B. Al-kazemi and C. Mohan. *Multi-phase Discrete Particle Swarm Optimization*. In the Third International Workshop on Frontiers in Evolutionary Algorithms, Atlantic City, New Jersey, USA, 2000.
- **[Mohan et Al-Kazemi, 2001]** C. Mohan and B. Al-Kazemi. *Discrete Particle Swarm Optimization*. In Proceedings Workshop on Particle Swarm Optimization, Purdue School of Engineering and Technology, USA, 2001.
- **[Peer et al, 2003]** E. Peer, F. Van den Bergh and A. Engelbrecht. *Using Neighborhoods with the Guaranteed Convergence PSO*. In Swarm Intelligence Symposium, Piscataway, New Jersey, USA, pp. 235-242, IEEE Service Center, 2003.
- **[Higashi et Iba 2003]** N. Higashi and H. Iba. *Particle Swarm Optimization with Gaussian Mutation*. In Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003), Indianapolis, Indiana, USA. pp. 72-79, 2003.
- **[Esquivel et Coello Coello, 2003]** S. Esquivel and C. Coello Coello. *On the use of Particle Swarm Optimization with Multimodal Functions*. In Proceedings of IEEE Congress on Evolutionary Computation, pp 1130-1136, 2003.
- **[Navathe & Ra, 1989]** Navathe, S. B., & Ra, M. (1989). *Vertical partition for database design: A graphical algorithm*. ACM SIGMOD Record, 18(2), 440–450.
- **[Navathe et al, 1984]** Navathe, S. B., Ceri, S., Wiederhold, G., & Dou, J. (1984). *Vertical partition algorithms for database design*. ACM Transactions on Database Systems, 9(4), 680–710.

- **[S. Ceri et al, 1981]** S. Ceri, B. Pernici, and G. Wiederhold. *Optimization problems and solution methods in the design of data distribution*. Information Systems, 14(3):261-272, 1989.
- **[Hoffer & Severance, 1975]** Hoffer, J. A., & Severance, D. G. (1975). *The use of cluster analysis in physical database design*. Proceedings of the International Conference on Very Large Databases, pp. 69–86.
- **[Cornell & Yu 1990]** Cornell, D. W., & Yu, P. S. (1990). *An effective approach to vertical partition for physical design of relational database*. IEEE Transactions on Software Engineering, 16(2), 248–258.
- **[Chu & Ieong, 1992]** Chu, W. W., & Ieong, I. T. (1992). *A transaction-based approach to vertical partition for relational database systems*. IEEE Transactions on Software Engineering, 19(8), 804–812.
- **[Hammer & Niamir, 1979]** Hammer, M., & Niamir, B. (1979). *A heuristic approach to attribute partition*. Proceedings of ACM-SIGMOD, pp. 93–100.
- **[Song & Gorla 2000]** Song, S., & Gorla, N. (2000). *A genetic algorithm for vertical fragmentation and access path selection*. The Computer Journal, 43(1), 81–93.
- **[Ruskey, 1993]** Ruskey, F. (1993). *Simple combinatorial gray codes constructed by reversing sublists. Algorithms and Computation*, Lecture Notes in Computer Science 762, pp.201–208, Berlin Heidelberg New York: Springer.
- **[Chakravarthy et al, 1992]** Chakravarthy, S., Muthuraj, J., Varadarjan, R., & Navathe, S. B. (1992). *A formal approach to the vertical partition problem in distributed database design*. Technical Report, CIS Department, University of Florida, Gainesville, Florida.
- **[Kennedy et Eberhart, 1995]** J. Kennedy, R.C. Eberhart, *Particle swarm optimization*, in: Proceedings of the IEEE International Conference on Neural Networks, vol. IV, 1995, pp. 1942–1948.
- **[S. Ceri et al,1982]** S. ceri, M. Negri, et G. Belagatti. *Horizontal data partitioning in database design*. Proceding of ACM SIGMOD international conférence on management of Data. SIGPLAN Notices, pages 128-136, 1982.
- **[S. Ceri et G. Belagatti, 1984]** S. ceri, G. Belagatti. *Distibuted Databases : Principales and Systems*. McGraw Hill international Editions 1984.
- **[D. Cornell and P.S. Yu, 1987]** D. Cornell and P.S. Yu. *A vertical partitioning algorithm for relational Databases*. Proceedings of the third international Conference in data ingeneering, pages 30-35, 1987.
- **[S.B. Navathe et al, 1984]** S.B. Navathe, S. Ceri et G. WiederHold, Dou J. *A vertical partitioning algorithm for Databases design*. ACM transaction on database Systems, 9(4) : 681-710, December 1984.
- **[K. Karlapalem et al, 1994]** K. Karlapalem, S.B. Navathe, and M.M.A. Morsi. *Issues in distributed design of object-oriented databases*. In Distributed Object Management, pages 148-165. Morgan Kaufman Publishers Inc., 1994.
- **[C.I. Ezeifeet al, 1995]** C.I. Ezeife and K. Barker. *A comprehensive approach to horizontal class fragmentation in distributed object based system*. International Journal of Distributed and Parallel Databases, 247-272, 1995.
- **[Ozsu et Valduriez, 1991]** Ozsu, M. T., & Valduriez, P. (1991*). Principles of distributed database systems*. Prentice Hall.
- **[Collette, 2002]** Collette, Y. (2002). *Contribution à l'évaluation et au perfectionnement des méthodes d'optimisation multiobjectif*. Application à l'optimisation des plans de rechargement de combustible nucléaire. PhD thesis, Université de Paris XII Val-de-Marne.
- **[M-C. Wu et al, 1997]** M-C. Wu and A. Buchmann*. Research issues in data warehoursing*. In datenbanksustem inBuro, technik unk Wisssenscharft, 61-82, march 1997.

- **[K. Karlapalem et al, 1995]** K. Karlapalem, Q. Li. *Partitioning schemes for object oriented databases*. In Proceedings of the Fifth International Workshop on research Issues in Data Engineering- Distributed Object Management, RIDE-DOM'95, pages 42-49, March 1995.
- **[L. Bellatreche et al, 1999]**L. Bellatreche, K. Karlapalem, and G.B. Basak, *Query-driven horizontal class partitioning in object –oriented databases* in 9th International Conference on database and Expert Systems Applications (DEXA'98), pages 692-701, August 1998.
- **[L. Bellatreche et al, 1996]** L. Bellatreche, A. Simonet, and M. Simonet. *An algorithm for vertical fragmentation in distributed object database systems with complex attributes and methodes*. In International Workshop on Database and Expert Systems Applications (DEXA'96), Zurich, pages 15-21, September 1996.
- **[W.T. McCormicket al, 1972]**W.T. McCormick, P.J. Schweidtzer, and T.W. White. *Problem decomposition and data reorganization by a clustering technique*. Operation research, 993-1009, September 1972.
- **[P. O'Neil et al, 1997]** P. O'Neil and D. Quass. *Improved query performance with variant indexes.* Proceedings of the ACM SIGMOD International Conference on Management of Data, Pages 38-49 , May 1997.
- **[M. Golfarelli, 1999]** M. Golfarelli, D. Maio, and S. Rizzi. *Vertical fragmentation of views in relational data warehouses*. Procedings of Sistemi Evolui per Basu diDati, pages 19-33, 1999.
- **[A. Y. Noaman et al, 1999]** A. Y. Noaman and K Barker. *A horizontal fragmentation algorithm for the faxct relation in a distributed data warehouse*. In th e8th International Conference on Information and Knowledje Management(CIKM'99) pages 154-161, November 1999.
- **[G. Gardarin et al, 1990]** G. Gardarin an P. Valduriez. *SGBD AVANCEES: Bases de données objets, deductive, répartiés*. Edition Eyrolles, 1990.
- **[S. Chakravarthy et al, 1994]** S. Chakravarthy, J. Muthuraj, R. Varadarajan, and S.B. Navathe. *An Objective function for vertically partitioning relations in distributed databases and its analysis*. In Distributed and Parallel Databases Journal, 2(2): 183-207, April 1994.
- **[D. Tsichritzis, 1978]** D. Tsichritzis and A. Klug. *The ansi/x3/sparc framework*. Press, Montval, N.J., 1978.
- **[J. Darmont, 1999]** J. Darmont. *Etude des performances de méthodes de regroupement dynamique dans les bases de données orientées-objet*. Thèse de doctorat, Unversité Clermont-Ferrand II, Janvier 1999.
- **[H. Jagadish et al, 1999]** H. Jagadish , L. V. S. Lakshmanan, and D. Srivastava. *Snakes and sandwichs: Optimal clustering strategies for a data warehouse*. Proceedings of theACM SIGMOD International Conference on Management of Data, pages 37-48, June 1999.
- **[S.B. Navathe et al, 1995]** S.B. Navathe, K. Karlapalem, and M. Ra. *A mixed partitioning methodology for distributed databases design*. Journal of Computer and Software Engineering, 3(4): 395-426, 1995.
- **[M. J. Eisner, 1976]** M. J. Eisner and D. G. Severance. *Mathematical techniques for efficient record segmentation in large shared databases*. Journal of the ACM, 23(4):619-635, October 1976.
- **[J-C. Nicolas, 1991]** J-C. Nicolas. *Machines bases de données parallèles*: *Contribtion aux problems de la fragmentation et de la distribution*. Thèse de doctorat, Université des Sciences et Techniques de Lille, Flandres Artois, Janvier 1991.

Sites Internet:

1. Particle Swarm Central - a directory of resources pertaining to PSO
   http://www.projectcomputing.com/resources/psovis/index.html
2. Méthode des essaims particulaires
   http://www.particleswarm.net/
3. Séminaire OEP'2003
   http://www.particleswarm.net/oep_2003/
4. Groupe de travail sur les métaheuristiques
   http://www.lifl.fr/~talbi/META
5. Livre récent sur les métaheuristiques
   http://www.eyrolles.com/php.informatique/Ouvrages/ouvragem.php3?ouv_ean13=9782212113686
6. –Livre sur les métaheuristiques
   http://www.eyrolles.com/Informatique/Livre/9782212113686/
7. Livre sur la théorie de l'auto-organisation :
   http://www.scottcamazine.com/personal/selforganization/SOMain.html
8. Thèse sur la programmation à mémoire adaptative :
   http://ina.eivd.ch/collaborateurs/etd/articles.dir/IDSIA-52-98.ps
9. Méthode des essaims particulaires :
   http://www.particleswarm.info/
10. Groupe de travail sur les algorithmes évolutionnaires :
    http://afia.lri.fr/node.php?lang=fr&node=285
11. Méthode des colonies de fourmis :
    http://iridia.ulb.ac.be/~mdorigo/ACO/