REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE MOHAMED KHIDER

BISKRA

Faculté des sciences et des sciences de l'ingénieur

Département d'informatique

Mémoire de Magistère

Option : Intelligence Artificielle et Systèmes d'Information Avancés

Résolution des problèmes difficiles par optimisation distribuée

Présenté par :

Fouad Bekkari

Proposé par

Pr. Mohamed.C Batouche

Soutenu le : 18/02/2009

Dr. Chaouki Babahennini	Maître de Conférences	Université de Biskra	Président
Pr. Mohamed Batouche	Professeur	l'Université de Constantine	Rapporteur
Dr. Chikhi Salim	Maître de Conférences	l'Université de Constantine	Examinateur
Dr. Kazar Okba	Maître de Conférences	l'Université de Biskra	Examinateur
Dr. Chrif Foudil	Maître de Conférences	l'Université de Biskra	Examinateur

2008

Remerciements

Je tiens à remercier

Monsieur **Batouche Mohamed**, Professeur à l'Université de Constantine, pour m'avoir encadré pendant les années de ce travail. Je le remercie aussi pour m'avoir orienté vers un nouveau domaine de recherche.

Monsieur le docteur **Melkemi Kamel Eddine**, Docteur à l'Université de Biskra pour ses conseils fructueux et son soutien moral.

Monsieur **Chaouki Babahennini**, Maître de conférences à l'Université de Biskra, pour m'avoir fait l'honneur d'être président du jury.

Messieurs le Docteur **Chikhi Salim**, Maître de conférences à l'Université de Constantine, le Docteur **Kazar Okba**, et le Docteur **Chrif Foudil**, Maîtres de conférences à l'Université de Biskra, pour m'avoir fait l'honneur d'être membres de jury.

Le directeur du lycée IBN ELHAYTAM et le gérant de l'école ESIG pour m'avoir permis d'utiliser leurs salles machines.

Monsieur **Amrane Said**, Mme **Miloudi Samira**, Mme **Bekkari Om Elkhir**, Monsieur **Khlili Taher** pour la relecture de ce mémoire.

Dédicace

A

mes parents, père et mère

mes frères et sœurs :Ouafia, Messaoud, Amine,

Yacine et Sara.

ma femme Mouzdalifa

ma Douce Petite Perle Rayhana

mes tantes Maternelles Aicha et Halima

ma grande famille

la famille Miloudi

Monsieur Thabet Abd Elmadjid

mon frère Samir, mon cher ami Abd Elkamel, mes

amis Abd Elouaheb, Abd Eslam, Abd Ennour , Nour

Eddine, Lakhder,Nadjib, Saad et Toufik.

mes Collègues Nadjib, Hakim. Elhadj, Ahmed.

Résumé

L'hybridation et le parallélisme des métaheuristiques sont utilisés avec succès pour résoudre les problèmes difficiles de grande taille. Les métaheuristiques parallèles et hybrides exigent une énorme puissance de calcul. Les grilles de calcul représentent un environnement distribué qui peut offrir cette puissance. Les grilles sont caractérisées par leur hétérogène, volatilité des ressources et leur temps de latence. Ces caractéristiques doivent être respectées lors de la conception des méthodes parallèles et hybrides.

Ce travail présente une méthode qui résulte de l'hybridation entre l'optimisation par essaim de particules et la recherche local itérée. Cette méthode est souhaitée être adaptée à un environnement grille de calcul.

Mots clés :

Problème difficile, métaheuristique, parallélisme, hybridation, grille de calcul, optimisation par essaim de particules, recherche locale itérée.

Abstract

The hybridization and parallelism of metaheuristics are used successfully to solve large difficult problems. Parallel hybrid metaheuristics require enormous computing power. The grids are a distributed environment that can provide that power. The grids are characterized by their diverse, volatility of its resources and time latency. These features must be respected when designing methods and parallel hybrids.

This work presents a result of hybridization between the particle swarm optimization and local search iterated. This method required to be adapted to grid computing environment

Keywords:
difficult problem, Métaheuristics, parallelism, hybridization, grid computing, particle swarm optimization, iterated local search.

Table de matières

Liste de figures

Introduction

Contexte de travail

Ce travail s'inscrit dans le domaine de résolution des problèmes d'optimisations difficiles avec l'utilisation des méthodes approximatives dites métaheuristiques. Dans ce type d'approches, on cherche à trouver des solutions pour des problèmes exagérément en terme de temps de calcul pour les résoudre. Généralement, on peut dire qu'un problème est difficile si on ne peut pas localiser la meilleure solution de ce problème dans un temps raisonnable.

Deux approches sont généralement utilisées. La première cherche à trouver des solutions exactes, tandis que la seconde cherche à rapprocher le plus possible à la solution optimale, cette approche est concrétisée par des méthodes dites heuristiques, parmi elles, il existe des heuristiques applicables pour la majorité des problèmes d'optimisation, on les appelle les métaheuristiques.

Motivations

Ce travail est guidé par deux motivations principales :

La première est que les métaheuristiques classiques ne peuvent pas trouver de bonnes solutions pour des problèmes difficiles à grande taille d'une manière efficace. Afin de pallier à ce problème deux techniques sont actuellement utilisées, on parle du parallélisme et d'hybridation des métaheuristiques. Les méthodes basées sur l'hybridation et le parallélisme des métaheuristiques demandent une puissance du calcul considérable.

La seconde motivation est représentée par l'émergence de la plateforme dite grille. La grille est une large collection des dispositifs informatique dispersés sur une grande surface et connectés via un réseau WAN. Cette plateforme offre une très grande puissance de calcul et/ou de stockage, mais elle a doté d'un ensemble de caractéristique qui rend la conception et l'implémentation des applications parallèles sur une telle plateforme plus difficile qu'une plateforme simple de calcul distribué.

Dans ce travail le but est d'étudier les techniques de parallélisme et d'hybridation des métaheuristiques, ensuite proposer une méthode parallèle et hybride capable d'exploiter la puissance donnée par la grille. Elle doit être adaptée aux caractéristiques de celle-ci.

Organisation du travail

Dans le premier chapitre, on commence par quelques définitions relatives aux problèmes d'optimisation, optimisation difficiles, métaheuristiques et autres ; puis une

classification de métaheuristiques ; finalement on termine par une vue générale sur les métaheuristiques les plus connues.

Dans le deuxième chapitre, on donne deux classifications qui englobent les manières ou les sortes du parallélisme et l'hybridation des métaheuristiques. Généralement on donne les critères de classification puis nous citons les types, les modèles et les classes.

Le troisième chapitre, commence par donner une définition des grilles, suivie par leurs caractéristiques, types et architecture. Après avoir présenté un état de l'art sur l'utilisation des grilles de calcul dans le domaine d'optimisation des problèmes difficiles.

Au début du quatrième chapitre, on propose une méthode coopérative, parallèle et hybride ; elle est le résultat d'une hybridation entre un modèle parallèle d'optimisation par essaime de particules dite le modèle cellulaire avec une méthode de recherche locale, dite recherche locale itérée.

Chapitre I

Les problèmes difficiles et les métaheuristiques

État de l'art

1 Introduction

La réussite d'un projet ou le développement d'une entreprise est dans plusieurs cas lié à la capacité des ingénieurs et des décideurs de résoudre des problèmes de type : minimiser les coûts ou maximiser la production, avec la prise en compte d'un certain nombre de paramètres. Ce type de problème appelé problème d'optimisation, où on veut minimiser une fonction de coût ou maximiser une fonction objectif.

Les problèmes d'optimisation apparaissent dans plusieurs domaines, tels que la conception de systèmes mécaniques, le traitement des images, l'électronique ou la recherche opérationnelle. Résoudre un tel problème consiste à donner les bonnes valeurs aux paramètres pour avoir une solution optimale. Ces valeurs doivent respecter les contraintes associées au problème.

2 Problèmes d'optimisation difficiles

Dans la littérature, deux sortes de problèmes reçoivent cette appellation, non définie strictement (et liée, en fait, à l'état de l'art en matière d'optimisation) [SDPT03]:

- Certains problèmes d'optimisation discrète, pour lesquels on ne connaît pas d'algorithme exact polynomial (c'est à dire dont le temps de calcul est proportionnel à N^n, où N désigne le nombre de paramètres inconnus du problème, et n est une constante entière). C'est le cas, en particulier, des problèmes dits « NP-difficiles », pour lesquels on suppose qu'il n'existe pas de constante n telle que le temps de résolution soit borné par un polynôme de degré n.

- Certains problèmes d'optimisation à variables continues, sont ceux pour lesquels on ne connaît pas d'algorithme permettant de repérer un optimum global (c'est à dire la meilleure solution possible) à coût sûr et en un nombre fini de calcul.

3 Méthodes de résolutions

Il existe généralement deux approches pour la résolution des problèmes difficiles : une approche de résolution exacte et une approche de résolution approximative. Dans la première on cherche à trouver la solution exacte d'un problème donné, les algorithmes de type B&X représentent cette approche. Pratiquement l'application d'un tel mode de résolution est limitée pour les petites instances des problèmes difficiles. La deuxième tendance est la résolution approximative ; là, on cherche à trouver de bonnes solutions en se basant sur des méthodes stochastiques dites *heuristiques* ; parmi ces heuristiques on trouve des méthodes assez générales pour être appliquées sur un grand spectre de problèmes, ces méthodes sont les *métaheuristiques*.

4 Les métaheuristiques

4.1 Définition

Le mot métaheuristique est composé de deux mots ; le mot méta qui est un préfixe signifiant "au-delà" ou bien "dans un niveau supérieur"; et le mot heuristique qui vient du verbe heuriskein et qui signifie 'trouver' [HGH99].

Dans la littérature on trouve des définitions qui expliquent bien la notion de ' métaheuristiques', citons :

"A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."[1]

"A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a constructive method."[2]

"A **metaheuristic** is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems

[1] I.H. Osman et G. Laporte, Metaheuristics: a bibliography. Annals of Operations Research 63, 513-623, 1996
[2] S. Voß, S. Martello, I.H. Osman et C. Roucairol (Eds), Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, Dordrecht, The Netherlands, (1999)

with relatively few modifications to make them adapted to a specific problem. Examples of metaheuristics include simulated annealing (SA), tabu search (TS), iterated local search (ILS), evolutionary algorithms (EC), and ant colony optimization (ACO)."[1]

A partir de ces définitions, on peut dire que les métaheuristiques sont des stratégies qui permettent de guider la recherche d'une solution optimale, leur but est de faire explorer l'espace de recherche d'une manière efficace pour atteindre des solutions presque optimales ; elles sont en général non déterministes (pas de garantie), généralement, elles contiennent des techniques pour échapper de l'optimum local, leurs concepts de base peuvent être décrits de manière abstraite (indépendamment d'un problème spécifique). Même si on utilise des heuristiques qui tiennent compte de la spécificité du problème traité, ces heuristiques resteront contrôlées par une stratégie de niveau supérieur [HGH99].

4.2 Classification

On peut classer les métaheuristiques selon plusieurs points de vue ; par conséquent on peut trouver une métaheuristique qui appartient aux différentes classes selon le critère de la classification [BPSV01], donc on a comme critères :

- **Le nombre des solutions manipulées à la fois** : selon ce point de vue, on distingue deux classes des métaheuristiques :
  - Les métaheuristiques basées sur la manipulation (évaluation et modification) d'une solution à la fois. Dans ce type de méthodes la notion de voisinage d'une solution est pertinente. On trouve dans cette classe la méthode de recuit simulé, la recherche tabou, la recherche locale itérée, la recherche avec voisinage variable et autres.
  - Les métaheuristiques basées sur la manipulation d'un ensemble de solutions à la fois dite généralement population de solutions ; citons dans cette classe les algorithmes génétiques, l'optimisation par colonies de fourmis et l'optimisation par essaim des particules.
- **Type de parcours** : les métaheuristiques sont des méthodes qui explorent l'espace de recherche soit avec le déplacement dans l'ensemble de voisinage ou avec l'échange de l'état des individus d'une population. Ce déplacement forme une sorte de parcours qui peut être continu comme dans le recuit simulé et la recherche tabou ; ou discontinu comme dans la recherche locale itérée, les algorithmes génétiques, l'optimisation par colonies de fourmis et GRASP.

[1] Metaheuristics Network Website http://www.metaheuristics.net visité en septembre 2008

- **Emploi de mémoire** : l'emploi de la mémoire signifie l'utilisation de l'historique de la recherche pour déduire l'état courant de l'optimisation. Dans ce contexte, on peut distinguer deux types : les métaheuristiques dites ''sans mémoire'' où l'état actuel de la recherche est déterminé uniquement suivant l'état précédent, c'est le cas de la descente récursive, le recuit simulé. Par contre, il existe des méthodes qui utilisent l'historique pour guider la recherche telle que la recherche tabou. la mémoire peut être figurée selon différentes formes, par exemple, on peut considérer que l'ensemble des individus d'un algorithme génétique forme une mémoire, la phéromone dans l'optimisation par colonies de fourmis aussi et les positions dans l'optimisation par essaim des particules.
- **La Source de la méthode**: on trouve généralement deux classes : d'un côté les méthodes inspirées de la nature tel que le recuit simulé, les algorithmes génétiques, l'optimisation par colonies de fourmis et l'optimisation par essaim des particules, et de l'autre côté, on trouve des méthodes qui ont des origines non naturelles, telle que la recherche avec tabou, la recherche avec voisinage variable, la recherche locale itérée et GRASP.
- **Nombre et structure de voisinage :** le voisinage est l'ensemble des solutions (états) atteignables à partir de la solution (état) courante ; le voisinage peut être fixe ou variable. Les métaheuristiques, selon ce critère, sont classifiées en deux : les métaheuristiques avec un voisinage fixe, c'est le cas de la majorité des heuristiques ; et les métaheuristiques avec un voisinage variable, c'est le cas de la recherche avec voisinage variable.

5 Les méthodes basées sur une seule solution

Appelées aussi méthodes de trajectoire, le processus de recherche trace une sorte de chemin ou un parcours le long de leur progression, il se déplace d'une solution vers une autre dans *l'espace de recherche* ; ce déplacement est possible grâce à la notion de voisinage. Les sections suivantes donnent une vision sur les méthodes de recherche locale les plus connues, de la descente récursive qui représente une méthode de base au recuit simulé et la recherche tabou.

5.1 La descente récursive

C'est un exemple typique des méthodes de recherche locale, elle progresse à travers l'ensemble des solutions X par le choix de la meilleure solution voisine de la solution courante et ainsi de suite ; ce processus s'interrompt lorsqu'un minimum local est atteint [HGH99].

Cette méthode est caractérisée par sa simplicité mais elle a deux inconvénients :

- Suivant la taille et la structure du voisinage N(s) considérées, la recherche de la meilleure solution voisine est un problème qui peut être aussi difficile que le problème (P) initial.

- elle est incapable de progresser au-delà du premier minimum local rencontré. Par contre les problèmes d'optimisation combinatoire comportent en générale plusieurs optima locaux pour lesquels la valeur de la *fonction objectif* peut être fort éloignée de la valeur optimale.

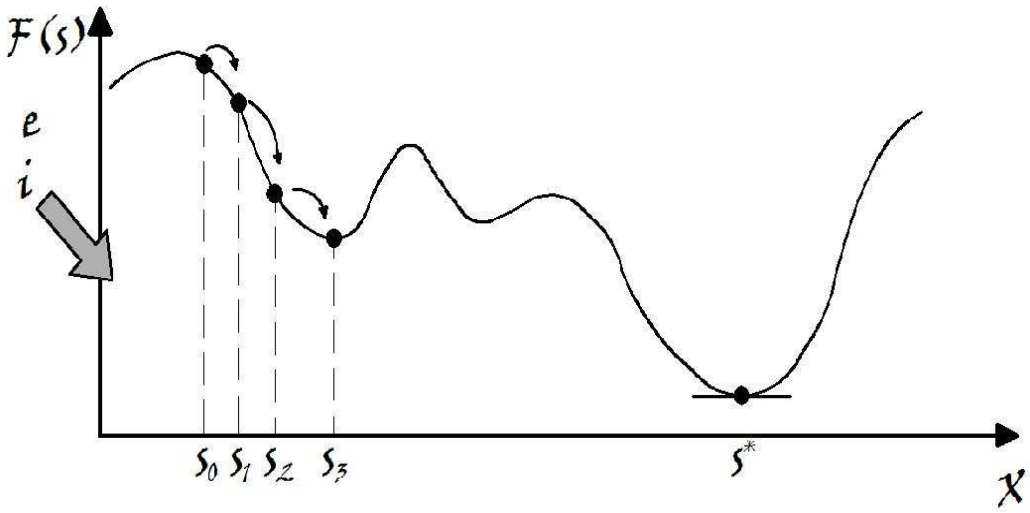

Figure 1. 1 L'algorithme de descente ne peut pas remonter pour sortir d'un optimum local

Pour palier à ces problèmes, l'idée est de faire accepter des solutions moins bonnes (pour échapper du minimum local); en plus on exploite une portion de voisinage d'une solution pour faire diminuer la taille du problème; ce que font les autres méthodes de recherche locale [HGH99].

5.2 Le recuit simulé

Le recuit simulé (SA) est parmi les plus vieilles métaheuristiques et sûrement l'un des premiers algorithmes qui ont une stratégie explicite pour échapper de minima locaux [BR03]. Les origines du recuit simulé remontent aux expériences réalisées par Metropolis et al [KGV83]. Dans les années 50 pour simuler l'évolution d'un tel processus de recuit physique [Wid01] ; 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. Soit ΔE la

différence d'énergie occasionnée par une telle perturbation. Le nouvel état sera accepté si l'énergie du système diminue ($\Delta E<0$). Sinon, il sera accepté avec une probabilité définie par :

$$\mathbf{p(\Delta E,T) = exp^{(-\Delta E/(Cb \cdot T))}}$$

Où T est la température du système et Cb est un constant physique connu sous le nom de *constant de Boltzmann*.

Le recuit simulé est une méthode de recherche locale dont le mécanisme de recherche est calqué sur l'algorithme de Metropolis, et les principes du recuit thermodynamique ; il a été, la première fois, présenté comme un algorithme de recherche pour les problèmes d'optimisation combinatoire par S. Kirkpatrick et autres [KGV83]. L'idée fondamentale est de permettre au processus de recherche de se déplacer vers des solutions ayant une qualité plus mauvaise que la qualité de la solution courante (mouvements ascendants) ; afin de s'échapper du minimum local. La probabilité de faire un tel déplacement est diminuée pendant la recherche.

L'idée consiste à utiliser l'algorithme de Metropolis pour atteindre des états avec une énergie aussi faible que possible. Le refroidissement progressif d'un système de particules est simulé en faisant une analogie entre l'énergie du système et la fonction objectif du problème d'une part, et entre les états du système et les solutions admissibles d'autre part [Wid01]. Notons que chaque état du système représente une solution potentielle du problème.

5.2.1 Déroulement

Initialement, on donne au système une très haute température ; puis on le refroidit petit à petit. Le refroidissement du système doit se faire très lentement pour avoir l'assurance d'atteindre un état d'équilibre à chaque température T.

Le voisinage N(s) d'une solution S_X s'apparente à l'ensemble des états atteignables depuis l'état courant en faisant subir des déplacements infinitésimaux aux atomes du système physique.

A chaque itération, une seule solution voisine s' est générée et elle est acceptée si elle est meilleure que la solution courante, dans le cas contraire on a les cas suivants :

- si T est grande, $\exp^{(-\Delta E/T)}$ est de l'ordre de 1, on garde toujours le mouvement, même s'il est mauvais.
- si T est très petit, $\exp^{(-\Delta E/T)}$ est de l'ordre de 0, donc les mouvements qui augmentent l'énergie (la différence) seront disqualifiés.

Méthode : on tire au hasard un nombre dans l'intervalle [0,1[, si le nombre est < $\exp^{(-\Delta E/T)}$, on garde la solution sinon on rejette.

La meilleure solution trouvée est mémorisée dans la variable s* [Wid01].

La méthode recuite simulée donne de très bons résultats pour le problème de voyageur de commerce mais pas pour le problème d'affectation ; leur performance liée au schéma de refroidissement [Wid01].

```
Choisir une solution initiale S
Choisir une température initiale Ti > 0
Choisir une température finale Tf > 0 // Tf < Ti
Choisir un nombre d'itération NB (à une température Fairennée)
Choisir le coefficient de diminution de la température F ∈ [0,1[
T ← Ti
TQ (Tf < T) faire
Pour (k ← 1 à NB)
        S' ← voisin aléatoire de S
        Δ ← f(S') - f(S)
        Si (Δ ≤ 0) alors
                S ← S'
                Si non
                S ← S'  avec la probabilité e-Δ/T
        Fin si
Fin pour
T ← T *Φ
Fin TQ
```

Figure 1. 2 pseudo code de la méthode du recuit simulé

5.2.2 Les applications de recuit simulé

Cette méthode est très utilisée dans les milieux industriels, parmi les applications citons :

- l'optimisation combinatoire
- La CAO : conception des circuits et installation des composants
- le traitement d'images (réstituation d'images brouillées)

Elle est aussi appliquée dans les domaines de la physique, la biologie, la géophysique, la finance, ...etc.

5.2.3 Les avantages de recuit simulé:

Parmi les avantages de la méthode, on cite:

- Traite les fonctions de coût avec les degrés tout à fait arbitraires de non linéarité, la discontinuité, et l'imprévisibilité.
- Processus assez arbitraire sur les conditions limites et les contraintes imposées sur les fonctions de coût.
- Simple à implémenter
- Garantie statistique de trouver une solution optimale

5.2.4 Les inconvénients de recuit simulé

La méthode comprend quelques inconvénients parmi lesquels :

- le non déterminisme
- Difficulté de choisir les paramètres efficaces ; par exemple le schéma de refroidissement.
- Le compromis entre la vitesse et l'optimisation.

5.3 La Recherche Tabou

La recherche tabou ou recherche avec tabou est l'une des métaheuristiques les plus utilisées dans le domaine de l'optimisation des problèmes difficiles. Elle est introduite par Glover (1986) basé sur ses premières idées formulées en (1977). La méthode de recherche tabou utilise l'historique de la recherche pour implémenter un mécanisme permettant d'échapper de l'optimum local et aussi d'éviter les cycles dans la recherche. Ce qui donne comme résultat une bonne exploration de l'espace de recherche.

5.3.1 Principe et démarche

Initialement, la méthode de recherche avec tabou se comporte comme la descente récursive. Elle applique une recherche locale basée sur la règle d'amélioration de la fonction objectif, c'est-à-dire choisir à partir de l'ensemble des voisins de la solution courante, la meilleure solution, et qui doit être mieux que la solution courante. Lorsqu'on atteint un optimum local, la recherche tabou permet d'accepter la meilleure solution qui appartient à l'ensemble des voisines même si elle a une qualité inférieure de la solution actuelle. En d'autres mots, on a choisi le moins mauvais. La recherche tabou ne s'arrête donc pas au premier optimum trouvé.

Une telle méthode basée sur une telle stratégie peut fortement tomber dans une recherche cyclique ; c'est-à-dire si un minimum local S se trouvant au fond d'une vallée

profonde est atteint , il sera impossible de ressortir de celle-ci en une seule itération, et un déplacement de la solution s vers une solution s'ϵ N(s) avec f(s') > f (s) pourra provoquer le déplacement inverse à l'itération suivante, puisqu'en général s ϵ N(s') et f(s)<f(s'); dans ce cas on a un cycle de longueur 2 : $s \rightarrow s' \rightarrow s \rightarrow s'$ [HGH99].

Pour régler ce problème, une mémoire à court terme est mise en application, cette mémoire stocke les dernières meilleures solutions déjà visitées sous forme d'une liste dite liste tabou, où tout mouvement vers une solution qui appartient à cette liste est interdit. À chaque itération, la meilleure solution de l'ensemble des solutions permises (solutions voisines de la solution courante et n'appartenant pas à la liste tabou) est choisie comme une nouvelle solution courante. En plus, cette solution est ajoutée à la liste de tabou et une des solutions qui étaient déjà dans la liste de tabou est enlevée (habituellement dans un ordre de FIFO) [BR03].

L'utilisation de la mémoire permet d'éviter tout cycle de longueur inferieur ou égale à la taille de la liste tabou, par conséquent, elle oblige le processus de recherche à explorer de nouvelles régions dans l'espace de recherche. Le pseudo-code de l'algorithme tabou classique est présenté dans la figure (1.3).

La mémorisation des solutions entières sera trop coûteuse en terme d'espace et en terme de gestion de la mémoire et ne sera pas la plus efficace. Pour cela la liste tabou mémorise des attributs de solutions au lieu de solutions complètes. Les attributs sont habituellement des composants des solutions, des mouvements ou des différences entre deux solutions. Notons que plus d'un attribut peut être considéré ; dans ce cas une liste tabou est présentée pour chacun d'eux. L'ensemble d'attributs et les listes tabou correspondantes définissent les conditions tabou qui sont employées pour filtrer le voisinage d'une solution et pour produire l'ensemble des solutions permises. Le stockage des attributs des solutions au lieu des solutions complètes est beaucoup plus efficace, mais il présente une perte d'information, car la mémorisation d'un attribut dans la liste de tabou peut interdire plus d'une solution [BR03], un exemple simple de mémorisation des attributs est présenté dans [HGH99].

5.3.2 Fonction d'aspiration

A un moment donné, les mécanismes employés réduisent l'espace de recherche et interdisent au processus de recherche d'explorer des zones qui peuvent être intéressantes, pour cette raison un mécanisme dit aspiration ou fonction d'aspiration est utilisé ; il permet de ressortir une solution de la liste tabou et l'utilise comme une solution candidate à la prochaine

itération. Cette solution devra satisfaire un critère d'aspiration (f(s')<A(f(s))) [Wid01] ; la fonction la plus simple consistera à enlever le statut tabou d'un mouvement si ce dernier permet d'atteindre une solution de qualité supérieure à celle de la meilleure solution trouvée jusqu'alors [HGH99].

```
s ← solution aléatoire
fmin ← f(s)
s* ← s
TABOU ← liste de solutions s', de longueur L
TABOU ← VIDE
REPETER
        Générer un N-échantillon TEL QUE s' ∈
        voisinage N(s) et {s, s'} ∉ TABOU ou satifie le
        critère d'aspiration
        f(s') ← min[f(s')]
        Ajouter ({s, s'}, TABOU)
        s ← s'
        SI f(s) < fmin
                fmin ← f(s)
                s* ← s
        FIN DE SI
JUSQU'A conditions d'arrêt satisfaites
FIN.
```

Figure 1. 3 pseudo code de la méthode recherche tabou

5.3.3 Intensification et diversification

L'intensification et la diversification sont deux techniques (parmi d'autres) utilisées pour améliorer la recherche avec tabou. L'intensification consiste à explorer en détail une région de l'espace de recherche qui comprend des solutions qui partagent quelques propriétés ; l'idée est d'élargir l'ensemble du voisinage de la solution courante. Par contre la diversification consiste à diriger l'exploration vers des régions inexplorées de l'espace de recherche ; il faut simplement faire redémarrer le processus de recherche avec une solution initiale aléatoire ou choisie judicieusement dans une région non encore visitée de l'ensemble des solutions admissibles [Wid01].

5.3.4 Applications de la recherche tabou

On a dit que la méthode de recherche tabou est l'une des métaheuristiques les plus utilisées, elle a été appliquée dans plusieurs domaines : dans le domaine de la télécommunication (conception des réseaux…), la conception (conception assistée par ordinateur, conception des réseaux de transport,…..), le routage (routage des véhicules,

voyageur de commerce…), optimisation des graphes (coloration des graphes..), les problèmes d'allocation (problème d'assignement quadratique…) et autres [Glo95].

6 Les méthodes basées sur une population des solutions

Les métaheuristiques basées sur la manipulation d'une population des solutions essayent d'augmenter la qualité moyenne d'un ensemble des solutions via un mécanisme défini. Généralement ces méthodes sont inspirées de la nature, du domaine de la biologie, l'éthologie et autres.

On distingue deux approches utilisées par les méthodes basées sur une population de solutions : la première est l'utilisation des mécanismes d'évolution naturels, cette approche est représentée par les AGs, EV, et autres. La deuxième est l'utilisation de l'intelligence collective, cette approche est représentée par ACO, PSO et autres.

Dans les deux sections suivantes, on va donner un exemple de chacune de ces stratégies qui sont les AGs et l'ACO.

6.1 Les Algorithmes Génétiques

Les algorithmes génétiques (AGs) appartiennent à un ensemble de méthodes dites algorithmes évolutifs. Le terme algorithmes évolutifs englobe toute une classe de métaheuristiques. Ces algorithmes sont basés sur le principe du processus d'évolution naturelle. Un algorithme évolutif typique est composé de trois éléments essentiels [HGH99]:

1. une *population* constituée de plusieurs individus, chacun représente une solution potentielle du problème donné.
2. un *mécanisme d'évaluation* de l'adaptation de chaque individu de la population à l'égard de son environnement extérieur.
3. un *mécanisme d'évolution* composé d'opérateurs permettant d'éliminer certains individus et de produire de nouveaux individus à partir des individus sélectionnés ; les AGs s'appuient sur des techniques dérivées de la génétique : croisements, mutations, sélection, etc.

Les GAs ont été inventés par John Holland dans les années 60 et ont été développés par Hollande et ses étudiants et collègues à l'université du Michigan dans les années 60 et les années 70. Contrairement aux stratégies d'évolution et programmation évolutionnaire, le but original de Holland n'était pas de concevoir des algorithmes pour résoudre des problèmes spécifiques, mais plutôt pour étudier formellement le phénomène de l'adaptation, et comment

il se produit dans la nature ; et pour développer des méthodes dont les mécanismes de l'adaptation normale pourraient être importés dans les systèmes informatiques. En 1975, Holland présente dans son livre *Adaptation in Natural and Artificial Systems* les algorithmes génétiques comme une abstraction de l'évolution biologique, et il a donné un cadre théorique pour l'adaptation basé sur les algorithmes génétiques [Mel99].

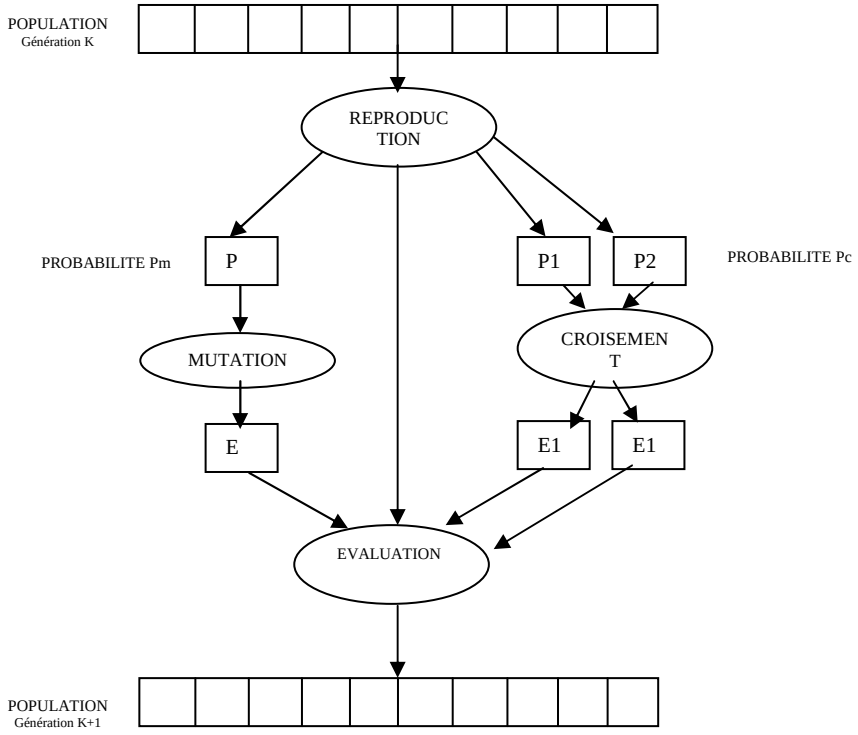

Figure 1. 4 principe et démarche des algorithmes génétiques

6.1.1 Déroulement

Dans les algorithmes génétiques, on essaye de simuler le processus d'évolution d'une population. Au début on génère une population de solutions d'une manière aléatoire, cette population forme la population initiale. Le degré d'adaptation de chaque individu à l'environnement (exprimé par la valeur de la fonction coût dite fonction fitness) est calculé. Après, des couples des individus P1 et P2 (appelés parents) sont sélectionnés en fonction de leurs adaptations ; ensuite un opérateur de croisement est appliqué sur P1 et P2 avec une probabilité P_c et génère des couples C_1 et C_2 (dits enfants). D'autres individus P sont sélectionnés en fonction de leur adaptation ; un opérateur de mutation est appliqué sur P avec une probabilité P_m (Pm est généralement très inférieur à P_c) et génère des individus mutés P_0. Le niveau d'adaptation des enfants (C_1, C_2) et des individus mutés P_0 sont ensuite évalués avant de les insérer dans la nouvelle population ; l'insertion est faite par le choix de N

individus parmi la population des parents et la population des enfants. En itérant ce processus jusqu'à ce qu'un critère d'arrêt soit atteint. Un critère d'arrêt peut être un nombre fixe d'itérations ou lorsque la population n'´evolue pas d'une manière plus ou moins suffisamment rapide [Dur04] ; la figure (1.4) montre ce déroulement.

6.1.2 Le codage utilisé par les algorithmes génétiques

Un algorithme génétique repose sur l'utilisation d'un codage des informations du problème à résoudre, afin qu'il puisse appliquer les opérateurs génétiques sur ces informations ; il existe plusieurs manières pour coder les données du problème, entre autres on trouve le code binaire naturel, le code binaire de Gray et le code réel.

Le codage en binaire :

Dans ce type de codage, pour un individu on code ses variables et on les concatène, par exemple, la chaîne binaire 1000| 0110| 1101 correspond à un individu défini par 3 variables (8, 6, 13) en codage binaire naturel sur 4 bits chacune.

C'est le codage le plus utilisé pour plusieurs raisons : pour des raisons historiques, ce codage a été utilisé par J. Holland et ses étudiants ; plusieurs résultats théoriques sont basés sur ce codage, et il est facile de mettre en place les opérateurs génétiques avec ce codage. Cependant si la longueur de la chaîne augmente la performance de l'algorithme diminuera.

Le codage réel

Il s'agit de concaténation des variables x_i d'un individu x. par exemple un individu x (25, 31, 8) est codé 25| 31| 8 ; ce codage est plus précis que le codage binaire, l'espace de recherche est le même que l'espace du problème, l'évaluation de la fonction coût est plus rapide ; mais son alphabet est infini et il a besoin d'opérateurs appropriés.

6.1.3 Les opérateurs génétiques

Dans un algorithme génétique il existe trois principaux opérateurs : l'opérateur de sélection, l'opérateur de croisement, et l'opérateur de mutation. Ces opérateurs jouent un rôle prépondérant dans la possible réussite d'un AG.

L'opérateur de sélection

C'est le choix des individus pairs qui vont participer à la reproduction de la génération suivante. Il existe plusieurs méthodes pour la reproduction. Comme il ya plusieurs méthodes pour sélectionner un individu ; la méthode la plus connue et utilisée est *la roue de loterie*

biaisée (*roulette wheel*) de Goldberg (1989). Selon cette méthode, chaque chromosome sera dupliqué dans une nouvelle population proportionnellement à sa valeur d'adaptation. Donc si on considère la fitness d'un chromosome particulier, comme $f(d(ci))$, la probabilité avec laquelle il sera réintroduit dans la nouvelle population de taille N est :

$$\frac{f(d(c_i))}{\sum_{j=1}^{N} f(d(c_j))}$$

Dans ce cas les individus ayant une grande fitness ont donc plus de chance d'être sélectionnés [VY01] ; cette stratégie pose le problème de convergence prématurée si la population contient un super individu qui ne sera pas forcément un optimum global ou proche de l'optimum global.

Il existe d'autres méthodes, par exemple celle du *tournoi* (*tournament selection*) : on tire deux individus aléatoirement de la population et on reproduit le meilleur des deux dans la nouvelle population. On refait cette procédure jusqu'à ce que la nouvelle population soit complète. Cette méthode donne de bons résultats [VY01].

L'opérateur de croisement

Cet opérateur permet de créer de nouveaux individus ; il permet en quelque sorte l'échange d'information entre les individus.

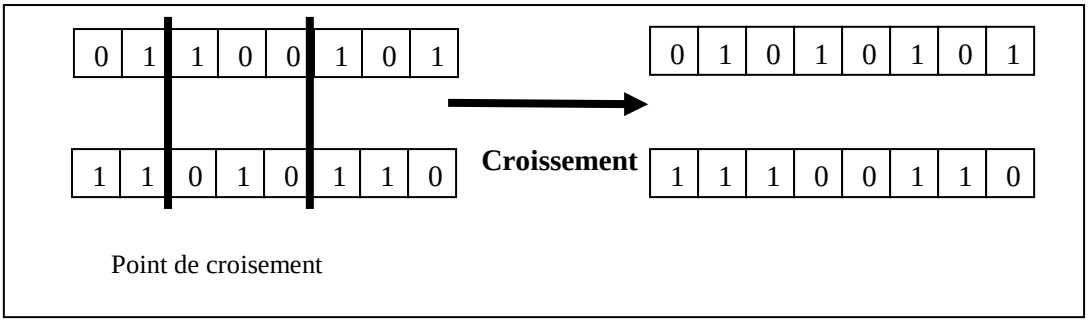

Figure 1. 5 principe de l'opérateur de croisement L'opérateur de mutation

Au début, on tire au hasard deux individus de la population courante, et on définit aléatoirement un ou plusieurs points de croisement. Ces derniers marquent des sites de croisement. Après, selon une probabilité Pc, les segments des deux parents sont alors échangés autour de ces sites (voir figure 1.5) ; le résultat est la création de deux nouveaux individus (dans le cas d'un site de croisement). Notons qu'un individu sélectionné ne subit pas forcément l'action d'un croisement. Ce dernier ne s'effectue qu'avec une certaine probabilité. Plus cette probabilité est élevée et plus la population subira de changement [Dur04].

La mutation consiste à changer aléatoirement un composant (gène) de l'individu, cette opération est faite avec une certaine probabilité Pm (généralement inférieure de la probabilité de croisement Pc), par exemple, si on utilise un codage binaire, un composant qui porte la valeur 1, on lui affecte une valeur 0 et vice versa (voire la figure).

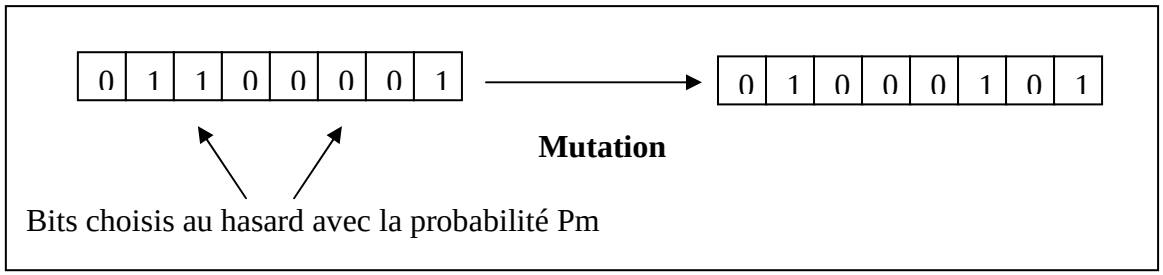

Figure 1. 6 principe de fonctionnement de l'opérateur de mutation

La mutation est un mécanisme qui protège les algorithmes génétiques contre les pertes prématurées d'informations pertinentes, ces informations ont pu être perdues lors de l'opération de croisement. La mutation maintient aussi la diversité de l'algorithme pour l'exploration des nouvelles régions [Dur04].

6.1.4 Les avantages des algorithmes génétiques

Les algorithmes génétiques ont plusieurs avantages, citons:

- Ils sont adaptables à plusieurs types des problèmes
- Robustes
- Faciles à implémenter
- Faciles à hybrider
- Faciles à paralléliser

6.1.5 Les inconvénients des algorithmes génétiques:

Parmi les inconvénients des algorithmes génétiques, on note :

- Pas de garantie de convergence
- Temps de calcul important (en cas où la taille de population est grande)

6.2 L'optimisation par colonies de fourmis

L'optimisation par colonies de fourmis ou Ant Colony Optimization (ACO) est une méthode d'optimisation basée sur la manipulation d'un ensemble de solutions ; cette méthode représente toute une classe des métaheuristiques qui reposent sur la notion de *l'intelligence*

collective. La solution finale est plus complexe que celle d'un composant simple. Plusieurs mots apparaissent dans ce domaine : l'auto-organisation, émergence et autres.

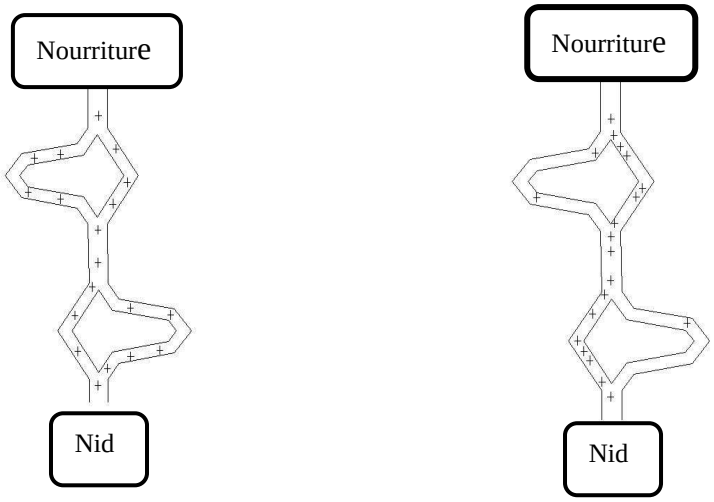

Figure 1. 7 l'influence de l'expérience sur le choix des fourmis

6.2.1 L'origine de la méthode

Les fourmis ont la particularité d'employer pour communiquer des substances volatiles appelées *phéromones*. Elles sont très sensibles à ces substances qu'elles perçoivent grâce à des récepteurs situés dans leurs antennes. Ces substances sont nombreuses et varient selon les espèces. Les fourmis peuvent déposer des phéromones au sol, grâce à une glande située dans leur abdomen, et former ainsi des pistes odorantes, qui pourront être suivies par leurs congénères [SDPT03].

Les fourmis utilisent les pistes de phéromones pour marquer leur trajet, par exemple entre le nid et une source de nourriture. Une colonie est ainsi capable de choisir (sous certaines conditions) le plus court chemin vers une source à exploiter, sans que les individus aient une vision globale du trajet [SDPT03].

6.2.2 Description et algorithme

Pour bien comprendre comment fonctionne l'optimisation par colonie de fourmis, on va retourner vers le premier algorithme proposé, c'est l'algorithme « Ant System » proposé par **Colorni et autres** en **1992** ; au début chaque fourmi est mise aléatoirement sur une ville et elle a une mémoire qui stocke la solution partielle qu'elle a construit jusqu'ici (au commencement la mémoire contient seulement la ville du début). À partir de sa ville de début, une fourmi se déplace itérativement d'une ville vers une autre mais avec une règle de probabilité. Étant donné à une ville i une fourmi k choisit d'aller à la ville j avec une probabilité donnée près :

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}} \qquad \text{if } j \in N_i^k \qquad (1)$$

Où $\eta_{ij} = 1/d_{ij}$ est l'information heuristique a priori disponible, et N_i^k est l'ensemble de villes que la fourmi *k* n'a pas encore visité, cet ensemble forme le voisinage faisable de la fourmi *k*. α et β sont deux paramètres qui déterminent le degré d'influence de la densité de phéromone et de l'information heuristique sur le choix de la prochaine ville.

Si α = 0, les probabilités de choix sont proportionnelles à [le *β d'ij* de] et les villes les plus proches seront plus probablement choisies: dans ce cas AS correspond à un algorithme *glouton* stochastique classique (avec multi-départ puisque les fourmis au commencement sont aléatoirement distribuées sur les villes).

Si *le* β = 0, seulement l'amplification de phéromone est au travail: ceci mènera à l'apparition rapide d'une convergence prématurée vers un optimum local [DS00].

Lorsque chaque fourmi termine la construction d'un chemin complet (de longueur n) ; on passera à la phase de mise à jour de la phéromone. Cette dernière se décompose en deux parties, la première consiste à baisser la densité de phéromone (c'est l'évaporation de la phéromone), la deuxième permet à chaque fourmi de déposer la phéromone sur les arcs qui appartiennent à son excursion:

$$\tau_{ij}(t+1) = (1-p) * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}(t) \quad \forall (i,j) \qquad (2)$$

Où 0 < ρ = 1 est le taux d'évaporation de la phéromone et *le m* est le nombre de fourmis. Le paramètre ρ est employé pour éviter l'accumulation illimitée de phéromone, et permet à l'algorithme d'"oublier" les mauvaises décisions faites précédemment. Avec ce mécanisme la force de phéromone associée aux arcs qui ne sont pas choisis par les fourmis, sera diminuée exponentiellement avec le nombre d'itérations. *l'ij du τ k* (t) est la quantité de dépôts de la fourmi *k* de phéromone sur les arcs; elle est définie par :

$$\Delta\tau_{ij}(t) = \begin{cases} 1/L^{k}(t) & \text{If arc (i,j) is used by ant k} \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

Où *L k* (t) est la longueur de la k^{eme} excursion de la fourmi. On remarque que plus l'excursion de la fourmi est courte, plus la phéromone reçue par les arcs de l'excursion est importante. En général, les arcs qui sont employés par beaucoup de fourmis et qui sont

contenus dans des excursions plus courtes recevront plus de phéromone et auront plus de chance d'être choisis dans les futures itérations de l'algorithme [DS00].

```
Pour t = 1,…..tmax
        Pour chaque fourmi k=1,….m
                Choisir une ville au hasard
                Pour chaque ville non visitée i
                        Choisir une ville j dans la liste jk i  dans les villes restantes,
selon la formule 4.1
                Fin Pour
                Déposer une piste Δ (t) sur le trajet conformément à l'équation 4.2
        Fin Pour
Evaporer les pistes selon la formule 4.3
Fin pour
```

Figure 1. 8 Algorithme de colonies de fourmis de base : the « Ant System »

6.2.3 Variations de l'algorithme

Après l'apparition du premier algorithme AS plusieurs variations ont été introduites [SDPT03], on note :

- Ant system et élitisme : dans cette version la fourmi qui effectue le chemin le plus cours dépose une quantité de phéromone plus que les autres afin d'accroitre la probabilité des autres à explorer la solution la plus prometteuse.
- Ant-Q : considérée comme une préversion de « Ant colony system ». la seule modification est que la règle de mise à jour locale est inspirée du 'Q-learing'.
- Ant colony system : ACS version comporte plusieurs modifications afin de permettre de résoudre des problèmes de grande taille ; ces modification **concernant** : l'introduction d'une règle de transition qui dépend d'un paramètre q_0 ($0=<q_0=<0$), la gestion des pistes est séparée en deux niveaux : une mise à jour locale et une mise à jour globale, utilisation d'une liste **de candidats stocke pour** chaque ville, les n villes les plus proches.
- ACS et 3-Opt : est une hybridation entre la version précédente et une recherche locale de type 3-Opt ; afin d'améliorer les solutions trouvées par les fourmis.
- Max-Min Ant system : basée sur AS avec les additions suivantes :
    - Seule la meilleure fourmi met à jour une piste de phéromone.
    - Les valeurs des pistes sont bornées par t_{min} et t_{max}
    - Les pistes sont initialisées à la valeur maximum.

- La mise à jour des pistes se fait de façon proportionnelle, les pistes les plus fortes étant moins renforcées que les plus faibles ;
- Une réinitialisation des pistes peut être effectuée.

6.2.4 Applications de l'optimisation par colonie de fourmis

L'optimisation par colonie de fourmis est appliquée sur plusieurs problèmes de différents types [DS00] on note ;

- NP- complet, pour ces problèmes très souvent ACO des algorithmes sont couplés avec d'autres possibilités, tel qu'un optimiseur local qui prend les solutions trouvés par les fourmis à un optimum local.
- Les problèmes de plus court chemin dans lesquels les propriétés des problèmes sont dynamiques et varient dans le temps (comme dans des problèmes de réseaux), donc le processus d'optimisation doit s'adapter. Dans ce cas-ci nous conjecturons que l'utilisation des algorithmes d'ACO devient de plus en plus appropriée [Li06].

7 Conclusion

Les métaheuristiques sont des méthodes générales qui peuvent être appliqués sur un grand spectre des problèmes difficiles pour donner des solutions efficaces dans un temps raisonnable. Ces méthodes sont des méthodes stochastiques, elles partagent le même principe, générer une (ou plusieurs) solution(s) et tester que cette (ces) solution(s)est (sont) bonne(s) ou non. Les métaheuristiques se distinguent dans la manière d'appliquer ce principe et dans les techniques additionnelles pour échapper d'un optimum local ou pour se déplacer dans l'espace de recherche.

On peut classer les métaheuristiques selon plusieurs critères : nombre des solutions manipulées, source de la méthode, type de voisinage utilisé et autre. Mais le critère le plus utilisé est le nombre des solutions manipulées par la méthode où on distingue deux grandes classes, les méthodes de recherche locale qui manipulent une solution à la fois et les méthodes **basées population** qui manipulent un ensemble de solutions à la fois. Cette classification influe considérablement sur la manière d'appliquer d'autres techniques telles que le parallélisme et l'hybridation afin d'augmenter l'efficacité des métaheuristiques.

Chapitre II

Parallélisme et hybridation des métaheuristiques : vue générale et classification

1 Introduction

Le parallélisme et l'hybridation des métaheuristiques sont deux techniques généralement utilisées pour traiter des grandes instances des problèmes difficiles ; pour le parallélisme, au début, le but était d'utiliser le calcul parallèle pour réduire le temps, mais l'apparition des modèles basés sur la notion de multi-cherche coopérative a conduit non seulement à réduire le temps de calcul mais aussi à trouver des solutions de qualité supérieure.

Pour l'hybridation l'idée est simple : utiliser les points forts de chaque méthode pour avoir des solutions plus satisfaisantes, par exemple combiner la puissance d'exploration d'espace de recherche des méthodes basées population tel que AG et OEP avec la bonne exploitation des méthodes de recherche locale tels que TS, RS.

Les métaheuristiques parallèles et hybrides nécessitent généralement une puissance additionnelle de calcul, une plateforme telle que les grilles du calcul peuvent donner cette puissance avec le moindre des coûts. Le problème qui se pose est que les grilles du calcul ont des caractéristiques qui imposent plus d'efforts sur le concepteur et le développeur.

Dans ce chapitre on va donner un aperçu sur les stratégies et les modèles du parallélisme et de l'hybridation des métaheuristiques.

2 Le parallélisme des métaheuristiques

On peut définir le *calcul parallèle* comme l'utilisation simultanée des ressources multiples de calcul (un ordinateur simple avec les processeurs multiples, un nombre arbitraire d'ordinateurs reliés par un réseau) pour résoudre un problème informatique, et on peut imaginer un programme qui s'exécute à l'aide des unités centrales de traitement multiples. Un problème décomposé en parties discrètes qui peuvent être résolues concurremment, chaque partie est une série d'instructions. Ces instructions s'exécutent simultanément sur différentes unités centrales de traitement.

Le parallélisme des métaheuristiques est l'une des deux techniques utilisées pour augmenter l'efficacité des métaheuristiques ; une idée simple est d'utiliser le calcul parallèle pour exécuter les instructions d'un algorithme d'une métaheuristique d'une manière parallèle (parallélisme bas niveau) ou décomposer l'espace de recherche en parties et en affecter chacune à un processus de recherche (approche par décomposition de l'espace de recherche).

3 Classification sur le parallélisme des métaheuristiques

Il existe plusieurs stratégies pour paralléliser les métaheuristiques ; et plusieurs classifications des métaheuristiques parallèles, chaque classification décompose les métaheuristiques parallèles selon une vision indépendante ; par exemple dans [CT03] Crainic et autres décomposent le parallélisme des métaheuristiques en 3 types : parallélisme de bas niveau, parallélisme par décomposition de l'espace de recherche et parallélisme haut niveau ; on remarque que cette classification engendre des classes non exclusives, on parle de type 1et2 et de type 2 et 3 car le critère de cette classification n'est pas unique ; par conséquent, on trouve la décomposition de l'espace et le parallélisme bas niveau qui appartiennent à la même classe dans une autre classification [CMRR02] ; cette dernière décompose les métaheuristiques parallèles selon les modèles suivants : parallélisme avec trajectoire unique et parallélisme avec trajectoire multiple ; celle-ci est décomposée en deux : parallélisme avec trajectoire multiple indépendant et parallélisme avec trajectoire multiple coopérative ; cette classification est mieux adaptée au métaheuristiques basées solution unique que les métaheuristiques basées population, car même la version séquentielle de ces dernières ne définit pas une sorte de trajectoire. Des classifications ont été expliquées dans [AT02] et [Can98] pour les modèles parallèles des métaheuristiques évolutionnaires ; on trouve le modèle maitre-travailleur, le modèle insulaire et le modèle cellulaire.

Dans la suite on va donner une classification rassemblant les différents points de vue. En premier lieu on maintient le typage de Crainic [CT03] et [CT98], en second lieu, au sein de chaque type on citera les modèles de parallélisme selon la nature de la métaheuristique :

- **le niveau du parallélisme :** on trouve une classification basée sur un tel critère dans [CT03] et [CT98] ; ce point de vue décompose le parallélisme des HM en deux types :
  - **parallélisme bas niveau** : dans ce type le parallélisme se fait au niveau des opérations de chaque itération, le seul but de cette stratégie est de faire accélérer le calcul, les solutions obtenues ont la même qualité des solutions obtenues par les algorithmes séquentiels.

- **parallélisme haut niveau** : cette stratégie vise une bonne exploration de l'espace de recherche par le lancement de plusieurs processus qui explorent l'espace de recherche, d'une manière synchrone ou asynchrone, coopérative ou indépendante.

- **la nature de la métaheuristique de base :** c'est-à-dire si la méthode basée population ou solution unique, on trouve une classification basée sur ce critère dans [RPE01].

L'application de ces deux repères donne un ensemble des modèles de parallélisme appliqués sur les métaheuristiques.

3.1 Parallélisme type I : parallélisme bas niveau

3.1.1 Le modèle d'accélération :

Les méthodes de recherche locale sont généralement construites d'un ensemble d'instructions exécutées itérativement. Le modèle d'accélération repose sur un paradigme maître-travailleur ; le processus maître décompose l'ensemble d'instructions en parties qui peuvent être exécutées simultanément, et il envoie chaque partie à un processus esclave ; puis chaque processus esclave exécutera la partie qui lui est assignée indépendamment sur des contraintes de synchronisation entre-parties [CT98].

Ce modèle peut être appliqué aussi si l'évaluation de la fonction de coût est parallélisable et/ou les entrées sorties sont coûteuses en temps. La fonction de coût sera remplacée par des fonctions des coûts partielles évaluées simultanément ; la fonction globale atteinte par l'application d'une fonction d'agrégation [CT98].

3.1.2 Le modèle d'évaluation parallèle du voisinage

Le voisinage d'une solution est l'ensemble des solutions atteignables à partir de celle-là après l'application d'un mouvement. L'étape d'évaluation des solutions voisines est une étape pertinente dans les méthodes basées solution unique, et peut consommer la plus part de temps d'exécution, donc nous reposons sur un modèle *maître- travailleurs* pour établir une évaluation parallèle d'un tel voisinage. Le processus maître engendre l'ensemble des solutions voisines de la solution courante et envoie chaque partie à un processus travailleur, qui à son tour fait les calculs pour évaluer la partie qui lui est associée, après il renvoie le résultat au processus maître ; ce dernier compare les résultats reçus et choisit le meilleur (voire la figure 2.1) [Mal05].

Ce modèle exploite la puissance de calcul parallèle mieux que le modèle précédent, surtout si la fonction à évaluer était coûteuse en termes de calcul [Mal05].

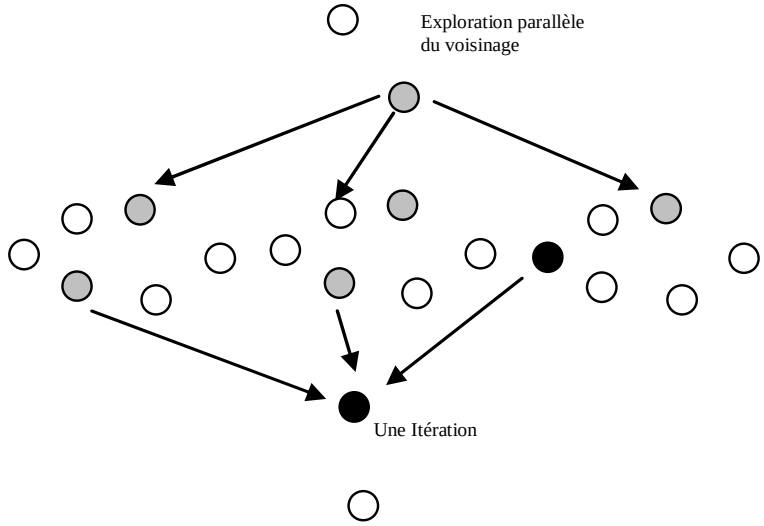

Figure 2. 1 le modèle d'évaluation parallèle de voisinage

3.1.3 Le modèle d'évaluation parallèle d'une population

Dans les algorithmes génétiques, un processus maître exécute **l'opérateur** de la sélection, le croisement et la mutation sur l'ensemble de la population ; après il envoie une partie de la population aux processus travailleurs, à leur tour, ces derniers évaluent les parties qui leur sont affectées , puis ils renvoient les résultats au maître, celui-ci calcule la moyenne totale ; car l'ordre employé pour calculer la fitness (ou la qualité) des individus est non pertinent à la moyenne finale de la population. La figure (2.2) montre la fonctionnalité de ce modèle.

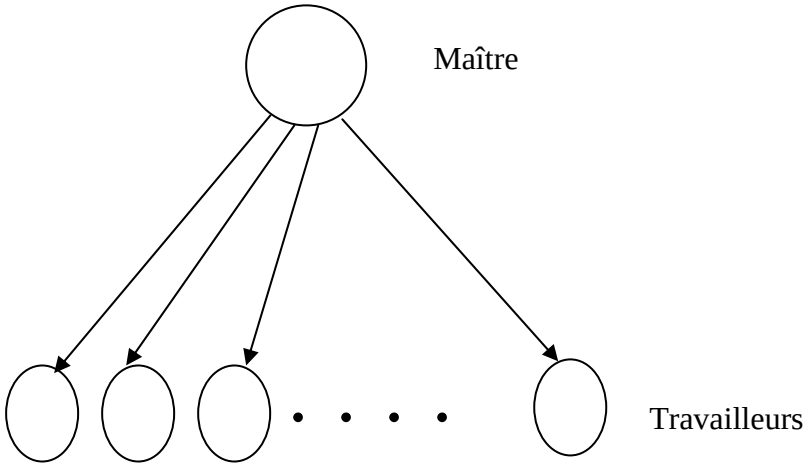

Figure 2. 2 le modèle d'évaluation parallèle d'une population

3.1.4 Le modèle d'évaluation parallèle d'une fonction objectif

C'est un autre modèle basé sur le paradigme maitre-travailleurs, appelé aussi modèle d'évaluation parallèle d'un individu [Mal05]. Chaque individu est répliqué sur plusieurs machines et il reçoit une évaluation partielle de leur fonction objectif, puis les parties

calculées de la fonction objectif sont envoyées à la machine maître pour appliquer une fonction d'agrégation. Ce modèle de parallélisme sera utile si et seulement si la fonction objectif est très coûteuse en fonction de calcul. Maleb [Mal05] dit que l'utilisation de ce modèle avec d'autres améliore le degré de parallélisme d'une manière significative ; il dit aussi que pour un modèle insulaire à 10 îles de 100 individus, la prise en compte du modèle d'évaluation parallèle d'une solution fait passer le nombre d'évaluation pouvant être effectuées simultanément de 1000 à 20000.

3.2 Parallélisme type II : avec décomposition de l'espace de recherche

Dans ce type, des sous problèmes sont résolus en parallèle, et la solution finale sera obtenue par une technique de composition ou d'extraction. La mise en application de ce type de parallélisme est généralement assurée par un modèle maître-esclave où un processus maître divise l'espace de recherche avec la décomposition de l'ensemble des variables de décision en sous-ensembles disjoints ; les variables en dehors de chaque sous-ensemble sont considérées fixes, ensuite le maître envoie chaque partie à un processus esclave**,** ce dernier applique l'heuristique à leur partition assignée concurremment et indépendamment puis il renvoie le résultat au maître qui composera le résultat final.

Le maître peut effectuer des modifications sur les partitions pendant la recherche, soit à des intervalles fixés en avant ou déterminés pendant l'exécution ; ou au moment de redémarrage de processus de la recherche.

Un inconvénient d'une telle approche basée sur la division des variables de décision peut laisser des grandes parties de l'espace de recherche inconnues. Par conséquent la plupart des applications répètent la division pour créer des différents segments du vecteur de variable de décision et redémarre la recherche [CT03].

3.3 Le parallélisme type III : parallélisme haut niveau

3.3.1 Le modèle multi cherche

Il consiste à lancer simultanément plusieurs processus de recherche locale pour trouver une meilleure et robuste solution. Les processus de recherche peuvent exécuter la même méthode heuristique, dans ce cas on a une approche dite *homogène ;* comme ils peuvent aussi exécuter des heuristiques différentes, en ce moment, on a une approche dite hétérogène. Ainsi ils peuvent commencer à partir de la même solution initiale (*mono dép*art) ou à partir des différentes solutions (multi *départs*) ; et peuvent communiquer seulement à l'extrémité de la

recherche pour identifier la meilleure solution globale, ainsi on l'appellera méthode *de recherche indépendante.* Comme ils peuvent inter-changer des informations pendant la recherche, on l'appellera *méthode de recherche coopérative*. Les communications peuvent être exécutées d'une manière synchrone ou asynchrone et peuvent être déclenchées par des événements ou exécutées aux moments prédéterminés ou dynamiquement décidés [CT03].

Ce modèle exige plus d'efforts de programmation et des maîtrises pour l'implémentation. Les processus échangent l'information rassemblée le long de la recherche, Cette information partagée est mise en application en tant que : variables globales stockées dans une mémoire partagée, ou par passage de message si on utilise une architecture de mémoire distribuée.

Dans ce modèle, où les processus de recherche coopèrent ; l'information rassemblée le long de chaque trajectoire est employée pour améliorer les autres. Cependant on compte non seulement accélérer la convergence vers une solution optimale mais, aussi, trouver une meilleure solution que la solution trouvée par les stratégies de recherche indépendantes avec le même temps de calcul.

L'aspect le plus difficile est la détermination de la nature d'information à partager ou à être échangée pour améliorer la recherche, sans prendre trop de mémoire ni de temps additionnel pour rassembler cette information. Généralement on utilise les meilleures solutions trouvées, des mouvements, des listes de tabou, et tailles de population, entre d'autres. Ces informations peuvent donner une vue plus générale sur l'espace de recherche et peuvent être employées pour renforcer les stratégies de diversification et d'intensification [CMRR02].

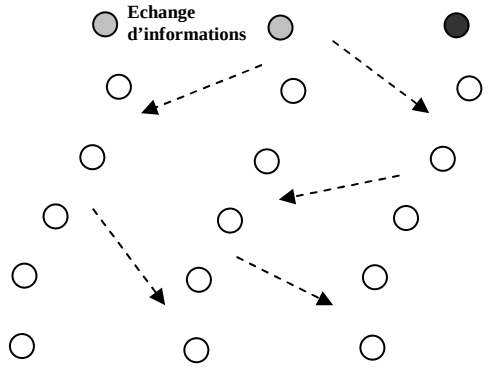

Figure 2. 3 le modèle multi-cherche

Le calcul est accéléré avec l'emploi d'une stratégie à recherche multiple, on essaye généralement de réduire l'espace de recherche pour chaque processus par rapport à l'espace de recherche exploré par un algorithme séquentiel. La mise en application d'une telle

technique varie selon la méthode métaheuristique. Si on a P le nombre de processeurs ; pour la recherche de tabou par exemple, on peut effectuer à chaque processus T/P itérations, où T est le nombre d'itérations du processus séquentiel correspondant. Pour le recuit simulé, le nombre d'itérations de la boucle d'équilibrage est réduit proportionnellement de L à L/P [CT03].

3.3.2 Le modèle d'îles (Island model)

Ce modèle est appelé aussi le modèle parallèle des algorithmes génétiques distribués ou le modèle parallèle avec une granularité grossière [Can98].

Le modèle Island consiste à diviser la population globale en sous populations, où chaque sous-population est affectée à un processeur. Les opérations de sélection, croisement et mutation sont faites au niveau de chaque sous-population. La communication entre les sous population est assurée par la migration des individus entre elles (voire la figure) ; sans celle-ci les sous-populations convergent rapidement vers des solutions non optimales.

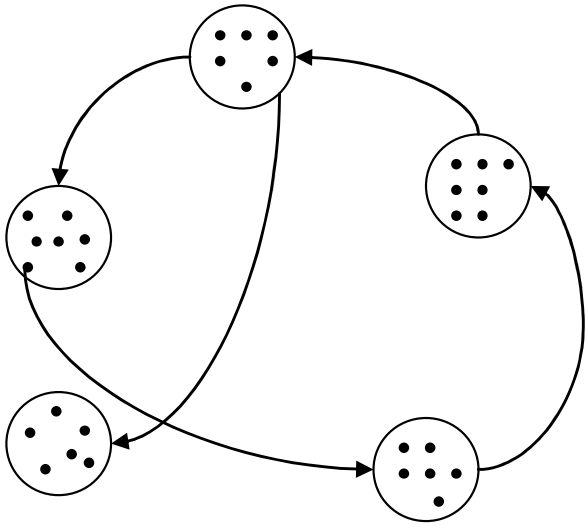

Figure 2. 4 le modèle d'îles

Une bonne stratégie de migration peut donner des résultats comparatifs ; la migration est contrôlée par plusieurs paramètres tels que :

- la Structure de voisinage : détermine la topologie et la densité de la connexion inter sous population. Si la densité de la connexion est très forte l'ensemble se comportera comme un algorithme unique ; et si la connexion est faible chaque sous population se comportera comme une population indépendante.
- le choix des individus à être migrés : définit la nature des individus à être migrés ; ils peuvent être les meilleurs, les mauvais ou aléatoires.

- le temps de migration : définit quand les individus doivent être immigrés ; la migration inter populations est soit synchrone et déterminée à l'avance avec des intervalles fixes, ou asynchrone.

3.3.3 Le modèle cellulaire

Dans ce modèle la population est divisée en sous populations où chacune contient un nombre très réduit d'individus. Chaque ensemble est affecté à un processeur, qui habituellement manipule un ou un nombre réduit d'individus avec une communication intensive entre les ensembles. On note que les ensembles sont rangés selon une topologie bien déterminée de 1, 2 ou 3 dimensions, La topologie 2D rectangulaire (la figure) est la plus utilisée parce que dans la plupart des ordinateurs massivement parallèles, les processus sont reliés selon cette topologie.

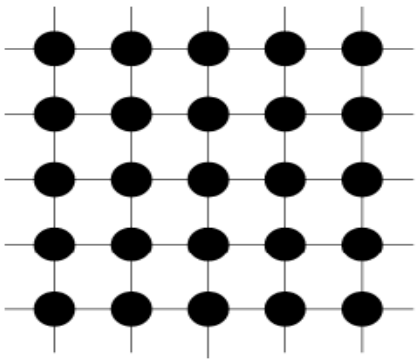

Figure 2. 5 le modèle cellulaire

Chaque ensemble a un voisinage, dans ce modèle, la notion de voisinage est pertinente car les opérations de sélection, croisement et mutation sont faites localement.

Ce modèle est mieux adapté à une structure informatique massivement parallèle SIMD, comme la machine Maspar200. En outre ce modèle peut être mis en application sur n'importe quel multiprocesseur. Ses paramètres principaux sont : la taille de population, la structure de population, la structure et la taille de voisinage [Can98].

4 L'hybridation des métaheuristiques

La motivation derrière une telle hybridation des déférentes métaheuristiques est d'avoir des méthodes plus efficaces. Ces dernières exploitent et rassemblent les avantages des méthodes de base, par exemple coupler la puissance d'exploration des méthodes basées population avec la bonne exploitation de l'espace de recherche par les méthodes de recherche locale. Le mot hybridation signifie n'importe quelle chose composée des éléments

hétérogènes ; donc une méthode métaheuristique hybride est une méthode composée de deux méthodes dans l'une est au moins une métaheuristique.

L'idée d'hybrider des métaheuristiques n'est pas nouvelle, mais elle remonte aux origines des métaheuristiques elles-mêmes. L'hybridation des métaheuristiques a subi un vrai lancement après l'apparition du théorème « no free lunch » proposé par (Wolpert et Macready), ces derniers ont prouvé qu'il n'existe pas une métaheuristique plus efficace que toutes les autres dans tous les problèmes [Rai06].

5 Stratégies d'hybridation des métaheuristiques

Au cours de ces dernières années plusieurs travaux ont été publiés dans le domaine de l'hybridation des métaheuristiques ; on trouve dans la littérature quelques efforts pour classifier ces travaux ; dans [PT97] Talbi et autres donnent une classification pour les métaheuristiques hybrides ; elle est fondée sur deux critères : l'ordre d'exécution (séquentiel ou parallèle) et sur la synchronisation pour les hybridations parallèles ; cette classification est enrichie et complétée dans [Tal99] [CTA05] où on trouve toute une taxonomie contenant plusieurs classifications, chacune d'elles se base sur différents critères ; Talbi distingue entre l'hybridation de point de vue conceptuel de l'hybridation de point de vue implémentation.

Une classification hiérarchique et une classification plate représentent les deux composants de l'aspect conceptuel ; la classification hiérarchique se fonde sur le niveau de l'hybridation (bas ou haut) et sur son application (en relais ou concurrente). La classification plate répond aux questions : hybridation homogène ou hétérogène, globale ou partielle et générale ou spéciale.

5.1 Classification hiérarchique

Selon Talbi [Tal99] cette classification fait la différence entre :

- Hybridation bas niveau et hybridation haut niveau : dans une hybridation de *bas niveau*, une fonction donnée d'une métaheuristique est remplacée par une autre métaheuristique. Dans le cas du *haut niveau*, le fonctionnement interne des métaheuristiques n'est pas modifié.
- Exécution en relais ou en concurrence : dans l'exécution en relais (séquentielle) les métaheuristiques sont exécutées l'une après l'autre, chacune utilise les résultats sortants de la précédente comme des entrées. l'exécution concurrente (parallèle) constitue plusieurs processus de recherche à résoudre le problème d'une manière coopérative ou compétitive.

L'application de ces critères donne quatre classes des métaheuristiques hybrides :

- **Bas niveau & relais (abrégé LRH en anglais)** on peut avoir cette hybridation dans le cas où une métaheuristique est incorporée dans une métaheuristique basée solution unique, par exemple incorporer une recherche locale dans le recuit simulé.
- **Bas niveau & co-évolution (abrégé LCH)** : dans cette classe d'hybridation on peut avoir par exemple une intégration d'un algorithme de recherche locale tel que la descente récursive, recuit simulé ou recherche avec tabou dans un algorithme évolutionnaire (méthode de recherche basée population). Le but est de rassembler la puissance d'exploration des algorithmes évolutionnaires avec la puissance d'exploitation des algorithmes de recherche locale ; cette stratégie d'hybridation peut être réalisée par le remplacement d'un opérateur évolutionnaire par une recherche locale. Dans les algorithmes génétiques les opérateurs à remplacer sont la mutation et le croisement, dans le cas de mutation l'opérateur est appelé *lamarckian* (voir la figure). L'utilisation d'une telle stratégie a donné de bons résultats pour un ensemble de problèmes. Pour l'opérateur de croisement on peut imaginer un algorithme glouton pour la reconstruction des individus.

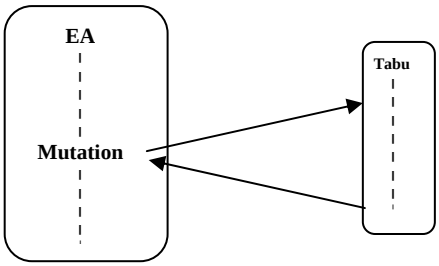

Figure 2. 6 la recherche avec tabou comme un opérateur de mutation

- **Haut niveau & relais (HRH)** Dans ce type d'hybridation les métaheuristiques sont exécutées l'une après l'autre ; par exemple on combine entre la puissance des méthodes évolutionnaires pour la détection des régions prometteuses avec la puissance des méthodes de recherche locale concernant la finition de la recherche. Cette idée est fondée sur une remarque sur les algorithmes évolutionnaires. Après un certain nombre d'itérations, la population se stabilise et arrête de progresser, on dit que la recherche se situe dans un bassin d'attraction où il existe une faible possibilité d'échapper ; là on peut utiliser une recherche locale telle que la recherche avec tabou pour mieux exploiter le bassin et avoir un optimum de très bonne qualité.

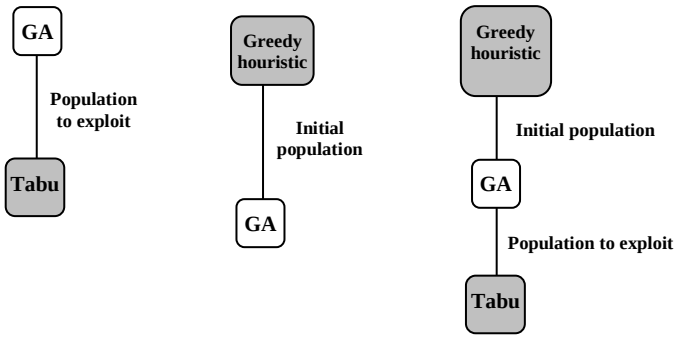

Figure 2. 7 hybridation séquentielle haut niveau

- ***Haut niveau & co-évolution (HCH)*** : elle consiste à exécuter plusieurs métaheuristiques en parallèle ; ces métaheuristiques coopérent pour trouver une solution optimale, la performance d'un processus de recherche basé sur cette hybridation est au moins égale à la performance d'un de ses composants. le modèle island basé sur la coopération des plusieurs algorithmes génétiques est un exemple de cette approche. GRASP (Greedy Randomised Adaptative Sherche) est un autre exemple. on peut le considérer comme une hybridation entre un algorithme glouton pour construire des solutions initiales et une méthode de recherche locale.

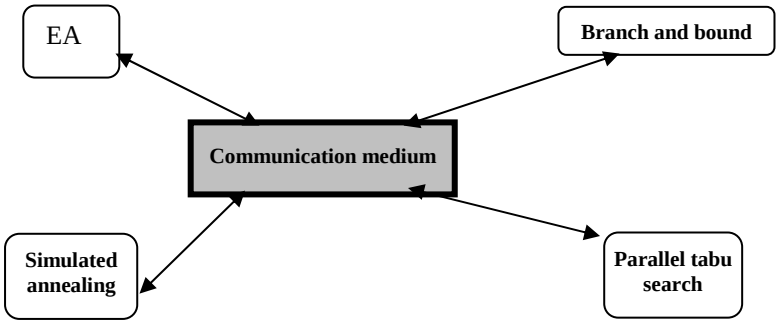

Figure 2. 8 hybridation parallèle haut niveau

5.2 La classification plate

La classification plate dégage autres critères, pouvant caractériser les hybridations :

- *Homogène Vs hétérogène* si l'hybridation se fait entre plusieurs instances d'une même métaheuristique, elle est *Homogène*, sinon, elle est *hétérogène.*
- *Globale Vs partielle* si les méthodes cherchent dans tout l'espace de recherche, on parlera d'hybridation *globale*, par contre si le problème est décomposé en sous problèmes où chacun possède son espace de recherche, et chaque composant de l'hybridation résout un sous-problème dans se cas là l'hybridation est dite *partielle.*

- *Générale Vs spécialisée* si les algorithmes mis en jeu travaillent tous pour résoudre le même problème, on parlera d'approche générale, mais s'ils sont lancés sur des problèmes différents, l'hybridation est alors spécialisée.

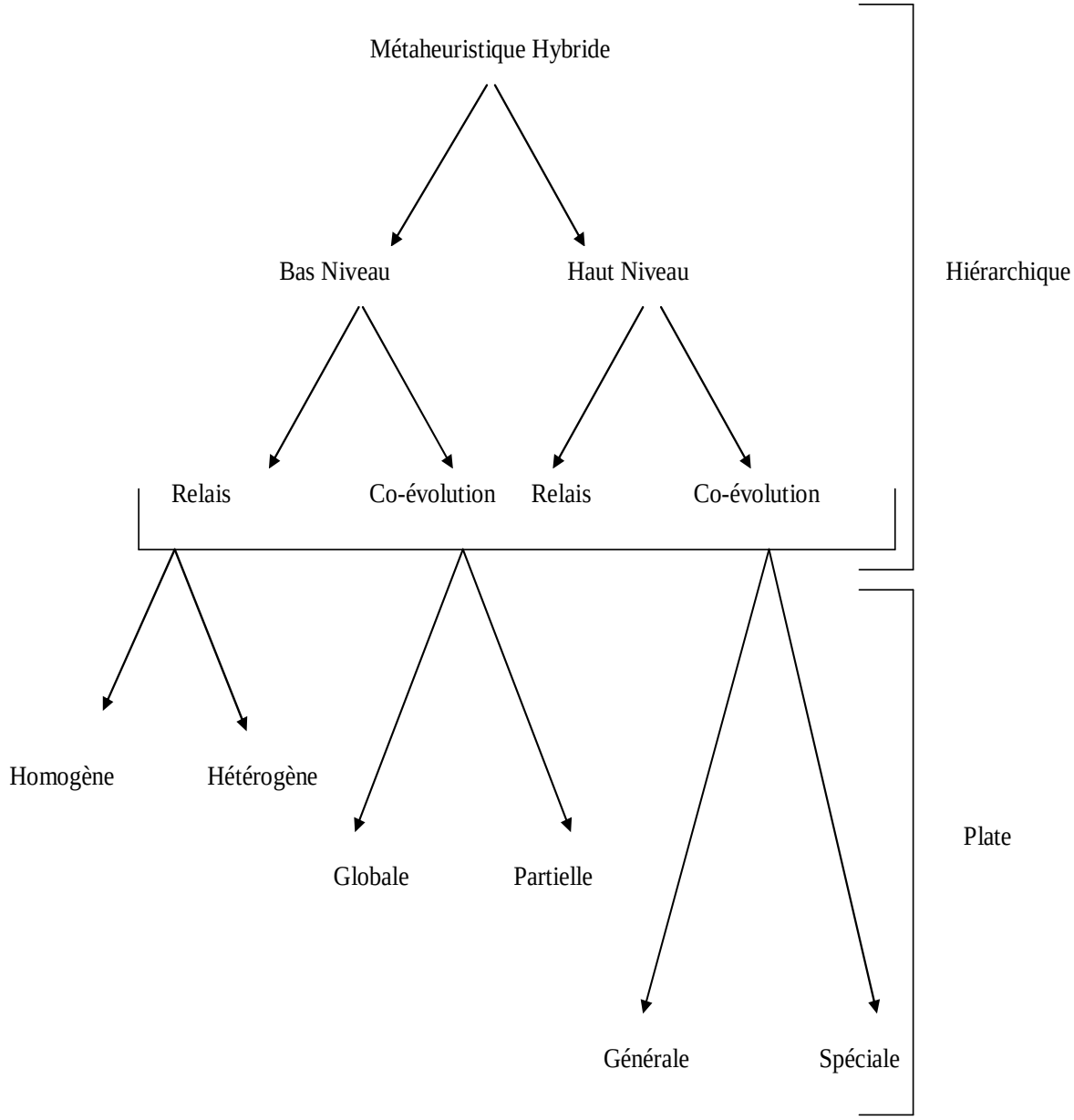

Figure 2. 9 la classification hiérarchique et la classification plate

6 Conclusion

Lorsque la taille des données d'entrée d'un problème difficile augmente, la taille de l'espace de recherche augmente aussi ; dans ce cas là l'utilisation des métaheuristiques dans leurs versions originales ne donne pas les résultats souhaités ; pour augmenter la puissance de

recherche des métaheuristiques deux techniques sont généralement utilisées, le parallélisme et l'hybridation.

Le parallélisme consiste à exécuter plusieurs instructions simultanément sur une ou des machine(s) capable (s) de faire ça. Le parallélisme des métaheuristiques est réalisé de plusieurs manières ; celles-ci engendrent suite à une classification différents types :

- le parallélisme bas niveau : basé généralement sur un modèle maitre-travailleur et tente toujours de réduire le temps d'exécution occupé par le processus de recherche afin d'élaborer des grandes instances des problèmes difficiles ; les solutions données par ce type de parallélisme sont les mêmes données par les versions séquentielles mais avec un temps inferieur.
- Le parallélisme avec la décomposition de l'espace de recherche : basé aussi sur le modèle maitre-travailleur, il nécessite un effort supplémentaire pour avoir les résultats finaux.
- Le parallélisme haut niveau : consiste à lancer plusieurs processus de recherche qui tentent d'explorer la totalité de l'espace de recherche. Dans ce type de parallélisme on ne cherche pas seulement de réduire le temps d'exécution mais aussi de trouver des solutions plus efficaces et robustes. Ce type comprend une grande variété des méthodes et il nécessite plus d'efforts soit dans la conception (choix et adaptation des paramètres), soit dans l'implémentation (synchronisation, modes de migration …). Ce type de parallélisme reste le plus promoteur pour donner des bons résultats.

Le but de l'hybridation des métaheuristiques est de construire des méthodes qui comprennent les points forts de ses composantes. On peut hybrider les métaheuristiques avec des métaheuristiques ou avec d'autres méthodes qui appartiennent aux autres approches de résolutions, tels que les algorithmes recherche exacte. Plusieurs hybridations entre les métaheuristiques concentrent, sur le point que les métaheuristiques de recherche locale exploitent mieux une portion de l'espace de recherche, et les métaheuristiques basées population explorent mieux l'espace de recherche. L'investissement sur cette idée est réalisé sur différentes formes qui peuvent être séquentielles ou parallèles, intégratives (bas niveau) ou coopératives (haut niveau).

Les méthodes résultent de l'utilisation des deux techniques (le parallélisme et l'hybridation) donnent les meilleurs résultats, mais ces dernières exigent une grande puissance de calcul. Dans le chapitre suivant on va parler d'une plateforme qui peut donner cette puissance, avec une simple accessibilité et un coût presque nul, cette plateforme est la grille.

Chapitre III

L'optimisation des problèmes difficiles et les grilles de calcul

1 Introduction

Au début du 20e siècle, la génération personnelle d'électricité était technologiquement possible et de nouveaux équipements utilisant la puissance électrique avaient déjà fait leur apparition. Paradoxalement, le fait que chaque utilisateur possède son propre générateur d'électricité était un obstacle majeur à la généralisation de son utilisation. La véritable révolution a nécessité la construction d'une grille électrique, c'est-à-dire la mise en place de réseaux de transmission et de distribution d'électricité. C'est l'accessibilité et la baisse de coût qui ont finalement permis de généraliser l'exploitation industrielle de cette énergie.

Par analogie, l'utilisation des technologies informatiques connaissait une énorme évolution : les PC, les réseaux et l'Internet sont partout, dans le domaine administratif, industriel, éducatif et autre. La puissance de calcul et l'espace de stockage continuent à grandir et la vitesse de transmission de données aussi subit une évolution rapide et importante. On parle de quelques Gigas/s avec l'utilisation des fibres optiques.

Cependant les ressources informatiques sont rarement utilisées à leurs capacités maximales, soit pour la puissance de calcul ou pour la capacité de stockage. Des statistiques montrent que les processeurs restent en chômage dans 47% du temps.

D'autre part on a des applications informatiques qui nécessitent une grande puissance de calcul telles que les applications liées au domaine d'optimisation des problèmes difficiles; alors pourquoi ne pas utiliser les ressources déjà existantes et inexploitées.

2 Les grilles

2.1 Définition

Le terme **Grid** a fait son apparition pour la 1ere fois dans le monde académique au milieu des années 1990 et s'inspire de la grille d'électricité (*power grid*). Selon I.Foster **«** *Une*

grille informatique est une infrastructure de matériel et de logiciel qui fournit l'accès sûr, conformé, dominant, et peu coûteux aux ressources informatiques à extrémité élevée » [FK99].

Comme la grille est un nouveau concept elle devient un point d'attraction pour tout ce qui est parallèle et distribué. Une précision du terme 'grille' est nécessaire. Dans [Fos02] on trouve une liste de conditions pour distinguer la grille des autres concepts :

1. ne mettre en jeu aucun contrôle administratif centralisé (ce qui élimine les grappes)
2. utiliser des protocoles universels (ce qui élimine les plateformes spécifiques à une application tel que Seti@home)
3. assurer une haute qualité de service.

Dans notre travail ; on considère que la grille ressemble à une large collection des éléments informatiques (PC, unités de stockage, capteurs, outils de visualisation et autre) hétérogènes et largement dispersés; on peut imaginer l'existence d'un grand nombre de stations connectées entre elles à travers le monde qui donnent un pouvoir de calcul énorme avec un coût très réduit.

2.2 Les Types de grilles

Les types des grilles diffèrent selon le critère de typage ou de classification, dans ce cas là on peut avoir deux classifications :

- classification par infrastructure
- classification par objectif

2.2.1 Classification par infrastructure

Cette classification distingue les grilles selon la nature du matériel utilisé, le type de participation, l'organisme d'administration, la durée et le rayon d'application de la grille. Avec cette classification on trouve deux approches :

Approche communautaire

Il s'agit dans ce type de grille de récupérer les ressources de calcul inutilisées par les propriétaires des machines. Ces ressources sont exploitées grâce au réseau Internet. Elles sont mises à la disposition de manière volontaire, individuelle et gratuite, sur un modèle d'action charitable ou communautaire. Il s'agit de montages à court terme, techniquement légers, et le plus souvent spécialisés pour un certain type d'application d'intérêt commun pour la

communauté concernée. Un exemple de l'utilisation de ce type de techniques est le projet *Seti@home*[1] de détection de la vie extraterrestre. Ce projet a été lancé par des universitaires américains. L'idée est d'analyser les signaux recueillis par un très grand télescope radio pour détecter des corrélations internes. Le postulat est que même si ces signaux sont censés être aléatoires, ils cachent une cohérence interne. Alors ils pourraient être émis par une intelligence extraterrestre tentant de rentrer en contact avec nous. Les signaux reçus sont découpés par l'ordinateur maître en petits fragments significatifs et soumis aux PC serveurs participants au programme. Chaque serveur exécute une série de tests statistiques sur les fragments qu'il reçoit et renvoie les résultats au maître. Les PC serveurs participant au programme ne traitent les fragments transmis par le maître que lorsqu'ils sont inactifs. Dès que leur propriétaire les réveille, ils laissent ce travail en attente, ce qui rend cette activité pratiquement indétectable pour l'utilisateur. Ceci est implémenté simplement en demandant aux participants d'utiliser un économiseur d'écran particulier au lieu d'afficher des poissons rouges sur leur écran comme le font la plupart des humains. Ainsi, les procédures d'activation et de mise en veille de l'activité d'analyse sont entièrement sous le contrôle du système d'exploitation et de l'utilisateur par les outils de paramétrage habituels. Ce programme apparemment un peu futile a en fait connu un succès extraordinaire, en octobre 2001, le nombre de participants dépassait 3.000.000, et plus de 700 années de calcul avaient été effectuées. Le temps moyen *donné* par chaque PC participant est de 17,5 heures par jour environ, soit plus de 70 % du temps disponible. Le chiffre le plus frappant est certainement la puissance développée : 23,5 téraFlop/s, soit plus de 3 fois la puissance de la machine la plus puissante du monde, pour un coût nul, ou presque [Bou02].

Approche institutionnelle

Il s'agit dans ce cas de mettre en commun des infrastructures matérielles et logicielles lourdes, gérées en général par des institutions publiques (centre de recherche, centre de calcul, etc.). Ces infrastructures peuvent être très diverses :

- outils d'acquisition de données (microscope électronique, accélérateur de particules, satellite, etc.).
- moyens de calcul et de stockage (ordinateur, système d'archivage, etc.).
- réseaux de communication dédiés à très haut débit, outils d'exploitation des résultats (environnement de réalité virtuelle, etc.).

[1] http://setiathome.berkeley.edu/

Il s'agit de montages lourds, planifiés à long terme. Un exemple majeur de mise en place d'une grille de ce type est le projet européen *DataGRID*[1]. Son objectif est de fournir au niveau européen une infrastructure pour le calcul intensif et aussi l'analyse de très grandes bases de données (de l'ordre du pentaoctet, 10^{15}) partagées par des communautés scientifiques dispersées sur l'ensemble du continent. Les applications pilotées sont les suivantes :

- analyse des données du *Large Hardon Collider* en cours de construction au **CERN** (organisation européenne de recherches nucléaires).
- traitement d'images dans le domaine de la biologie et de médecine, piloté par le **CNRS (**Centre National de la Recherche Scientifique**).**
- exploitation des données satellites d'observation terrestre, piloté par l'Agence spatiale européenne (**ESA**).

2.2.2 Classification par objectif

Dans cette classification on distingue trois classes de grilles selon leur objectif ou nature :

1. les grilles d'information : peut être la première utilisation de la grille, consiste à partager la connaissance via une large infrastructure : la concrétisation de ce type est assurée par l'apparition du Web.
2. les grilles de données : consiste à stocker et à traiter des grandes bases de données de l'ordre de petabyte. Ces dernières sont générées soit par de grandes expériences scientifiques (physique des particules, biologie, observation spatiale,…) ; soit résultant d'actions spécifiques telles que les données commerciales ou économiques. Ces données nécessitent des capacités de stockage inaccessibles pour un organisme seul. Par ailleurs, il apparaît souvent inutile de rapatrier en un même lieu l'ensemble des données alors que seule une faible partie est nécessaire pour le traitement envisagé. par conséquent une infrastructure Grille apporte une solution efficace. Un exemple sur un tel type de grilles est le projet DataGrid.
3. les grilles de calcul : consiste à agréger la puissance de calcul via l'exploitation des temps morts des clusters (ou des simples PC) au sein d'une communauté dont les ressources sont utilisées de manière discontinue dans le temps. Plusieurs applications ont besoin d'une telle puissance, citons :

[1] http://eu-datagrid.web.cern.ch/eu%2Ddatagrid/

- les modèles météo
- les études sur le changement climatique global
- les simulations et outils de conception en aéronautique, automobile, chimie ou nucléaire.

Et ça, pour ne citer que quelques exemples industriels, sans oublier les domaines de la finance, notamment avec les calculs de risque et de la biologie.

2.3 Caractéristiques des grilles

On peut facilement constater que la grille est une plateforme exceptionnelle, car elle est composée d'éléments hétérogènes (du point de vue fonctionnel, architectural ou logiciel) et ces éléments sont distribués sur une grande surface géographique ; par conséquent, la grille est dotée d'un ensemble de caractéristiques [Li06]. Ces derniers produisent un ensemble de critères qui doivent être respectés par toute application implémentée sur une grille:

- **la grille est multi-domaines d'administration :** c'est-à-dire que les ressources de la grille sont reparties sur plusieurs domaines d'administration et gérées par différentes organisations ; elles peuvent utiliser différentes stratégies de gestion, n'oublions pas le problème d'identification et le problème de sécurité ; ce qui justifie la nécessité d'identifier les utilisateurs et les fournisseurs sur la grille.
- **la grille est hétérogène** : le fait que la grille est constituée par un nombre important de machines qui appartiennent à des domaines qui différent, alors elle contient des ressources (matériel et logiciel) hétérogènes. La solution est d'utiliser les standards d'échange de données tel que XML et les langages qui sont supportables par plusieurs plateformes tel que JAVA.
- **La grille est dynamique:** à cause de la probabilité de tomber en panne, la récupération par le propriétaire et le retour à la grille on peut dire que les ressources sont volatiles. Donc pour développer une application sur la grille on doit fournir des mécanismes pour :
  - découvrir dynamiquement les ressources sur la grille.
  - la tolérance aux pannes.
  - la sauvegarde et la restauration des données et la synchronisation.

  Ces concepts peuvent être pris en compte par les "intergiciels" ou par les concepteurs au niveau des applications.

- **Le temps de latence :** c'est la période de passage des messages entre les processeurs. Dans un environnement grille cette période peut être haute, variable et imprévisible.

2.4 Architecture des grilles

D'une manière synthétique, il est généralement possible de définir l'architecture d'une grille de calcul à l'aide de quelques éléments représentés dans la figure. Cette architecture a été proposée par Foster et autres [FKT01]. Les divers composants de cette architecture sont : le tissu (support matériel), l'intergiciels et les tâches (applications).

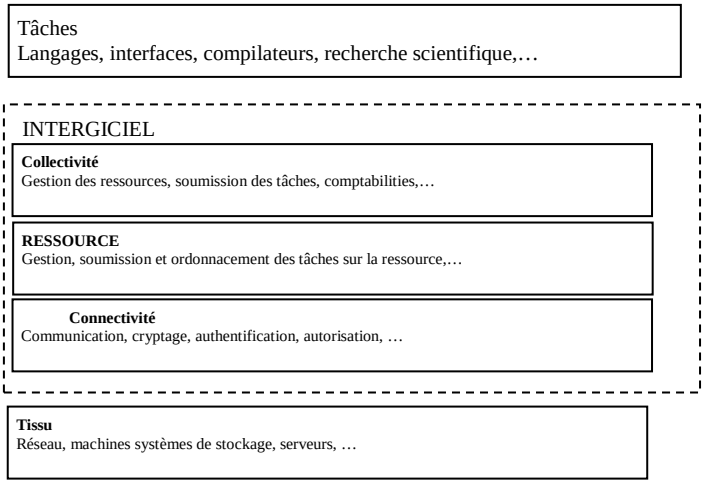

Figure 3. 1 L'architecture de la grille de calcul

Le tissu comprend les ressources de calcul (clusters, machines de bureau, . . .), les systèmes de stockage, le réseau, *etc.*, qui vont être partagés via les protocoles contenus dans les intergiciels de la grille.

L'intergiciel (*middleware*) est en quelque sorte la glu qui se charge de connecter les diverses ressources de la fabrique afin de les transformer en une entité unique, *l'organisation virtuelle* (VO). Il se décompose en trois sous-couches : la sous-couche connectivité, la sous-couche ressource et la sous-couche collectivité.

- La sous-couche **connectivité** fournit les mécanismes de sécurité nécessaires à la protection des ressources. Elle contient les protocoles de communication et d'authentification utilisés dans les transactions à l'intérieur du réseau spécifique à la grille. Les protocoles de communication permettent l'échange de données entre les diverses ressources de la fabrique. Les protocoles d'authentification sont construits sur les services de communication pour fournir des mécanismes sûrs et cryptés afin de vérifier l'identité des utilisateurs et des ressources. Idéalement, dans un environnement de VO, les solutions d'authentification devraient présenter les caractéristiques de

«*single sign on*» et de délégation des droits à un autre programme : le client s'authentifie une fois et ses applications peuvent s'exécuter sur n'importe quelle ressource de la grille. Dans les grilles, la sécurité est un souci nettement plus important que dans les environnements informatiques traditionnels, à cause du grand nombre de ressources concernées, qui sont de plus généralement réparties sur divers sites géographiques.

- La sous-couche **ressource** comprend des protocoles permettant d'obtenir des informations sur la structure et l'état d'une ressource, tel que sa configuration, sa charge courante et sa politique d'utilisation. Cette couche comprend aussi les protocoles de gestion et de négociation de l'accès à une ressource partagée.
- la sous-couche **collectivité** permet de coordonner des ressources multiples. Elle contient des protocoles et des services qui sont associés aux interactions entre une collection de ressources. Elle doit permettre entre autres de :
    - découvrir l'existence et les propriétés des différentes ressources d'une VO (service d'information et d'annuaire).
    - faire la coallocation, l'ordonnancement et le courtage, c'est-à-dire permettre aux clients de la VO de demander l'allocation d'une tâche particulière à une ou plusieurs ressources, selon une certaine logique.
    - transférer les données nécessaires aux tâches d'un site vers un autre.
    - gérer la facturation des services.

La dernière couche concerne les **tâches (applications)** des clients. Elle contient des outils susceptibles d'aider les développeurs à concevoir et à écrire des applications adaptées à la grille. Il s'agit en particulier de compilateurs, de libraires, d'outils de conception d'applications parallèles ainsi que d'interfaces de programmation.

3 L'optimisation et les grilles de calcul : état de l'art

Comme les autres domaines, le domaine d'optimisation combinatoire a bénéficié de la capacité donnée par les grilles de calcul ; malgré la difficulté de la conception et de l'implémentation des applications tournant sur une telle plateforme.

Ce qui nous intéresse sont les grilles de calcul, ces grilles immergent en trois approches :

- **Grid Computing (Virtual Supercomputing)**

- **Internet Computing**
- **Global Computing (MetaComputing)**

Le *grid computing* consiste à regrouper un ensemble des ressources pour construire un supercalculateur virtuel à l'échelle de l'internet. Plusieurs intergiciels permettent de construire des grilles de calcul avec une telle stratégie ; par exemple l'intergiciel condor, Globus [FK97] et ligeon.

Dans [Ste00] une description du projet METANOS qui a débuté en 1997 comme un effort de collaboration entre les spécialistes en optimisation et les groupes de Condor et de Globus. Le but était la conception et l'implémentation des algorithmes et outils logiciels pour la résolution des problèmes d'optimisation complexes et difficiles dans un environnement grille de calcul. Ces efforts ont conduit au développement de la plateforme MW [GL05] pour mettre en application des algorithmes avec un mode de control 'maître-travailleur' (mastre-worker).

Avec l'utilisation de l'intergiciel Condor comme support, des travaux sont réalisés sur des algorithmes et des codes de la programmation linéaire, le problème d'assignement quadratique des tâches (QAP) et la programmation stochastique.

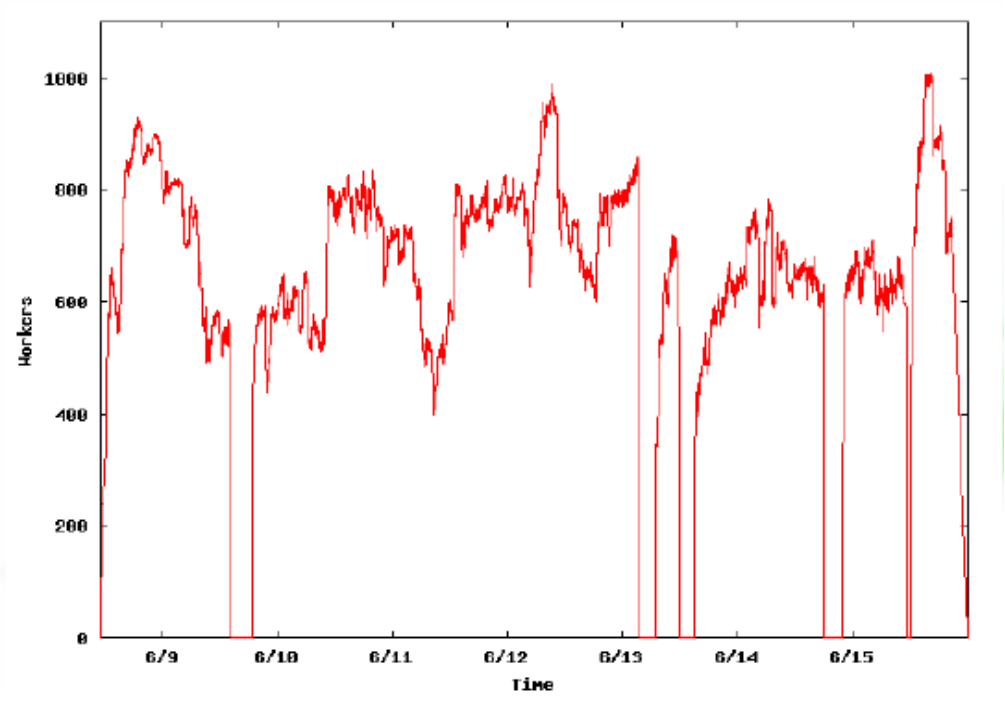

Figure 3. 2 variation de nombre des PC participants dans la résolution de NUG30

Dans [ABGL01], un groupe de chercheurs à l'université d'Iowa et du laboratoire national Argonne a résolu une instance du QAP appelé NUG30. La résolution de cette

instance est concrétisée grâce à l'emploi de l'intergiciel Condor, qui est développé à l'université du Wisconsin. NUG30 est connue comme une instance énorme du QAP. NUG30 aura besoin de plus de 10 ans de temps de calcul par une unité centrale de traitement simple. Les chercheurs ont obtenu une solution optimale au bout de sept jours en utilisant 653 CPUs en moyenne.La figure 3.2 montre la variation de nombre des PC participants à l'expérience.

L'Internet computing est une autre stratégie pour construire les grilles de calcul. Elle consiste à récupérer les cycles inutilisés sur l'ensemble des PC connectés via Internet et à les utiliser en calcul utile. Parmi les intergiciels qui utilisent cette technique on trouve l'intergiciel EasyGrid [BR04].

Dans [[AUB+05] on trouve un exemple typique pour la conception et l'implémentation d'un algorithme parallèle et hybride appliqué sur un environnement grille de calcul selon une telle stratégie.

Une présentation de quatre stratégies pour le parallélisme de l'algorithme GRASP étendue par la méthode ILS (recherche locale itérative) pour le problème de déplacement reflété de tournois.

Les quatre stratégies sont PAR-I (Parallel strategy with independent processes), PAR-O(Parallel strategy with one-off cooperation), PAR-1P(Parallel strategy with one elite solution) et PAR-nP(Parallel strategy with a pool of elite solutions) ; elles Ont été mises en application avec l'utilisation du langage C++, la bibliothèque MPI-LAM (une implémentation de MPI) et l'intergiciel EasyGrid AMS [AUB+05].

Les résultats montrent que ces réalisations parallèles sont capables de trouver des meilleures solutions par rapport à leurs versions séquentielles. L'utilisation des processus multiples et d'un ensemble des solutions élus offre une diversité des solutions de haute qualité avec une convergence plus rapide.

Les résultats montrent aussi que la version PAR-nP implémentée sur la grille est capable d'être exécutée d'une manière efficace et robuste. Les résultats prouvent que cette nouvelle implémentation dynamique s'exécute aussi bien que la version statique équivalente. Cette version de grille permet l'exécution pendant des périodes de temps étendues où l'utilisateur n'est pas concerné par l'échec de ressources, ni de processus ni d'utilisation de ressources.

Le global computing repose sur une autre technique pour utiliser les ressources à distance. Dans ce mode on invoque les fonctions déjà installées sur les machines extrêmes.

On utilise généralement le GridRPC [SNM+02], qui prolonge le mécanisme des appels des procédures à distance (RPC) pour la grille. GridRPC est un modèle de programmation typique pour développer des applications sur les grilles. Dans le GridRPC, les supports d'application arrangent leurs propres applications sur les serveurs à distance à l'avance. Puis, les utilisateurs appellent l'un d'eux par le réseau étendu (WAN) en employant les APIs des GridRPC, qui ont été normalisés par GGF (Global Grid Forum) [SHMD05].

Le GridRPC a deux avantages. En premier lieu, il a une excellente rentabilité pour les utilisateurs. En second lieu, il permet aux supports d'application d'arranger et d'éditer des applications déjà existantes sur la grille de façon facile et sans risque. En outre, quelques systèmes sophistiqués qui mettent en application le GridRPC APIs tel que NetSolve, Ninf et Ninf-G sont déjà développés. [SHMD05].

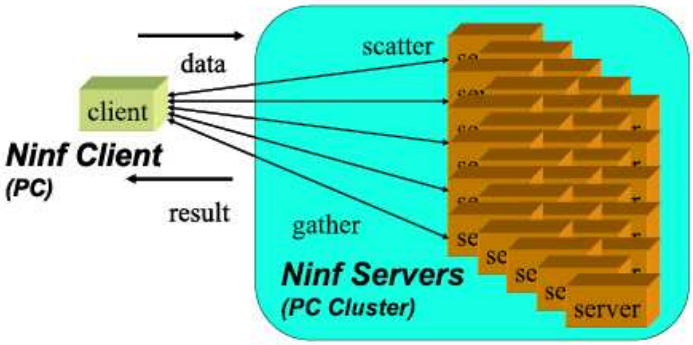

Figure 3. 3 principe global computing avec Ninf

On trouve un exemple dans [FKTY03] pour l'utilisation des GridRPC pour les problèmes d'optimisation difficiles. Dans cet article, K. Fujisawa et autre présentent l'utilisation de la grille et les clusters pour la résolution de quelques problèmes d'optimisation. Par exemple pour le problème QOP, ils ont mis en application un SCRM (Successive Convex Relaxation Method) fortement parallèle implémenté sur l'intergiciel Ninf. A chaque itération le client de Ninf applique le SCRM à un QOP et produit un grand nombre de sous-problèmes. Chacun forme un problème SDP. Tous les problèmes produits de SDP sont envoyés aux machines du serveur de Ninf. Puis chaque serveur de Ninf résout le problème de SDP en utilisant le logiciel SDPA. Notez que chaque serveur de Ninf ne résout qu'un problème SDP à la fois. Après la terminaison de l'exécution du SDPA, le résultat est envoyé de nouveau à la machine cliente de Ninf.

Les expériences sur un environnement de grille formé de deux clusters de PC qui sont situés à Kyoto et à Tokyo. Le cluster de Kyoto a 4 nœuds qui sont reliés à un client de Ninf par un LAN (100BASE-tx) et le cluster de Tokyo a 4 nœuds qui sont reliés au PC de client de

Ninf par l'Internet appelé SINET avec une vitesse moyenne entre Kyoto et Tokyo environ de 2 à 4 Mbps.

4 Les plateformes pour implémenter des applications sur les grilles de calcul

On a dit que la grille de calcul demande des efforts additionnels lors de la conception et l'implémentation des applications sur elles. Pour simplifier la conception et l'implémentation des méthodes d'optimisation basée sur les métaheuristiques (ou autres méthodes), quelques plateformes ont été développées. Même si le but est commun, chaque plateforme a ses propres caractéristiques et diffèrent dans plusieurs points : les métaheuristiques utilisées, les modes de leur d'utilisation, le langage d'implémentation entre autres.

4.1 ParadisEO-CMW

ParadisEO-CMW [Mal05] est une gridificaStion de la plateforme ParadisEO [CMT04] avec l'utilisation de la plateforme MW et l'intergiciel Condor.

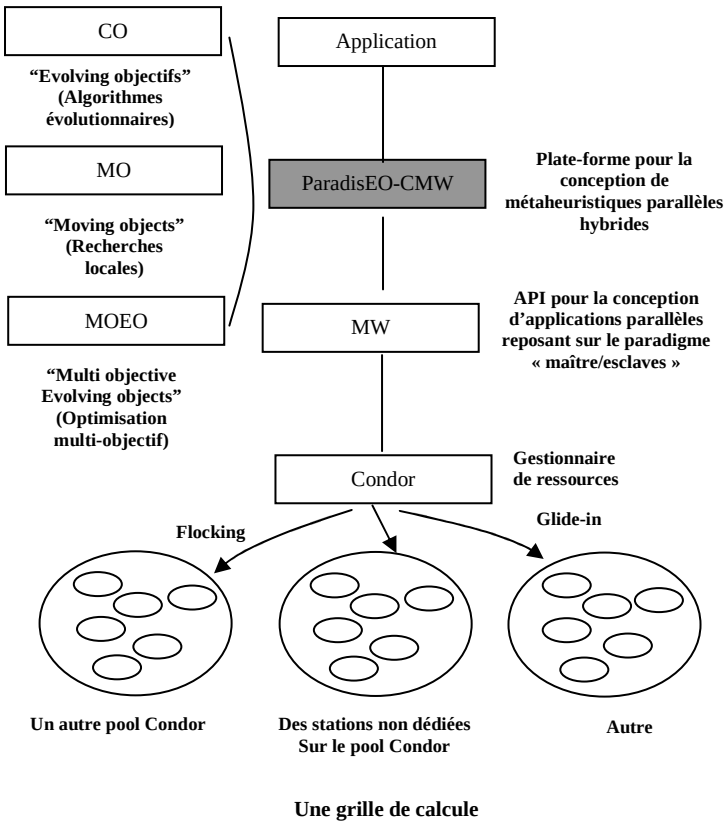

Figure 3. 4 L'architecture de PradesEO-CMW

ParadisEO est une plateforme développée par l'équipe OPAC du laboratoire LIFL de l'université de Lille. Elle offre à l'utilisateur de concevoir et d'implémenter des algorithmes parallèles et hybrides pour résoudre différents problèmes. ParadisEO n'impose pas un modèle de parallélisme car elle offre une multitude de modèles : le modèle insulaire (islande modèle), le modèle d'évolution parallèle d'une population et le modèle d'évaluation parallèle d'un individu pour les algorithmes basés population. Pour les algorithmes basés solution unique, on trouve le modèle d'évaluation parallèle du voisinage, le modèle d'évaluation parallèle d'un mouvement et le modèle parallèle multi-départs.

ParadisEO fait une séparation entre la partie générale partagée entre tous les problèmes et la partie spécifique à chaque problème. Cette plateforme utilise la notion de la classe abstraite pour minimiser le travail de l'utilisateur. Elle respecte le paradigme maitre-travailleur et elle est implémentée avec le langage C++.

ParadisEO-CMW repose sur MW et l'intergiciel Condor pour pouvoir exécuter des applications sur une grille de calcul selon l'architecture représentée dans la figure.

4.2 DREAM (Distributed Resource Evolutionary Algorithm Machine)

Le projet européen DREAM [ACE+02] (Distributed Resource Evolutionary Algorithm Machine) propose une plateforme permettant l'utilisation du modèle insulaire pour l'implémentation parallèle pair à pair des algorithmes évolutionnaires.

La plateforme est construite de 5 niveaux et offre 5 points d'entrée selon l'expertise de l'utilisateur ; dans le haut niveau l'utilisateur fait la spécification et l'exécution de l AE simplement avec la manipulation des outils graphiques, ce niveau est simple mais il est limité du point de vue flexibilité ; dans le bas niveau la plateforme offre un système pair à pair d'agents mobiles qui peut être utilisé pour construire des applications distribuées basées sur des processus autonomes, ce niveau demande une bonne expérience dans le domaine de la programmation parallèle et l'optimisation combinatoire.

La plateforme est basée sur un intergiciel de calcul global pair à pair *ad hoc* développé en Java.

4.3 HEURISTICLAB GRID

Heuristiclab-grid [WA04] est une plateforme inspirée du projet SETI@home pour l'exécution des applications parallèles (à base de heuristiques) sur une grille de calcul. Heuristiclab-grid est une gridification de la plateforme Heuristiclab.

HeuristicLab est un environnement extensible, flexible et multi-paradigme pour concevoir rapidement des prototypes pour les nouveaux algorithmes et problèmes d'optimisation.

Différents algorithmes et problèmes ont été déjà réalisés dans le cadre de HeuristicLab parmi eux les algorithmes génétiques, programmation génétique, stratégies d'évolution, optimisation d'essaim de particules, optimisation par colonies de fourmis, recherche Tabou, problème de voyageur de commerce, problème d'allocation multiprocesseurs Etc.

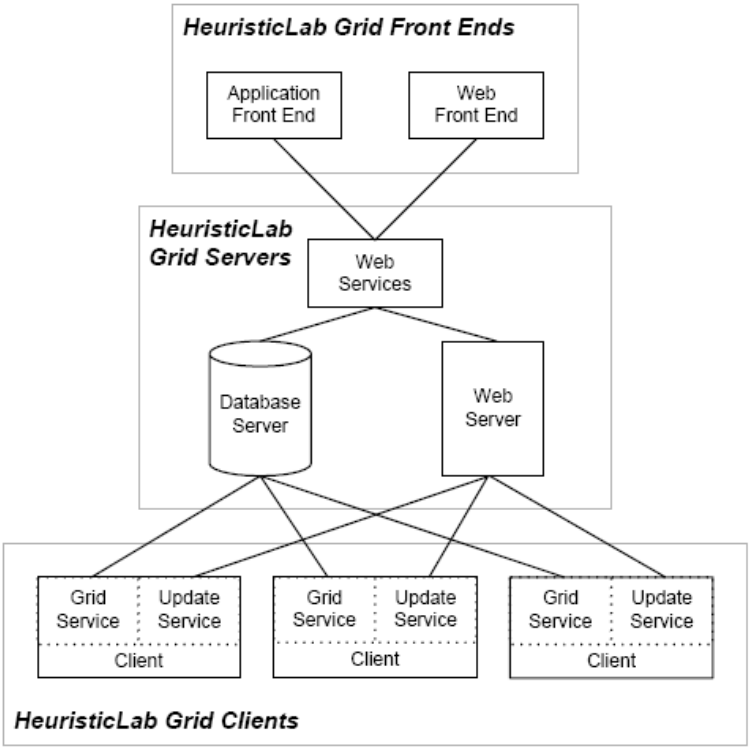

Figure 3. 5 L'architecture de Heuristic-Lab Grid

L'idée principale derrière la gridification de HeuristicLab était de prolonger le cadre de HeuristicLab d'une manière inspirée du calcul de grille ; par analogie à SETI@home un client est installé sur chaque ordinateur participant à la grille de HeuristicLab qui fonctionne avec une priorité très basse afin d'handicaper l'utilisateur le moins possible. Ce client cherche les travaux d'un serveur central de base de données. Il fait le traitement et il écrit les résultats dans la base de données. Les résultats calculés peuvent être encore traités par n'importe quel autre client et peuvent par exemple être employés pour produire de nouveaux travaux. De cette façon on constate que la communication est faite par l'intermédiaire de la base de données, par conséquent aucun raccordement direct entre les clients n'est nécessaire. En

conséquence la complexité du client est parfaitement réduite et le client peut être maintenu très mince [WA04].

5 Conclusion

Une grille dans le monde des systèmes distribués représente une collection des ressources informatiques hétérogènes et distribuées sur une grande surface géographique. Ces ressources sont liées via un réseau étendu ou via internet.

Les grilles diffèrent selon le mode de construction où on trouve les grilles communautaires et les grilles institutionnelles ; ou selon la fonction de la grille, où on distingue les grilles d'information, les grilles de stockage et de traitement de données et les grilles de calcul. Nous nous intéressons de cette dernière.

Notons que la construction d'une grille est impossible sans l'apparition des intergiciels. L'intergiciel représente le cœur de n'importe quelle architecture de grille ; il repose sur les informations collectées sur les ressources participantes dans la grille (couche inférieure) **pour offrir toute service nécessaire pour les applications tournantes** sur grille (couche supérieure).

Dans le domaine de l'optimisation, les grilles de calcul son utilisées pour résoudre des instances géantes de problèmes difficiles, soit en tant qu'Internet computing, global computing ou super calculateur virtuel ; mais les caractéristiques de la grille tels que l'hétérogénéité et la volatilité des ressource et le temps de latence, rendent la conception ou l'implémentation d'une application grille plus difficile ; donc n'importe quelle application doit respecter ces caractéristiques. Dans le chapitre suivant on essaye de concevoir une méthode parallèle et hybride adaptée aux caractéristiques des grilles.

Chapitre IV

Une méthode parallèle et hybride pour l'optimisation difficile

Cas : le problème de voyageur de commerce

1 Introduction

Dans ce chapitre, on va proposer une méthode coopérative, parallèle et hybride, pour résoudre de grandes instances des problèmes difficiles ; cette méthode doit respecter le plus possible les critères imposés pour concevoir une application sur une plateforme grille de calcul.

Le choix des méthodes de base n'est pas arbitraire ; la méthode d'optimisation par essaim de particules est choisie à cause de leurs propriétés d'auto-organisation et d'émergence ; la méthode ILS se base sur un mécanisme intéressant pour échapper de l'optimum local.

Dans ce qui suit, une description du problème du voyageur de commerce sera faite. Après, on discutera quelques choix pour l'implémentation suivis d'un aperçu sur la plateforme d'implémentation et avant conclusion tests et résultats seront présentés.

2 Recommandations

Quelques recommandations sont à respecter pour qu'une méthode soit adaptée à un environnement grille [LS01]:

A. la méthode doit être orientée calcul plutôt qu'orientée données : les algorithmes orientés donnée doivent communiquer de grandes quantités de données entre les processeurs, et pourront être affectés sérieusement si les propriétés de latence de la plateforme sont pauvres ; donc la méthode proposée doit avoir un minimum de taille de données et un minimum de quantité à communiquer.

B. la méthode doit avoir des faibles conditions de synchronisation : Les algorithmes fortement synchrones, qui exigent la coordination stricte entre les différentes parties du calcul global, peuvent fonctionner inefficacement quand la plateforme est

fortement hétérogène, ou ont des propriétés faibles de latence, ou si les processeurs deviennent constamment indisponibles. Dans tels algorithmes, le calcul entier peut être suspendu tout en attendant un processeur lent ou suspendu simple d'ouvrier pour accomplir sa tâche.

C. la méthode doit nécessiter la puissance de grille de calcul et justifier l'effort pour développer une application sur les grilles.

3 Premier pilier : l'optimisation par essaim de particules

L'optimisation par essaim de particules est une méthode d'optimisation basée population, elle est introduite pour la première fois par Dr. Russell C, Eberhart et Dr. James Kennedy en 1995 **Qu'ils ont inspirés cette méthode de la nature** [KE95].

La méthode se base sur un ensemble d'individus qui font l'exploration de l'espace de recherche avec un mécanisme d'interaction. On appelle chaque individu une particule. Chaque particule est caractérisée par deux propriétés principales:

- sa position courante (solution actuelle appartenant à l'espace de recherche)
- sa vitesse

Chaque particule survient toujours à deux positions, la meilleure position qu'elle a connue dans son évaluation, et la meilleure position connue dans son voisinage [Set05].

A la différence des algorithmes génétiques qui utilisent des mécanismes génétiques dans leurs évolutions, l'OEP fait l'appel à un mécanisme coopératif, utilise les informations position et vitesse pour améliorer les solutions visitées.

3.1 Démarche

Au début, l'essaim (l'ensemble de particules) est initialisé aléatoirement. C'est-à-dire associer à chaque particule une solution initiale ensuite à chaque itération. Chaque particule fait deux opérations principales : une adaptation de sa vitesse(1) et une adaptation de sa position(2). Les deux opérations faites appellent à la position courante, la vitesse courante, la meilleure position connue par la particule et la meilleure position connue par tout l'ensemble de voisinage de la particule (la figure 4.1). Les adaptations suivent les équations suivantes:

$$\mathbf{V_i(t)= V_i(t-1) + P_1 (X_{ipbest}-X_i) + P_2 (X_{gbest}-X_i)} \ \ldots\ldots\ldots\ldots\ldots..(1)$$

$$\mathbf{X_i(t)= X_i(t-1) + V_i(t).} \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2)$$

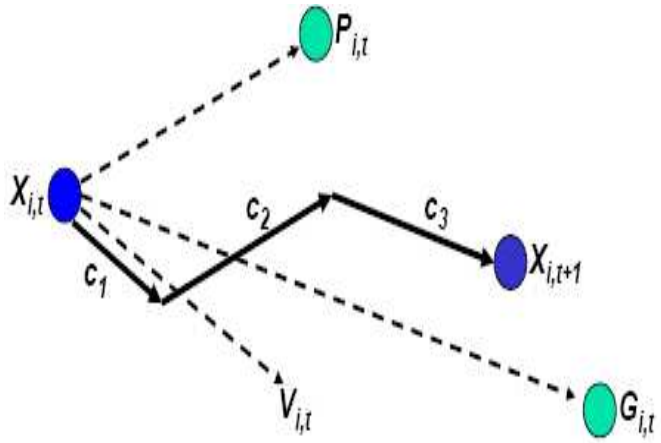

Figure 4. 1 Principe de calcul de la position

Le processus stoppe lorsque la condition d'arrêt est atteinte. Cette condition peut être un nombre maximal d'itérations, une fitness ou si l'évolution de processus ne fait pas augmenter la qualité des solutions avec un algorithme [Set05].

3.2 Les paramètres

Comme tous les métaheuristiques, l'efficacité de l'optimisation est fortement liée au paramètres choisis lors de l'implémentation ; pour l'optimisation par essaim de particules, on trouve des paramètres numériques tels que les coefficients P1 et P2 et des paramètres généraux tels que la façon d'initialisation de la population, le nombre des particules, la structure et le nombre des particules qui constituent l'ensemble de voisinage du chaque particule de la population.

On a P_1 et P_2 peut être définie comme :

- $P_1=R_1 * C_1$
- $P_2= R_2 * C_2$ ou R_1 et $R_2 \in [0\ 1[$.
- C_1, C_2 sont des constants avec $C^1 + C^2 \leq 4$.

Il existe d'autres paramètres tels que:

- Le nombre de particule.
- La vitesse maximale
- Le facteur d'inertie.

Si on prend les valeurs suivantes :
N nombre de particules
X_i position de la particule P_i
V_i vitesse de la particule P_i
pbesti meilleure fitness obtenue pour la particule P_i
gbest meilleure fitness obtenue dans tout le voisinage de P_i
X_{pbest}i position de la particule Pi pour la meilleure fitness
X_{gbest} position de la particule ayant la meilleure fitness de toutes
P_1 et p_2 valeurs aléatoires positives
Initialisation aléatoire de la population
Tant que la condition d'arrêt n'est pas atteinte
    Pour i de 1 à N faire
        Si ($f(X_i)$>$pbest_i$) alors
            $pbest_i \leftarrow f(X_i)$
            $X_{pbesti} \leftarrow X_i$
        Fin si
        Si ($f(X_i)$>gbest) alors
            $gbest \leftarrow f(X_i)$
            $X_{gbest} \leftarrow X_i$
        Fin si
    Fin pour
    Pour i de 1 à N faire
        $V_i \leftarrow V_{i-1} + p_1(X_{pbesti} - X_i) + p_2(X_{gbest} - X_i)$
        $X_i \leftarrow X_{i-1} + V_i$
    Fin pour
Fin tant que

Figure 4. 2 Pseudo code de la méthode OEP

3.3 Topologies des voisinages

Il y a 3 topologies principales de voisinage utilisées dans PSO (voir la figure): cercle, rayon et étoile. Le choix de la topologie de voisinage détermine quel X_{gbest} employer pour l'individu Xi.

Dans la topologie de cercle, chaque individu est connecté socialement à ses k voisins topologiques les plus proches (X gbest de Xi = mieux résultat individuel parmi ses k voisins plus proches, k égale typiquement 2).

La topologie de rayon, isole efficacement les individus les uns des autres, car l'information doit être communiquée par un individu focal.

La topologie d'étoile est la meilleure topologie connue ; ici chaque individu est relié à chaque autre individu (X gbest de Xi = mieux différents résultats dans la population) [Set05].

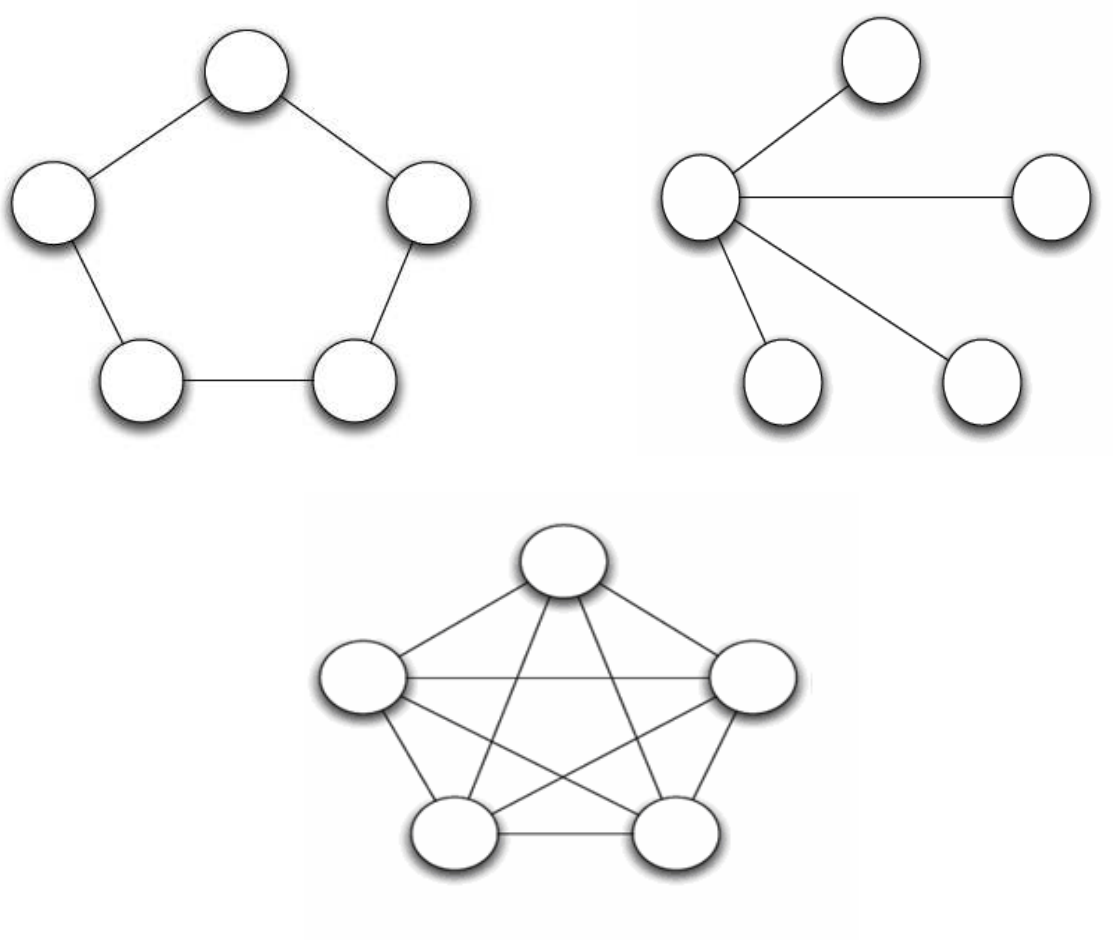

Figure 4. 3 topologies de voisinage

3.4 Les avantages et les inconvénients

L'avantage majeur de l'optimisation par essaim de particules est qu'elle donne de bons résultats avec un nombre réduit de calcul. Par rapport à une méthode basée population telle que les AGs, l'OEP exécute un nombre inférieur d'opération pour avoir des solutions avec la même qualité que celles obtenus par les AGs, ou peut-être mieux [HCWV04].

D'un autre coté, l'OEP souffre du problème de la convergence prématurée, c'est-à-dire la recherche converge rapidement vers un optimum local loin de l'optimum globale en terme de qualité. En plus l'OEP nécessite un effort additionnel lors du traitement des problèmes de nature discontinue.

3.5 Les modèles parallèles de l'OEP

Selon [BEG04], et comme les algorithmes génétiques, il existe 3 modèles parallèles d'OEP : le modèle global, le modèle à base de migration et le modèle diffus.

3.5.1 Le modèle global (à base de maître /esclave)

Dans ce modèle l'optimum global est situé toujours dans le maître et les opérations d'évaluation de la fonction de coûts, calcul de la vitesse et mis à jour de position sont faites au niveau des esclaves. Il est clair que le coût de communication est une fonction linaire avec le nombre de particules, donc on a un grand gain en terme de temps total de calcul à cause de cette réduction. D'autre part, si une particule trouve un optimum global elle fait le partager de cette information avec les autres par le biais du maître.

```
Tant que la condition d'arrêt n'est pas atteinte  Faire
        Meilleure_valeure ← une très grande valeur
        Pour tout x∈  esclave Faire
                Recois (valeur_esclave) de x
                Si valeur_esclave < valeur_courante
                        Meilleure_valeure ←
                valeur_esclave
                Fin Si
        Fin Pour
        Pour tout x ∈  slaves Faire
                Envoie (Meilleure_valeure) à x
        Fin Pour
Fin Tant que
```

Figure 4. 4 pseudo code du processus maître

```
Tant que la condition d'arrêt n'est pas atteinte  Faire
        Recois (Meilleure_valeure) du maître;
        calcule( x[],w,c1,c2);
        valeur_courante ← evalue( x[] );
        Envoie (valeur_courante) au maître;
Fin Tant que
```

Figure 4. 5 pseudo code du processus esclave

3.5.2 Le modèle de migration

Ce modèle est similaire au modèle d'ils des algorithmes génétiques. Dans ce modèle la population est subdivisée en un ensemble de sous-populations ; chacune d'elle est affectée à un élément de calcul. Chaque sous-population fait la recherche de l'optimum, la mise à jour de la vitesse et la mise à jour de la position. Après un nombre fini d'itérations la meilleur solution est migrée ver une population voisine.

Ce modèle se divise en deux parties; une partie fait l'émission et la réception du message, l'autre fait l'évaluation de coût et la mise à jour de la vitesse et la position.

```
Pour tout x∈ voisinage Faire
        Envoie (Ma_Meilleure_valeure) à x
Fin Pour
Pour tout x∈ voisinage Faire
        Recois (valeure_voisin) de x
        Si valeure_voisin < Ma_Meilleure_valeure
                Ma_Meilleure_valeure ← valeure_voisin
        Fin Si
Fin Pour
```

```
Tant que not optimal_is_reached()
        calcule( x[],w,c1,c2 );
        evalue( x[] );
        Update(L[])
        Ma_Meilleure_valeure:= choisie_meilleur( x[])
Fin Tant que
```

Figure 4. 6 pseudo code du modèle de migration de l'OEP

3.5.3 Le modèle diffus

Dans ce modèle toutes les opérations s'effectuant localement. Chaque particule a une information très limitée (pas de connaissance globale sur toute la population).

```
Tant que la condition d'arrêt n'est pas atteinte Faire
        Meilleure_valeure ← Evaluate(valeur_courante)
        Pour tout x∈ voisinage Faire
                Envoie(Meilleure_valeure) à x
        Fin Pour
        PourALL x∈ voisinage Faire
                Recois (valeure_voisin) de x
                Si valeure_voisin < valeur_courante
                        Meilleure_valeure ← valeure_voisin
                Fin Si
        Fin Pour
        valeur_courante ← w* valeur_courante + c1*random()*(Meilleure_valeure -
        valeur_courante)
Fin Tant que
```

Figure 4. 7 pseudo code d'un processus de modèle diffuse de l'OEP

Initialement chaque particule considère que sa valeur de coût est la meilleure. Ensuite à chaque itération elle subie l'information de son voisinage et fait une mise à jour. Si dans une

itération elle trouve que sa valeur est meilleure que la solution globale elle passe l'information à ses voisines.

4 Recherche locale itérée

4.1 Introduction

La recherche locale itérée (Iterated Local Search ILS) [LMS02] est une métaheuristique de recherche locale, simple et efficace.

Une recherche locale efficace est une recherche locale capable de trouver un bon minimum local. Si l'espace de recherche est grand ou/et le nombre des bassins d'attraction est grand dans ce cas là une recherche locale avec une réinitialisation aléatoire est insuffisante. L'idée proposée par ILS est au lieu de tracer une trajectoire à travers l'espace de recherche on trace une à travers l'ensemble d'optima locaux. Le problème est qu'il n'existe pas une méthode pour définir un voisinage de s* ∈ S* (l'ensemble d'optima locaux), ILS peut tracer une telle trajectoire en utilisant le pseudo suivant :

1. exécute une recherche locale à partir d'une solution initiale s jusqu'à ce que un optimum local s * soit trouvé.
2. perturbe s* et obtenue s'.
3. exécute une RL à partir de s' jusqu'à ce que un optimum local s'* soit trouvé
4. à base de critère d'acceptation on juge le passage s*← S'*.
5. allez vers 2.

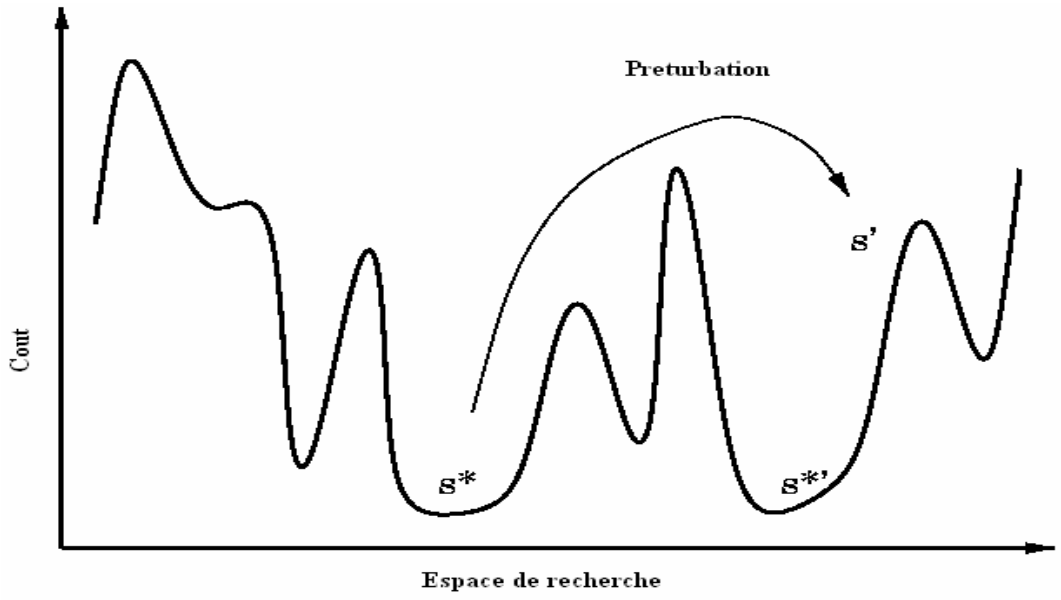

Figure 4. 8 mode d'exploration de l'espace de recherche en ILS

4.2 Déroulement

ILS applique une recherche locale sur une solution initiale jusqu'à ce qu'elle trouve un optimum local ; puis elle perturbe la solution trouvée et refait la recherche locale.

La construction d'une solution initiale doit être rapide et non coûteuse. Une manière simple et efficace est de générer la solution initiale aléatoirement ; on peut aussi utiliser une méthode constructive.

La perturbation a comme but de produire une solution initiale pour démarrer une nouvelle recherche locale. Cette dernière doit conduire à un optimum local différent du premier et plus proche de lui qu'un optimum obtenu par un redémarrage aléatoire de la recherche locale.

L'importance de la perturbation est évidente ; si la perturbation est petite donc elle ne permettra pas d'échapper d'un bassin d'attraction de l'optimum local ; par contre une grande perturbation rendra l'algorithme comme une recherche avec réinitialisation aléatoire.

Pour éviter les cycles, la perturbation est généralement non déterministe. Un aspect important est l'importance de la perturbation qui peut être constante ou variable, dans la première la distance entre s* et s' reste constante indépendamment de la taille du problème. Une perturbation variable est plus efficace et adaptable au problème à grande taille. On peut aussi varier l'importance de la perturbation, on l'augmente lorsqu'on veut diversifier la recherche et on la baisse si on veut intensifier la recherche.

Le critère d'acceptation joue le rôle de contrebalance, comme il filtre et fait le retour en arrière vers la perturbation selon les caractéristiques du nouvel optimum local.

Le troisième composant de la méthode est le critère d'acceptation, il varie entre les deux extrêmes :

1. accepter les solutions avec améliorations
2. accepter toutes les solutions optimales.

On peut utiliser d'autres techniques, par exemple une acceptation basée sur un critère semblable à celui du recuit simulé, c à d :

$$\mathbf{p}(Accept(\hat{s},\hat{s}',history)) = \begin{cases} 1 & \text{if } f(\hat{s}') < f(\hat{s}) \\ \exp(-\frac{f(\hat{s}')-f(\hat{s})}{T} & \text{otherwise} \end{cases}$$

Le chemin de refroidissement peut être monotonic (pas d'augmentation dans le temps) ou non-monotonic (adaptable pour équilibrer entre la diversification et l'intensification).

La figure 4.9 donne un pseudo-code de la méthode de recherche itérée.

```
s(0) = GenerateInitialSolution
s* =  LocalSearch(s_0)
REPEAT
        s' =  Perturbation(s*,history)
        s*' = LocalSearch}(s')
        s* =  AcceptanceCriterion(s*,s*',history)
UNTIL termination condition met
```

Figure 4. 9 pseudo code la méthode ILS

5 *La méthode proposée*:

La méthode proposée a pour but de vaincre deux inconvénients de l'optimisation par essaim de particules implémenté sur un environnement largement distribué ; ces inconvénients sont :

- une grande quantité de communication inter-processus (particule).
- la convergence primature à cause du chemin mince exploré par cheque particule.

La méthode basée sur une population d'individus (solutions) coopérant entre eux pour résoudre un problème. Chaque individu exécute un algorithme identique aux autres et contient trois mécanismes :

- un mécanisme pour la coopération entre ses éléments, ce mécanisme est inspiré de l'OEP.
- un mécanisme pour échapper d'un optimum local inspiré du ILS.
- un mécanisme de recherche locale basé sur la descente récursive.

5.1 Description

Au début, chaque individu construit une solution initiale (*Sinit*) d'une manière aléatoire, ou avec une méthode constructive. Après il exécute une recherche locale avec la méthode descente – par exemple- pour obtenir rapidement un optimum local qui devient la nouvelle solution (initialisation du nouvel optimum local Noptlocal). Ensuite il envoie leur (Noptlocal) au voisinage et reçoit les autres optimums pour calculer l'optimum global.

L'étape suivante consiste à affecter la valeur de (Noptlocal) au (Eoptlocal), c.-à-d. la nouvelle solution devient une ancienne solution. Ensuite une perturbation est appliquée sur (Eoptlocal*)* pour construire une nouvelle Sinit selon l'équation :

Sinit = Eoptlocal + Q*(optglobal - Eoptlocal)

Où « Q » est un cœfficient de perturbation, après il exécute une recherche locale, là on a deux situations :

- Si on revient au même optimum local, on augmente le pas de perturbation avec l'augmentation de Q et on réfait la recherche locale.
- Si on obtient une nouvelle solution (Noptlocal) – pas forcément mieux que l'ancienne- on fait le retour à l'étape de la communication et mise à jour de l'optimum global.

Ce processus est répété jusqu'on a un état de convergence vers une solution globale ou si on exécute un nombre fixe d'itérations. La figure 4.10 représente le pseudo code de la méthode.

La conception de cette méthode a plusieurs buts, on note :

1. la minimisation de la communication inter-processus : dans la plupart du temps les instructions seront exécutées sur des données locales ; car chaque processus ne fait la communication que s'il trouve un nouvel optimum local, donc on a un nombre réduit de points d'envoi et de réception, ce qui donne un degré de liberté à chaque processus et minimise les contraintes de synchronisation.
2. une bonne exploitation de l'espace de recherche : par l'intensification assurée par la méthode de recherche locale ILS. À chaque itération chaque processus exploite leur voisinage ou une grande portion de voisinage.
3. une exploration guidée de l'espace de recherche : chaque processus visite une grande partie de l'espace de recherche par le mécanise d'échapper de l'optimum locale ; en une autre expression, la méthode offre un mécanisme de perturbation guidé (en puissance et direction) pour la méthode ILS.
4. un mécanisme de coopération qui permet de dire que la solution finale est une solution émergente de tout le processus de recherche.

```
Phase d'initialisation
Initialisation (Sinit, Qinit, facteur,…….)
Noptlocal ← recherche locale (Sinit)
Fin d'initialisation
TQ la condition d'arrêt ne pas atteinte faire
        Phase de communication
        Best-value ← évaluation (Noptlocal)
        Mètre à jour (optglobal)
        Pour x ϵ voisinage faire
                Envoyez (Best-value) à x
                Envoyez (optglobal) à x
        Fin pour
        Pour x ϵ voisinage faire
                Reservoir (neighbour -value) de x
                Reservoir (neighbour-solution) de x
                Si neighbour _value < Best-value
                        optglobal← neighbour-solution
                Fin si
        Fin pour
        Fin phase de communication
        Si (Noptlocal <> optglobal)
                Eoptlocal ← Noptlocal
                Q ← Qinit
                TQ Eoptlocal = Noptlocal faire
                        Sinit ← perturbation (Sen, Q, optglobal ……)
                        Snou← recherche local(Sinit)
                        Q ← Q+Q*facteur
                Fin TQ
        FinSi
Fin TQ
```

Figure 4. 10 pseudo code de la méthode proposée

6 Le problème de voyageur de commerce

Imaginer un vendeur de marchandise qui veut faire une tournée à travers N villes pour vendre sa marchandise et rentre chez lui à condition que chaque ville soit visitée une seule fois. Le passage d'une ville à une autre se fait avec un coût donné. Le but est de trouver le chemin qui présente le coût minimum.

Le problème de voyageur de commerce (PVC) ou Travelling Salesman Problem (TSP) est largement étudié en informatique. Le PVC a tenu l'intérêt des informaticiens et des mathématiciens parce que ce problème n'a pas été complètement résolu jusqu'à présent

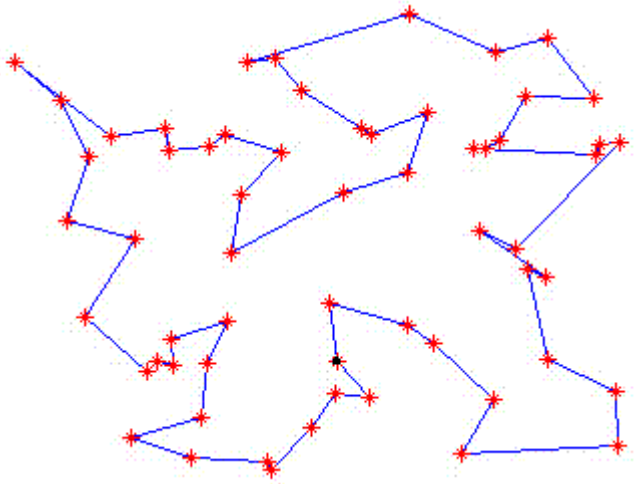

Figure 4. 11 une tourné de voyageur de commerce

.Le PVC appartient à la classe des problèmes difficiles ou NP-complet ; il peut être appliqué pour résoudre beaucoup de problèmes pratiques dans notre vie quotidienne [Zam06].

6.1 Définition mathématique

Du point de vue formel, le PVC est énoncé comme ; étant donné G un graphe complet, avec un ensemble V des sommets, un ensemble E des arcs, et C_{ij} représente le coût associé à l'arc qui lie le sommet **i** et le sommet **j**. Une solution au PVC doit renvoyer le cycle hamiltonien de G qui donne le coût le plus bas. Un cycle Hamiltonien est un cycle qui visite chaque nœud dans un graphe seulement une fois.

Une autre formulation de PVC, étant donné $D = d_{ij}$ la matrice symétrique n*n, où d_{ij} représente la distance entre i et j, le problème consiste à arranger les points dans un ordre cyclique de telle manière que la somme de d_{ij} entre les points consécutifs soit minimale [Zam06].

6.2 Complexité

Ce problème appartient à la classe NP-difficile et pour montrer la difficulté de ce problème, supposant qu'on a 50 villes donc il existe 49! Solutions possibles, ce qui est égal $6{,}08*10^{62}$ tournées possibles. Prenons un ordinateur de π Millimètres calculant un milliard de solutions par seconde. Sachant que le diamètre équatorial de la terre est de 12756 kilomètres, mettons 12756000000 de ces ordinateurs les uns à la suite des autres sur l'équateur. On peut ainsi calculer 12576000000000000000 solutions par seconde. Pour être certain de trouver la tournée la plus courte ; il faut considérer toutes les tournées possibles. Il nous faudra alors

$4{,}766*10^{43}$ secondes avec le super ordinateur que l'on vient de présenter. Notons que $4{,}766.10^{43}$ est l'équivalent de $1{,}51*10^{34}$ siècles.

D'une manière formelle, pour démontrer que PVC est NP-complete nous emploierons la méthode populaire de la réduction. Nous prendrons un problème, l'appelons H, on sait déjà que H est NP-complete. Puis, nous prouverons qu'on peut résoudre H en le ramenant au PVC. Une fois que nous avons démontré ceci, nous pouvons dire que si PVC est soluble dans un temps polynomial, alors nous pouvons résoudre H dans un temps polynomial. Lorsque nous atteignons une contradiction, nous **concluons** alors que PVC ne peut pas être résolu dans un temps polynôme parce qu'on a déjà montré que H ne peut pas être résolu dans un temps polynomial. D'ailleurs, PVC doit être NP-complete parce que H est NP-complete [Zam06].

Pour démontrer que PVC est NP-complete, nous choisirons H le problème populaire de cycle hamiltonien, qui est connu NP-complete. À un regard étroit, il est évident que le problème de cycle hamiltonien est un cas spécial de PVC. On doit tout simplement modifier le graphe d'entrée de l'algorithme de PVC en plaçant les coûts de tous les arcs existants pour être à un certain constant fixe. Puis, si n'importe quelle excursion est trouvée, cette excursion est un cycle hamiltonien. Ainsi, le PVC doit être NP-complete parce qu'on a montré que le problème de cycle hamiltonien est NP-complete[Zam06].

6.3 Historique

L'origine de ce problème est quelque peu obscure. Dantzig, Fulkerson, et paroles de Johnson [1954]:'' It appears to have been discussed informally among mathematicians at mathematics meetings for many years.'' c à d, Il semble avoir été discuté officieusement par les mathématiciens dans les réunions de mathématiques pendant beaucoup d'années.

Il se peut que le problème PVC a été déclenché par le jeu de Hamilton, où Les joueurs devaient trouver une tournée passant par 20 points en utilisant uniquement les connections prédéfinies [DFJ54].

- **Dans les années 1930** le PVC est traité plus en profondeur par Karl Menger à Harvard. Il est ensuite développé à Princeton par les mathématiciens Hassler Whitney et Merril Flood [Sch60].
- **1954** Solution du PVC pour 49 villes par Dantzig, Fulkerson et Johnson par la méthode du *cutting-plane* [DFJ54], les villes représentent 48 capitales états d'ETATS-UNIS plus Washington.

- **1975** Solution pour 100 villes par Camerini, Fratta and Maffioli
- **1987** Solution pour 532, puis 2392 villes par Padberg et Rinaldi
- **1998** Solution pour les 13 509 villes des Etats-Unis.
- **2001** par Applegate, Bixby, Chvàtal et Cook des universités de Rice et Princeton appliquent une version parallèle de la méthode de Danzig, Fulkerson et de Johnson sur une grande instance de PVC. Ils ont obtenu la solution optimale du PVC qui a 15.112 villes (nœuds) en Allemagne, Le calcul a été exécuté sur un réseau de 110 processeurs et ils ont estimé que tout le temps de calcul était de 22,6 ans, mesurés à un processeur d'alpha de Compaq EV6 (21264) fonctionnant à 500MH [FKTY03]
- **En mai 2004** le PVC qui visite chacune des 24.978 villes en Suède a été résolu: une excursion de longueur approximative de 72.500 kilomètres a été trouvée et on a démontré qu'aucune excursion plus courte n'existe.
- **En mars 2005**, le PVC qui visite chacun des 33.810 points dans une carte a été résolu en utilisant CONCORDE. une excursion d'une longueur de 66.048.945 a été trouvée et on a démontré qu'aucune excursion plus courte n'existe. Le calcul a pris approximativement 15,7 années d'unité centrale de traitement (Cook et al. 2006).

6.4 Problèmes réels se modélisant sous forme du PVC

Outre les problèmes de transport et de livraison des marchandises, plusieurs problèmes de la vie quotidienne peuvent être modélisés sous forme du PVC. Parmi ces problèmes, on cite :

- La tournée des avions
- Le tour d'un supporteur de base-ball
- Collection de la monnaie des cabines téléphoniques
- La chaîne d'ADN
- Conception des réseaux à fibres optiques
- la mise en place de plomberie dans un bâtiment ou le câblage `électrique`
- le câblage d'un circuit imprimé

6.5 Les variantes du PVC

Le problème de voyageur de commerce se présente sous différentes formes selon la nature de la distance entre villes (symétrique, asymétrique, euclidien…etc.) ou l'ensemble des villes à visiter (tous les N villes ou bien un sous-ensemble de N…etc.).

Le PVC Symétrique (STSP: Symetric Traveling Salesman Problem)*:* Le PVC symétrique est le PVC le plus simple, car la distance entre deux villes quelconques x et y est égale à celle entre y et x.

Le PVC Asymétrique (ATSP **: Asymetric Traveling Salesman Problem*)* :** Avec le PVC asymétrique, la distance entre deux villes x et y n'est pas forcément égale à celle entre y et x.

Le PVC Avec Collection (PCTSP:Prize-Collection Traveling Salesman Problem): Dans les autres variantes du PVC le tour doit passer, exactement, une fois par chaque ville du problème, ce qui n'est pas le cas avec le PCTSP. Dans ce cas on ne visite qu'un sous ensemble k de l'ensemble des villes N.

Le PVC Euclidien : Les villes sont représentées dans un espace bidimensionnel, la distance entre deux villes quelconques x, y est calculée à partir des abscisses et des ordonnées des deux villes (**distance=** $\sqrt{(x2 - x1) + (y2 - y1)}$) ou x_1 et y_1 sont les coordonnées de la ville x et x_2 et y_2 représente les coordonnées de la ville y).

7 Implémentation

L'implémentation et le test d'une méthode sur un problème donné nécessitent de faire plusieurs choix, ces choix sont liés soit à la méthode, soit à l'environnement matériel de l'implémentation ou soit au problème. Les paragraphes suivants expliquent et justifient les choix que nous avons adoptés pour l'implémentation de la méthode sur un environnement distribué.

7.1 Choix du mode de déploiement

Il existe deux modes pour le déploiement des tâches sur la grille, on a :

7.1.1 Le modèle Maître -travailleur (mastre-worker)

Là, les nœuds sont constitués par un client (maître) et plusieurs serveurs (les travailleurs). Le maître fait la distribution des tâches, l'initialisation d'application, le contrôle de déroulement et la récupération des résultats. Les travailleurs font les calculs et renvoient

les résultats au maître; ce paradigme est le plus utilisé pour l'implémentation des algorithmes parallèles sur les grilles.

7.1.2 Le modèle pair à pair (Peer to Peer)

Il n'existe pas de contrôle central lors de calcul. Chaque nœud de calcul est constitué d'un client et un serveur. Peu de travaux utilisent ce mode dans le développement des applications sur la grille.

Pour la méthode proposée j'ai choisi le modèle pair à pair pour deux raisons

- Le modèle pair à pair est plus général que le modèle maître-travailleur.
- La nature de la méthode n'exige pas un contrôle central, elle est décentralisée et auto-organisatrice

7.2 Choix du langage de la programmation

Le choix du langage est un facteur pertinent. Pour le développement des applications parallèles sur un environnement grille de calcul on trouve deux langages qui sont largement utilisés ; le langage C++ et le langage Java [Mal05].

- le langage C++ : plusieurs travaux présentés dans le chapitre précédent utilisent C++ comme un langage d'implémentation, à titre d'exemple dans [AUB+05], PradisEO [CMT04] et Malba [AtMG02]. Le langage C++ caractérisé par son efficacité avec l'existence d'une bibliothèque riche pour les calculs scientifiques et son adaptation au calcul parallèle [Mal05], mais C++ a un problème de portabilité.

- le langage Java : Les résultats de beaucoup de recherches récentes ont montré que l'efficacité des applications implémentées avec java peut venir très près de celle de C ou de Fortran ; en plus la portabilité de java lui donne un bon avantage pour être un langage d'implémentation pour les applications sur la grille de calcul. On trouve Java par exemple dans [ANT02] et dans [ACE+02] pour la plateforme DREAME.

Pour l'approche proposée et parce que la grille est hétérogène on a choisi le langage java.

Le langage Java a plusieurs mécanismes intégrés qui permettent au parallélisme inhérent à un programme donné d'être exploité.

Les threads sont bien adaptés aux machines avec mémoire partagée, mais non aux machines de mémoire distribuée. Pour les applications réparties, Java fournit les sockets et le mécanisme de RMI (Remote Method Invocation). Dans le monde de calcul parallèle, les sockets sont souvent de niveau trop bas et le RMI est orienté trop vers le modèle client/serveur. Ce modèle ne soutient pas spécifiquement le modèle de communication symétrique adopté par beaucoup d'applications parallèles. Heureusement il y a un modèle dans l'ensemble des modèles de programmation fournis par Java particulièrement pour supporter la programmation parallèle sur les clusters et les machines à mémoire distribuée ; la solution de ce problème construite inévitablement autour du modèle de passage de message, qui a été un des paradigmes les plus populaires de programmation parallèle depuis les années 80[GLPF01].

Contrairement aux sockets et au RMI, le passage de message soutient directement les communications symétriques comprenant le pair à pair, les opérations collectives (broadcast, gather, allto-al) et d'autres, comme défini par la norme de MPI [GLPF01].

La programmation avec MPI est facile parce qu'elle soutient le modèle multiple données programme simple (SPMD) du calcul parallèle et c'est le cas de notre approche, où un groupe de processus coopèrent en exécutant des images identiques de programme sur des valeurs locales de données [GLPF01].

7.3 Passage de message

Le passage de message est probablement le modèle aujourd'hui le plus largement répandu de programmation parallèle. Les programmes *Message-passing* créent des tâches multiples, chacune encapsule ses données locales et chacune d'elles est identifiée par un nom unique. Les tâches agissent l'une sur l'autre en envoyant et en recevant des messages à et des tâches appelées.

Théoriquement ce modèle permet la création dynamique des tâches, l'exécution de plusieurs tâches par un processeur, ou l'exécution de différents programmes par différentes tâches. Cependant, dans la pratique la plupart des systèmes passage de message créent un nombre fixe de tâches identiques au démarrage de programme et ne permettent pas aux tâches d'être créées ou détruites pendant l'exécution du programme, P2PMPI [Rat04] est un exemple typique.

On dit que ces systèmes mettent en application un modèle de programmation *programme simple données multiples (*SPMD) parce que chaque tâche exécute le même programme sur des données différentes. Ce modèle est suffisant pour un éventail de

problèmes de programmation parallèle, mais il gêne quelques développements d'algorithme parallèles.

Du point de vue programmation, ce modèle immerge sous forme de bibliothèques des sous-programmes, Ces bibliothèques fournissent des routines pour l'initialisation et la configuration de l'environnement de transmission de messages aussi bien que l'envoi et la réception des paquets des données. Actuellement, deux bibliothèques de passage de messages de haut niveau sont les plus populaires pour les applications scientifiques et industrielles :

- les PVM (machine virtuelle parallèle) du laboratoire national d'Oak Ridge
- MPI (interface de passage de messages) défini par le forum de MPI.

7.3.1 PVM

Le PVM [AT02] est un support logiciel qui soutient le modèle passage de message et permet l'utilisation d'un ensemble hétérogène d'ordinateurs parallèles ou séquentiels comme une seule ressource informatique concourante flexible simple.

PVM offre une suite des primitifs d'interface utilisateur pour la communication et d'autres services, et il comporte deux avantages :

- il supporte les collections des calculateurs hétérogènes.
- les applications implémentées avec PVM sont en général tolérantes aux pannes.

La gestion d'une collection dynamique de ressources informatiques potentiellement hétérogènes comme simple ordinateur parallèle est le vrai festin attrayant de PVM.

Malgré ses avantages, PVM a commencé d'être non soutenu. Les utilisateurs décalent de PVM à des paradigmes plus efficaces, tels que MPI.

7.3.2 MPI

MPI [AT02] est un standard composé de deux documents, MPI-1 et MPI-2, qui spécifie une bibliothèque des fonctions pour permettre le passage des messages entre des nœuds dans un environnement de calcul parallèle. Bien que les standards de MPI spécifient plus de 300 fonctions, une grande classe des applications peut exécuter tout le passage du message exigé en utilisant moins de 10 fonctions communes de MPI.

MPI offre un ensemble riche de fonctions qui peuvent être combinées de manières simples ou complexes pour résoudre n'importe quel type de calcul parallèle.

MPI a été mis en application sur une grande variété de plateformes, systèmes d'exploitation, clusters et supercalculateurs.

Grace à sa fonctionnalité et sa flexibilité riches, MPI est devenue le standard pour les applications parallèles basées sur le modèle passage de message.

Il y a beaucoup d'implémentations de MPI; certaines sont dérivées des projets de sources ouvertes telle que MPICH et d'autres sont des produits commerciaux.

7.4 Plateforme

Le système proposé consiste à implémenter la méthode sur un ensemble des ordinateurs pas forcement homogènes. Les trois choix précédemment faits (le mode de déploiement pair à pair, le langage JAVA et le mode de programmation avec MPI) ont conduit à choisir la plateforme P2PMPI pour implémenter et tester la méthode proposée.

<table>
<tr><td colspan="3">La couche application</td></tr>
<tr><td colspan="3">P2PMPI</td></tr>
<tr><td colspan="3">JXTA</td></tr>
<tr><td colspan="3">La machine virtuelle Java</td></tr>
<tr><td>Linux</td><td>Windows</td><td>……..</td></tr>
</table>

Figure 4. 12 architecture de la plateforme de l'implémentation

7.4.1 P2PMPI

P2P_MPI[1] [Rat04] est un intergiciel (Middleware) de type disktop_grid développé par l'équipe TAG du laboratoire ICPS de Strasbourg. L'objectif de P2PMPI est de fournir un environnement pour la programmation et l'exécution d'applications parallèles en grille. P2PMPI a deux rôles, c'est un intergiciel (Middleware) donc il offre des services au niveau système à l'utilisateur tels que la découverte des ressources, le transfert des fichiers, le lancement des taches...etc. Le deuxième rôle est celui de l'API (Application Programming Interface) de programmation parallèle qu'il fournit au développeur.

L'API : Contrairement de la plupart des travaux destinés au calcul parallèle sur les grilles qui se basent sur le modèle client/serveur ou les appels RPC ; P2PMPI offre un modèle plus général basé sur le passage de message, le modèle client serveur sera un cas particulier.

[1] http://grid.u-strasbg.fr/p2pmpi/

On trouve dans la distribution de P2PMPI une librairie de communication conforme à la spécification de MPJ [CGJ+00].

Middleware : Si un utilisateur veut faire son ordinateur joindre une grille P2P-MPI il pourra le faire en tapant simplement *mpiboot* qui lance un processus local *gatekeeper*. Le *gatekeeper* peut jouer deux rôles:

(i) il annonce son ordinateur local aussi disponible au reste de la communauté, et décide d'accepter ou décliner les demandes de travail d'autres pairs.

(ii) quand l'utilisateur émet une demande de travail, le *gatekeeper* est chargé de trouver le nombre demandé de pairs et ensuite d'organiser le lancement du travail.

Le lancement d'un travail MPI exige l'assignation d'un identifiant unique à chaque tâche (processus) ; puis la synchronisation de tous les processus à la barrière de MPI_Init.

Quand un utilisateur (le présentateur) émet une demande du travail impliquant plusieurs processus, le gatekeeper lance une découverte pour trouver le nombre demandé de ressources pendant une période limitée. P2P-MPI utilise JXTA [SunMS05] pour effectuer toutes les opérations habituelles de pair-à-pair telle que la découverte des ressources qui peuvent être trouvées parce qu'elles ont annoncé leur présence et leurs caractéristiques techniques quand elles ont joint le groupe des pairs.

Une fois qu'assez de ressources ont été choisies, le *gatekeeper* contrôle d'abord, si les participants sont encore disponibles (avec des Ping) et fait la construction d'un tableau contenant des nombres assignés à chaque processus participant (appelé le communicateur). Puis, à l'aide d'un service spécifique, le *gatekeeper* envoie à chaque ordinateur choisi le programme et les données d'entrée ou le URL des données. Chaque ordinateur choisi fait l'acquisition du transfert et démarre l'exécution du programme reçu, si quelques ordinateurs échouent avant d'envoyer l'acquisition**,** un débordement du temps expirera du côté de l'émetteur et le travail sera arrêté.

Pour vaincre la nature volatile des grilles P2PMPI dispose d'un mécanisme de détection et de tolérance contre les pannes basé sur la réplication des processus qui permettent d'augmenter la robustesse des applications. Avec ce mécanisme, P2PMPI peut éviter le temps supplémentaire qui résulte de l'utilisation de la technique de point de restauration utilisée par d'autres intergicièls.

7.4.2 JXTA

JXTA[1] [SunMS05] est un projet Open Source initié par Sun Microsystems en avril 2001. JXTA vient du mot anglais « Juxtapose ». Le but du projet est de développer un ensemble de protocoles permettant à un ensemble d'appareils (PC, téléphone mobile, serveur…) de s'interconnecter et de collaborer avec les autres. Les peers JXTA créent un réseau virtuel au dessus du réseau physique, cachant ainsi la complexité de celui-ci. Dans un réseau virtuel JXTA, chaque peer peut interagir avec tous les autres indépendamment de leurs localisations physiques et ses infrastructures centralisées.

L'ensemble des protocoles JXTA est indépendant de tout langage de programmation, de toute plateforme réseau et tout système d'exploitation, car ils sont Basés sur des standards tels que TCP/IP, HTTP et XML, mais leurs implémentations diffèrent selon l'environnement.

Plusieurs implémentations de JXTA existent, celle qui nous intéresse particulièrement est celle spécifique à Java.

Terminologie

Le projet JXTA utilise une terminologie spécifique dont on peut énoncer les éléments principaux :

- Un *peer* représente un utilisateur individuel et/ou sa machine (PC, PDA, téléphone mobile), un processeur ou un processus.
- Un *peer group* définit un ensemble de peers qui collaborent.
- Un *advertisement* est constitué d'un document XML qui contient des informations (nom, ID, propriétés, services disponibles) sur un peer ou un service.
- Une *pipe* définit un canal de communication asynchrone.
- Un *message* constitue l'unité d'échange d'informations est également représenté par un document XML.

Les protocoles JXTA

L'ensemble de protocoles JXTA est actuellement composé de six protocoles situés comme dans la figure ci-dessous:

- Peer Resolver Protocol (PRP) : utilisé pour envoyer une requête à un nombre indéterminé de peers et recevoir une réponse.

[1] http://grid.u-strasbg.fr/p2pmpi/

- Peer Discovery Protocol (PDP) : utilisé pour annoncer et découvrir du contenu.
- Peer Information Protocol (PIP) : utilisé pour obtenir l'état d'un peer.
- Pipe Binding Protocol (PBP) : utilisé pour créer un chemin de communication.
- Peer Endpoint Protocol (PEP) : utilisé pour trouver une route entre deux peers.
- Rendezvous Protocol (RVP) : utilisé pour propager les messages dans le réseau.

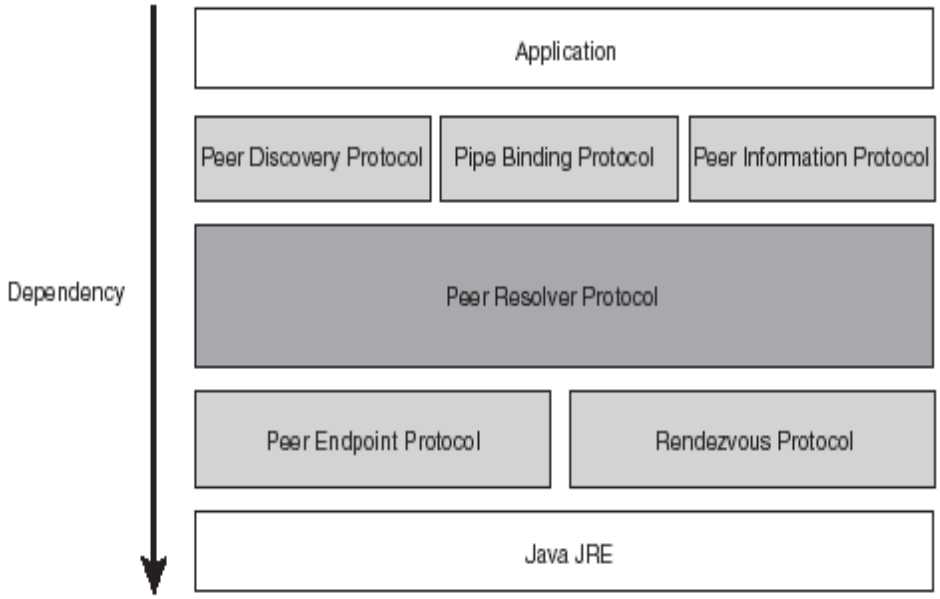

Figure 4. 13 les protocoles du JXTA

Les protocoles JXTA fournis aux peers les services suivants :

- Se découvrent les uns les autres.
- Se constituent en groupes.
- Publient et découvrent des services disponibles.
- Communiquent entre eux.

7.5 Spécifications pour le problème de voyageur de commerce

Dans cette section on parle de deux points :

1. la méthode de recherche locale.
2. la structure de données.

7.5.1 La structure des données

Pour la structure de données et pour des raisons de simplicité j'ai fait les choix suivants :

a. Chaque solution est représentée à l'aide d'un tableau de la taille N+1 où N est le nombre des villes, ce qui donne O(1) comme complexité de consultation et O(n) pour l'opération d'inversion des segments.

b. Même structure pour la matrice des positions des villes ; cette matrice contient le numéro de chaque ville suivie de leurs indices, cette matrice a la taille N*3.

c. la vitesse (l'élément principal de la perturbation), qui représente une liste de permutation entre les segments, est une structure de taille variée nommée ListArray.

L'utilisation du tableau comme structure pour les chemins est justifié par la taille des problèmes pris comme exemples (<1000 villes) ; mais si N> 1000 villes, on constate qu'une autre structure de données est nécessaire. Il faut utiliser un arbre à deux niveaux. Cet arbre est très difficile à implémenter ; mais il donne O($\sqrt{n}$) de temps par inversion, ces arbres feront le travail pour des tailles de problèmes allant jusqu'à 1000000 villes.

Une autre structure, les arbres étendus qui ont une complexité, dans le pire des cas égale à O($\log_2(n)$) pour les mouvements et les consultations; ces arbres surpasseront les deux structures précédentes pour les problèmes de grandes dimensions[FJMO95].

7.5.2 La méthode de recherche locale

Pour cette application j'ai choisi la méthode de la descente récursive grâce à leur simplicité, en plus elle obtient rapidement un optimum local. Pour l'implémentation de cette méthode on utilise le voisinage 2-opt (voir la figure). Donc à chaque itération, on génère un ensemble de voisinage 2-opt de la solution courante par la permutation de deux arcs de cette solution, ensuite on fait le choix de la meilleure solution pour qu'elle soit la solution courante de la prochaine itération de la descente. On note que si on considère que N est la taille de données (dans notre cas le nombre de villes) l'ensemble de voisinage 2-opt contient N*(N-3)/2 solution voisine.

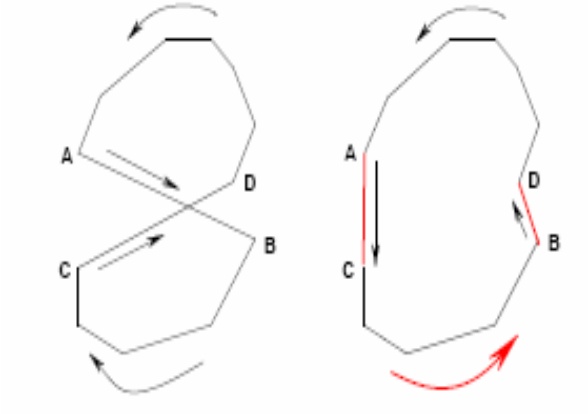

Figure 4. 14 permutation 2-Opt

7.6 Phase expérimentale

Une implémentation de l'application est faite avec l'outil JDK 1.5.0 :

L'exécution sur une plateforme logicielle :

- Windows XP SP2
- jre-6u1-windows-i586-p-s
- jxta-lib-2.4.1
- p2pmpi-0.20.1

Sur une plateforme matérielle de huit postes connectés de la configuration suivante:

Poste	Processeur	RAM
Poste1	Intel pentium 1.80 GH	224 Mo
Poste2	Intel celeron 2.00 GH	480 Mo
Poste3	Intel pentium 2.40 GH	192 Mo
Poste4	Intel celeron 1.78 GH	96 Mo
Poste5	Intel celeron 2.00 GH	96 Mo
Poste6	Intel celeron 2.10 GH	96 Mo
Poste7	Intel celeron 2.00 GH	96 Mo
Poste8	Intel celeron 2.00 GH	96 Mo

Les tests sont faits sur des problèmes de TSPLIB[1] (a280 de 280 villes ; KorA100, KorB100, KorC100 de 100 villes ; KorA150, KorB150, KorC150 de 150 villes ; KorA100, KorB200, KorC200 de 20 villes) avec des distances euclidiennes 2D l'affichage des résultats est en mode console.

[1] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

7.7 Exécutions et tests

Test N°1 amélioration donnée par la méthode

Ce test a pour but de montrer le gain de cette méthode par rapport à la méthode de la descente récursive, et on a les conditions suivantes :

- un voisinage total pour les solutions
- chaque exécution est de 200 itérations
- nombre de processeurs égal à 4 (réduit)

Les résultats figurent dans le tableau suivant :

Le problème	La valeur initiale	La décente	La valeur finale
Eil51_tsp	1639	577	467
Eil76_tsp	2580	766	587
Eil101_tsp	3675	824	707
kroA100_tsp	185813	32109	30463
kroA150_tsp	267140	45953	34130
kroA2000_tsp	386458	8832	7251
A280	34563	3689	3515
Pr1002_tsp	6451168	516464	416720

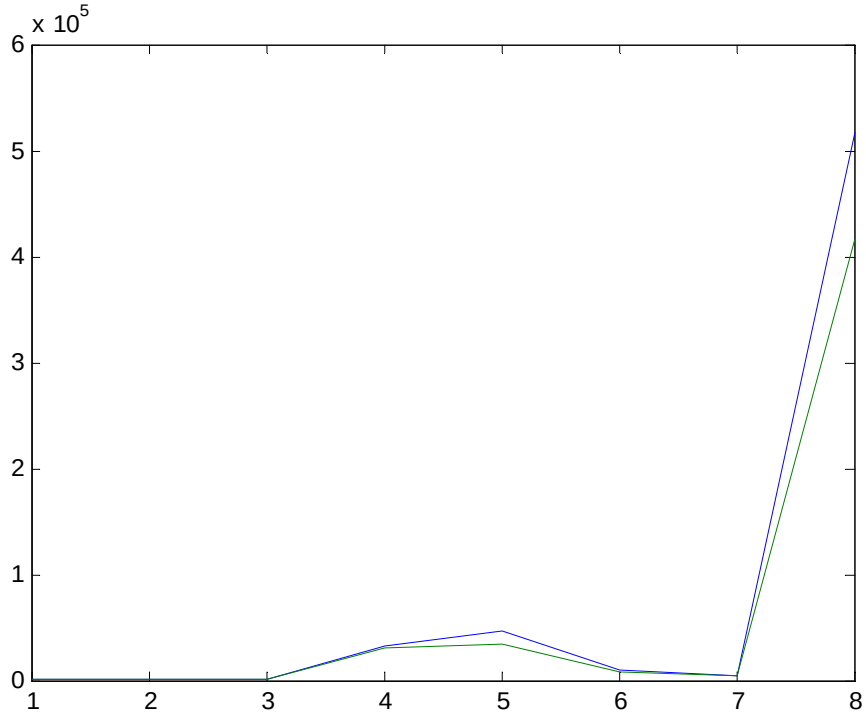

Figure 4. 15 comparaison entre la méthode et le méthode de la descente

Test N° 2 effets du nombre de processus :

Ce test a pour but de découvrir l'influence du nombre de processus en coopération sur le temps d'exécution totale et la qualité de solution.

Conditions : pour ce test on a pris les conditions suivantes :

- le test est fait sur le problème a280 de 280 villes
- nombre des processeurs égal au nombre des processus
- un voisinage total pour chaque processus
- la condition d'arrêt est un nombre d'itérations égales à 200
- l'exécution se fait cinq fois

Les résultats figurent dans le tableau suivant :

		1ere exécution	2eme exécution	3eme exécution	4eme exécution	5eme exécution
2	Mov-temps	14687	38687	15906	21140	13969
	Mill-solu	3742	3536	3801	3591	3957
3	Mov-temps	27484	28141	19781	21719	27578
	Mill-solu	3619	3607	3763	3531	3767
4	Mov-temps	34296	41688	27672	39969	40390
	Mill-solu	3690	3455	3621	3711	3718
5	Mov-temps	48203	39250	40391	45438	105469
	Mill-solu	3736	3485	3492	3787	3616
6	Mov-temps	52766	47907	46734	58812	46469
	Mill-solu	3569	3599	3418	3431	3440
7	Mov-temps	52078	54593	51750	52297	50687
	Mill-solu	3341	3475	3391	3522	3812
8	Mov-temps	61391	51266	71063	67750	65141
	Mill-solu	3637	3463	3559	3365	3318

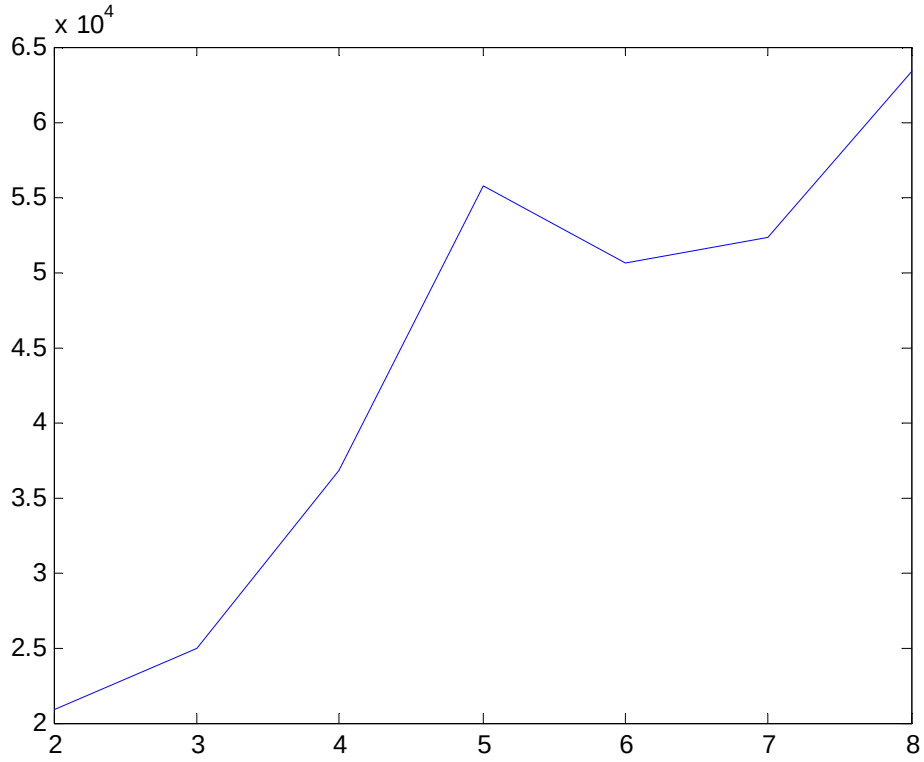

Figure 4. 16 variation de temps d'exécution par rapport le nombre de processus

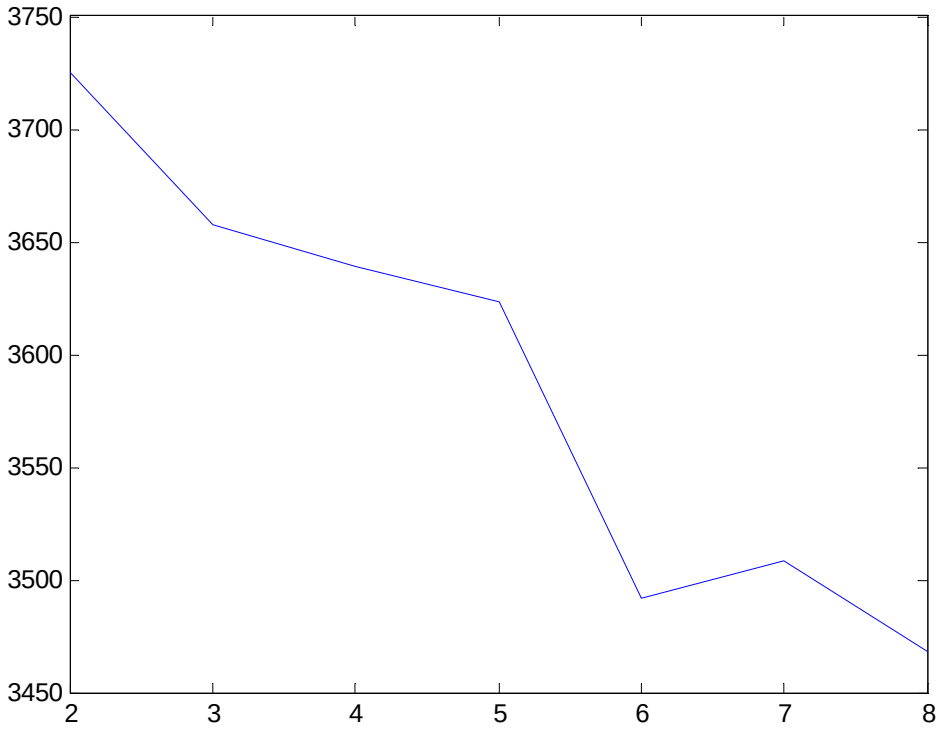

Figure 4. 17 variation de la qualité des solutions par rapport le nombre des processus

Test N°3 : augmentation du nombre de processeurs :

Ce test montre la scalabilité de l'approche c.-à-d l'effet de l'augmentation du nombre de processeurs en travail sur le temps d'exécution, on a les conditions suivantes :

- le test est fait sur le problème (a280) de 280 villes
- un voisinage total pour chaque processus
- la condition d'arrêt est un nombre d'itérations égal à 200

Les résultats figurent dans le tableau suivant :

problème	Nombre de processus	Nombre de processeurs	Max Temps d'exécution (ms)
A280	12	4	276360
		5	259875
		6	184406
		7	178969
		8	186078

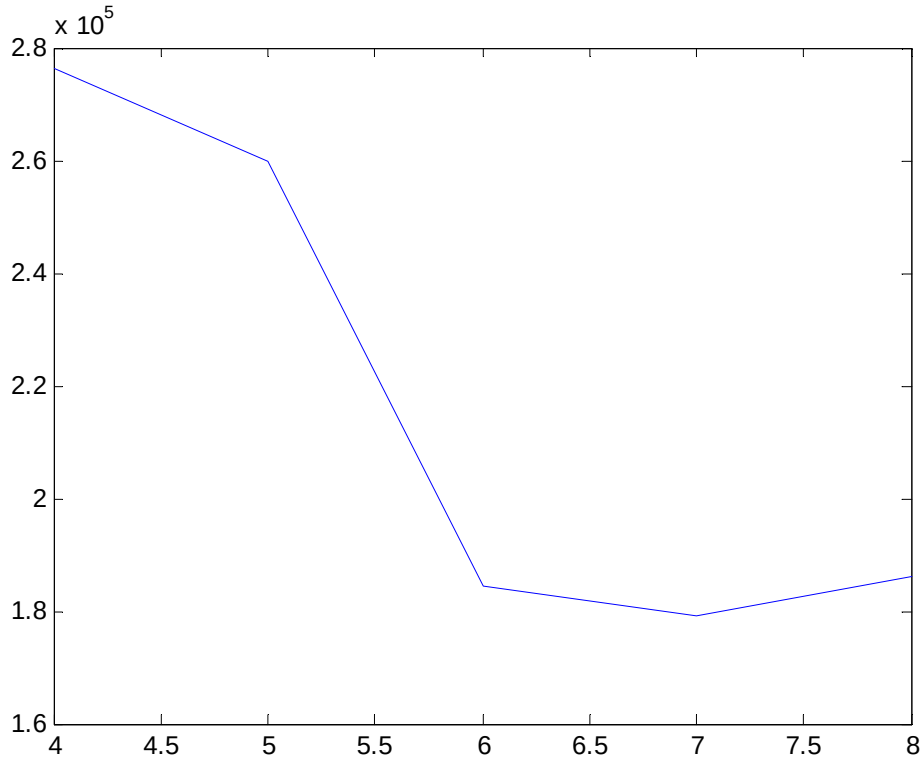

Figure 4. 18 variation de temps d'exécution suivant la variation de nombre des processeurs

7.8 Discussions

1. les résultats du 1er test montrent que la méthode proposée donne des solutions meilleures que la méthode de base (la descente).
2. le 2 eme test montre que l'augmentation du nombre des processus augmente aussi le temps d'exécution (augmentation du temps de communication inter processus) mais elle donne des solutions de qualité de plus (une exploration plus fine de l'espace de recherche).
3. le test N° 3 montre que pour un nombre donné de processus, l'augmentation du nombre des processeurs en exécution diminue en général le temps d'exécution, mais le facteur le plus essentiel est le rapport

$$\frac{nombredeprosussus}{nombredeprocesseur}$$

8 Conclusion

Dans ce chapitre on a proposé une méthode coopérative, parallèle et hybride **souhaitée** d'être adaptée aux caractéristiques des grilles. Cette méthode représente une hybridation entre un modèle parallèle de l'optimisation par essaim de particules dite **le** modèle cellulaire et la méthode de recherche locale itérée (ILS). La méthode a pour buts de minimiser la fréquence de la communication et la taille des informations à communiquer et de guider la recherche suivant un mécanisme supérieur fourni par l'OEP.

Une implémentation était réalisée sur le problème du voyageur de commerce, respectant le mode de déploiement pair à pair ; utilisant le langage java et le passage de message comme un modèle de programmation parallèle. Le résultat est une application MPJ développée avec la plateforme P2PMPI qui repose sur JXTA. Les tests ont été faits sur des instances euclidiennes 2D de TSPLIB et ils montrent des améliorations par apport aux méthodes de base, et une scalabilité suivant le nombre des processeurs participants.

Conclusion et perspectives

Dans ce travail, on a essayé de faire une étude sur les stratégies du parallélisme et d'hybridation des métaheuristiques afin d'augmenter l'efficacité de celle-ci. Ensuite on a proposé une méthode résultant de l'hybridation entre le modèle cellulaire de l'optimisation par essaim de particules et la méthode de recherche locale ILS. Cette hybridation a pour but diaboutir une méthode capable d'être exécutée d'une manière efficace dans un environnement grille de calcul.

Les tests sont faits sur une plateforme matérielle qui ne représente pas une véritable grille à cause de la distance entre ses ressources, mais on peut dire que la méthode proposée est une méthode prometteuse pour plusieurs raisons. Si on fait une comparaison entre cette méthode et ses composants on trouve qu'elle offre une intensification de recherche plus que celle fournie par le modèle parallèle de l'optimisation par essaim de particules ; elle offre aussi un mécanisme de perturbation qui permet à la méthode de recherche itérée d'explorer l'espace de recherche pas à pas, ce qui a pour résultat une exploration de l'espace des solutions par des zones plus ou moins adjacentes ; elle permet d'exécuter les instructions sur des données locales dans la plus part du temps, par conséquent le degré de synchronisation est minimisé. Les solutions fournies par cette méthode sont généralement de qualité meilleure.

Cependant, la méthode reste ouverte pour plus d'analyse et d'amélioration ; on parle de : l'adaptation des paramètres de la méthode telle que l'équation de mise à jour du pas de la perturbation, l'utilisation des autres langages de programmation tel que C++, faire tester la méthode sur une véritable grille pour un meilleur jugement, l'utilisation des autres plateformes telles que JACE et P2PJACE pour implémenter et tester une version complètement asynchrone.

En fin l'approche proposée peut être étendue à d'autres méthodes telles que la carte auto-organisatrice (la carte de Kohonen).

Bibliographie

[ABGL01] K. Anstreicher, N. Brixius, J-P. Goux, J. Linderoth. **Solving Large Quadratic Assignment Problems on Computational Grids.** To appear in Mathematical Programming, available at http://www.optimizationonline.org /DB_HTML/ 2000 /10 /233. html. 2001.

[ACE+02] M.G. Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo, B. Paechter ; M. Preuss, and M. Schoeanauer. **A framework for distributed evolutionary algoritmes.** In Procedings of PPSN VII, september 2002.

[Dur04] N. Durand. **Algorithmes génétiques et autres outils d'optimisation appliqués à la gestion de trafic aérien.** Thèse Phd. 5 octobre 2004

[ANT02] E. Alba, A. J. Nebro, and J. M. Troya **Heterogeneous Computing and Parallel Genetic Algorithms** Journal of Parallel and Distributed Computing 62, 1362–1385 (2002)

[AT02] E. Alba and M. Tomassini. **Parallelism and Evolutionary Algorithms.** IEEE transactions on evolutionary computation, vol. 6, no. 5, OCTOBER 2002

[AtMG02] E. Alba and MALLBA Group. **MALLBA: A library of skeletons for combinatorial optimisation.** In R.F.B. Monien, editor, Proceedings EuroPar , volume 2400 of LNCS, pages 927-932 , Paderborn,2002 Springer-Verlage.

[AUB+05] Alet´eia P.F. Ara´ujo, Sebasti´an Urrutia, Cristina Boeres, Vinod E.F. Rebello, Celso C. Ribeiro. **Towards Grid Implementations of Metaheuristics for Hard Combinatorial Optimization Problems.** 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'05), 2005

[BEG04] M. Belal and T. El-Ghazawi **PARALLEL MODELS FOR PARTICLE SWARM OPTIMIZERS.** *IJICIS, VOL. 4, NO. 1, January 2004*

[Bou02] L. Bougé. **Si tous les ordinateurs du monde... Calcul scientifique *vraiment très haute performance* sur la grille mondiale.** Département info-télécoms ENS Cachan, Antenne de Ker Lann 25 octobre 2002

[BPSV01] M. Birrttari, L. Paquete, T. Stutzle and K. Varrentrap. **Classification of metaheuristics and Design of Experiments for the analysis of component**. Technical Repport AIDA -01-05. November 2001.

[BR03] C. Blum And A. Roli. **Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison.** ACM Computing Surveys, Vol. 35, No. 3, pp. 268–308,

September 2003.

[BR04] C. Boeres and V. Rebello. **Easygrid: Towards a framework for the automatic grid enabling of legacy mpi applications.** *Concurrency And Computation Practice And Experience*, 17(2):173.190, 2004.

[Can98] E. Cantu-Paz. **A Survey of Parallel Genetic Algorithms**. Calculateurs parallèles, réseaux et systèmes répartis 10, 141-171, Hermès, Paris, 1998

[CGJ+00] B. Carpenter, V. Getov, G. Judd, A. Skjellum and G. Fox. **MPJ: MPI-like Message Passing for Java.** Concurrency:Practice and Experience. 12(11): 1019-1038; 2000

[CMRR02] V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. **Strategies for the Parallel Implemetation of Metaheuristics**. Essays and Surveys in Metaheuristics. C.C. Ribeiro, P. Hansen (Eds.), 263-308, Kluwer Academic Publishers, Norwell, MA, 2002.

[CMT04] S. Cahon N. Melab E.-G. Talbi. **ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics.** Journal of Heuristics, Vol 10 :353-376, May 2004

[CT98] T.G. Crainic and M. Toulouse. **Parallel Metaheuristics**. In T.G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 205–251. Kluwer Academic Publishers, Norwell, MA. 1998

[CT03] T.G. Crainic and M. Toulouse. **"Parallel Strategies for Meta-heuristics"**, State-of-the-Art Handbook in Metaheuristics, F. Glover, G. Kochenberger (Eds.), 475-513, Kluwer Academic Publishers, Norwell, MA, 2003.

[CTA05] C. Cotta, EG. Talbi, and E. Alba. **Parallel hybrid metaheuristics**. In Alba, E., ed.: Parallel Metaheuristics, a New Class of Algorithms. John Wiley (2005) 347–370.

[DFJ54] G. Dantzig, R. Fulkerson, And S. Johnson **solution of a large-scale traveling-salesman problem** *The Rand Corporation, Santa Monica, California* (Received August 9, 1954)

[DS06] J. Dréo et P. Siarry. **Métaheuristiques pour l'optimisation et auto-organisation dans les systèmes biologiques** *Technique et Science Informatiques, Vol. 25, No. 2. (2006), pp. 221-240. - 1 January 2006*

[DS00] M. Dorigo and T. Stutzle. **The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances.** Technical Report IRIDIA-2000.. To appear in *Metaheuristics Handbook*, F. Glover and G. Kochenberger (Eds.), International Series in Operations Research and Management Science, Kluwer, 2001.

[FJMO95] M.L Fredman, D.S. Johnson, L.A. McGeoch, G. Ostheimer. **Data Structures For Traveling Salesmen**. *J. ALGORITHMS 18,* 1995, pp. 432-479.

[FK97] I. Foster et C. Kesselman **Globus: A Metacomputing Infrastructure Toolkit.** Intl. J. of supercomputer application, 11(2) : 115-128,1997

[FK99] I. Foster et C. Kesselman. **The Grid: Blueprint for a New Computing Infrastructure**. Morgan Kaufmann, 1999.

[FKT01] I. Foster, et C. Kesselman et S. Tuecke. **The Anatomy of the Grid: Enabling Scalable Virtual Organization.** The International Journal of High Performance Computing Applications, 15(3), 2001

[FKTY03] K. Fujisaway, M. Kojimaz, A. Takeda and M. Yamashita. **High Performance Grid and Cluster Computing for Some Optimization** Problems Research Report B-396, October 2003. Revised December 2003

[Fos02] I. Foster. **What is the Grid? A Three Point Checklist.** Grid Today, 1(6), july 22 2002

[GL05] W. Glankwamdee, J. T. Linderoth. **MW: A Software Framework for Combinatorial Optimization on Computational Grids** October 20, 2005

[Glo95] F. Glover. **tabu search fundamentals and uses.** supported in part by the National Science and Engineering Council of Canada under Grants 5-83998 and 5-84181; juin 1995

[GLPF01] V. Getov, G. V. Laszewski, M. Philippsen and I. Foster **Multiparadigm Communications in Java for Grid Computing.** *Communications of ACM*, 2001 http://www.globus.org/cog/documentataion/papers/

[HCWV04] R. Hassan, B.Cohamin, O. de Weck and G. Venter. **A COMPARISON OF PARTCLE SWARM OPTIMIZATION AND THE GENETIC ALGORITHME.** Copyright © 2004 by Rania Hassan, published by AIAA.

[HGH99] J.-K Hao, P. Galinier and M. Habib. **Méthaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes.** Revue d'Intelligence Artificielle Vol : No. 1999

[KE95] J. Kennedy and R. Eberhart. **Particle Swarm Optimization**. In *Proceedings of IEEE International Conference on Neural Network,* volume IV., pages 1942–1948, 1995

[KGV83] S. Kirkpatrick, C. D. Gelatt, Jr. and M. P. Vecchi. **Optimization by Simulated Annealing.** *SCIENCE* **;** 13 May 1983, Volume 220, Number 4598

[Li06] Y. Li. **A Bio-inspired Adaptative Job Scheduling Mechanisme on a Computational Grid.** 1IJCSNS International Journal of Computer Science and Network Security, VOL .6 No. 3B, March 2006.

[LMS02] H. R. Lourenço, O. Martin, and T. Stützle. **Iterated Local Search**. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321-353, Kluwer Academic Publishers, Norwell, MA, 2002.

[LW01] J. Linderoth, J. S. Wright. **Computational Grids for Stochastic Programming Optimization.** Technical Report 01-01 October 10, 2001.

[Mal05] N. Meleb. **contributions à la résolution des problèmes d'optimisation combinatoire sur grilles de calcule.** Thèse Phd soutenue le 28 Novembre 2005.

[Mel99] M. Melanie. **An Introduction to Genetic Algorithms.** A Bradford Book The MIT Press Cambridge, Massachusetts. London, England Fifth printing, 1999.

[Rai06] G. R. Raidl A **Unified View on Hybrid Metaheuristics**. Third International workshop, HM 2006, Gran Canaria, Spain, October 13-14, 2006.

[Rat04] C. Rattanapoka **P2P-MPI: A Peer-to-Peer Framework For Robust Execution of Message Passing Prallel Programs on Grids** Mémoire de DEA Univercité Louis-Pasteur de Strasbourg 2003/2004.

[RPE01] M. G. C. Resende, P. M. Pardalos and S. D. Eksioglu. **Parallel Metaheuristics for Combinatorial Optimization.** Advanced Algorithmic Techniques of Parallel Computation with Applications, R. Correa et al. (Eds.), 179-206, Kluwer, 2001.

[Sch60] A. Schrijver **On the history of combinatorial optimization (till 1960)**

[SDPT03] P. Siarry, J. Dréo, A. Pétrowski and É. Taillard. **métaheuristiques pour l'optimisation difficile** Éditions Eyrolles 2003 ISBN : 2-212-11368-4

[Set05] M. Settles **An Introduction to Particle Swarm Optimization** Department of Computer Science, University of Idaho,Moscow, Idaho U.S.A 83844 November 7, 2005

[SHMD05] H. Shimosaka, T. Hiroyasu, M. Miki, and J. Dongarra. **Optimization Problem Solving System using GridRPC.** IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 1, NO. 1, OCTOBER 2005

[SNM+02] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, H. Casanova. **GridRPC: A Remote Procedure Call API for Grid Computing.** Advanced Programming Models Research Group, Tokyo Institute of Technology.
http://www.eece.unm.edu/˜apm/docs/APM GridRPC 0702.pdf. July 2002

[PT97] P. Preux and E-G. Talbi **Towards hybrid evolutionary algorithms** Preprint submitted to Elsevier Preprint, 16 October 1997.

[SunMS05] **JXTA v2.3.x: Java™ Programmer's Guide** Sun Microsystems, Apr 7, 2005

[Tal99] E-G. Talbi. **A Taxonomy of hybrid metaheuristics.** Journal of combinatorial optimization 1999.

[TTM03] D. Thain, T. Tannenbaum, and Miron Livny**. Condor and the Grid.** *Grid Computing – Making the Global Infrastructure a Reality*. Edited by F. Berman, A. Hey and G. Fox 2003 John Wiley & Sons, Ltd ISBN: 0-470-85319-0

[VY01] T. Vallée et M. Yıldızo-glu. **Présentation des algorithmes génétiques et de leurs applications en économie.** 7 septembre 2001, v. 1.2

[WA04] S. Wagner, M. Affenzeller. **Heuristiclab Grid– a flexible and extensible environment for parallel heuristic optimization.** *Proceedings of the 15th International Conference on Systems Science*.2004

[Wid01] M. Widmer **les metaheuristiques : des outils performants pour les roblemes industriels**. *3e Conférence Francophone de MOdélisation et SIMulation "Conception, Analyse et Gestion des Systèmes Industriels" MOSIM'01 – du 25 au 27 avril 2001 - Troyes (France).*

[Wri 00] J. S. Wright. **Solving Optimization Problems on Computational Grids** November 17, 2000.

[Zam06] L. Zambito. **The Traveling Salesman Problem: A Comprehensive Survey.** Submitted as a project for CSE 4080, Fall 2006