Chapitre III

recherche locale a mémoire adaptative parallèle pour le problème de voyageur de commerce

Chapitre III : recherche locale a mémoire adaptative parallèle pour le problème de voyageur de commerce

3.1 Introduction

Dans ce chapitre on va parler sur un problème d'optimisation combinatoire, c'est le problème de voyageur de commerce, c'est un problème typique de la classe NP-complet, il nous aide comme une pierre de test de la méthode implémenté par nous même, cette méthode base sur la méthode de recherche locale à mémoire adaptativela méthode de base sera représentée en second lieu, en suite on va introduire la méthode parallèle proposée, puis les détails de l'implémentation, tests et résultats et finalement les discutions et la conclusion.

3.2 Le problème de voyageur de commerce

Le problème du voyageur commerce (PVC) est l'un des problèmes les plus connus dans le domaine de l'optimisation combinatoire car en plus de sa simplicité il est typiquement NP-complet, donc il est largement utilise pour juger l'efficacité des méthodes proposées pour résoudre un tel type des problèmes, il s'agit de trouver le chemin le plus court qui relie n villes données, chaque ville ne doit être visitée qu'une et une seule fois, et revenir à la ville de départ.

Le PVC peut être appliqué à résoudre un grand nombre des problèmes pratiques dans notre vie quotidienne.

3.2.1 Définition

Le problème du voyageur de commerce", est le suivant : un représentant de commerce ayant *n* villes à visiter souhaite établir une tournée qui lui permet de passer exactement une fois par chaque ville et de revenir à son point de départ pour un moindre coût, c'est-à-dire en parcourant la plus courte distance possible.

Si on utilise les notions des graphes. Soit $G = (X,U)$, un graphe dans lequel l'ensemble X des sommets représente les villes à visiter, ainsi que la ville de départ de la tournée, et U, l'ensemble des arcs de G, représente les parcours possibles entre les villes. A tout arc $(i,j) \in U$, on associe la distance de parcours d_{ij} de la ville i à la ville j. La longueur d'un chemin dans G est la somme des distances associées aux arcs de ce chemin. Le PVC se ramène alors à la recherche d'un circuit hamiltonien (un chemin fermé passant exactement une fois par chacun des sommets du graphe G) et de longueur minimale[HW85].

Plus formellement, le problème du voyageur de commerce peut s'écrire comme un programme linéaire. Ci -dessous, *V* est l'ensemble des *n* sommets du graphe. *X* désigne l'arc (*i, j*) et vaut 1 s'il fait partie de la solution, 0 si non .d_{ij} représente le poids l'arc *(i,j).*

$$\begin{cases} \min z = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\, x_{ij} \\ \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall i \in V \qquad (1) \\ \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall j \in V \qquad (2) \\ \sum_{(i,j)\in p^2} x_{ij} \leq |p| - 1 \;\forall\, p \subset V, p \neq \emptyset \qquad (3) \end{cases}$$

$$x_{ij} \in \{0;1\} \;\forall (i,j) \in V^2 (4)$$

(1) Chaque sommet *i* ne peut accepter qu'un arc sortant vers un autre sommet *j*.

(2) Chaque sommet *j* ne peut accepter qu'un arc entrant d'un autre sommet *i*.

(3) Dans tout sous ensemble *P* de sommets, sélectionner |*P*| arcs permettrait de former des circuits hamiltoniens. On interdit cela.

3.2.2 Historique

Des problèmes mathématiques liés au problème de voyager de commerce (en anglais, TSP : Traveling Salesman Problem) ont été étudiés dans les 1800's par le mathématicien Irlandai Sir William Rowan Hamilton et par le mathématicien Britannique Thomas PenyngtonKirkman. Hamilton a créé le jeu Icosian en 1857qui nécessite un lecteur de terminer une tournée en utilisant seulement connecteurs spécifiés par 20 points [CT03].

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 capitale état 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 Allemagne 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 situés et ils ont estimé tout le temps de calcul étaient de 22,6 ans, mesurés à un processeur d'alpha de Compaq EV6(21264) fonctionnant à 500MH[FKTY03].

En mai 2004, le problème de représentant de commerce de visiter chacune des 24.978 villes en Suède a été résolu: une excursion de longueur approximativement 72.500 kilomètres a été trouvée et on l'a montrée qu'aucune excursion plus courte n'existe.

En mars 2005, le problème de représentant de commerce de visiter chacun des 33.810 points dans une carte a été résolu en utilisant *CONCORDE* une excursion des unités de la longueur 66.048.945 a été trouvée et on l'a montrée qu'aucune excursion plus courte n'existe. Le calcul a pris approximativement 15,7 années d'unité centrale de traitement (Cook et al. 2006).

3.2.3 Complexité du problème

Ce problème appartient à la classe NP-difficile et pour montrer la difficulté de ce problème supposant qu'on a 50 villes ; donc ils existent 49! Solutions (tourne) possibles, i.e. $6{,}08.10^{62}$ tournées possible. 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. Note : $4{,}766.10^{43}$ est l'équivalent de $1{,}51*10^{34}$ siècle.

D'une manière formelle pour démontrer que PVC est NP-complet nous emploierons la méthode populaire de la réduction. Nous prendrons un problème, l'appelons H, on s'est déjà

avéré qu'il est NP-complet. Puis, nous prouverons que vous pouvez résoudre H en le ramenant au PVC. Une fois que nous avons montré ceci, nous pouvons dire que si PVC est soluble dans un temps polynôme, alors nous pouvons résoudre H dans un temps polynôme. À quel point nous atteignons une contradiction nous concluons alors que PVC ne peut pas être résolu dans un temps polynôme parce qu'on l'a déjà montré que H ne peut pas être résolu dans le temps polynôme. D'ailleurs, PVC doit être NP-complet parce que H est NP-complet[Zam06].

En montrant que PVC est NP-complet, nous choisirons H pour être le problème populaire de cycle hamiltonien, qui est connu pour être NP-complet. À un regard étroit, il est évident que le problème de cycle hamiltonien soit un cas spécial de PVC. Tous simplement en modifiant le graphe d'entrée selon l'algorithme de PVC en plaçant les coûts de tous les bords existants pour être une certaine constante finie fixe. Puis, si n'importe quelle excursion est trouvée, cette excursion est un cycle hamiltonien. Ainsi, le PVC doit être NP-complet parce qu'on l'a montré que le problème de cycle hamiltonien est NP-complet[Zam06].

3.2.4 Problèmes réels se modélisent 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 monnaie des cabinés téléphoniques publics.
- La chaîne d'ADN.
- Conception des réseaux à fibres optiques.

3.2.5 Problèmes symétrique et asymétrique

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 xet y n'est pas forcément égale à celle entre y et x.

3.3 La recherche à mémoire adaptative

La recherche à mémoire adaptative est une méthode proposée par Rochat et Taillard en 1995. Est une extension de la Recherche Tabou qui permet de réaliser automatiquement une diversification et une intensification de la recherche[TGGP97].

Cette méthode facilite les modifications des données du problème qui peuvent être intégrées rapidement et au même temps les informations contenues dans la mémoire restent pertinentes.

Elle repose sur trois étapes :

- **Information**

Correspond à la représentation des informations. On le modélise sous la forme d'une mémoire. Le but étant de mémoriser tout ou une partie des solutions générées par la recherche.

- **Intensification**

Correspond à l'exploitation des informations. On approfondit la recherche au niveau local en tentant d'améliorer la pertinence des informations disponibles.

Sa stratégie consiste à :

Les meilleures solutions rencontrées sont mémorisées. Les propriétés communes en sont dégagées. On oriente la recherche vers les régions ainsi définies.

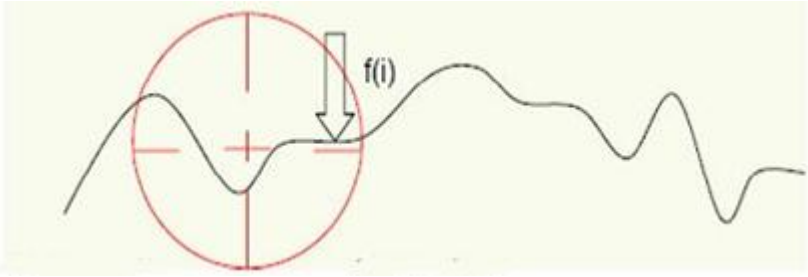

Figure 15: Intensification.

❖ **Diversification**

Correspond à la recherche de nouvelles informations. On approfondit la recherche sur tout l'espace de recherche (niveau global) pour accroître la quantité de ces informations, en explorant de nouvelles régions (niveau global).

Sa stratégie consiste à :

On mémorise les solutions les plus visitées On impose un système de pénalités.

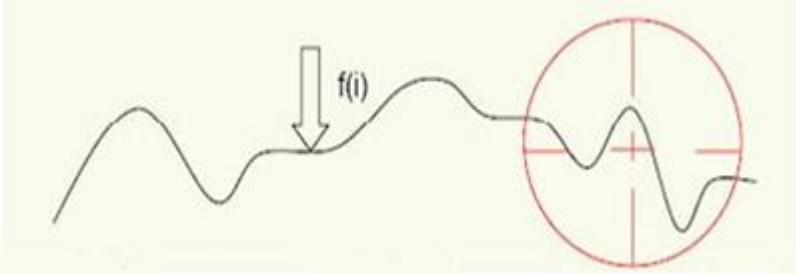

Figure 16: Diversification.

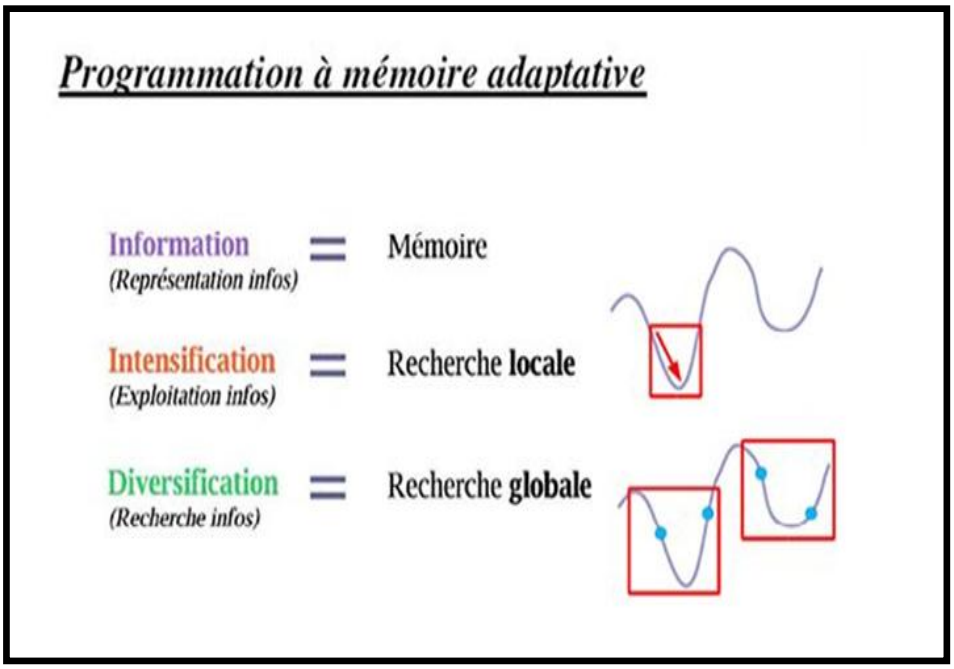

Figure17 : les étapes de méthode de recherche à mémoire adaptative.

3.3.1 Le principe de fonctionnement de la méthode a mémoire adaptative

Cette méthode fonctionne avec une mémoire centrale chargée de stocker les composantes des meilleures solutions rencontrées. Ces composantes sont combinées

afin de créer de nouvelles solutions. Si la combinaison ne produit pas une solution admissible, un processus de réparation est mis en œuvre.

Un algorithme de Recherche Locale est ensuite appliqué et les composantes de la solution ainsi obtenues sont considérées pour éventuellement faire partie dela mémoire centrale.

Au début de la recherche, la mémoire centrale contient des composantes provenant de solutions très diverses et le processus de combinaison aura donc tendance à créer une diversité de nouvelles solutions. Plus la recherche avance et plus la mémoire centrale aura tendance à ne mémoriser que les composantes d'un souensemble très restreint de solutions. La recherche devient donc petit à petit un processus d'intensification[TGGP97].

- **M← ∅.**
- **2) Répéter, tant qu'un critère d'arrêt n'est pas satisfait:**

  2a) M'← tous les arcs individus , s ← ∅, s '← ∅.

  2b) Répéter, tant que M'≠ ∅.

  2b1)Choisir une arc T∈M', aléatoirement

  2b2)Poser s'← s'∪{T}.

2b3)Pour toute arc T'∈M' telle que T∩ T' ≠∅, poser M'←M'\{T'}.

2c) Compléter la solution partielle s' et l'améliorer à l'aide d'une recherche locale pour obtenir une solution s.

2d) Pour toute arc T de s, poser : M← M∪{T}.

- **3) Trouver une solution s aussi bonne que possible en considérant les arcs contenues dans M.**

Programme à mémoire adaptative pour problème de voyageur de commerce

3.3.2 Recherche à mémoire adaptative parallèle

Bien que nous n'ayons pas formulé l'algorithme général de la recherche à mémoire adaptative sous une forme concurrente, il est cependant facile de le paralléliser. En effet, il suffit pour cela d'exécuter la boucle principale de manière indépendante par plusieurs processus qui ne communiquent entre eux que par l'intermédiaire de la mémoire. Cela signifie qu'une machine idéale pour l'exécution concurrente d'un programme à mémoire adaptative doit posséder une mémoire partagée.

Le schéma d'une telle parallélisassions est présenté en figure 18. Il est généralement très efficace car le processus de lecture de la mémoire et de sa mise à jour est beaucoup plus rapide que les autres, en particulier celui de la recherche locale qui consomme typiquement presque l'intégralité des ressources de calcul.

On suppose cependant que chaque processus d'optimisation possède une copie locale des données du problème. Nous pouvons illustrer une implantation typique d'un programme à mémoire adaptative parallèle sur le problème de l'affectation quadratique. La solution d'un tel problème, pour un exemple de taille *n*, peut être représentée par une permutation de *n* éléments. Les données du problème sont constituées de deux matrices carrées de dimension $n \times n$. Dans nos récentes implantations , de même que dans l'implantation, la mémoire adaptative est elle aussi constituée d'une matrice carrée de dimension $n \times n$(ou éventuellement de $O(n)$ permutations de *n* éléments, ce qui correspond à un encombrement similaire). Cela signifie que la quantité d'informations transférée pour consulter la mémoire ou la mettre à jour est au pire en $O(n^2)$, mais souvent en $O(n)$, le temps nécessaire pour calculer les nouvelles valeurs de la mémoire étant du même ordre de complexité.

Par contre, le processus de recherche locale est d'une complexité supérieure :

$O(n^3)$. Ainsi, des problèmes de congestion dans les accès à la mémoire partagée pourront apparaître pour un nombre de processus d'optimisation de l'ordre de $O(n)$, voire $O(n^2)$.

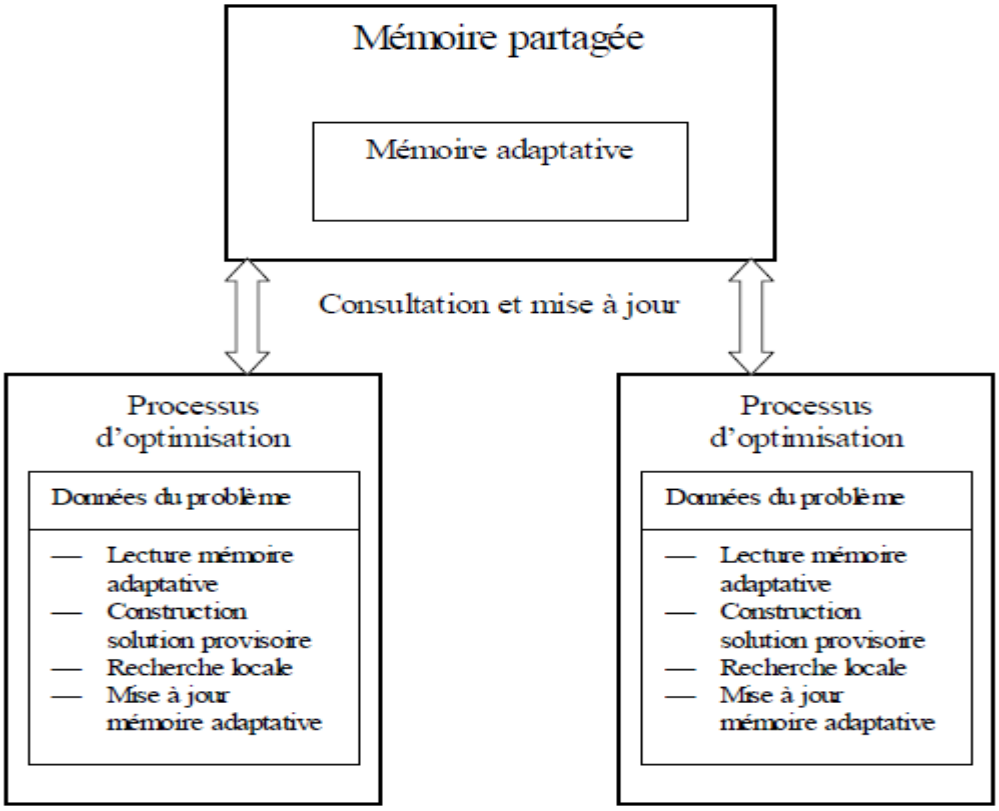

Figure 18 : Parallélisassions avec une mémoire commune

3.4 Recherche parallèle a mémoire adaptative pour le problème de voyageur de commerce

La variante parallèle proposée de la méthode de recherche a mémoire adaptative respecte la description du parallélisme type 3, cad une recherche parallèle basée sur un modèle multi-départs homogène et coopératif.

La méthode consiste à lancer plusieurs processus de recherche identiques, chaque processus exécute une recherche locale a mémoire adaptative, cad un processus de recherche locale plus une mémoire, cette mémoire est mise à jour à la fin de chaque itération, et elle est utilisée pour réinitialiser la solution de la prochaine itération. La coopération est assurée par le moyend'une autre mémoire située au niveau de chaque processus de recherche, leur rôle est de garder une trace de la recherche des autres processus, les informations contenues dans cette mémoire seront aussi utilisées pour la prochaine réinitialisation.

Le tôt d'utilisation des informations de chaque mémoire représente le degré de la coopération ou de l indépendance de la recherche de l ensemble.

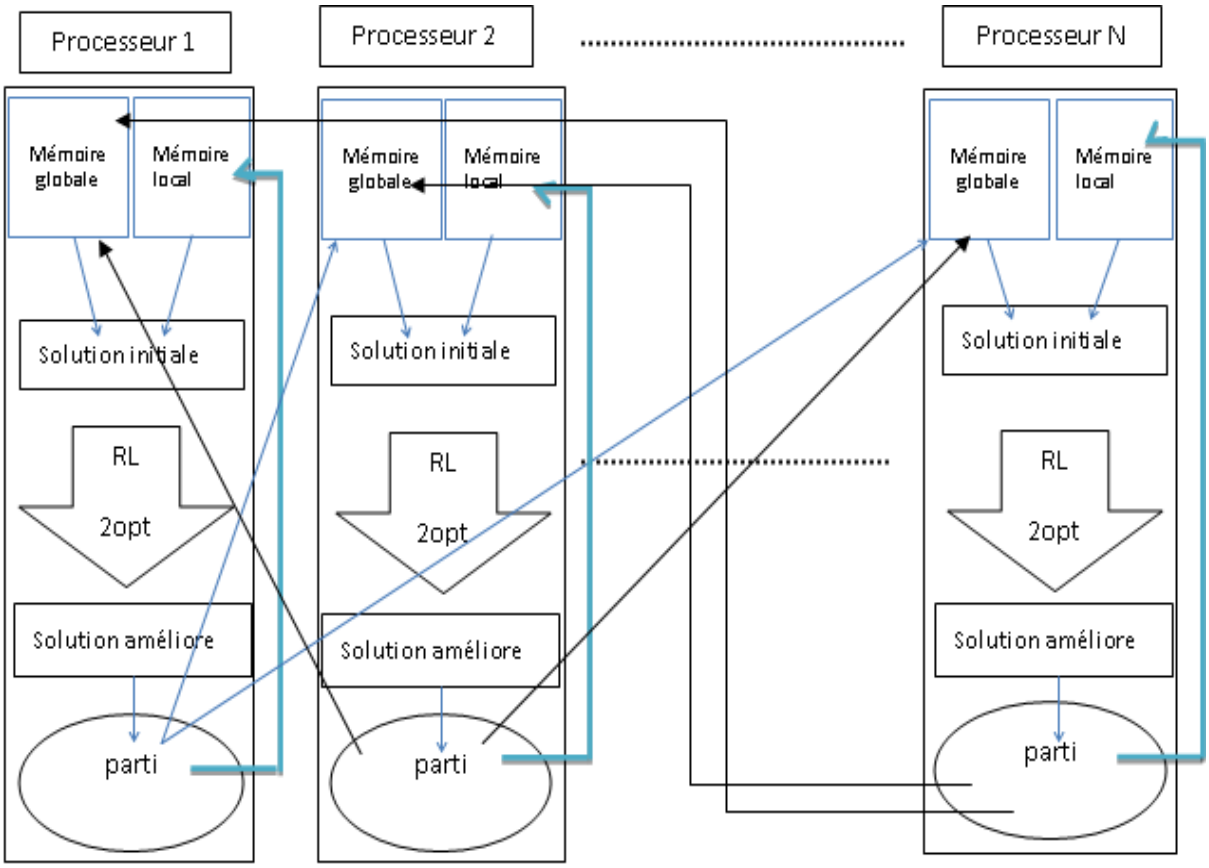

Figure 18: schéma générale qui représente la méthode proposer.

3.4.1.1 Implémentation

La phase implémentation se caractérise généralement par un ensemble des choix faits par le concepteur ou le programmeur afin de répondre à un ensemble des critères lies à la méthode proposée, dans notre cas on va spécifier le modèle de programmation parallèle et le langage de programmation ;par conséquence on va designer l'outils ou la plateforme de programmation et d'exécution adapte au modèle de la programmation et au langage choisis.

Autres choix liés à la méthode et qui doivent être justifiés qui est la structure de données supportant une solution, supportant la mémoire locale et la mémoire globale ; ainsi que le choix de type de voisinage dune solution.

3.4.1.2 Le choix de modèle de programmation parallèle

On a utilisé le passage de message comme un modèle de programmation parallèle parce qu'il soutient le modèle *Simple Programme - Multiple Données* (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].

Une autre raison est que le seul moyen disponible d'avoir une plateforme parallèle et d'interconnecter un ensemble d'ordinateur via un réseau, afin d'avoir un système parallèle de type mémoire distribuée.

3.4.1.3 Le choix du langage de programmation

Il existe plusieurs langages de programmation supportant le passage de message, mais pour des raisons de portabilité la méthode a été implémentée avec java, la portabilité de java lui donne un bon avantage pour être un langage d'implémentation pour les applications parallèles sur les systèmes distribues.

Le langage Java offre plusieurs mécanismes qui permettent au parallélisme d'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.

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]. il existe plusieurs implémentations java du modèle de passage de message et on s'intéresse au mpi java.

Concernant l'efficacité du java les résultats de beaucoup de recherches récents ont montré que l'efficacité des applications implémentées avec java peut venir très près de cela de C ou de Fortran.

3.4.1.4 Plateforme

Le système proposé consiste à implémenter la méthode sur un ensemble des ordinateurs pas forcement homogènes. Les trois choix précédemment faits (le modèle passage de message, le langage JAVA et l'implémentation MPI java) ont conduit à choisir la plateforme P2PMPI pour implémenter et tester la méthode proposée.

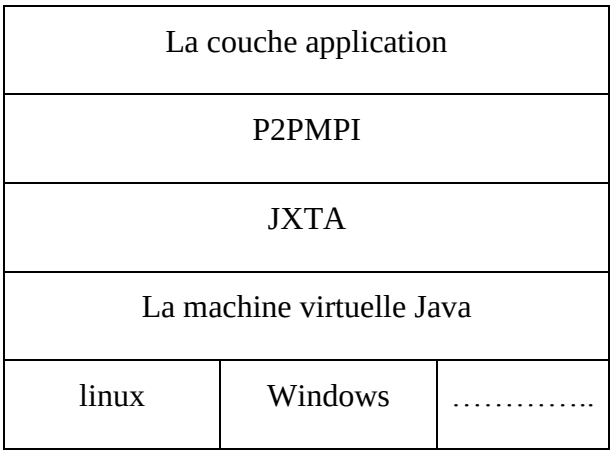

Figure 19 : architecture de la plateforme de l'implémentation.

P2PMPI

P2P_MPI [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.

JXTA

JXTA 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 se collaborer avec les autre. 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 autres indépendamment de leurs localisations physiques et ses infrastructures centralisées.

L'ensemble des protocoles JXTA est indépendant de tout langage de programmation, de tout plateforme réseau et tout système d'exploitation, car ils sont Basés sur des standards tel 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.

3.4.1.5 Les structures des données

Pour la structure de données et pour des raisons de simplicité on a 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. Pour la mémoire locale et la mémoire globale dans chaque processus on a choisi une structure de taille variée nommée ListArray.

L'utilisation du tableau comme structure pour les chemins est justifiée 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(√n) de temps par inversion, ces arbres feront le travail pour des tailles de problème 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.

3.4.1.6 La méthode de recherche locale

Pour cette application on a 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 donnée (dans notre cas le nombre de villes) l'ensemble de voisinage 2-opt contient N*(N-3)/2 solution voisine.

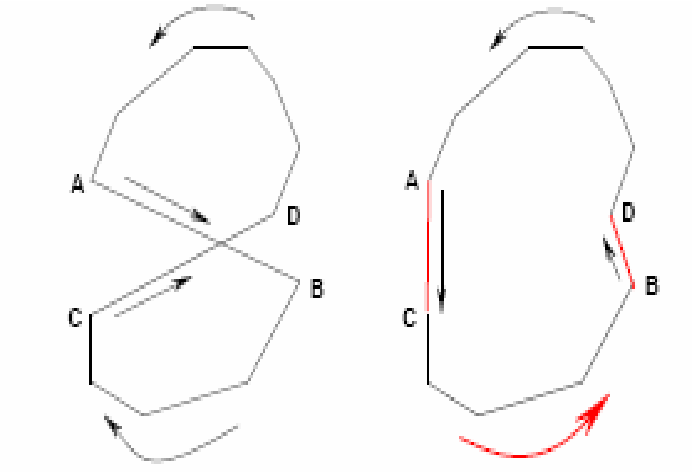

Figure 20 : permutation 2-Opt.

3.5 Phase expérimentale

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

L'exécution sur une plateforme logicielle :

- Windows 7.
- jre-6u1-windows-i586-p-s.
- jxta-lib-2.4.1.
- p2pmpi-0.21.0.

Sur une plateforme matérielle de 3 postes connectés de la configuration suivante:

Poste	Processeur	RAM
Poste1	Intel pentium Dual-CoreT4500 2.30 GH	2 Go
Poste2	Intel(R)pentium(R)CPU960 @2.20GHz 2.20GHz	2 Go
Poste3	Intel pentium Dual-Core T3300 2.00 GH	2 Go

3.5.1.1 Tests et résultats

Test N° 1amélioration 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 3000 itérations (500 itérations pour 1002 villes).

Nbr It	nb ville	solution ini	Mémoire adptative	temps	déscante
3000	51	1644	437	20	658
3000	76	2650	585	51	681
3000	101	3301	670	85	824
3000	150	263253	32546	212	40278
3000	200	337726	37572	401	45309
500	1002	6602834	409334	2h,41m,03s	478957

Test N° 2 pour calculer l'accéliration donnée par le parallelisme :

Ce test a pour but de découvrir l'influence de l'augmantation de 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 les problème 51 76 101 villes.
- nombre des processeurs égal au nombre des processus.
- un voisinage total pour chaque processus.
- nombre d'itérations parallèles égales 1/3 nombre itération séquentiel.
- l'exécution se fait quatre fois.

Les résultats se représentent dans les tableaux suivants

51 villes 03 processeurs

300	T(s)	5	6	6	6	5.75
	Q	447	448	440	444	444.75
600	T(s)	12	11	11	10	11.00
	Q	439	431	441	444	438.75
900	T(s)	16	16	17	18	16.75
	Q	439	440	440	440	439.75
1200	T(s)	22	21	23	22	22.00
	Q	442	436	445	435	439.5

51 villes 1 processeur

900	T	2	3	2	3
	Q	445	446	449	444
1800	T	4	6	6	4
	Q	439	438	434	436
2700	T	7	8	9	9
	Q	440	442	444	441
3600	T	11	8	12	10
	Q	437	431	438	440

76 ville 3 processeurs

300	T	7	8	6	6
	Q	595	572	598	592
600	T	12	12	12	12
	Q	591	589	590	588
900	T	27	18	22	19
	Q	584	587	590	588
1200	T	26	24	26	24
	Q	581	586	586	577

76 villes 1 processeurs

900	T	7	9	9	8
	Q	600	580	591	579
1800	T	16	18	16	16
	Q	580	585	587	583
2700	T	22	25	20	25
	Q	585	585	596	587
3600	T	31	34	33	33
	Q	580	588	573	571

101 ville 3 processeurs

300	T	10	13	10	10
	Q	689	708	711	700
600	T	24	19	21	20
	Q	691	693	680	693
900	T	28	29	30	32
	Q	696	691	686	677
1200	T	38	38	38	38
	Q	671	684	685	682

101 ville 1 processeurs

Nbr It		Ex1	Ex2	Ex3	Ex4
900	T (s)	20	24	23	23
	Q	708	712	709	703
1800	T (s)	49	48	47	52
	Q	682	694	704	799
2700	T (s)	68	68	71	81
	Q	695	692	685	693
3600	T (s)	108	97	98	91
	Q	695	700	706	701

51 ville 3 pro

300	T	5	6	6	6	5.75
	Q	447	448	440	444	444.75
600	T	12	11	11	10	11.00
	Q	439	431	441	444	438.75
900	T	16	16	17	18	16.75
	Q	439	440	440	440	439.75
1200	T	22	21	23	22	22.00
	Q	442	436	445	435	439.5

51 villes 1pro

900	T	2	3	2	3	
	Q	4	446	4	4	
1800	T	4	6	6	4	5
	Q	4	438	4	4	4
2700	T	7	8	9	9	
	Q	4	442	4	4	4
3600	T	1	8	1	1	
	Q	4	431	4	4	

76 ville 3 pro

300	T	7	8	6	6	6.75
	Q	595	572	598	592	589.25
600	T	12	12	12	12	12
	Q	591	589	590	588	589.5
900	T	27	18	22	19	21.5
	Q	584	587	590	588	587.25
1200	T	26	24	26	24	25
	Q	581	586	586	577	582.5

76 ville 3 pro

900	T	7	9	9	8	8.25
	Q	600	580	591	579	587.5
1800	T	16	18	16	16	16.5
	Q	580	585	587	583	583.75
2700	T	22	25	20	25	23
	Q	585	585	596	587	588.25
3600	T	31	34	33	33	32.75
	Q	580	588	573	571	578

101 ville 3 pro

300	T	10	13	10	10	10.75
	Q	689	708	711	700	702
600	T	24	19	21	20	21
	Q	691	693	680	693	689.25
900	T	28	29	30	32	29.75
	Q	696	691	686	677	687.5
1200	T	38	38	38	38	38
	Q	671	684	685	682	680.5

101 Villes 1 pro

900	T	20	24	23	23	22.5
	Q	108	712	709	703	558
1800	T	49	48	47	52	49
	Q	682	694	704	799	719.75
2700	T	68	68	71	81	72
	Q	695	692	685	693	691.25
3600	T	108	97	98	91	98.5
	Q	695	700	706	701	700.5

Calcul d'accélération et d'efficacité

Considérons un algorithme qui s'exécute sur un ordinateur parallèle comportant p processeurs (identiques) dans un temps Tp, et soit Ts son temps d'exécution séquentiel.

L'accélération est définie par le rapport: Sp = Ts/Tp

Généralement on a : $1 \leq Sp \leq p$

L'efficacité d'un algorithme parallèle est le rapport Ep = Sp/p

51 villes				
nombre d'itération	temps pour 3 pros	temps pour 01 pro	accélération	efficacité
900	5.75	2.5	0.43	0.14
1800	11.00	5	0.45	0.15
2700	16.75	8.25	0.49	0.16
3600	22.00	10.25	0.47	0.16

76 villes				
nombre d'itération	temps pour 3 pros	temps pour 01 pro	accélération	efficacité
900	6.75	8.25	1.22	0.41
1800	12.00	16.5	1.38	0.46
2700	21.50	23	1.07	0.36
3600	25.00	32.75	1.31	0.44

101 villes				
nombre d'itération	temps pour 3 pros	temps pour 01 pro	accélération	efficacité
900	10.75	22.5	2.09	0.70
1800	21.00	49	2.33	0.78
2700	29.75	72	2.42	0.81
3600	38.00	98.5	2.59	0.86

3.5.1.2 Discutions :

Dans le premier tableau les résultats montrent une ameloiration consédirble donnée par la methode recherche à mémoire adaptative par rapport descante recursive.

Dans le deuxième et le toisième tableaux qui représentent des tests sur une instance de 51 villes, le temps d'exécution de l'algorithme séquentiel est moins que le temps d'exécution de la version parallèle avec 03 processus sur 03 processeurs et la qualité des solutions sont très comparables.

Mais avec l'augmentation du nombre des villes (dans le cas de 76 villes) le temps d'exécution de la version séquentielle dépasse le temps de la version parallèle.

Des que la complexité augmente avec le nombre de villes 101 la différence entre les deux versions apparaisse explicitement et on peut parler de l'accélération donnée par le parallélisme proposé.

3.6 Conclusion

Dans ce chapitre on a implémenté une parallélisassions de la méthode de recherche a mémoire adaptative basée sur le modèle multi-cherche coopérative, on veut prouver que ce type de parallélisme augmente l'efficacité de la méthode (donne une accélération au calcul et trouve des bonnes solutions). L'implémentation était réalisée sur le problème du voyageur de commerce, utilisant le langage java et le passage de message comme un modèle de programmation parallèle. Le résultat est une application parallèle testée sur la plateforme P2PMPI qui repose sur JXTA. Les résultats montrent que la méthode de recherche a mémoire adaptative donne des résultats meilleurs que la méthode descente relancée plusieurs fois, et montre une très bonne accélération entre le modèle parallèle et le modèle séquentiel quand le nombre de villes augmente et avec une amélioration de la qualité des solutions.

Conclusion générale

Notre travail s'inscrit dans le domaine de la résolution des problèmes d'optimisation combinatoire difficiles avec l'utilisation des méthodes dites Metaheuristiques. Les Meta heuristiques sont des méthodes qui utilisent des processus stochastiques et itératifs pouravoir des bonnes solutions pour les problèmes d'optimisations.

Les metaheuristiques se diffèrent en ce qui concerne leurs origines (physique, biologique,…), leurs nombre des solutions à traiter à la fois, et leurs stratégies d'échapper à un optimum local.

L'augmentation de la taille des problèmes abordés a conduit à une situation où il faut exploiter mieux la puissance des métaheuristiques.

Le parallélisme est l'un des techniques les plus logiques à utiliser car la structure algorithmique des metaheuristiques motive le parallélisme. De là plusieurs modèles des metaheuristiques parallèles ont été conçus et implémentés, parmi ces modèles on a le modèle multicherche coopératif. Ce modèle ne tente pas seulement de réduire le temps d'exécution sur les grandes instances des problèmes difficiles mais aussi d'augmenter la qualité des solutions trouvées.

On se basant sur ce modèle, on a proposé et implémenté une version parallèle de la méthode de recherche locale à mémoire adaptative, et on a choisi le problème de voyageur de commerce pour les tests.

Les tests montrent que le parallélisme est très efficace lorsque la taille du problème augmente, et comme on a dit dans le deuxième chapitre on a procédé à diviser le nombre total d'itérations de l'algorithme séquentiel sur le nombre des processus participants à la recherche dans la version parallèle. On obtient des résultats plus efficaces que les résultats trouvées par l'algorithme séquentiel.

Pour les petites instances le temps de la communication interprocessus occupe une bonne partie du temps total de l'exécution, pour ce-là l'exécution séquentielle apparait plus rapide que l'exécution parallèle.

Noter qu'il existe d'autres aspects influents quasiment sur l'efficacité de la méthode telle que la gestion de la mémoire locale et la mémoire globale, la taille des portions, le taux d'utilisation des partie de la mémoire globale et la mémoire locale.

Cette méthode peut être exploitée sur des plateformes de calcul intensif pour mieux juger leur efficacité et pour l'améliorer.

Références

[AL97] **E.H.L. Aarts & J.K .Lenstra,** Local Search in Combinatorial Optimization.Discrete Mathematics and Optimization. Wiley-Interscience, Chichester, England, June 1997**.**

[AK89] **E.H.L. Aarts& J. Korst**, Simulated annealing, boltzmann machines: a stochastic approach to combinatorial and neural computing. Wiley edition , Chichester, 1989.

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

[BAF08] **B. Bontoux & C. Artigues & D. Feillet,** A Memetic Algorithmwith a Large Neighborhood Crossover Operator for the Generalized Traveling Salesman Problem, University of Technology of Troyes, France,2008.

[BAP06] **A. BAPTISTE,** Les métaheuristiques en optimisation combinatoire, conservatoire national des arts et metiers paris,2006.

[BDJM94] A. Beguelin & J. Dongarra & W. Jiang & R. Manchek , PVM_ Parallel Virtual Machine A Users_ Guide and Tutorial for Networked Parallel Computing Al Geist , Vaidy Sunderam Massachusetts Institute of Technology, 1994.

[BR03] J .BOIGONTIER & N.RICHASSE , LE CALCUL PARALLELE Structures & Applications Travail d'études Licence , 2002/2003.

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

[CLM06] **A. COSTANZO & T.** LUONG & G. MARILL, «Optimisation par colonies de fourmis », 2006.

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

[CT03] **T.G. Crainic & 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.

[CT03] T.G .Crainic & 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.

[DFJ54] G. DANTZIG & R. FULKERSON & S. JOHNSON, SOLUTION OF A LARGE-SCALE TRAVELING-SALESMAN PROBLEM* *The Rand Corporation, Santa Monica, California* (Received August 9, 1954)

[DS00] **M. Dorigo & 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.

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

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

[Fos95] I. Foster, Designing and building parallel programs: Concepts and tools for parallel software engineering, Addison-Wesley, 1995.

[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.

[HL03] **L. Haouari & T. Ladhari,** A branch-and-bound based local search method for the flowshop problem, The Journal of the Operational Research Society 54(10), 1076–1084, 2003.

[HS05] **H. Hoos & T. Stutzle**, Stochastic Local Search: Foundations and Applications, Morgan Kaufmann, 2005.

[LV08] **V.Laurence & L. Violaine,** Généralités sur parallélisme, Ecole d'Automne ((Informatique Scientifique pour le Calcul)), 3Décembre2008.

[Mel05] **N. Melab,** Contributions à la résolution de problèmes d'optimisation combinatoire sur grilles de calcul, HDR, Lille France, 2005.

[Noe07] **S. NOËL**, métaheuristiques hybrides pour la resolution du probleme d'ordonnancement de voitures dans une chaine d'assemblage automobile, memoire presente comme exigence partielle de la maitrise eninformatique,2007.

[PBN07] **C. Pessan, J.L. Bouquard & E. Néron,** Genetic Branch-and-Bound or Exact Genetic Algorithm, Artificial Evolution, 136–147, 2007.

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

[Sbi03] **A. Sbihi,** Les méthodes hybrides en optimisation combinatoire : algorithmes exacts et heuristiques,thèse de doctorat, université de paris, le18 décembre 2003.

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

[PS82] **C.H. PAPADIMITRIOU & K. STEIGLITZ,** Combinatorial optimization – algorithms and complexity. Prentice Hall, 1982.

[VD04] **C.P. Valentin & T.M. Djenet,**Optimisation Par Colonies de Fourmis,2004.

[WD96] D .Walker & J. Dongarra , MPI: The Complete Reference by Marc Snir, Steve Otto, Steven Huss-Lederman, Massachusetts Institute of Technology , 1996.

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

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

Table des matières

Résumé

les métaheuristiques sont des méthodes approchée utilisées avec succès pour trouver des bonnes solutions pour les problèmes dites difficiles, le parallélisassions de métaheuristiques peut augmenter la taille des problèmes traites ; dans ce travail on propose une implémentation parallèle d'une métaheuristique de recherche locale appliquée sur le problème de voyageur de commerce

Mots Clés : *problèmes difficiles, métaheuristiques, parallélisassions des métaheuristiques, supports, problème de voyageur de commerce.*

Abstract

Metaheuristics are approximate methods successfully used to find good solutions for difficult problems say, the parallelization of metaheuristics can increase the size of the treated problems in this work we propose a parallel implementation dune metaheuristic local search applied to the traveling salesman problem

Keywords: *difficult problems, metaheuristics, parallelization of metaheuristics, media, traveling salesman problem.*

UNIVERSITE KASDI MERBAH OUARGLA -
B.P. 511, 30 000, Ouargla. Algérie