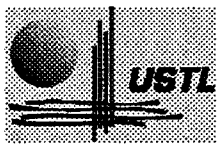


UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

U.F.R. D'I.E.E.A.

Numéro d'Ordre: Année: 1999

THÈSE

pour obtenir le grade de

DOCTEUR DE L'U.S.T.L.

DISCIPLINE : INFORMATIQUE

présentée et soutenue publiquement,

le 16 décembre 1999, par

VINCENT BACHELET

Titre:

MÉTAHEURISTIQUES PARALLÈLES HYBRIDES : APPLICATION AU PROBLÈME D'AFFECTATION QUADRATIQUE

Jury:

Président:	M. Bernard TOURSEL	Université de Lille I
Rapporteurs:	M. Gerd FINKE	Université Joseph Fourier - Grenoble
	M. Arnaud FRÉVILLE	Université de Valenciennes Hainaut-Cambrésis
Examinateurs:	M. Michel HABIB	Université de Montpellier II
	M. Philippe PREUX	Université du Littoral Côte d'Opale
Directeurs:	M. El-Ghazali TALBI	Université de Lille I
	M. Jean-Marc GEIB	Université de Lille I

Remerciements

Mes remerciements vont en premier lieu à Philippe Hasbroucq, Directeur du Département Informatique de l'École des Mines de Douai, et à Jean-Marc Geib, Directeur du Laboratoire d'Informatique Fondamentale de Lille pour m'avoir accueilli durant 4 ans.

Je tiens à exprimer ma gratitude à Gerd Finke et à Arnaud Fréville pour avoir accepté de rapporter mon travail. Je remercie vivement les membres du jury, Michel Habib et Philippe Preux, et son président, Bernard Toursel, pour leur participation très enthousiaste.

Je tiens à remercier particulièrement, Jean-Marc Geib et El-Ghazali Talbi, pour l'intérêt qu'ils ont accordé à la direction de mon travail de recherche, pour leurs conseils avisés. Je suis aussi très reconnaissant à Philippe Preux, pour ses suggestions et ses encouragements amicaux.

Je remercie mes collègues de l'École des Mines de Douai et du Laboratoire d'Informatique Fondamentale de Lille pour leur soutien tout au long de ces années. Je remercie spécialement Clarisse pour ses nombreuses relectures.

Mes derniers « mercis », mais non les moindres, je les adresse à ma famille : ma femme, nos enfants, nos parents et les autres. Merci de votre appui. Merci de votre patience quand je vous ai manqué.

Table des matières

Introduction générale

Ce mémoire rapporte des travaux de recherche dans le domaine de l'optimisation combinatoire. La communauté de l'optimisation combinatoire regroupent des chercheurs autour d'un intérêt commun qu'est la résolution de problèmes de nature combinatoire. Les caractéristiques qui définissent la nature combinatoire d'un problème sont plutôt vagues. Cependant, les problèmes dits combinatoires présentent majoritairement les propriétés suivantes: les variables de décision sont discrètes[1] et une solution au problème est un ensemble ou une suite d'objets discrets. Ainsi, résoudre un problème d'optimisation combinatoire, c'est trouver une solution qui optimise une fonction réelle, appelée fonction coût. Pour illustrer ce qu'est un problème combinatoire, considérons, à titre d'exemple, le problème du voyageur de commerce, qui est, de loin, le problème le plus étudié en optimisation. La formulation du problème du voyageur de commerce, dans un espace euclidien, est plutôt simple: un voyageur de commerce doit visiter un certain nombre de villes; soit n ce nombre; dans quel ordre doit-il parcourir les n villes pour effectuer le trajet le plus court? Plus formellement, il s'agit de trouver le circuit hamiltonien[2] de plus faible coût dans un graphe pondéré de n noeuds complètement connecté. Habituellement, on numérote les villes de 1 à n et on représente une solution par une permutation sur l'intervalle d'entiers $[1, n]$. Par exemple, pour une instance[3] à $n = 4$ villes[4], la permutation $(3, 1, 2, 4)$ représente le circuit pour lequel la ville 3 précède la ville 1, qui elle-même précède la ville 2, qui précède la ville 4, cette dernière précédant la ville 3. On constate aisément que la représentation des solutions (des circuits) par des permutations n'est pas injective; car $(3, 1, 2, 4)$ et $(1, 2, 4, 3)$ correspondent au même circuit. Ainsi, en prenant garde à ces redondances, on dénombre[5] $(n - 1)!$ circuits hamiltoniens différents parmi lesquels on recherche le meilleur, le plus court. Pour 4 villes, on a donc $3! = 6$ circuits; pour 8 villes, on a $7! = 5040$ circuits; pour 20 villes,

1. C'est-à-dire définies sur des domaines non continus (souvent un ensemble d'entiers).

2. Un circuit passant par tous les nœuds du graphe. On considère, ici, que les arcs du graphe ont des valeurs positives correspondant aux distances entre les villes.

3. On appelle instance, un problème pour lequel les variables de données sont instanciées. Par exemple, une instance du problème du voyageur de commerce est spécifiée par le nombre de villes et les distances entre elles deux à deux.

4. Généralement, les instances optimisées concernent plutôt plusieurs centaines ou milliers de villes.

5. Dans le cas d'un graphe non orienté.

on a $19! \simeq 10^{17}$ circuits. On constate que le nombre de circuits croît très fortement (exponentiellement) avec le nombre de villes : c'est le phénomène dit de l'explosion combinatoire. Pour des tailles de problème (nombre de villes) modestes, l'espace de recherche (les circuits) est très vaste, ce qui constitue la principale difficulté dans la résolution de ces problèmes. Ainsi, pour explorer tous les circuits d'un problème de 25 villes, même un ordinateur très puissant, capable de traiter 1 milliard de tours (circuits) par secondes, mettrait 20 millions d'années.

Le phénomène d'explosion combinatoire constitue une difficulté majeure en optimisation combinatoire. La plupart des problèmes étudiés en optimisation appartiennent à la classe des problèmes NP-durs. Cette classe rassemble des problèmes pour lesquels il n'existe[6] pas d'algorithme qui fournissent la solution optimale, à coup sûr, en un temps d'exécution polynômial en la taille de la donnée. Par exemple, pour le voyageur de commerce, on ne connaît pas d'algorithmes, donnant toujours le tour le plus court, dont le temps d'exécution soit borné par un polynôme de degré quelconque en n, le nombre de villes. Face à cette problématique : trouver la meilleure solution dans un espace de recherche immensément vaste, quels plans d'attaque adopter pour résoudre un problème d'optimisation combinatoire? Puisque le phénomène d'explosion combinatoire nous interdit l'exploration complète de l'espace de recherche dans un temps raisonnable[7] (quelques heures, voire quelques jours), des méthodes ont été imaginées pour restreindre la portion de l'espace explorée. Ces méthodes, dites heuristiques, s'appuient sur des propriétés spécifiques au problème à résoudre. Ces propriétés sont connues *a priori* ou sont acquises par l'heuristique au cours de l'exploration de l'espace de recherche. Fort de ces connaissances, les méthodes d'optimisation tentent de caractériser des zones de l'espace où l'optimum ne peut se trouver, dont l'exploration n'est pas nécessaire ; et/ou tentent d'identifier des zones prometteuses, dans lesquelles l'optimum a de grandes chances d'être, et qu'il faut exploiter en priorité. Du coup, le temps de résolution est réduit. En toute généralité, une bonne heuristique est une méthode qui réussit un juste équilibre entre l'exploration de nouvelles régions de l'espace de recherche et l'exploitation des bonnes solutions déjà trouvées. Une autre approche, pour obtenir des temps de calculs raisonnables, est l'emploi de calculateurs puissants. Aujourd'hui, des contraintes technologiques limitant la puissance d'un processeur unique, les plates-formes puissantes sont des machines parallèles, c'est-à-dire intégrant plusieurs processeurs. Ainsi, pour gagner en efficacité, il convient d'intégrer à l'optimisation combinatoire des techniques de parallélisme pour concevoir des méthodes d'optimisation parallèles, plus performantes. Ces dernières années, quelques « heuristiciens » se sont orientés vers les machines parallèles. En effet, seuls les calculateurs parallèles ont permis d'obtenir des résultats satisfaisants sur des instances de grandes tailles.

Cette thèse a été réalisée dans le cadre du groupe de recherche PERFORM[8]. Ce

6. Cette propriété des problèmes NP-durs, bien qu'elle soit consensuelle, n'a pas été démontrée.
7. C'est-à-dire exploitable à l'échelle humaine.
8. http://www.lifl.fr/~talbi/perform.html

groupe rassemble les efforts de chercheurs de la région Nord sur le thème de la résolution de problème combinatoires. Le groupe PERFORM est composé d'une dizaine de chercheurs dont 5 maîtres de conférence. L'objectif principal de notre groupe est la mise en œuvre conjointe de méthodes heuristiques et de techniques de parallélisation pour aboutir à la résolution de problème d'optimisation complexes de grande dimension. Les travaux réalisés par le groupe ont porté sur l'étude et la réalisation de nombreux algorithmes d'optimisation parallèles (algorithme génétique, recuit simulé, recherche tabou, colonies de fourmis, ...), selon une conception parallèle et distribuée (régulation de charge, parallélisme adaptatif, mécanisme de sauvegarde/restauration, processus légers, ...) pour résoudre des problèmes complexes (voyageur de commerce, affectation quadratique, couverture d'ensembles, partitionnement de graphe, ordonnancement, ...) par des techniques d'hybridation et de coévolution. Dans cette thèse, nous nous sommes intéressés à la classe des métaheuristiques itératives. Les métaheuristiques sont des heuristiques qui ne sont pas spécifiques à un problème particulier; elles n'ont aucune connaissance *a priori* du problème à résoudre[9]. Ces heuristiques intègrent des principes de résolution de problèmes combinatoires très généraux; de ce fait, elles s'adaptent aisément à la résolution de nombreux problèmes. Le fonctionnement des métaheuristiques itératives est plutôt simple : d'une manière générale, ces méthodes s'appuient sur une ou plusieurs solutions, de qualités (coûts) quelconques, qu'elles tentent d'améliorer, pas à pas, au fil des itérations, en les modifiant partiellement. Parmi les méthodes itératives les plus connues, on trouve les algorithmes génétiques, le recuit simulé et la recherche tabou. Depuis quelques années, on constate que les métaheuristiques les plus performantes sont des méthodes hybrides, c'est-à-dire combinant plusieurs (très souvent deux) métaheuristiques. Sont à l'origine de l'hybridation des métaheuristiques, des constatations selon lesquelles certaines méthodes auraient tendance à trop explorer l'espace de recherche, alors que d'autres, au contraire, auraient une exploitation dominante. Ainsi, la conjugaison dans une méthode hybride, de métaheuristiques aux comportements complémentaires, fournirait l'équilibre recherché entre exploration et exploitation, donnant une heuristique performante. L'objectif principal de cette thèse, est de mieux comprendre le fonctionnement des métaheuristiques parallèles hybrides afin de proposer des méthodes plus efficaces. Pour ce faire, nous avons conduit notre étude selon trois fronts :

- l'étude de la structure intrinsèque des instances;
- l'étude des métaheuristiques et des mécanismes d'hybridation,
- la conception et l'implantation sur machines parallèles.

En comprenant mieux le comportement des métaheuristiques et la nature des espaces dans lesquels elles évoluent, nous avons voulu expliquer les écarts de performances

9. À l'inverse, les heuristiques dédiées sont spécifiques à la résolution d'un problème donnée; leurs concepteurs tentent d'y intégrer un maximum de leurs connaissances du problème à résoudre.

entre différentes heuristiques sur des instances de structures différentes et justifier la robustesse des méthodes hybrides. De plus, nous avons souhaité développer des compétences dans le domaine du parallélisme pour concevoir des méthodes parallèles hybrides, dont nous savions qu'elles seraient plus performantes. Comme support de notre étude, nous nous attaquons à la résolution du problème d'affectation quadratique (QAP). C'est un problème classique difficile, reconnu pour la communauté de l'optimisation combinatoire. L'intérêt pour le QAP provient, sans doute, de sa grande difficulté, mais aussi, et surtout, du fait qu'il contient un grand nombre d'autres problèmes (par exemple, le voyageur de commerce) et parce qu'il modélise de très nombreux problèmes réels dans des domaines variés.

Ce mémoire est articulé en trois parties principalement. La partie I concerne le paysage du QAP ; la partie II, les métaheuristiques et leurs parallélisations ; la partie III, les techniques d'hybridation et les mécanismes de coévolution.

• La partie I est consacrée au QAP. Notre objectif est de mieux cerner les différentes natures intrinsèques des instances car nous avons constaté, pour une même méthode, des variations de performances selon l'instance à résoudre. L'objectif final est d'adapter la méthode à l'instance à résoudre pour gagner en efficacité. Dans cette partie, outre un rapide état de l'art du problème et de ses méthodes de résolution, nous présentons l'étude expérimentale que nous avons menée des paysages du QAP. Dans cette étude statistique, basée sur de très nombreuses expérimentations, nous sélectionnons et proposons des indicateurs qui permettent de caractériser la nature des paysages des instances. Enfin, nous donnons une taxinomie des instances en fonction de ces indicateurs. Cette taxinomie, obtenue par calcul, est assez proche d'autres classements obtenus de manière empirique.

• La partie II relate le travail que nous avons réalisé autour de la mise en œuvre de métaheuristiques parallèles sous des formes non-hybrides. Nous détaillons les environnements et plates-formes parallèles et distribuées que nous avons utilisés. Nous soulignons la grande adéquation de l'environnement de parallélisme adaptatif MARS sur réseau de stations de travail pour l'évaluation des métaheuristiques parallèles. Dans cette partie, nous commentons notre approche des métaheuristiques qui distingue les méthodes itératives à solution unique des méthodes basées sur une population de solutions. Pour chacune de ces deux catégories, nous proposons un modèle qui fédère les méthodes les plus connues. Nous donnons les caractéristiques et les évaluations sur le QAP des métaheuristiques parallèles que nous avons étudiées et constatons des variations de performances selon la catégorie de la métaheuristique et en fonction du type de l'instance identifié en partie I. Nous concluons sur l'intérêt de l'hybridation pour gagner en robustesse et en performance, principalement sur les instances de grande taille.

• La partie III concerne les métaheuristiques parallèles hybrides. Dans cette partie, nous proposons une classification originale des méthodes hybrides. Cette taxinomie, illustrée de nombreux exemples, apporte un peu de clarté dans la multitude de

méthodes hybrides plus ou moins complexes que nous avons recensées. Notre taxinomie est complétée par une grammaire qui permet de décrire d'une manière concise tous les types d'hybridation que nous avons rencontrés. Par ailleurs, nous relatons dans cette partie, notre travail de conception de métaheuristiques hybrides coévolutionnistes. Nous proposons une métaheuristique basée sur la coévolution d'un agent de recherche, d'un agent de diversification, et d'un agent d'intensification coopérant par l'intermédiaire d'une mémoire adaptative. Cette mémoire, renseignée par l'agent de recherche, contient une information sur les zones prometteuses de l'espace de recherche et sur les zones à éviter. L'agent de diversification accélère la découverte de nouvelles zones de l'espace, non encore explorées. L'agent d'intensification augmente l'exploitation de régions prometteuses. Cette métaheuristique coévolutionniste, appliquée à la résolution du QAP, se montre très efficace, spécialement sur les instances de grande taille.

Première partie

Le QAP et son paysage

Cette partie s'articule en deux chapitres. Le premier présente le QAP et les méthodes de résolution. Le second chapitre s'intéresse à la structure intrinsèque du problème. Il décrit la notion de paysage et relate une étude expérimentale qui conduit à une taxinomie des instances.

Les travaux décrits dans cette partie ont fait l'objet d'une présentation à la Conférence Internationale ECCO X et d'un rapport interne du Laboratoire d'Informatique du Littoral [Bachelet et al.97] [Preux et al.99].

Chapitre 1

Le problème d'affectation quadratique

Dans ce chapitre, nous présentons le problème d'affectation quadratique qui sert de support à notre étude. Notre objectif est de faire un rapide tour d'horizon des aspects pratiques et théoriques du problème. Nous donnons quelques exemples de problèmes réels et présentons quelques résultats fondamentaux sur sa complexité. Enfin, nous nous intéressons aux méthodes de résolution, anciennes ou récentes, pour conclure sur l'efficacité des métaheuristiques itératives.

Bien qu'il soit un problème ancien, puisqu'on le connaît depuis les années 50, le problème d'affectation quadratique (QAP – *Quadratic Assignment Problem*) suscite encore aujourd'hui beaucoup d'intérêts. À cela, deux raisons principales : d'une part, beaucoup de problèmes du monde industriel sont modélisés par des QAP ; d'autre part, de nombreux autres problèmes d'optimisation combinatoire s'expriment comme des cas particuliers du QAP[1]. Ainsi, le problème d'affectation quadratique apparaît comme un problème très « général » ; et de ce fait, résoudre le (seul) QAP, c'est résoudre de nombreux problèmes industriels ou académiques.

Par ailleurs, il y a de nombreux problèmes combinatoires que la théorie montre comme « difficiles » mais qui se révèlent « faciles » en pratique. Pour sa part, le QAP n'est pas seulement un problème « difficile » en théorie, c'est aussi un problème « difficile » en pratique. À titre de comparaison, on sait aujourd'hui résoudre certaines instances du problème du voyageur de commerce comportant plusieurs milliers de villes ; on sait trouver la clique maximale dans un graphe comportant des milliers de sommets et des millions d'arcs ; mais pour le QAP, on ne sait pas, à notre connaissance, résoudre des instances de taille supérieure à 25. Ce facteur de difficulté

1. Entre autres, le voyageur de commerce, le partitionnement de graphe, la clique maximale d'un graphe.

augmente l'intérêt que la communauté de l'optimisation combinatoire porte au QAP.

1.1 Approche pratique du problème

1.1.1 Un peu d'histoire

Pour notre étude, le QAP est traité en tant que problème formel. Cependant, la formulation mathématique du QAP provient de la modélisation de certains problèmes de placement qu'on rencontre dans le domaine de la production industrielle [Koopmans et al.57]. Très souvent, des industriels se demandent comment agencer les différentes unités d'un site de production pour minimiser le coût inhérent au transport des produits à l'intérieur du site. « Faut-il implanter l'unité d'extrusion à côté du secteur d'assemblage ou plutôt à côté du magasin des matières brutes? ». Avant l'ère informatique, on utilisait des méthodes graphiques pour concevoir la disposition d'une usine mais ces méthodes ne permettent pas une subdivision du site en plus d'une dizaine d'unités. Dans les années cinquante et soixante, quelques chercheurs se sont intéressés à la méthodologie du placement pour répondre aux besoins des industriels. Ils ont proposé une méthode heuristique qu'ils ont implantée sur un ordinateur [Buffa et al.64] [Vollman et al.66]. Pour notre étude, nous avons appréhendé le QAP à travers une définition formelle ; mais nous l'avons presque toujours imaginé comme un problème d'optimisation du placement d'éléments sur des emplacements.

1.1.2 Les instances du QAP

Presque toutes les instances du QAP que nous avons traitées dans notre étude sont des instances « standard », issues de la bibliothèque QAPLIB [Burkard et al.91]. Cette bibliothèque, reconnue par la communauté œuvrant sur le QAP, facilite grandement la comparaison des méthodes de résolution, c'est pourquoi nous l'utilisons. Les instances QAPLIB sont d'origines variées : ce sont des instances générées par des programmes informatiques et/ou des instances modélisant des problèmes réels. Les instances générées ont été produites à l'aide de générateurs de nombres pseudo-aléatoires ; cependant, certains auteurs ont introduit des contraintes qui leur procurent une nature non uniforme, imitant, pour certains, la nature des instances réelles. On trouve des instances réelles du QAP dans une grande variété de domaines dont voici quelques exemples : l'architecture [Elshafei77] [Whitehead et al.64] [David et al.73], le sport : élaboration d'une équipe de relais [Heffley77], l'imagerie : trames de niveaux de gris [Taillard95], le câblage en électronique [Steinberg61], la conception de claviers et baies de contrôle [Burkard et al.77], la conception de turbines hydrauliques [Laporte et al.88], l'analyse de données statistiques ou l'ordonnancement de chaînes de production parallèles [Geoffrion et al.76]. Nous détaillons,

ci-après, quelques uns de ces problèmes qui ont été modélisés par des QAP. Tous ces exemples peuvent être imaginés comme un problème de placement de n éléments sur n sites pour lequel les données sont les distances entre les sites et les flux entre les éléments.

1.1.2.1 Problèmes de câblage

Le problème de câblage peut s'énoncer comme suit: on dispose d'un certain nombre, soit n, de composants électroniques à connecter entre eux, et n emplacements. Comment agencer les différents composants sur les différents emplacements pour minimiser la longueur totale des fils de câblage? Les données sont les distances entre les emplacements pris deux à deux et les nombres de connexions que nécessite le montage entre les composants pris deux à deux. En 1961, Steinberg présente la résolution d'un problème de câblage d'ordinateur à 36 composants; ce problème, de taille 36, est désigné par **Ste36x** dans la QAPLIB [Steinberg61].

1.1.2.2 Conception de claviers de machine à écrire

Sur les machines à écrire mécaniques, des problèmes de fonctionnement surviennent lorsqu'on frappe trop vite; de ce fait, on doit respecter un certain délai entre deux touches. L'objectif de Burkard et Offermann était d'optimiser l'organisation des lettres sur un clavier de machine à écrire de façon à minimiser ces incidents [Burkard et al.77]. Ils modélisent ce problème par un QAP selon le principe d'optimisation suivant: une séquence de deux touches qui peut s'exécuter rapidement (délai court entre les touches) doit correspondre à une séquence de deux lettres qui est fréquemment frappée, donc fréquemment présente dans la langue écrite; et inversement, une séquence de deux touches qui exigent un délai plus long entre elles doit correspondre à une séquence de deux lettres peu fréquente. Les données du problème sont les différents temps de repos nécessaires entre les différentes touches (qu'on peut assimiler à des distances), et les fréquences d'occurences des séquences de deux lettres (qu'on peut assimiler à des flux) dans la langue à dactylographier. Ce problème, de taille 26 (car 26 lettres!), est décliné en huit variantes en fonction du type de machine à écrire et en fonction de la langue. Ces instances sont désignées par **Bur26a, Bur26b, ..., Bur26h** dans la QAPLIB.

1.1.2.3 Architecture des centres hospitaliers

Les hôpitaux modernes sont de grands complexes, dotés de très nombreux services spécialisés (admission, radiologie, urgences, chirurgie, analyses, etc.). En architecture, se pose le problème du placement de ces différentes unités afin de minimiser les flux des patients et des personnels soignant. Lors de la conception de l'hôpital

Maher au Caire, un des objectifs fût de limiter la charge inhérente au déplacement des patients entre les différents services [Elshafei77]. Dans la QAPLIB, ce problème est désigné sous le nom **Els19**. Ainsi, les services furent agencés de façon à minimiser la distance totale (en mètres-patients) parcourue en un an. Les données qui furent exploitées étaient la distance (en mètre) entre les différents emplacements susceptibles de recevoir un service, et le flux de patients entre les services. D'autres études sur la répartition des services au sein des hôpitaux sont relatées dans [Whitehead et al.64] [David et al.73].

1.1.2.4 Conception de trames de niveaux de gris

Le problème du rendu d'un niveau de gris, par tramage de points noirs et de points blancs, a été formulé comme un QAP par Taillard [Taillard95]. Dans la bibliothèque QAPLIB, les instances relatives au problème de tramage sont notées **Taixxc**. Le problème peut être exposé comme suit : il s'agit de créer un niveau de gris en répartissant des cases noires sur une grille blanche, à la manière d'un damier. L'idée n'est pas nouvelle, mais les tramages couramment employés font apparaître un quadrillage ou des motifs qui rendent le gris obtenu peu homogène et insatisfaisant. Pour obtenir une répartition homogène des cases noires dans la grille blanche, Taillard imagine que les cases noires se repoussent comme des électrons et interagissent entre elles, deux à deux, avec une intensité inversement proportionnelle au carré de leurs distances. La taille du problème est le nombre de cases de la grille. Par exemple, pour l'instance **Tai256c**, il s'agit d'obtenir une trame représentant un niveau de gris d'environ 35% en plaçant régulièrement 92 cases noires sur une grille carrée 16/16. Les données du problème sont les interactions entre chaque paire de cases de la grille comme si elles étaient toutes noires, et le nombre de cases noires. Il est plus difficile, car peu naturel, d'imaginer le problème de tramage comme un problème de placement avec distance et flux ; d'ailleurs, ce problème est assez singulier, nous reviendrons sur ces particularités par la suite.

1.2 Aspects formels du QAP

1.2.1 Formalisation mathématique du QAP

Une instance du QAP est caractérisée par sa taille n et trois matrices carrées de réels. Lorsque le QAP est considéré comme un problème de placement de n éléments sur n emplacements, ces matrices sont désignées comme suit :

- $(d)_n$, la matrice des distances entre deux emplacements ;
- $(c)_n$, la matrice des coûts des flux de produits entre deux éléments ;

- $(s)_n$, la matrice des coûts de placement d'un élément sur un site.

Bien que le problème d'affectation s'énonce simplement : « Trouver un placement dont le coût est minimal », de nombreuses formulations mathématiques du QAP ont été proposées. Pour notre étude, nous utilisons la formulation la plus répandue, directement dérivée de la proposition originale de Koopmans et Beckmann [Koopmans et al.57]. Le lecteur pourra trouver d'autres formulations mathématiques dans [Pardalos et al.94] et [ela98].

1.2.1.1 La formulation de Koopmans-Beckmann

En 1957, Koopmans et Beckmann proposent une des toutes premières formulations du QAP [Koopmans et al.57]. Ils représentent un placement possible par un tableau où les éléments sont en ligne et les emplacements en colonne. Une croix dans ce tableau indique l'affectation d'un élément à un emplacement. Pour le QAP, chaque élément étant associé à un et un seul emplacement, il ne peut pas y avoir deux croix sur une même ligne ou sur une même colonne (Fig. 1.1).

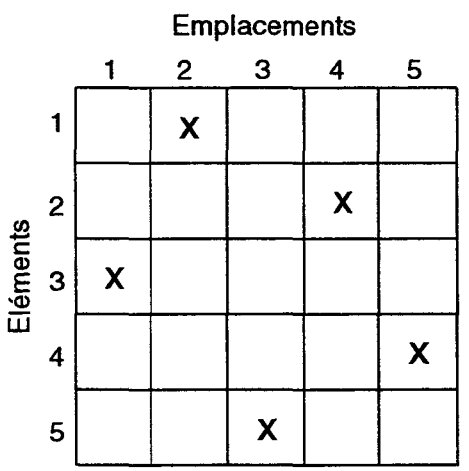


FIG. 1.1 - *Représentation matricielle d'un placement.*

Ainsi, en appelant i et j, les indices respectifs des éléments et des emplacements ; en posant $x_{ij} = 1$ si l'élément i est placé en j et $x_{ij} = 0$ sinon ; un placement possible est représenté par la matrice de permutation $(x)_n$ caractérisée par :

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, ..., n$$

$$x_{ij} \in \{0,1\} \quad i = 1,...,n \quad j = 1,...,n$$

Le problème revient à trouver la matrice $(x_*)_n$ qui minimise la fonction d'évaluation φ définie comme suit,

$$\varphi[(x)_n] = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} d_{ij}.c_{kl}.x_{ik}.x_{jl} + \sum_{i=1}^{n}\sum_{j=1}^{n} s_{ij}.x_{ij}$$

$$\varphi[(x_*)_n] = \min_{(x)_n \in \mathcal{P}_n} \varphi((x)_n)$$

où $\mathcal{P}_n$ est l'ensemble des matrices de permutations de taille n.

1.2.1.2 La formulation retenue

À partir de la proposition de Koopmans et Beckmann, on obtient une nouvelle formulation en « compactant » la représentation matricielle d'un placement. En effet, en indiquant les numéros de chaque élément à la place de chaque croix, puis en sommant les lignes on a une représentation du placement sous la forme d'un vecteur de taille n comme le montre la figure 1.2.

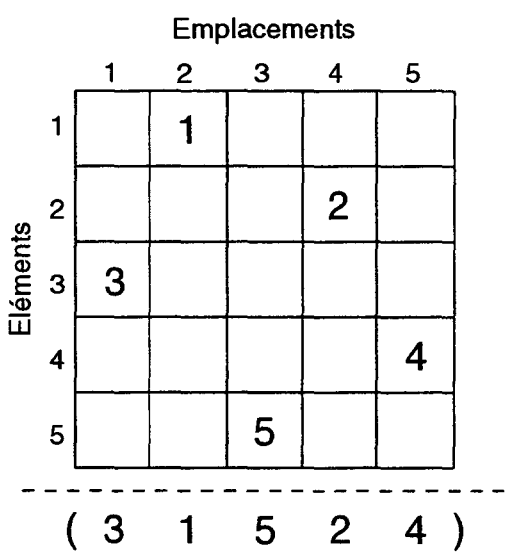


FIG. 1.2 - *Représentation vectorielle d'un placement.*

Ainsi, un placement est représenté par un n-uplet π qui est un élément de Π_n, l'ensemble des permutations de taille n. La formulation du QAP revient alors à trouver π_* tel que :

$$\varphi(\pi_*) = \min_{\pi \in \Pi_n} \left(\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}.c_{\pi(i)\pi(j)} + \sum_{i=1}^{n} s_{i\pi(i)} \right)$$

Pour notre étude, nous supposons toujours que la matrice $(s)_n$ est nulle. Ainsi, une instance du problème traité est totalement définie par une matrice de distance et une matrice de coût de flux ; le problème devient alors :

$$\varphi(\pi_*) = \min_{\pi \in \Pi_n} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}.c_{\pi(i)\pi(j)}$$

1.2.2 Propriétés du QAP

1.2.2.1 Le QAP est NP-difficile

Du point de vue de la complexité algorithmique [Garey et al.79], le QAP apparaît comme l'un des problèmes les plus difficiles à résoudre. En effet, de nombreux problèmes NP-difficiles bien connus peuvent être formulés sans réelle difficulté, comme des cas particuliers du QAP. Parmi ceux-là, citons le problème du voyageur de commerce (TSP – *Traveling Salesman Problem*), le problème de partitionnement de graphe (GP – *Graph Partitionning*) ou encore le problème de la clique maximale (MCP – *Maximum Clique Problem*). En 1976, Sahni et Gonzales ont montré que le QAP est NP-difficile [Sahni et al.76]. De plus, ils ont montré que trouver une ε-approximation est aussi NP-difficile. Cependant, pour quelques cas très particuliers de QAP, la complexité devient polynômiale ; le lecteur intéressé trouvera une bonne synthèse de ces cas particuliers dans la monographie de Çela consacrée au QAP [ela98].

1.2.2.2 Le QAP est PLS-complet

Dans les paragraphes précédents, nous avons toujours considéré que résoudre un problème d'optimisation consistait à trouver la meilleure solution de son espace de recherche. Cependant, en optimisation combinatoire, lorsqu'il est difficile de trouver la meilleure solution, on se contente, faute de mieux, de fournir une « bonne »[2] solution ; ce qui est plus facile. Pour ce faire, il est fréquent d'utiliser la recherche locale[3] (LS – *Local Search*). La recherche locale est une métaheuristique élémentaire qui est à la base d'autres méthodes plus sophistiquées comme, par exemple, la recherche tabou. Elle évolue dans l'espace de recherche muni d'une structure de voisinage. Elle commence avec une solution réalisable arbitraire, puis, cherche continuellement à progresser vers un voisin de meilleur coût ; elle s'arrête sur un optimum (local *a priori*). La recherche locale, encore appelée « méthode de descente », est présentée d'une manière plus détaillée à la section II:1.1.2.

La classe des problèmes PLS (*Polynomial-time Local Search*), proposée en 1988

2. On qualifie de « bonne », une solution qu'on estime satisfaisante. Généralement, c'est une solution de coût meilleur ou comparable aux solutions qu'on connaît déjà.

3. Dans ce document, le terme « recherche locale » désigne une méthode de descente.

par Johnson, Papadimitriou et Yannakakis [Johnson et al.88], regroupe les problèmes pour lesquels il est « facile » de trouver un optimum local par une recherche locale. D'une manière simple et informelle, un problème, muni d'un opérateur de voisinage, est PLS s'il existe trois algorithmes en temps polynômiaux qui fournissent :

- une solution réalisable ;
- le coût d'une solution réalisable ;
- la véracité de l'optimalité locale pour toute solution réalisable et une solution voisine de coût supérieur si elle existe.

Le QAP, muni d'un voisinage où ne sont voisines que les permutations qui diffèrent d'au plus une transposition, est un problème de la classe PLS. En effet, on vérifie aisément l'existence des trois algorithmes : pour une instance de taille n, toute permutation de taille n est une solution réalisable, cette solution est générée en temps polynômial ; pour obtenir le coût d'une solution de taille n, on calcule la double somme, ce qui se fait en n^2 ; enfin, pour décider de l'optimalité locale d'une solution, il suffit d'obtenir les coûts des $\frac{n(n-1)}{2}$ solutions voisines, soit une complexité de $\frac{n(n-1)}{2}.n^2$. De plus, on peut montrer que tout problème PLS est PLS-réductible au QAP. On dit qu'un problème PLS $\mathcal{P}$ est PLS-réductible à un problème PLS $\mathcal{Q}$ si et seulement s'il existe deux algorithmes A et B polynômiaux tels que : A fournit pour toute instance p de $\mathcal{P}$ en entrée, $A(p)$, une instance de $\mathcal{Q}$; B fournit pour toute instance de p de P et pour tout optimum local de $A(p)$ en entrée, un optimum local de p. La classe PLS-complet regroupe tous les problèmes PLS qui peuvent être obtenus par la réduction de chacun des autres problèmes PLS. Elle définit l'ensemble des problèmes PLS les plus difficiles (au sens de la complexité) auquel appartient le QAP .

Ces résultats théoriques, concernant la recherche locale et les optima locaux, sont difficilement exploitables en pratique. En effet, en optimisation, on s'intéresse plutôt à trouver un optimum de bonne qualité (sinon le meilleur) dans un temps raisonnable. Or, ces résultats ne concernent ni la qualité des optima locaux, ni la durée (polynômiale ?) d'une recherche locale jusqu'à l'optimum. Cependant, nous avons constaté, comme d'autres, qu'en pratique la longueur de marche (le nombre d'itérations) d'une recherche locale est relativement courte pour de nombreux problèmes combinatoires ; c'est bien là l'essentiel pour notre étude.

1.2.2.3 Comportement asymptotique du QAP

On s'intéresse ici au comportement asymptotique du QAP, c'est à dire aux propriétés des instances dont la taille tend vers l'infini. Plus précisément, on considère uniquement les instances uniformes, dont les coefficients des matrices des distances

et des flux sont uniformément distribués pour chaque matrice. Ces instances uniformes, ont une propriété asymptotique remarquable qui peut s'énoncer comme suit : le quotient du coût de l'optimum global π_* par le coût de la plus mauvaise solution de l'espace de recherche π_- tend vers 1.

$$\lim_{n\to\infty} \frac{\varphi(\pi_*)}{\varphi(\pi_-)} = 1$$

Autrement dit, pour de grandes instances uniformes, toutes les solutions sont très proches de l'optimum en terme de qualité ; ou encore toutes les solutions sont de même coût, à un ε près. Pour plus de détails concernant les propriétés asymptotiques du QAP, le lecteur pourra consulter les travaux de Fenck, Houweninge et Rinnoy Kan [Frenk et al.85] et Rhee [Rhee88] [Rhee91].

Cette propriété asymptotique est l'un des rares résultats théoriques, en optimisation combinatoire, qui peut être vérifié expérimentalement. En effet, cette propriété des instances uniformes est perceptible pour des tailles d'instances relativement modestes, vis à vis de l'infini. Nous avons constaté que les instances plutôt uniformes, comme Tai60a, Sko100a ou Tai100a, ont une amplitude[4] faible. Ces aspects, concernant la structures des instances, sont présentés avec plus de détails au chapitre suivant.

1.3 Méthodes de résolution du QAP

Communément, on distingue deux types d'approches dans les méthodes de résolution pour le QAP : les méthodes exactes et les méthodes suboptimales (voir Fig. 1.3). Les méthodes exactes résolvent, au sens propre le QAP. Autrement dit, elles fournissent, à coup sûr, un optimum global de l'instance. Cependant, les méthodes exactes conçues à ce jour ont un coût de calcul qui croît très vite avec la taille de l'instance à résoudre ; à partir des tailles $15-20$, pourtant modérées, le coût de calcul nécessaire est déjà presque prohibitif. Les méthodes suboptimales sont inspirées d'une philosophie tout autre : puisqu'il est très difficile de trouver la meilleure solution, on se contente de rechercher une « bonne » solution. Aujourd'hui, les méthodes suboptimales sont les plus efficaces pour le QAP ; notamment, certaines métaheuristiques itératives fournissent de très bonnes solutions (supposées optimales) pour des instances de grandes tailles alors qu'aucune méthode exacte, même très sophistiquée, n'a résolu un problème de taille supérieure à 25, à notre connaissance. Cependant, les méthodes exactes sont toujours employées, car contrairement aux méthodes suboptimales, elles démontrent l'optimalité des solutions qu'elles fournissent.

4. Écart entre les coûts de la pire solution et de l'optimum global.

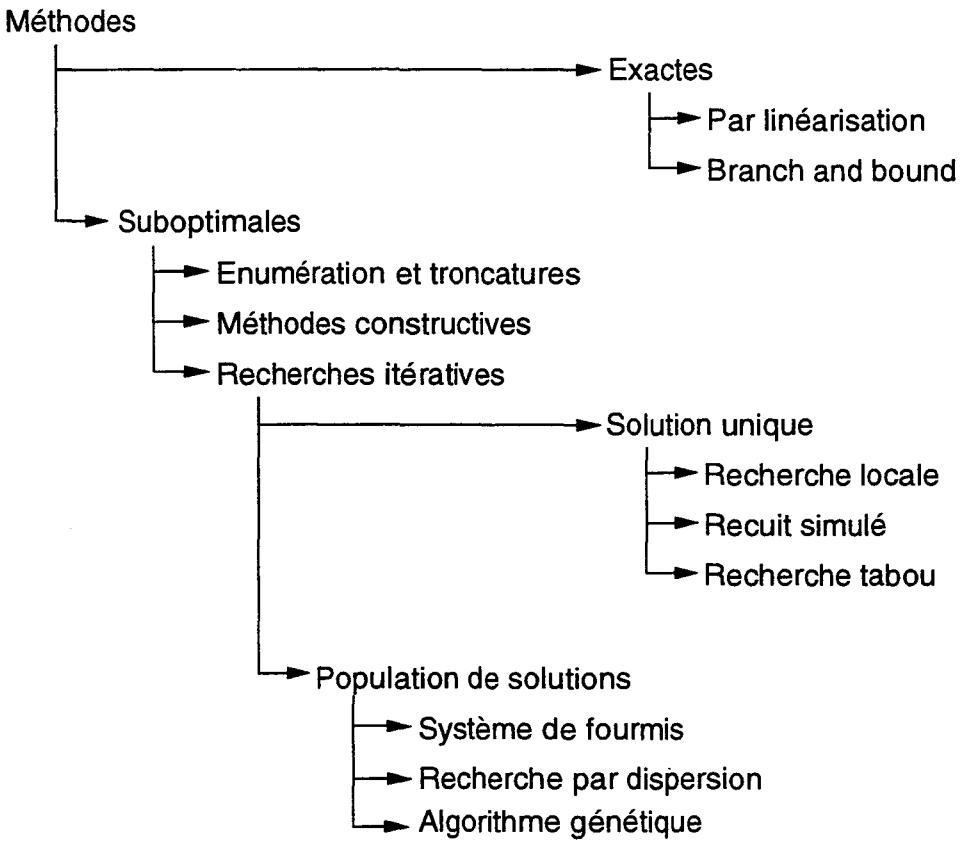


FIG. 1.3 - *Taxinomie des méthodes de résolution pour le QAP. On distingue deux branches principales : les méthodes exactes et les méthodes suboptimales. Notre intérêt se porte sur la sous-branche des méthodes suboptimales que constituent les recherches itératives. Dans cette dernière catégorie, nous distinguons les méthodes à population de solutions, des recherches à solution unique.*

1.3.1 Les méthodes exactes

Parmi les algorithmes exacts pour le QAP, on trouve les méthodes par linéarisation et les techniques de parcours d'arbres de type *branch and bound* qui se montrent les plus efficaces pour la résolution exacte du QAP.

1.3.1.1 Les méthodes par linéarisation

Les techniques d'optimisation par linéarisation sont basées sur l'expression du problème sous la forme d'un programme linéaire en variables 0-1. Le lecteur trouvera une bonne synthèse des techniques de linéarisation du QAP dans [ela98]. Ces techniques produisent de bons résultats sur certains problèmes d'optimisation combinatoire ; notamment sur le voyageur de commerce [Padberg et al.91]. Cependant, leur application au problème d'affectation quadratique fut plutôt décevante et n'a pas permis la résolution exacte de problèmes de tailles supérieures à 8 [Kaufman et al.78] [Bazaraa et al.80]. De ce fait, les méthodes par linéarisation pour le QAP sont plutôt délaissées depuis une vingtaine d'années au profit des méthodes *branch and bound.*

1.3.1.2 Les méthodes *branch and bound*

Les algorithmes *branch and bound* sont les méthodes exactes les plus efficaces pour le QAP [Mautor93]. Les méthodes de *branch and bound* reposent sur l'exploration d'une arborescence de construction des solutions. Au sommet de l'arbre, on place une solution partielle vide ; puis pour passer d'un niveau de l'arbre au suivant, on développe les différentes affectations possibles sur un site non affecté en amont[5]. L'ensemble des feuilles de l'arbre est constitué de toutes les solutions complètes ; c'est l'espace de recherche de l'instance à résoudre. Afin de limiter l'exploration de l'arbre, les méthodes *branch and bound* calculent une borne inférieure (évaluation minorant chacun des nœuds en aval) des sous-arbres. Plus la borne inférieure est précise et rapide à calculer, plus l'exploration est réduite. Les méthodes les plus performantes utilisent la borne inférieure de Gilmore-Lawler [Gilmore62] [Lawler63]. Cependant, cette borne de Gilmore-Lawler, bien que rapidement calculable manque encore de finesse. Par conséquent, il reste un grand nombre de sous-arbres à explorer dans l'arborescence et seules les instances de tailles modestes peuvent être résolues exactement.

5. Certaines méthodes, utilisant l'affectation d'une paire de sites par niveau, se sont montrées moins performantes.

1.3.2 Les méthodes suboptimales

Les méthodes suboptimales ne trouvent pas directement l'optimum, d'où leur nom. Leur objectif premier est d'obtenir une solution satisfaisante même si elle n'est pas la meilleure possible. Bien que ces méthodes n'offrent aucune garantie en ce qui concerne la qualité des solutions qu'elles produisent, certaines fournissent des solutions de bonnes qualités en des temps de calcul raisonnables. Par conséquent, ces méthodes sont très efficaces pour des applications pratiques pour lesquelles un bon résultat est acceptable, même s'il n'est pas optimal.

1.3.2.1 Les méthodes d'énumération limitée et de troncature

L'idée de ces méthodes est de commencer l'énumération systématique de tout l'espace de recherche, puis de s'arrêter après un temps raisonnable. On retient alors la meilleure solution rencontrée. Dans [Pardalos et al.94], les auteurs citent l'exemple de l'instance Nug15 pour laquelle l'optimum est obtenu en quelques secondes seulement[6]. Ces méthodes, encourageantes sur les petites instances, s'avèrent, en fait, très peu performantes. Cela provient sans doute du fait qu'elles explorent l'espace de recherche d'une manière totalement aveugle. D'autres techniques, plus « clairvoyantes », qui sont à rapprocher des méthodes exactes, suppriment certaines zones de l'espace de recherche de leur exploration en fonction de critères heuristiques. Il s'agit, par exemple, des techniques des plans de coupe et de l'exploration arborescente restreinte [Bazaraa et al.80] [Burkard et al.83a] [Hillier et al.66].

1.3.2.2 Les méthodes constructives

Les méthodes constructives [Gilmore62] partent d'une permutation vide et élaborent progressivement, affectation après affectation, une solution partielle, puis complète. Les affectations partielles sont choisies en fonction de certains indicateurs heuristiques. Cependant, à l'inverse du *branch and bound*, elles ne remettent jamais en cause les choix précédents. On peut citer, dans cette rubrique[7], l'une des plus anciennes heuristiques pour le QAP, datant des années 60 : la méthode CRAFT (*Computerized Relative Allocation of Facilities Technique*) de Armour et Buffa [Armour et al.63] [Buffa et al.64]. La méthode CRAFT optimise une configuration fournie par l'utilisateur en procédant itérativement à des échanges de deux ou trois sites. À chaque étape, l'algorithme retient de façon définitive l'échange qui produit la meilleure réduction de coût. CRAFT s'arrête lorsque plus aucun échange n'améliore la solution courante.

6. 23,8 secondes sur une machine de type CDC Cyber 76.

7. Bien que la méthode CRAFT ressemble fort à une recherche locale (méthode itérative), il est d'usage de la répertorier comme une méthode constructive.

1.3.2.3 Les méthodes itératives

À ce jour, les métaheuristiques de recherche itératives sont sans conteste les méthodes les plus efficaces pour une résolution approchée du QAP. Ces méthodes suboptimales commencent avec une ou plusieurs solutions arbitraires qu'elles tentent d'améliorer par étapes successives. Elles terminent généralement après un quota d'itérations ou quand elles ne progressent plus. Pour notre étude, nous avons retenu certaines de ces méthodes performantes pour servir de briques de base pour l'hybridation. C'est pourquoi, nous les présentons de façon plus détaillée en partie II.

Parmi les métaheuristiques itératives, nous distinguons (voir Fig. 1.3) les méthodes à solution unique :

- La recherche locale (LS – *Local Search*) [Papadimitriou et al.82] ;
- Le recuit simulé (SA – *Simulated Annealing*) [Kirkpatrick et al.83] ;
- La recherche tabou (TS – *Tabu Search*) [Glover89a].

et les méthodes à population de solutions :

- Les systèmes de fourmis (AS – *Ants System*) [Colorni et al.91] ;
- La recherche par dispersion (SS – *Scatter Search*) [Glover77] ;
- Les algorithmes génétiques (GA – *Genetic Algorithm*) [Holland75].

1.4 Conclusion du chapitre

Ce chapitre constitue un préalable à notre étude des métaheuristiques parallèles hybrides. En effet, le problème d'affectation quadratique a servi de support à notre travail : dans la suite de ce mémoire, nous y faisons référence de nombreuses fois, que ce soit pour illustrer certaines définitions et le fonctionnement d'algorithmes, et aussi, pour présenter les résultats de nos travaux.

Dans ce chapitre, nous avons présenté le problème d'affectation quadratique. Nous avons donné des définitions formelles, illustrées par des exemples pratiques. Nous avons donné un rapide état de l'art des méthodes de résolution du QAP en insistant sur la supériorité des métaheuristiques itératives.

Chapitre 2

Le paysage du QAP

vers une taxinomie des instances

Dans ce chapitre, nous nous intéressons à la notion de paysage en optimisation combinatoire en général. Puis, nous présentons une étude statistique des paysages du QAP dans laquelle nous avons défini des indicateurs de structure des instances. Enfin, nous donnons une taxinomie des instances en fonction de ces indicateurs. Notre objectif est de mieux cerner les différentes structures des instances du QAP pour mieux comprendre les écarts de comportement des heuristiques, puis adapter la méthode à la nature de l'instance à résoudre pour gagner en efficacité.

2.1 Introduction

Comme d'autres, nous croyons que la vraie question, en optimisation combinatoire, n'est pas de définir quel est le meilleur algorithme sur une large étendue de problèmes. Nous sommes convaincus qu'il est plus avantageux de rechercher quelle est la méthode la mieux adaptée à une instance ou à un groupe d'instances d'un problème donné. En effet, le propre de l'heuristique, est d'appréhender la structure d'une instance pour trouver une solution de bonne qualité. Donc, bien comprendre la structure d'une instance, d'un problème, c'est mieux comprendre le fonctionnement des méthodes d'optimisation afin d'en concevoir de plus performantes.

L'étude des paysages est une approche qui permet de différencier, de classifier les différentes instances d'un problème. Cette hiérarchisation des instances, permet de mieux comprendre pourquoi les heuristiques n'ont pas toujours le même comportement sur des instances différentes. Cette pensée est consolidée, par ailleurs, par un résultat théorique, dû à Wolpert et Macready, connu sous le nom de théorème NFL (*No Free Lunch*) [Wolpert et al.95]. Le théorème NFL montre qu'il n'existe pas

d'heuristique qui soit meilleure qu'une autre ; pour toute heuristique, aussi bonne soit-elle, on peut trouver au moins une instance pour laquelle une autre méthode lui est supérieure. Ainsi, le théorème NFL nous conforte dans l'idée qu'il n'existe pas de métaheuristique idéale de type « boîte noire »[1], et qu'il faut introduire dans les méthodes une certaine information sur l'instance à résoudre.

2.2 Le paysage d'une instance

Née au début du siècle, dans le cadre de l'étude de l'évolution des êtres vivants, la notion de paysage est décrite pour la première fois par Wright en 1932. Dans son article, Wright représente en deux dimensions les différentes combinaisons de gènes possibles et trace des contours de combinaisons de même *fitness*[2] (comme des courbes de niveaux sur une carte géographique) [Wright32]. Il définit la notion de paysage de *fitness* dans le domaine de l'évolution des espèces. Cette notion de paysage, issue de la biologie, a été, par la suite, transposée dans d'autres domaines dont celui des algorithmes évolutionnistes, puis de l'optimisation combinatoire. Le lecteur est renvoyé à [Stadler95] pour plus de précisions sur la notion de paysage comme nous l'entendons ici.

Nous définissons, ici, le paysage d'une instance d'un problème combinatoire et présentons quelques notions connexes. Bien que notre description reste générale, nous donnons des exemples appliqués au QAP pour illustrer les notions formelles. Les définitions qui suivent sont relatives à une instance I d'un problème de minimisation[3] pour laquelle on désigne par S l'ensemble des solutions, et $\varphi : S \mapsto \mathbb{R}$ est la fonction d'évaluation. Ainsi, répondre au problème, c'est trouver une solution s_*, telle que $\varphi(s_*) = \min_{s \in S} \varphi(s)$. Pour les exemples, on considère qu'une instance du QAP est définie par :

- n, la taille de l'instance ;
- $(d)_n$, $(c)_n$, les matrices des distances et des coûts ;
- $S = \Pi_n$, l'ensemble des permutations de taille n ;
- $\pi \in \Pi_n$, $\varphi(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}.c_{\pi(i)\pi(j)}$;
- π_*, l'optimum global : $\varphi(\pi_*) = \min_{\pi \in \Pi_n} \varphi(\pi)$.

1. Une méthode qui se suffit à elle même, très robuste, capable de résoudre tous les problèmes sans aucune connaissance *a priori*.

2. En biologie, la *fitness* désigne le niveau d'adaptation d'un individu à son environnement ; il s'agit de la qualité d'un individu.

3. Le choix d'un problème de minimisation, n'enlève rien à la généralité de la démarche qui est identique pour un problème de maximisation.

2.2.1 Espace de recherche d'une instance

2.2.1.1 Notion de voisinage

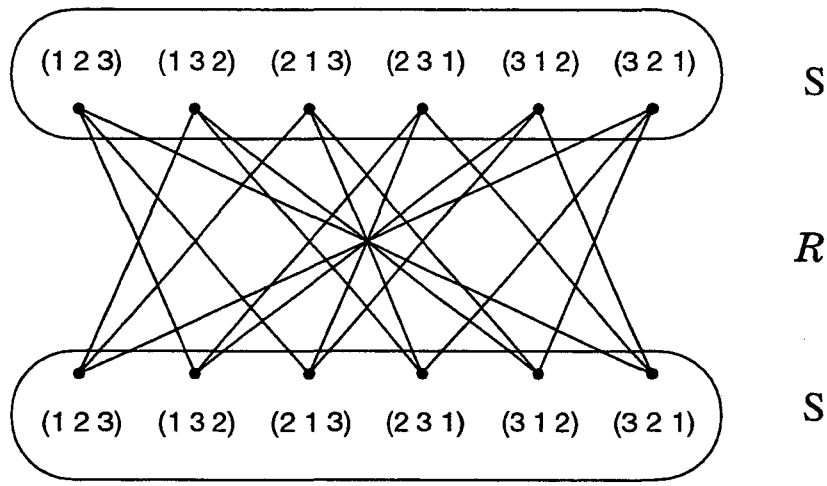


FIG. 2.1 - *Diagramme de la relation de voisinage basée sur l'opérateur de transposition pour une instance du QAP de taille 3. Pour obtenir un schéma plus clair, l'ensemble des solutions S est représenté deux fois. Sur ce schéma, deux solutions voisines sont reliées par un segment.*

On définit $\mathcal{R}$, une relation sur l'ensemble des solutions S telle que $s, t \in S$, $s\mathcal{R}t$ si t est voisin de s. $\mathcal{R}$ est appelée relation de voisinage. Pour le QAP, la relation de voisinage couramment utilisée est symétrique : deux permutations sont voisines si une transposition permet de passer de l'une à l'autre (Fig. 2.1). Soit $V(s)$, l'ensemble des voisins de $s \in S$. $V(s)$ est appelé voisinage de s.

$$s \in S\ ,\quad V(s) \;=\; \{t \in S,\, s\mathcal{R}t\}$$

La notion de voisinage peut être étendue au voisin du voisin. On définit alors le voisinage à l'ordre k comme suit :

$$\begin{aligned} s \in S\ ,\quad V_1(s) &= V(s) \\ s \in S\ ,\quad V_k(s) &= V_{k-1}(s) \cup \{t \in S,\, \exists u \in V_{k-1}(s),\, u\mathcal{R}t\} \end{aligned}$$

2.2.1.2 Définition de l'espace de recherche

L'opérateur de voisinage $\mathcal{R}$ étant spécifié, il permet de définir l'espace de recherche G relatif à $\mathcal{R}$ sur l'ensemble S des solutions. L'espace de recherche G est un graphe dont les sommets sont des solutions et dont les arcs connectent les voisins.

Soit E l'ensemble des couples de solutions voisines ; c'est-à-dire, l'ensemble des couples $(s,t) \in S^2$ tels que $t\mathcal{R}s$ (t est voisin de s). On appelle espace de recherche relatif à $\mathcal{R}$ sur S, le graphe $G(S,E)$. La figure Fig. 2.2 montre une représentation d'un espace de recherche pour une instance du QAP de taille 3.

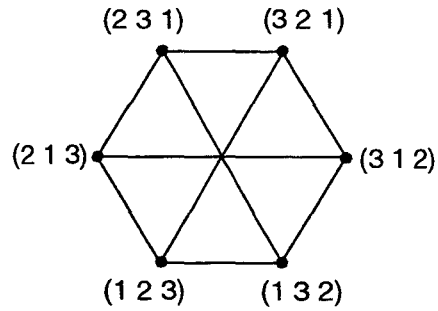


FIG. 2.2 - *Espace de recherche pour toute instance du QAP de taille 3.*

Pour le QAP, l'ensemble S des permutations de taille n, muni de l'opérateur de transposition, constitue un espace de recherche connexe (le graphe $G(S,E)$ est connexe). En effet, étant données deux permutations quelconques, il existe[4] toujours une séquence de transpositions qui permet de transformer l'une en l'autre. De plus, la transposition est un opérateur interne dans S. Cependant, pour certains problèmes, on ne retrouve pas ces bonnes propriétés. Il est fréquent d'utiliser des opérateurs de voisinage qui construisent des configurations n'appartenant pas à l'ensemble des solutions. Ces configurations sont appelés « solutions non réalisables » ; elles sont généralement intégrées à l'ensemble des solutions pour définir un nouvel espace de recherche ayant de bonnes propriétés.

2.2.1.3 Distance dans l'espace de recherche

Dans un espace de recherche, quand $\mathcal{R}$ est symétrique, il est possible de définir une distance. La distance entre deux solutions s et t est alors la longueur du plus court chemin entre s et t dans G. On vérifie aisément que c'est bien une distance au sens mathématique du terme :

- $\text{dist} : S^2 \mapsto \mathbb{N} \cup \{+\infty\}$;
- $\forall s,t \in S$, $\text{dist}(s,t) \geq 0$, par définition ;
- $\forall s,t \in S$, $\text{dist}(s,t) = \text{dist}(t,s)$, car $\mathcal{R}$ est symétrique ;

4. Voir preuve de la section 2.2.1.3

- $\forall s,t,u \in S,\ \text{dist}(s,t) \leq \text{dist}(s,u) + \text{dist}(u,t)$, preuve par l'absurde: soit $\mathcal{C}$ le plus court chemin (p.c.c.) de s à t, de longueur $\text{dist}(s,t)$ par définition, supposons $\text{dist}(s,t) > \text{dist}(s,u) + \text{dist}(u,t)$ alors le chemin de s à t composé du p.c.c. de s à u, suivi du p.c.c. de u à t est strictement plus court que $\mathcal{C}$, ce qui est absurde car $\mathcal{C}$ est le p.c.c. de s à t.

Pour le QAP, alors qu'il y a $n!$ solutions dans l'espace de recherche, toutes les solutions sont à une distance deux à deux d'au maximum $n-1$.

$$\forall s,t \in \Pi_n,\ 0 \leq \text{dist}(s,t) \leq n-1$$

Preuve:

$\forall\, s,t \in \Pi_n,\ s \neq t \quad \exists\, u \in V(s),$
$\exists\, j \in [1,n] - \{i,\ i \in [1,n],\ s(i) = t(i)\}, \quad u(j) = t(j)$

Pour transformer s en t, il existe toujours une transposition qui ajoute un élément à sa bonne place dans t. On en déduit immédiatement que la distance maximale est $n-1$, puisqu'il y a, dans le pire des cas, $n-1$ éléments à déplacer.

Pour calculer la distance entre deux permutations s et t de taille n, nous proposons un algorithme en $O(n)$ (voir Fig. 2.3). Le principe de cet algorithme est de construire le chemin le plus court entre s et t.

2.2.1.4 Diamètre de l'espace de recherche

On appelle diamètre d'une population P de S, la distance maximale entre deux éléments de P.

$$\text{diam}(P) = \max_{s,t \in P} \text{dist}(s,t)$$

Pour le QAP, $\text{diam}(\Pi_n) = n-1$. On en déduit que:

$$\forall u \in \Pi_n,\ V_{n-1}(u) = \Pi_n$$

Pour un usage ultérieur, nous définissons, pour une population P, son diamètre moyen $\text{dmm}(P)$ et son diamètre moyen relatif $\text{Dmm}(P)$:

$$\text{dmm}(P) = \frac{\sum_{s \in P} \sum_{t \in P} \text{dist}(s,t)}{(\text{card}P)^2} \qquad \text{Dmm}(P) = 100 \times \frac{\text{dmm}(P)}{n-1}$$

Le diamètre moyen (relatif) caractérise la concentration de la population P dans l'espace de recherche. Notamment, un diamètre faible indique que les points de la population P sont regroupés dans une région de l'espace.

```
Début
u ← s
Pour i de 1 à n
  faire
    v(t(i)) ← i
distance ← 0
Pour i de 1 à n
  faire
    Tant que v(u(i)) ≠ i
      faire
        change ← u(v(u(i)))
        u(v(u(i))) ← u(i)
        u(i) ← change
        distance ← distance + 1
Fin
```

FIG. 2.3 - *Algorithme de calcul de la distance entre deux solutions s et t en $O(n)$. Le principe de cet algorithme est de construire le chemin le plus court dans l'espace de recherche entre s et t.*

2.2.1.5 Dispersion dans l'espace de recherche

La notion d'entropie en optimisation combinatoire est très voisine de la notion de diversité. En fonction des problèmes traités, on rencontre différentes expressions mathématiques de l'entropie qui mesure la dispersion d'une population dans l'espace de recherche.

Pour le QAP, étant donné que le nombre d'objets aux mêmes emplacements dans deux solutions est fortement corrélé à la distance entre elles, nous utilisons une entropie basée sur le placement des objets, définie dans [Fleurent et al.94] par :

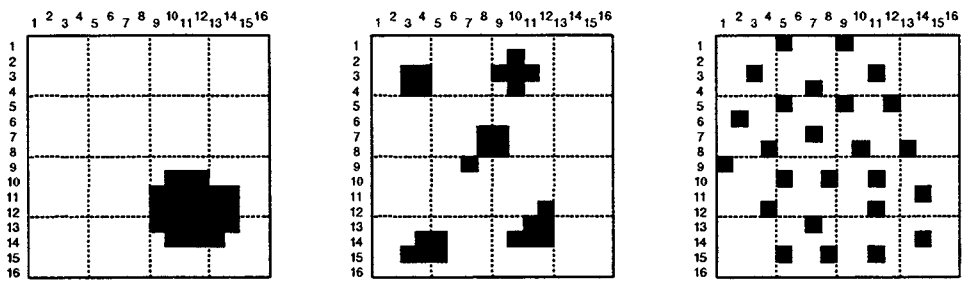
$$\text{ent}(P) = \frac{-1}{n \log n} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\frac{n_{ij}}{\text{card } P} \log \frac{n_{ij}}{\text{card } P} \right) \qquad \text{Ent}(P) = 100 \times \text{ent}(P)$$

où n_{ij} est le nombre d'affectations de l'objet i sur le site j dans la population P. Une entropie faible révèle des concentrations de solutions, alors qu'une entropie élevée témoigne d'une dispersion élevée. Par exemple, une population de 1000 solutions de taille 100, uniformément répartie dans l'espace de recherche, a une entropie de 98 %. À l'inverse, une population dans laquelle toutes les solutions sont identiques a une entropie de 0 %.

2.2.1.6 À propos du diamètre et de l'entropie

Bien que l'entropie et le diamètre moyen soient tous les deux des indicateurs de dispersion, les informations qu'ils fournissent sont complémentaires. Pour illustrer ces deux notions, nous avons calculé le diamètre et l'entropie pour plusieurs distributions de 25 points sur une grille carrée de taille 16 (voir Fig. 2.4) :

- une distribution groupée en un seul paquet (centrale).
- une distribution par petits paquets (en îlots) ;
- et une distribution répartie sur la grille ;



	distribution centrale	*distribution en îlots*	*distribution répartie*
Diamètre :	3	9	9
Entropie :	1	2	3

FIG. 2.4 - *Illustration des notions de diamètre et d'entropie. 25 points sont disposés selon trois distributions particulières (centrale, en îlot, et répartie) sur une grille carrée de taille 16. La table indique les valeurs des diamètres et des entropies.*

Le diamètre que nous avons considéré est la moyenne des distances de Manhattan entre chaque couple de points. Pour définir une entropie, nous avons divisé la grille 16/16 en 16 carrés de taille 4 (voir Fig. 2.4) et avons compté le nombre de points dans chacun de ces carrés. L'entropie que nous avons utilisée est la somme sur l'ensemble des carrés 4/4 des $x \log x$, où x désigne la contribution d'un carré. Calculs faits, (voir Fig. 2.4), on constate que le diamètre caractérise bien la distribution centrale, mais il ne permet pas de distinguer entre la distribution en îlots et la distribution répartie. Par ailleurs, l'entropie fournit des valeurs bien distinctes pour les trois configurations.

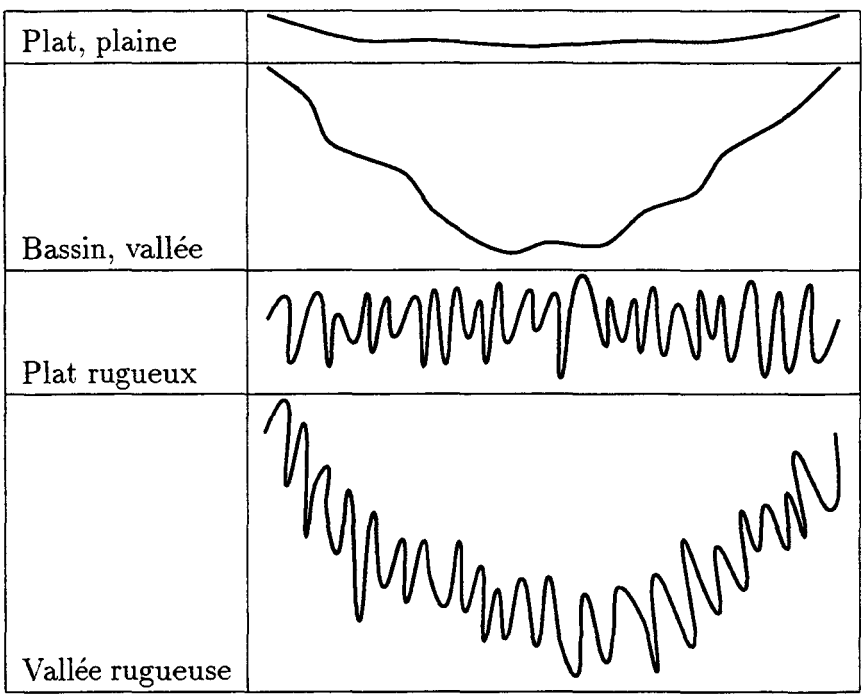
Ainsi, L'entropie et le diamètre fournissent des informations complémentaires sur la distribution. En effet, l'entropie apparaît comme une mesure de la granularité de la distribution, elle est un indicateur de la taille des regroupements. En outre, le

diamètre donne une mesure de la répartition des groupements dans l'espace, il est un indicateur de concentration centrale.

2.2.2 Le paysage d'une instance

2.2.2.1 Définition et métaphore géographique

Le paysage d'une instance d'un problème d'optimisation combinatoire est défini par l'association de son espace de recherche et de sa fonction coût : le couple (G, φ) est le paysage d'une instance. Cette notion de paysage est fortement apparentée à celle des paysages d'énergie connue en physique statistique.



TAB. 2.1 - *Quelques exemples de représentation des paysages selon la métaphore géographique pour un espace de recherche à une dimension.*

D'un point de vue pratique, il est commode et plutôt naturel de qualifier tout ou partie d'un paysage à l'aide de termes relatifs à la description de paysages tels que : vallée, plaine, pic, canyon, bassin, ... (voir Tab. 2.1). En outre, cette métaphore des paysages géographiques permet de reformuler le problème : « Trouver la solution de plus faible coût ? » en : « Quel est l'endroit de plus basse altitude dans le paysage ? ».

Cependant, nous ne savons pas si cette métaphore est pertinente en soi ; mais, nous n'en discutons pas ici la validité. Nous retenons seulement qu'il est très utile de se représenter une instance comme un paysage en une, deux ou trois dimensions.

Néanmoins, nous n'oublions pas que cette métaphore n'est qu'une image qui, bien qu'étant un guide intuitif utile, n'en reste pas moins une représentation erronée, source d'erreurs. Aussi, nous restons prudents quant à son interprétation.

2.2.2.2 Notions d'optimalité

Dans un paysage, un point qui est le plus bas dans son voisinage immédiat est qualifié d'optimum local et le point le plus bas de tout le paysage est appelé optimum global. La figure 2.5 illustre cette notion d'optimum.

$$s \text{ est un optimum local} \iff \forall t \in V(s),\ \varphi(t) \leq \varphi(s)$$

$$s \text{ est un optimum global} \iff \forall t \in S,\ \varphi(t) \leq \varphi(s)$$

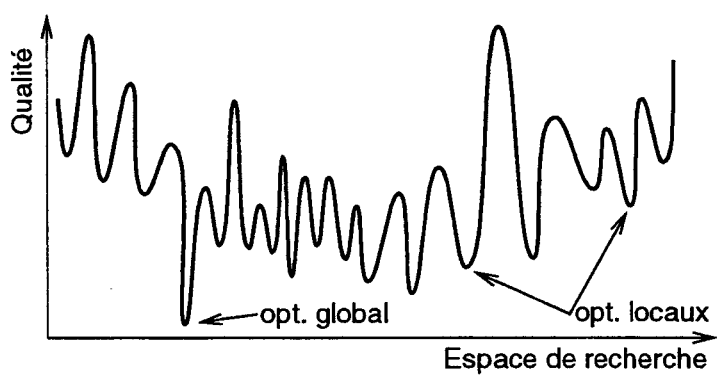


FIG. 2.5 - *Illustration des notions d'optima locaux et d'optimum global.*

2.2.2.3 Intérêt des paysages

Depuis quelque temps, la communauté de l'optimisation combinatoire s'intéresse de près aux paysages des instances. Une raison probable à cet engouement est l'émergence de nouvelles méthodes d'optimisation performantes basées sur le cheminement dans les paysages, comme, par exemple, la méthode tabou. En effet, mieux comprendre la structure des paysages, c'est mieux comprendre pourquoi ces méthodes sont efficaces et c'est mieux savoir comment les améliorer. Dans cet esprit, Hertz *et al.* étudient le nombre d'optima locaux et leur distribution au sein de vallées pour le problème de coloriage de graphes ; ce travail leur permet de proposer des améliorations de l'algorithme tabou pour ce problème [Hertz et al.94]. Pour

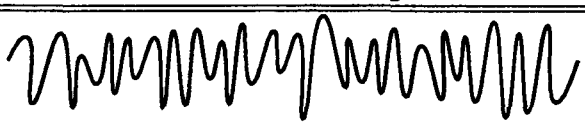
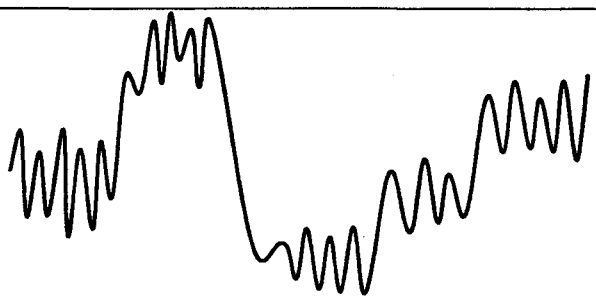
le problème du voyageur de commerce (TSP), on peut citer les travaux de Boese et de Kirkpatrick et Toulouse [Boese96] [Kirkpatrick et al.85]. Ces derniers, bien qu'ayant des approches différentes, parviennent à la même conclusion : l'allure du paysage de certaines instances est une vallée profonde (*big valley*). D'autres études, comme celles de Stadler ou de Weinberger, ont une approche formelle des paysages, en dehors de tout problème d'optimisation [Stadler95] [Weinberger91]. Cependant, ces études sont difficilement exploitables en pratique ; elles ne sont pas directement utilisables pour concevoir ou améliorer des techniques d'optimisation.

D'un point de vue pragmatique, pour l'optimisation par une méthode de cheminement dans le paysage, une instance facile est une instance dont le paysage est une vallée (voir Tab. 2.1) ; à l'inverse, une instance difficile est une instance dont le paysage est une plaine rugueuse. En effet, les méthodes de recherche parviennent facilement à découvrir le fond d'une vallée et il suffit alors d'intensifier la recherche dans cette région. En ce qui concerne les paysages plats et rugueux, il s'y trouve de très nombreux endroits de bonne qualité, mais ces derniers sont très dispersés et il devient difficile d'atteindre le meilleur ; on peut diversifier beaucoup mais ce n'est pas suffisant pour de grands espaces. Quand le paysage est intermédiaire, la méthode de recherche doit tenter d'équilibrer l'intensification et la diversification, mais ce n'est pas chose aisée. Fort de ces constatations, une approche prometteuse pour la résolution de problèmes combinatoires, consiste à modifier la structure du paysage pour qu'il devienne plus facile, c'est à dire formé d'un petit nombre de vallées bien profondes. Afin de modifier la structure du paysage, on ne peut agir que sur l'opérateur de voisinage ou sur la fonction coût. Dans le groupe de recherche PERFORM[5], dont l'objectif majeur est la résolution de problèmes d'optimisation combinatoire, nos travaux portent sur le TSP, le JSP (*Job Shop Scheduling Problem*) et le QAP. Pour le TSP, nous avons comparé les paysages bâtis à partir de deux opérateurs de voisinage différents : le *city-swap* et le 2-opt. Nous avons constaté que le paysage du 2-opt est bien meilleur[6] que l'autre. Le paysage du 2-opt est un « Massif Central », c'est à dire que c'est une vallée au fond de laquelle on trouve les optima locaux autour de l'optimum global, d'autant plus proches du centre qu'ils sont de bonne qualité. Ainsi, on constate que dans ce paysage, une simple méthode de descente permet d'obtenir un optimum de bonne qualité. Ce travail a permis la conception d'une heuristique performante [Fonlupt et al.97]. En ce qui concerne le JSP, nous avons découvert un paysage constitué de grands plateaux dont la traversée est fastidieuse et handicapante pour les méthodes de recherche. Nous avons proposé une fonction coût adaptée qui « gomme » ces plateaux et rend le paysage praticable [Duvivier et al.98] [Duvivier99]. Pour le QAP, notre approche est différente, nous n'avons pas entrevu de piste prometteuse basée sur la modification du paysage et nous nous sommes plutôt attachés à adapter les heuristiques à la structure du paysage.

5. Le groupe PERFORM est présenté dans l'introduction de ce mémoire.
6. Il est plus facile.

2.3 Étude de la structure du paysage du QAP

La structure d'une instance apparaît dans la bibliographie consacrée à l'optimisation comme une notion assez vague ; elle y est rarement explicitée. Souvent la notion de structure est assimilée à la complexité de l'instance. En effet, une heuristique tente d'appréhender une structure éventuelle de l'instance ; ainsi une instance peu structurée apparaît complexe pour une heuristique ; inversement, une instance structurée est plus facile. Pour notre part, nous définissons la structure d'une instance par rapport à son paysage. Ainsi, une instance structurée est une instance dont le paysage est constitué de régions de grande taille à des niveaux bien distincts. Au contraire, une instance peu structurée possède un paysage plutôt plat et rugueux (voir Fig. 2.2).

Instance	Allure du paysage
peu structurée	
structurée	

TAB. 2.2 - *Illustration de la notion de structure d'une instance par l'allure de son paysage.*

De nombreuses études ont été menées avec l'objectif de fournir un indicateur qui mesure la structure d'une instance. Ces études se distinguent selon deux types d'approches : globale ou locale. L'approche globale donne une information sur la globalité du paysage ; à l'inverse, l'approche locale consiste à caractériser la structure du paysage le long du cheminement d'une heuristique.

2.3.1 Approche globale du paysage

Pour l'approche globale, l'indicateur de structure est calculé à partir des données qui définissent l'instance. C'est le cas de la dominance de flux proposée par Vollmann et Buffa, de la fonction d'autocorrélation de Weinberger et du coefficient de rugosité défini par Angel et Zissimopoulos [Vollman et al.66] [Weinberger90] [Angel et al.96].

2.3.1.1 La dominance de flux

Pour le QAP, Vollmann et Buffa ont introduit la notion de dominance de flux [Vollman et al.66]. La dominance de flux est utilisée comme un indicateur de la complexité d'une instance. Il s'agit du coefficient normalisé fd de la matrice c des coûts de flux (100 fois l'écart type divisé par la moyenne).

$$fd(c) = 100 \times \frac{a}{b} \qquad a = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ij}-b)^2}{n^2}} \qquad b = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}}{n^2}$$

Ainsi, selon Vollmann et Buffa, une dominance élevée (plus de 200) caractérise une instance plutôt facile alors qu'une dominance faible caractérise une instance plus complexe. En effet, la dominance de flux caractérisant l'existence de coefficients prépondérants dans la matrice de flux, on imagine que ces valeurs dominantes produisent des accidents dans le paysage et le rendent, par là même, plus structuré donc plus facile. À l'inverse, une dominance faible indique une uniformité des coefficients de flux, et correspond à un paysage uniforme, peu structuré, donc plus difficile.

Cependant, plusieurs études critiquent la dominance de flux et proposent de nouveaux indicateurs du même type. Herroelen et Van Gils, présentent une synthèse de ces travaux et critiquent l'utilisation de la notion de complexité par la dominance [Herroellen et al.85]. Ils montrent que ces indicateurs ne fonctionnent pas très bien et recommandent beaucoup de prudence dans leur interprétation.

Pour notre part, bien que la dominance de flux ait été fortement critiquée en tant qu'indicateur de complexité d'une instance, nous ne l'avons pas rejetée. Nous considérons la dominance de flux comme une mesure à notre disposition sans la surestimer. En effet, nous avons constaté une certaine corrélation entre la fd et la « nature » intrinsèque des instances. La notion de nature d'une instance fait référence à une taxinomie des instances basées sur l'origine de l'instance. Cependant qu'il existe de nombreuses natures d'instances distinctes (voir Tab. 2.3), on distingue usuellement trois grandes classes :

- les instances de natures uniformes - ces instances sont produites par des algorithmes de génération uniforme de nombres pseudo-aléatoires. Ainsi les matrices qui définissent l'instance ont des coefficients uniformes ;
- les instances de nature grille - pour ces instances, les emplacements constituent une grille en 2 ou 3 dimensions. Il en résulte une structure particulière de la matrice des distances ;
- les instances réelles - ce sont des instances qui modélisent des problèmes réels (*e.g.* industriels). En général, ces instances ont une structure fortement marquée.

Instance		Nature		
Nom	Taille	Distance	Flux	fd/10
Lipa50a	50	a.n.	a.n.	4
Lipa90a	90	a.n.	a.n.	4
Esc128	128	réel	réel	5
Had20	20	grille	uniforme	5
Tai25a	25	uniforme	uniforme	6
Tai60a	60	uniforme	uniforme	6
Tai80a	80	uniforme	uniforme	6
Tai100a	100	uniforme	uniforme	6
Wil100	100	grille	aléatoire	6
Chr25a	25	a.n.	a.n.	6
Sko64	64	grille	aléatoire	11
Sko81	81	grille	aléatoire	11
Nug30	30	grille	aléatoire	11
Sko100a	100	grille	aléatoire	11
Tai64c	64	réel	réel	13
Tho150	150	grille	a.n.	15
Tai256c	256	réel	réel	22
Bur26d	26	réel	réel	23
Tai35b	35	pseudo-réel	pseudo-réel	31
Tai40b	40	pseudo-réel	pseudo-réel	32
Tai60b	60	pseudo-réel	pseudo-réel	32
Tai80b	80	pseudo-réel	pseudo-réel	32
Tai100b	100	pseudo-réel	pseudo-réel	32
Ste36c	36	réel	réel	40
Els19	19	réel	réel	53

TAB. 2.3 - *Jeu d'instances de différentes natures pour notre étude du paysage du QAP. Les instances sont ordonnées par dominance de flux croissante. On constate une corrélation entre la nature de l'instance et la dominance de flux (fd) : les instances uniformes ont une dominance faible alors que les instances réelles ont une fd élevée. Par exemple, les instances Taixxa, en haut de tableau (petite fd), sont des instances générées aléatoirement de façon uniforme [Taillard91] ; à l'opposé, le bas du tableau (grande fd) présente des instances réelles : Ste36c [Steinberg61] et Els19 [Elshafei77], ou pseudo-réelles : Taixxb générées aléatoirement mais présentant des caractéristiques proches des problèmes réels [Taillard93b].* *Légende des natures :* uniforme = *génération aléatoire uniforme,* aléatoire = *génération aléatoire,* grille = *distances sur une grille euclidienne,* réel = *problème réel,* pseudo-réel = *génération aléatoire imitant la nature réelle,* a.n. = *autre nature, voir la QAPlib pour plus d'information [Burkard et al.91] - par exemple, les instances Lipaxxa sont obtenues par génération de problèmes dont on connaît les solutions optimales [Li et al.92].*

Le tableau 2.3 montre quelques instances du QAP, de nature diverses, ordonnées par dominance de flux. Hormis quelques exceptions, on observe un coefficient de dominance de flux (fd) plutôt faible pour les instances de nature uniforme et une fd plutôt élevée pour les instances de nature réelle. Cette corrélation se résume dans le tableau suivant :

nature	dominance fd
uniforme	faible
grille	moyenne
réelle	élevée

Par ailleurs, nous comprenons mal pourquoi la dominance de distance n'est pas prise en compte[7] dans l'évaluation de la complexité d'une instance, étant donné le rôle symétrique des deux matrices, distances et flux. Aussi, nous considérons, en association avec la dominance flux, la dominance de distance dd comme mesure de la structure d'une instance.

$$dd(d) = 100 \times \frac{a'}{b'} \qquad a' = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(d_{ij}-m)^2}{n^2}} \qquad b' = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}}{n^2}$$

L'association de fd et dd permet de regrouper les instances par famille et constitue une taxinomie des instances du QAP. La figure 2.6 montre la répartition des instances du tableau 2.3 en fonction de leurs dominances de flux et de distance. On perçoit nettement que les instances d'une même famille y sont regroupées : les Skoxx sont ensemble, les Taixxa de même et les Taixxb aussi. On remarque aussi quelques instances disparates : Els19 est très à droite avec une forte fd, Chr25a et Tai64c se distinguent par une dd élevée alors que Tai256c est isolée avec une fd et une dd toutes deux supérieures à 200.

2.3.1.2 Le coefficient de rugosité du paysage

Weinberger propose la fonction d'autocorrélation $\rho(d)$ qui définit la rugosité d'un paysage [Weinberger90].

$$\rho(d) = 1 - \frac{\text{moy}_{(s,t)\in S^2,\, dist(s,t)=d}\,[(\varphi(s)-\varphi(t))^2]}{\text{moy}_{(s,t)\in S^2}\,[(\varphi(s)-\varphi(t))^2]}$$

Cette fonction ne contrarie en rien le sens naturel qu'on entend de la rugosité d'une surface. En effet, la fonction d'autocorrélation mesure l'influence de la distance d sur

7. Probablement que les instances traitées par Volmann et Buffa, pour lesquelles a été conçu l'indicateur fd présentaient une distribution de distances plutôt uniforme (distances sur une grille 2D) ; les matrices de distances étaient plutôt semblables rendant leur comparaison stérile. C'est pourquoi la matrice de distance n'a pas été prise en compte pour évaluer la complexité de l'instance.

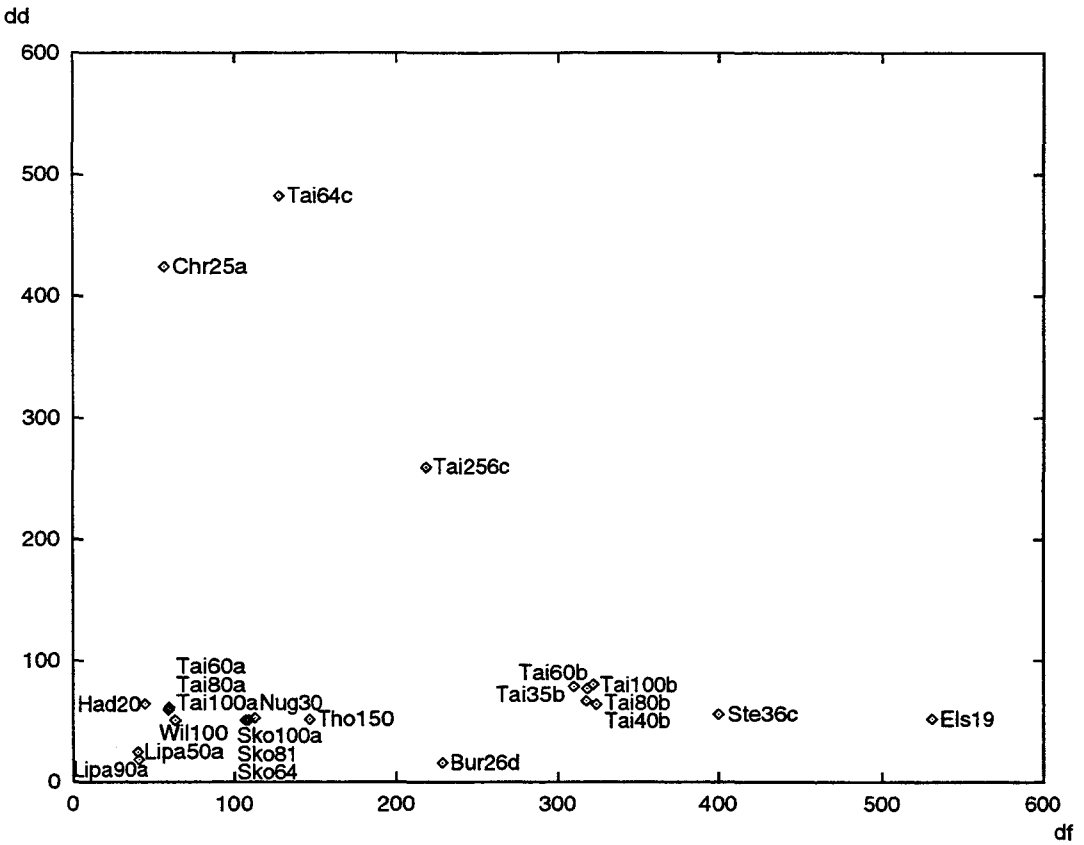


FIG. 2.6 - *Classement des instances par leurs dominances de flux et de distance. On perçoit nettement les regroupements des instances d'une même famille.*

les écarts de *fitness* des solutions. Considérons le cas des solutions immédiatement voisines ($d = 1$) : si $\rho(1)$ est proche de 0, cela signifie qu'en moyenne, deux solutions voisines n'ont pas des qualités plus proches que si elles étaient éloignées, on a alors un paysage plutôt rugueux ; à l'inverse, si $\rho(1)$ est proche de 1, alors deux solutions voisines ont en moyenne un coût plus proche que si elle étaient distantes, on a donc un paysage plus lisse.

À partir de la fonction d'autocorrélation de Weinberger pour le voisinage immédiat $\rho(1)$, Angel et Zissimopoulos ont défini un coefficient normalisé qui exprime la rugosité du paysage d'une instance du QAP [Angel et al.96]. Le coefficient de rugosité ζ évalue globalement l'homogénéité de la qualité du voisinage immédiat d'une solution.

$$\zeta = 100 - \frac{400}{n}(\xi - \frac{n}{4}) \qquad \xi = \frac{1}{1-\rho(1)}$$

Le coefficient ζ varie[8] de 0 à 100. Un ζ proche de 0 qualifie un paysage très doux

8. En supposant que $\xi \leq \frac{n}{2}$, vérifié expérimentalement par les auteurs.

alors qu'un ζ voisin de 100 correspond à un paysage rugueux.

2.3.1.3 Les limites de l'approche globale

Ces approches, dominance ou coefficient de rugosité, sont dites globales parce qu'elles s'intéressent au paysage dans sa globalité. Elle déterminent la structure d'un paysage par rapport à l'ensemble des solutions de l'espace de recherche. Cependant, en optimisation, on recherche les solutions de bonne qualité ; or, ces solutions ne représentent qu'une infime partie de l'espace de recherche. Lorsqu'on s'intéressent au fonctionnement des méthodes qui cheminent dans ces paysages (telles que les méthodes de descente), l'approche globale est insuffisante. En effet, ces heuristiques ont une vision locale du paysage, selon un ou plusieurs chemins dans l'espace de recherche ; elles focalisent sur les bonnes solutions. Il serait intéressant d'avoir une description de ces bonnes solutions et des régions ou elles se trouvent ; mais l'approche globale ne répond pas à ces questions car elle donne une information moyenne sur un trop grand nombre de points qui ne concernent pas l'heuristique.

Ainsi, l'approche globale de la structure des paysages, du fait de sa nature (globale), n'est pas adaptée à l'étude des heuristiques locales et par là même ne peut apporter que peu à la compréhension du fonctionnement de ces méthodes d'optimisation. C'est pourquoi nous n'avons pas développé cette approche dans notre étude de la structure des paysages du QAP.

2.3.2 Approche locale du paysage

Cette section présente une étude statistique du paysage d'instances du QAP. Pour étudier ces paysages, nous avons choisi une approche locale qui consiste à observer le paysage à travers le parcours d'une heuristique itérative simple, la recherche locale (LS – *Local Search*). Bien que cette étude présente une certaine originalité, notamment dans la proposition de nouveaux indicateurs de structure, elle est inspirée de travaux d'analyse locale des paysages du TSP, réalisés au sein de PERFORM notre groupe de recherche [Fonlupt et al.97] ou par ailleurs [Boese96].

2.3.2.1 Protocole expérimental

Pour notre étude, nous avons sélectionné des instances dans la bibliothèque QAPLIB [Burkard et al.91]. Cette bibliothèque apparaît comme un standard car elle regroupe les instances sur lesquelles portent la plupart des études récentes sur le QAP. Les instances que nous avons choisies (voir Tab. 2.3) sont différentes tant en taille qu'en nature. Elles constituent un échantillon qui représente la diversité de la QAPLIB de façon significative.

Pour chacune des instances sélectionnées, une population U de $p = 1000$ solutions est générée aléatoirement selon une distribution uniforme dans l'espace de recherche $G(V, E)$. Puis, avec chaque solution de U, on démarre une marche selon la plus forte pente (SDW – *Steepest Descend Walk*). La SDW est une méthode qui parcourt l'espace de recherche G. C'est une recherche locale (LS). Elle débute sur un sommet du graphe (ici une solution de U), et tant qu'il existe un sommet voisin de meilleure qualité, elle progresse vers le meilleur des sommets adjacents. Ainsi la SDW termine sur un optimum local (les méthodes de descentes sont décrites à la section II:1.1.2). À l'issue des marches, nous avons obtenu, pour chaque instance, une population O de p optima locaux, correspondant à la population uniforme initiale U. Au cours des marches, nous avons relevé certains indicateurs dont l'analyse nous a permis de caractériser les paysages des instances étudiées. Les résultats obtenus sont présentés dans le tableau 2.4 que nous commentons dans la suite de cette section.

Instance	Ent (U)	Δ_{Ent}	Dmm (U)	Δ_{Dmm}	Amp (U)	Δ_{Amp}	Gap (O)	Lmm (U)
Tai64c	99	0	93	0	80	4	0	11
Esc128	98	0	96	0	43	36	12	16
Tai256c	97	0	97	0	13	0	0	18
Tai25a	99	8	88	5	223	5	4	52
Lipa50a	99	1	92	0	0	0	1	56
Tai60a	99	8	93	0	5	3	4	58
Lipa90a	98	0	95	0	0	0	0	59
Tai80a	98	0	94	0	3	2	3	60
Tai100a	98	0	95	0	3	2	2	61
Chr25a	99	7	88	3	79	68	57	64
Had20	99	27	86	13	9	4	1	75
Bur26d	99	18	88	8	11	2	0	77
Nug30	99	11	89	4	17	8	4	77
Els19	99	18	85	9	94	82	26	79
Ste36c	99	14	90	4	55	22	9	89
Tai35b	99	9	90	4	40	21	8	94
Tai40b	99	9	91	3	30	22	10	100
Sko64	99	8	93	1	7	3	2	100
Sko81	98	9	94	2	5	2	2	109
Tai60b	99	19	93	3	23	16	8	118
Wil100	98	11	95	2	2	1	1	119
Sko100a	98	9	95	1	4	2	2	119
Tai80b	98	9	94	2	17	9	6	126
Tai100b	98	12	95	3	19	10	5	138
Tho150	98	9	96	1	4	2	2	139

TAB. 2.4 - *Tables des indicateurs relevés aux cours des 1000 marches de descentes pour chaque instance. Les instances sont classées par Lmm croissante.*

Les indicateurs retenus pour l'analyse des marches sont :

- $\mathrm{Ent}(U)$: l'entropie[9] de la population uniforme U des solutions de départ ;

- $\Delta_{\mathrm{Ent}} = \mathrm{Ent}(U) - \mathrm{Ent}(O)$: la variation d'entropie entre la population de départ U et celle des optima locaux O obtenus à la fin des marches SDW ;

- $\mathrm{Dmm}(U)$: le diamètre moyen relatif[10] de la population initiale U ;

- $\Delta_{\mathrm{Dmm}} = \mathrm{Dmm}(U) - \mathrm{Dmm}(O)$: la variation du diamètre moyen relatif entre la population uniforme U et la population des optima locaux O ;

- $\mathrm{Amp}(U)$: l'amplitude de la population initiale U. L'amplitude $\mathrm{Amp}(P)$ d'une population P quelconque de solutions étant l'écart relatif entre la qualité de la meilleure solution de P et la qualité de la pire solution de P ;

$$\mathrm{Amp}(P) = 100 \times \frac{\mathrm{card}\, P \,.\, (\max_{s \in P} \varphi(s) - \min_{s \in P} \varphi(s))}{\sum_{s \in P} \varphi(s)}$$

- Δ_{Amp} : la variation relative d'amplitude entre la population initiale U et la population d'arrivée O ;

$$\Delta_{\mathrm{Amp}} = \frac{\mathrm{Amp}(U) - \mathrm{Amp}(O)}{\mathrm{Amp}(U)}$$

- $\mathrm{Gap}(O)$: la moyenne des écarts relatifs des coûts des optima locaux de la population O par rapport à la meilleure solution connue π_* ;

$$\mathrm{Gap}(O) = 100 \times \frac{\sum_{s \in O} (\varphi(s) - \varphi(\pi_*))}{\mathrm{card}\, O \,.\, \varphi(\pi_*)}$$

- $\mathrm{Lmm}(U)$: la longueur de marche moyenne relative de la population U par les méthodes de descentes. Pour toute population P de solutions,

$$\mathrm{Lmm}(P) = 100 \times \frac{\mathrm{lmm}(P)}{n}$$

où $\mathrm{lmm}(P)$ est la moyenne des longueurs des marches SDW commençant avec les solutions de P.

9. L'entropie est définie à la section 2.2.1.5.
10. Le diamètre moyen est défini à la section 2.2.1.4.

2.3.2.2 La population uniforme

La diversité (entropie Ent (U)) de la population uniforme U décroît légèrement lorsque la taille n de l'instance augmente. On a une entropie de 99 pour les instances de tailles inférieures à 80, puis on descend à 98 pour les grandes instances et 97 pour Tai256c. Cela provient du fait que p est constant et qu'il devient petit par rapport à card $S = n!$ lorsque n augmente. Cependant, l'entropie demeure élevée (Ent $(U) \geq 97$) quelle que soit l'instance, ce qui montre bien la diversité de la population uniforme.

La variation du diamètre moyen en fonction de la taille de l'instance est bien marquée. On voit (Fig. 2.7) que Dmm (U) croît fortement avec n. Mais, rien ne nous permet d'affirmer que la variation de Dmm (U) est une bonne approximation de la variation de Dmm (S) car p est petit par rapport à $n!$ pour les grandes instances. Cependant, la variation est intéressante et mérite d'être soulignée.

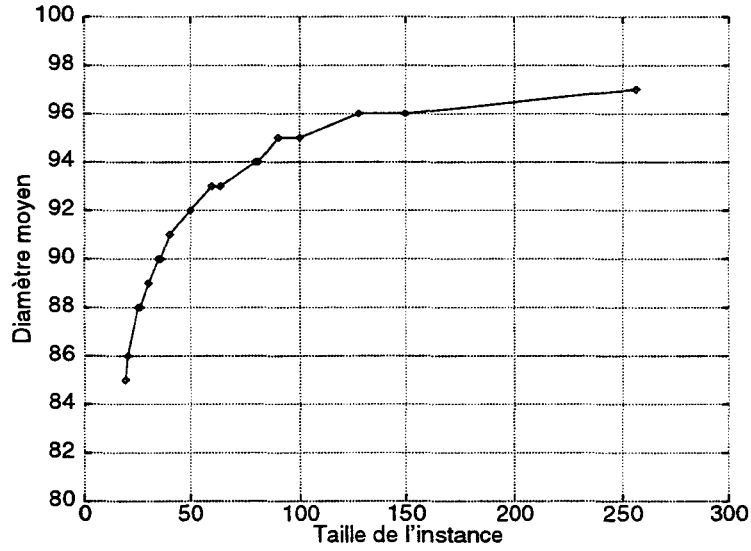


FIG. 2.7 - *Variation du diamètre moyen relatif Dmm (U) en fonction de n la taille de l'instance.*

Intéressons-nous maintenant à la qualité des solutions de la population uniforme. La figure 2.8 montre la répartition des solutions tirées aléatoirement pour l'instance Sko100a[11]. La répartition est donnée en fonction de la distance par rapport à la meilleure solution connue (axe des abscisses) et par rapport à la valeur de la solution (φ sur l'axe des ordonnées). On remarque que toutes les solutions sont regroupées dans le coin supérieur droit. Les solutions sont donc éloignées de la meilleure solution connue et sont de qualité médiocre. D'une façon générale, on retrouve cette situation

11. Des figures similaires, relatives à d'autres instances, sont présentées à l'annexe IV:A

pour toutes les instances testées. Ainsi, on voit bien que la probabilité d'obtenir une solution proche de l'optimum, par un tirage aléatoire uniforme est extrêmement faible. On en déduit immédiatement que la proportion des solutions de bonne qualité est infime.

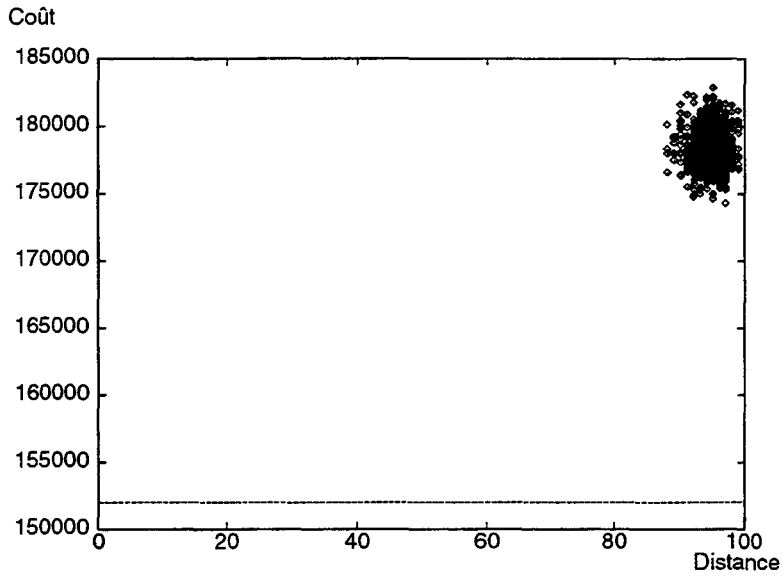


FIG. 2.8 - *Cette figure montre la situation des solutions générées aléatoirement selon une distribution uniforme pour l'instance Sko100a. La distance indiquée est calculée par rapport à la meilleure solution connue, soit o. La ligne en pointillé indique le coût de o. On constate que toutes les solutions sont très éloignées de o, aussi bien en distance qu'en coût.*

Ainsi, une très grande proportion des solutions ont une qualité médiocre. Donc, selon la métaphore des paysages, la majeure partie du paysage constitue un nuage en altitude. De plus, on constate (Tab. 2.4) que l'amplitude $\mathrm{Amp}(U)$ de ce nuage de solutions est assez faible; ce nuage peut donc être qualifié de couche. Ainsi, la majorité des solutions du QAP constitue une couche en altitude, plus ou moins fine.

2.3.2.3 La distribution des optima locaux

Où conduit la méthode de descente ?

La figure 2.9, relative à l'instance Nug30 [12], montre simultanément la distribution uniforme U et la distribution des optima locaux O obtenue par les marches SDW. On observe deux nuages bien distincts, la distribution uniforme est en haut à droite, les optima locaux sont en bas à droite. On note que le nuage O est moins épais que le

12. Des figures similaires, relatives à d'autres instances, sont présentées à l'annexe IV:A

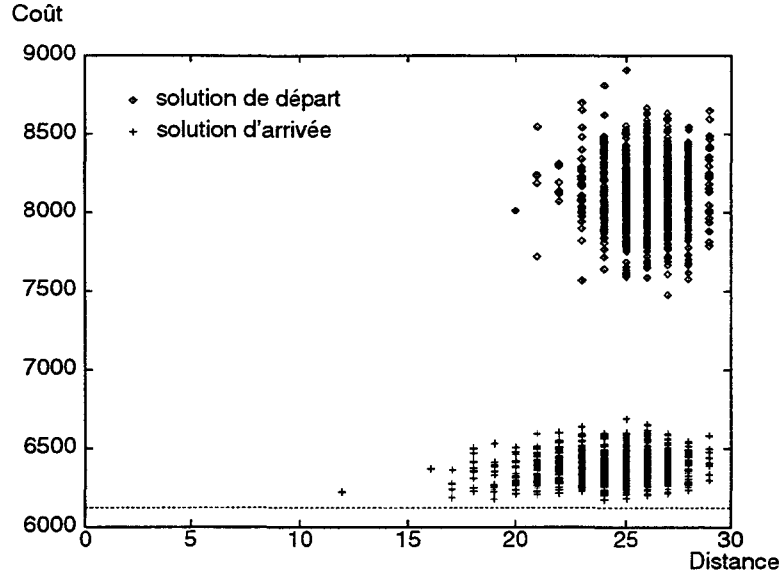


FIG. 2.9 - *Cette figure montre le déplacement d'une distribution uniforme de solutions par une descente suivant la plus forte pente SDW pour l'instance Nug30. La ligne en pointillé indique le coût de la meilleure solution connue. On constate que les points de départ, en haut, et les points d'arrivée, en bas, forment deux nuages bien distincts.*

nuage U (voir les amplitudes dans la table 2.4). On observe nettement que le nuage des optima locaux s'étale et s'amincit vers la gauche, en direction de l'optimum global. Pour toutes les instances de cette étude, nous avons toujours retrouvé cette situation des optima locaux, hormis quelques variations en ce qui concerne la qualité des optima locaux et leur décalage vers la gauche (voir annexe IV:A).

L'étalement du nuage O, pour certaines instances où il est très marqué, peut laisser penser qu'une partie des optima locaux se regroupent autour de la meilleure solution connue utilisée pour la distance. En fait, il y a bien des regroupements d'optima locaux comme nous le montrons plus loin. Regardons, par exemple la distribution des optima locaux pour l'instance Els19 (Fig. 2.10). On voit qu'une bonne part des optima locaux se rapproche de l'optimum global dont une majorité aux distances de 8 à 10. On s'aperçoit aussi que certaines marches atteignent l'optimum global (en bas à gauche sur le graphique).

2.3.2.4 Qualité des optima locaux

On remarque une bonne corrélation entre la variation d'amplitude Δ_{Amp} et la qualité relative moyenne des optima locaux Gap (O) ; la corrélation linéaire produit un coefficient de détermination linéaire de 0.95 (voir figure 2.11). En effet, quand la variation d'amplitude augmente, l'écart de qualité avec la meilleure solution connue augmente aussi, de façon presque linéaire. Autrement dit, un nuage d'optima locaux

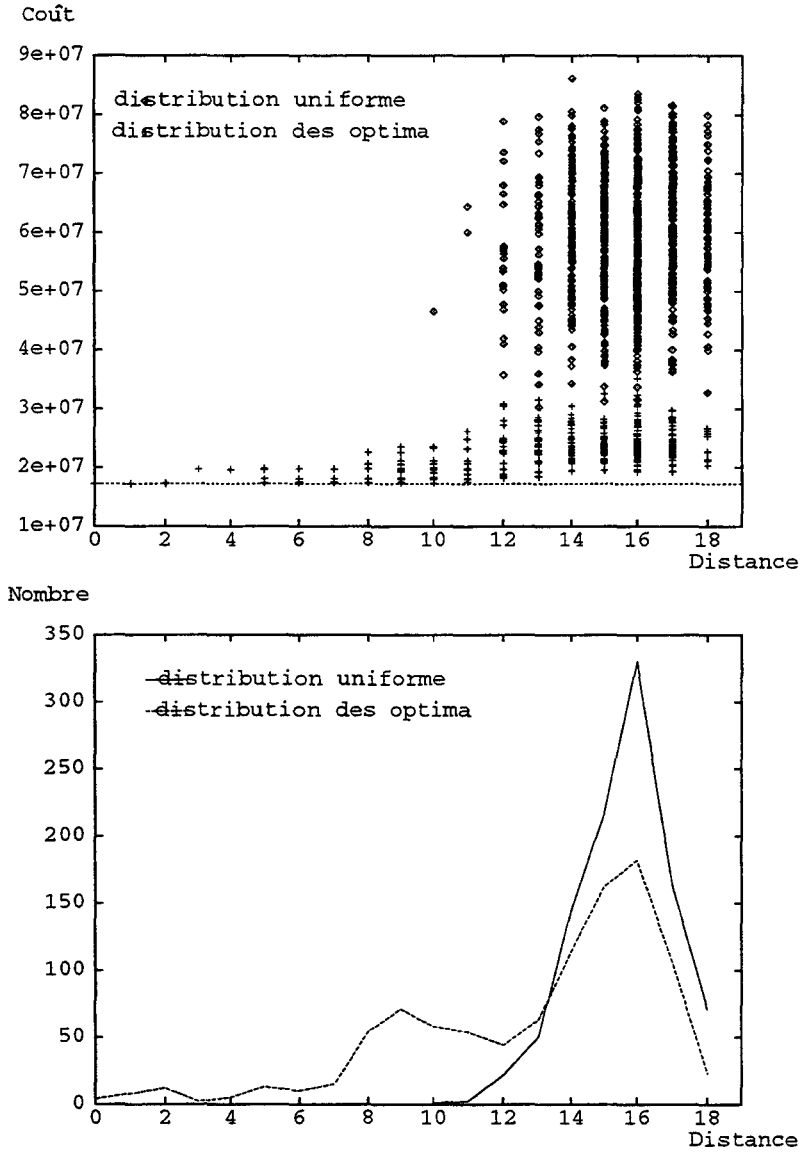


FIG. 2.10 - *Cette figure montre le déplacement d'une distribution uniforme par descente SDW pour l'instance Els19. Le graphique du bas donne le nombre d'optima locaux en fonction de la distance à l'optimum global. On note (figure du bas) une concentration d'optima locaux aux distances de 8 à 10.*

qui est plus mince (relativement à la distribution uniforme qui l'a engendré), est plus proche en qualité de l'optimum global. À l'inverse, à une variation d'amplitude faible correspond un nuage d'optima locaux de bonne qualité, puisque Gap (O) est faible. Par exemple, pour l'instance Lipa50, qui a une Δ_{Amp} à 0, le nuage d'optima locaux est d'une qualité très proche de l'optimum global (qui est connu) puisque l'écart moyen à l'optimum est de 1 % (Gap $(O) = 1$) ; d'ailleurs, la population O contient de nombreuses fois l'optimum global. En se reposant sur cette corrélation, constatée expérimentalement, on peut estimer la qualité de l'optimum global d'une instance. Par exemple, pour les instances Tai100a, Wil100, Tho150, dont les optima globaux ne sont pas connus[13], on peut penser que l'optimum global de chaque instance n'est pas d'une qualité très supérieure à celles des optima locaux puisque la variation d'amplitude de chaque instance est faible ($\Delta_{\text{Amp}} \leq 2$).

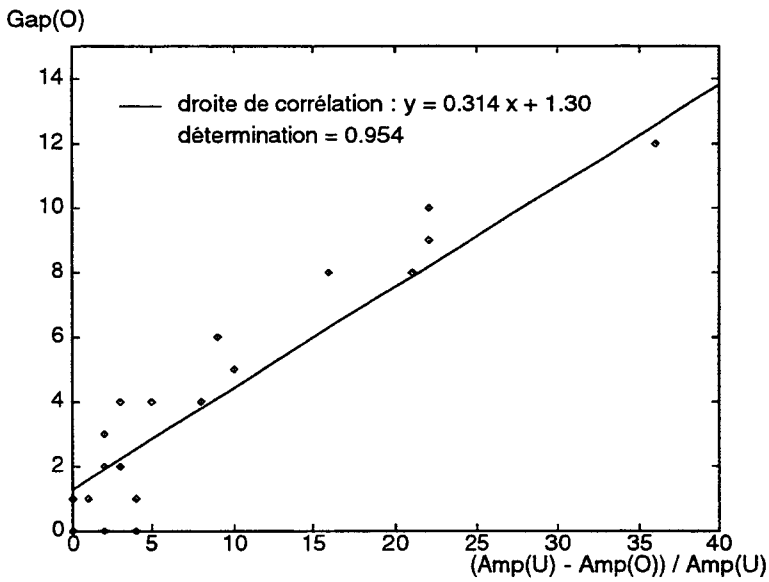


FIG. 2.11 - *Corrélation entre Δ_{Amp}, la variation d'amplitude par la marche SDW (axe des abscisses), et Gap (O), la qualité de la distribution des optima locaux. Les points correspondent aux instances de la table 2.4. Cependant, les instances Els19 et Chr25a ne sont pas représentées car très éloignées des autres. L'instance chr25a, fortement atypique, a été exclue dans le calcul de la corrélation linéaire.*

En rapprochant la nature des instances à leur variation d'amplitude, on observe que les instances uniformes ont plutôt une faible variation alors que les instances réelles en ont une plus élevée (par exemple, $\Delta_{\text{Amp}} = 55$ pour l'instance réelle Ste36c, $\Delta_{\text{Amp}} = 2$ pour l'instance uniforme Tai80a). Sur le plan de la résolution des instances, si on a l'objectif de trouver une bonne solution (objectif pratique), pas né-

13. Si l'optimum global d'une instance a été trouvé par une méthode suboptimale, il n'est pas connu comme tel mais seulement comme une bonne solution. Parfois, ces solutions, qu'on croit optimales sans que cela ait été montré, sont dites pseudo-optimales.

cessairement optimale, les instances à variation d'amplitude faible (uniformes) sont plutôt faciles. En effet, il suffit d'effectuer quelques descentes SDW pour obtenir des solutions de bonne qualité car la qualité dans le nuage des optima locaux est bonne et homogène. Au contraire, si on recherche la meilleure solution, ces instances sont difficiles car il ne peut exister de région privilégiée, identifiable par une heuristique, où les optima sont meilleurs. Pour les instances plus structurées (Δ_{Amp} plus grand), l'approche est différente. Une simple méthode de descente ne suffit pas pour trouver une bonne solution. Il y a différentes qualités d'optima locaux et on peut supposer (comme nous le montrons plus loin) qu'il existe une ou plusieurs régions de l'espace de recherche où les optima de bonne qualité se regroupent. Une bonne méthode de recherche doit trouver ses régions et y intensifier la recherche.

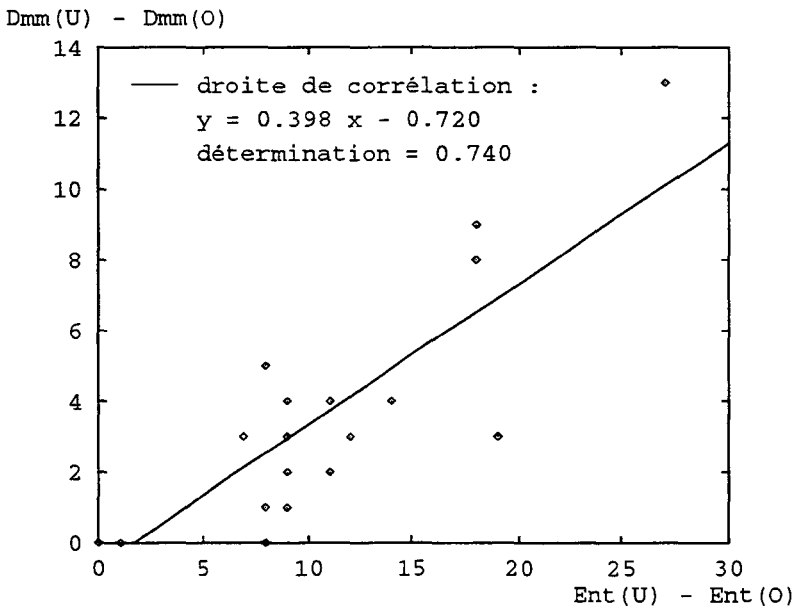
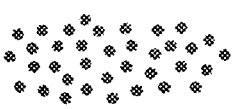


FIG. 2.12 - *Visualisation de la corrélation linéaire entre la variation d'entropie Δ_{Ent} et le diamètre moyen des optima locaux Δ_{Dmm}. Pour certaines instances, l'entropie décroît beaucoup plus que le diamètre moyen; c'est le signe de petits regroupements distants.*

2.3.2.5 Dispersion des optima locaux

Afin d'évaluer la dispersion des solutions obtenues par les descentes SDW, nous proposons de calculer leur entropie et leur diamètre moyen. L'entropie informe sur la diversité structurelle des solutions, elle permet de savoir si les points sont regroupés ou dispersés. Cependant, l'entropie ne renseigne pas sur le nombre de regroupements ; en effet, des regroupements multiples ou un regroupement unique donnent, dans les deux cas, une entropie faible (moins de diversité). Le diamètre moyen apporte une information complémentaire : il mesure la concentration de la distribution

(dans une région unique). Pour l'étude de la dispersion des optima, nous préférons observer les variations d'entropie et de diamètre moyen, entre U et O, afin de limiter l'influence de n, la taille de l'instance. Comme nous l'avons constaté à la section 2.3.2.2, la taille de l'instance influe peu sur l'entropie mais fort sur le diamètre moyen.

Δ_{Ent}	Petit	Grand	Grand
Δ_{Dmm}	Petit	Petit	Grand
	 *Uniforme*	 *Multi-Massif*	 *Uni-Massif*

TAB. 2.5 - *Interprétation de la valeur du couple entropie / diamètre moyen pour la répartition des optima locaux dans l'espace de recherche.*

L'analyse des dispersions montre que les variations d'entropie Δ_{Ent} et de diamètre moyen Δ_{Dmm} sont corrélées. La variation du diamètre moyen augmente avec la variation d'entropie (voir Fig. 2.12). Cependant, pour certaines instances, l'entropie varie (décroît) beaucoup plus que le diamètre moyen. C'est notamment le cas des instances Sko64, Sko81, Sko100a, Tai60b, Tai80b, Tai100b et Tho150 dont les rapports $\Delta_{Dmm}/\Delta_{Ent}$ sont inférieurs à 25 % (voir Tab. 2.4). Cette situation informe sur la distribution des optima: puisque l'entropie est faible, il y a des regroupements; mais le diamètre est modéré, donc il n'y pas de concentration. La table 2.5 montre la représentation graphique de trois types de distribution des optima locaux (uniforme, multi-massif, uni-massif) en fonction de l'entropie et du diamètre moyen. Pour de petites valeurs de variation, type uniforme, les optima locaux sont dispersés « uniformément » dans l'espace de recherche; les instances uniformes (Taixxa, Lipaxxa) sont dans cette catégorie avec des variations très faibles. À l'opposé, pour des variations élevées, type uni-massif, on a une concentration (centrale ou uni-massif) des optima; cette catégorie rassemble plutôt des instances réelles (Els19, Bur26d, Ste36c). Pour ce qui est du type intermédiaire, le type multi-massif, il rassemble les instances que nous avons citées plus haut, dont la variation d'entropie est supérieure à la variation de diamètre moyen. Pour ce type, les optima sont regroupés en petits massifs, dispersés dans l'espace de recherche. Bien que nous ayons identifié trois catégories, nous ne donnons pas de seuil numérique pour les différencier. Notre objectif n'est pas de caractériser finement ces catégories mais plutôt de donner des

indicateurs de tendance pour poser des jalons dans la grande diversité des paysages.

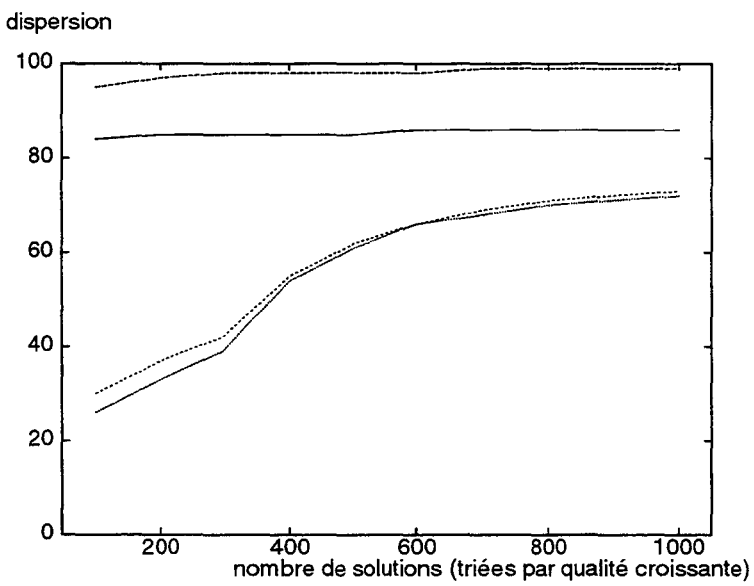


FIG. 2.13 - *Ce graphique montre la dispersions des optima locaux en fonction de leur qualité pour l'instance Had20. L'axe des abscisses donne le nombre de solutions impliquées dans les calculs de dispersion. La ligne pleine (resp. pointillé grand) montre le diamètre moyen (resp. l'entropie) des solutions initiales. La ligne en pointillé moyen (resp. pointillé petit) donne le diamètre moyen (resp. l'entropie) des optima locaux obtenus par la marche SDW. On constate que les meilleurs optima locaux sont regroupés.*

Par ailleurs, nous avons observé que le regroupement des optima locaux est accentué par la qualité des solutions. D'une manière générale, plus la qualité des optima locaux est bonne, plus ils sont proches. Cette observation résulte de l'expérience suivante: pour chaque instance, nous avons calculé la dispersion des k meilleurs optima locaux, k prenant successivement les valeurs 100, 200, 300, ..., 1000. Par exemple, pour l'instance Had20[14], on perçoit très nettement que la dispersion chute quand la qualité augmente, témoignant du regroupement des meilleurs optima (voir Fig. 2.13). Cependant, malgré ces regroupements, une certaine diversité persiste. En effet, pour presque toutes les instances, les marches SDW aboutissent à $p = 1000$ optima locaux différents pour p départs différents. Seules les deux plus petites instances se distinguent : Els19 et Had20 avec respectivement 487 et 801 optima locaux différents.

Nous avons donc constaté, en fonction des instances, que les optima locaux sont soit distribués, soit regroupés en un ou plusieurs amas. Quand les optima sont regroupés, ceux de meilleure qualité sont très regroupés et les autres les entourent, formant, selon la métaphore des paysages, un ou plusieurs massifs.

14. Des figures similaires, relatives à d'autres instances, sont présentées à l'annexe IV:A

2.3.2.6 La longueur de marche

La longueur d'une marche est le nombre d'itérations de la méthode de descente SDW. Intuitivement, la longueur de marche informe sur la rugosité du paysage. En effet, un paysage plutôt rugueux comporte beaucoup d'optima et les longueurs de marche sont courtes ; à l'inverse, un paysage plutôt doux a peu d'optima et les longueurs de marche sont plus longues. La figure 2.14 présente les longueurs de marche moyennes des SDW ($\mathrm{lmm}(U)$) en fonction de la taille de l'instance n. On constate que les points s'alignent sur des droites dont le pente est approximativement égale à $\mathrm{Lmm}(U) = \mathrm{lmm}(U)/n$. Ainsi, le coefficient $\mathrm{Lmm}(U)$ est un indicateur de la rugosité du paysage d'une instance ; il permet de classer les instances. C'est pourquoi, dans les tables 2.4 et 2.6, les instances sont classées par $\mathrm{Lmm}(U)$ croissante. En haut de table, pour les longueurs de marche très courtes, on a plutôt des instances uniformes. On en déduit, pour ces instances, un paysages très rugueux. En bas de table, les instances Skoxx, Taixxb, Will00 et Tho150 ont les marches les plus longues ; leur paysage est plus lisse. En milieu de table, pour des longueurs de marches intermédiaires, on trouve plutôt des instances réelles.

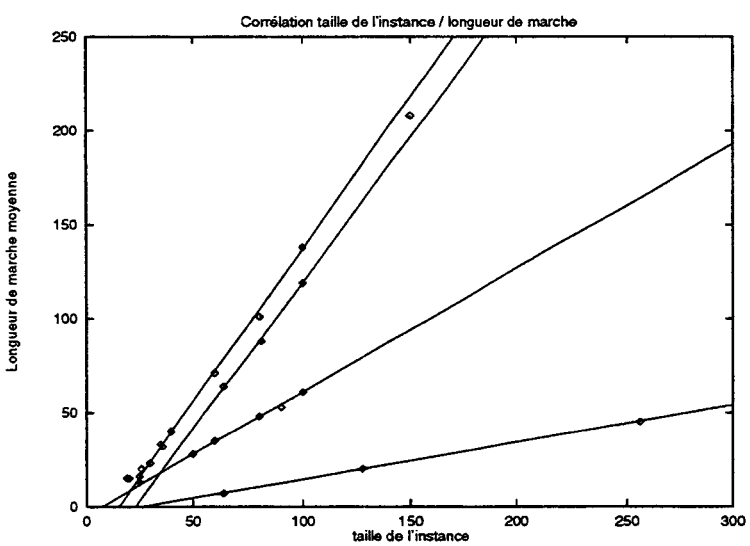


FIG. 2.14 - *Corrélation entre la longueur de marche moyenne lmm et la taille de l'instance n. Chaque point représente une instance. On note que les instances forment des alignements.*

2.3.2.7 Synthèse et taxinomie

Dans cette étude des paysages du QAP, selon une approche locale, nous avons sélectionné et proposé un certain nombre d'indicateurs qui permettent de caractériser

la structure du paysage d'une instance. Ces indicateurs ne sont ni aussi accessibles, ni aussi lisibles que la dominance de flux car ils sont multiples et nécessitent des calculs lourds. Mais ces indicateurs sont plus précis et sont surtout plus représentatifs de la vision du paysage par les heuristiques de recherche locale. Cependant, nous ne rejetons pas la dominance de flux. D'ailleurs, nous avons mis en évidence une certaine corrélation entre la dominance de flux et la dispersion des meilleurs optima locaux (voir Fig. 2.15). Cette corrélation est évoquée dans [Bachelet et al.96] ; elle peut s'expliquer par le fait que beaucoup d'instances ont une matrice de distance de nature équivalente et qu'ainsi seule la considération des flux suffit.

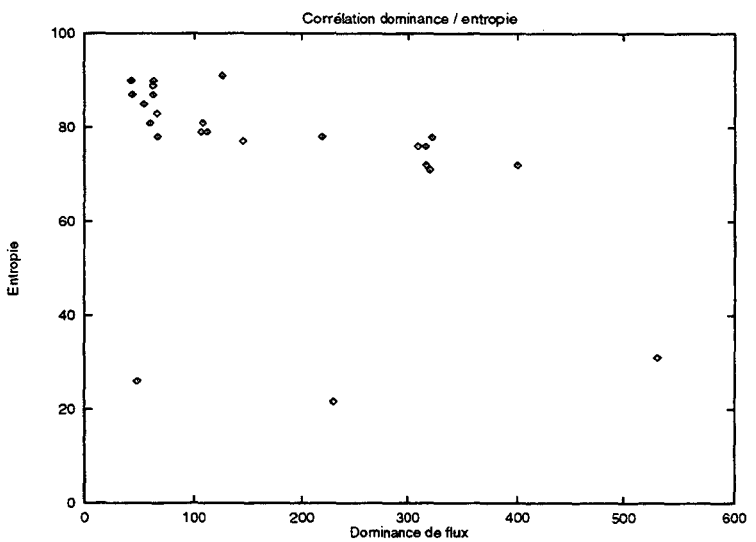


FIG. 2.15 - *Ce graphique fait apparaître une « certaine » corrélation entre l'entropie des 100 meilleurs optima obtenus par les SDW et la dominance de flux de l'instance.*

Nous avons montré ci-avant que la longueur de marche et la distribution des optima locaux (entropie et diamètre) permettent de distinguer différentes structures des paysages des instances. Lorsqu'on considère ces trois critères simultanément, on obtient une classification des instances en trois groupes distincts (voir Fig. 2.16). Nous avons donc caractérisé trois structures de paysages différentes que nous appelons type I, type II, et type III (voir Tab. 2.6).

Type I

Les instances de type I, sont en majorité des instances de nature uniforme. Les optima locaux sont très dispersés dans l'espace de recherche, l'amplitude de leur coût est faible et leur coût moyen est très bon. L'heuristique de recherche locale voit donc un paysage plutôt plat, rugueux, et de bonne qualité. Dans ces paysages peu

Inst.	Δ_{Ent}	Δ_{Dmm}	Lmm(U)	Paysage
Tai64c Esc128 Tai256c Tai25a Lipa50a Tai60a Lipa90a Tai80a Tai100a Chr25a	0	0	56	Uniforme Type I
Had20 Bur26d Nug30 Els19 Ste36c Tai35b Tai40b	14	4	79	Uni-Massif Type II
Sko64 Sko81 Tai60b Wil100 Sko100a Tai80b Tai100b Tho150	9	2	119	Multi-massif Type III

TAB. 2.6 - *Taxinomie des instances de notre étude. Les valeurs indiquées correspondent aux valeurs médianes dans chaque groupe.*

structurés, une recherche locale trouve facilement une bonne solution. Cependant, étant donné que toutes les régions de l'espace de recherche sont de coûts très semblables, il est difficile pour une heuristique de caractériser une bonne région. C'est pour cela qu'il est nécessaire de beaucoup diversifier la recherche pour obtenir une chance raisonnable de trouver des points de meilleure qualité. Notons que la présence de l'instance Tai256c dans la classe I est plutôt surprenante. En effet, c'est une instance réelle dont la dominance de flux est élevée (voir Tab. 2.3). Cependant, Tai256c est une instance de très grande taille et nous savons que la grande taille est un facteur qui diminue l'amplitude de l'espace de recherche (voir section 1.2.2.3). De plus, bien qu'elle soit réelle, l'instance Tai256c est assez atypique, ne correspondant pas directement à un problème de placement (voir section 1.1.2.4). Quoiqu'il en soit, puisque Tai256c apparaît dans le Type I, elle sera vu par une recherche locale comme une instance de type I.

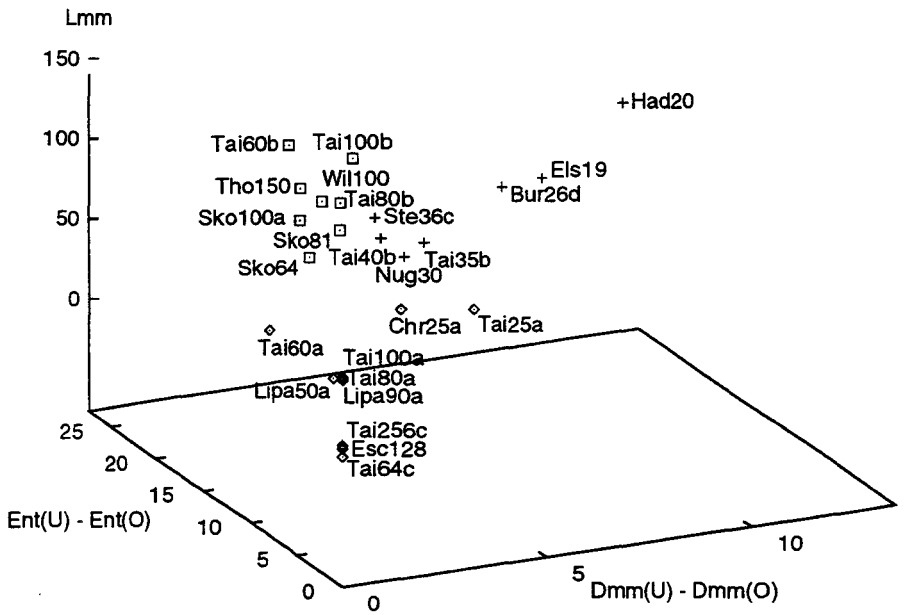


FIG. 2.16 - *Corrélation entre la longueur de marche et la distribution des optima locaux. On observe trois classes d'instances bien distinctes : le Type I représenté par des losanges, le Type II par des croix et le Type III par des carrés.*

Type II

En l'occurrence, le type II regroupe exclusivement des instances réelles ou pseudo-réelles. Pour ces instances, les marches sont plus longues et les optima locaux sont fortement regroupés en un petit nombre de massifs resserrés. Le paysage est constitué de quelques vallées profondes, dans une même région, au fond desquelles sont concentrés les meilleurs optima. Pour ces instances, une heuristique de recherche locale (la recherche tabou, par exemple) a peu de difficulté à trouver une vallée prometteuse, puis à l'exploiter. Contrairement à ce que nous avions imaginé avant de réaliser l'étude, on constate que le type II, bien qu'il labellise les paysages fortement structurés, ne correspond pas aux plus grandes longueurs de marche relatives (Lmm(U)), celles qui vont au fond des vallées profondes. Encore une fois, c'est peut-être un phénomène lié à la taille des instances ; en effet, le type II ne contient que des instances de petites tailles (< 50) alors que le type III ne contient que des instances de grande taille. De plus, les types II et III partagent des instances de même nature : Taixxb.

Type III

Le type III est sans aucun doute la classe d'instances la plus difficile à résoudre. C'est à dire la classe pour laquelle il est le plus difficile de trouver, par une recherche locale, une solution de qualité proche de celle de l'optimum global. D'abord, cette classe comprend exclusivement des instances de grande taille (> 60) ; ce qui est déjà une difficulté en soi, car les espaces de recherche sont grands. Ensuite, leurs paysages sont constitués d'une multitude de vallées, plus ou moins profondes, où sont regroupés les optima locaux. Une heuristique locale, pour être efficace sur ce genre d'instances, doit à la fois exploiter le fond des vallées et explorer suffisamment l'espace de recherche pour s'assurer qu'elle exploite la meilleure vallée. En effet, il ne sert à rien d'exploiter le fond d'une vallée profonde (comme le ferait une recherche tabou), s'il existe par ailleurs une vallée de meilleure qualité. Cependant, il est difficile pour une heuristique (ou pour ces concepteurs) de trouver un juste et efficace équilibre entre l'exploration et l'exploitation de l'espace de recherche; c'est pourquoi les instances de type III sont vraiment difficiles.

Deuxième partie

Métaheuristiques itératives parallèles pour le QAP

Cette partie est consacrée aux méthodes de recherches itératives parallèles. Nous donnons une description générale des métaheuristiques itératives et tentons de définir un modèle fédérateur. Nous comparons les différentes parallélisations des méthodes itératives. Puis, nous détaillons les heuristiques que nous avons mises en œuvre pour résoudre le problème d'affectation quadratique. Enfin, nous montrons les différents comportements de ces méthodes sur les différentes classes d'instance que nous avons identifiées en partie I.

Les travaux décrits dans cette partie ont fait l'objet d'une présentation au workshop ICGA 95 et aux conférences internationales POC 96 et PAREO 98 [Bachelet et al.95] [Bachelet et al.96] [Bachelet et al.98a].

Introduction

Il n'y a pas si longtemps, les métaheuristiques suboptimales étaient dénigrées. On leur reprochait de fournir des résultats sans aucune garantie de qualité. Pour certains, l'utilisation de telles méthodes apparaissait comme un constat d'échec par rapport à la résolution exacte du problème. Mais les méthodes itératives sont beaucoup mieux considérées maintenant ; car depuis, elles ont montré leurs nombreuses qualités. En effet, ces méthodes sont générales : elles s'appliquent à de nombreux problèmes ; elles sont faciles à mettre en œuvre car leur principe est simple ; et surtout, elles sont efficaces. À ce jour, les méthodes itératives sont, sans conteste, les métaheuristiques les plus performantes pour la résolution suboptimale du QAP et d'autres problèmes complexes. Parmi ces techniques performantes, nous avons distingué deux familles de méthodes : les méthodes itératives d'amélioration d'une solution unique (ISI – *Iterated Solution Improvement*) et les méthodes itératives à population de solutions (IPI – *Iterated Population Improvement*).

Les familles ISI et IPI se distinguent par deux approches différentes pour réaliser l'équilibre exploitation / exploration nécessaire à l'efficacité d'une heuristique de recherche. Pour les ISI, le squelette de l'heuristique est « exploiteur », c'est la recherche locale ; pour introduire un peu d'exploration, on doit greffer sur ce squelette des organes de diversification. Par exemple, la recherche tabou, décrite d'une manière simpliste mais juste, est une méthode de descente à laquelle on ajoute un organe qui mémorise le chemin parcouru pour éviter d'y retourner. À l'inverse, pour les méthodes IPI, le principe de base est « explorateur » de par la diversité de la population initiale. Cette diversité de la population est manipulée par des opérateurs intensificateurs qui concentrent la population dans une région de l'espace de recherche et exploitent cette dernière.

Certains chercheurs, Vaessens et al., Taillard et al., ont une approche unificatrice de ces deux familles [Vaessens et al.92] [Taillard et al.98]. En ce qui nous concerne, nous préférons étudier les ISI et les IPI séparément car cela permet de mieux comprendre l'intérêt de leur hybridation. En effet, les méthodes ISI ont tendance, par nature, à une plus grande exploitation d'une région de l'espace de recherche ; alors qu'à l'opposé, les recherches IPI sont exploratrices et elles caractérisent les régions prometteuses. C'est pourquoi, de nombreuses métaheuristiques robustes combinent une méthode ISI et une méthode IPI ; l'IPI explore l'espace et identifie des régions intéressantes, l'ISI exploite localement ces régions. La littérature relate un grand nombre de travaux concernant l'hybridation d'un algorithme génétique (IPI) et d'une recherche tabou (ISI) dans lesquels l'hybride obtenu est performant.

Chapitre 1

Les métaheuristiques itératives
et leurs parallélisations

Dans ce chapitre, nous décrivons le fonctionnement des principales méthodes itératives et de leur parallélisation. Nous présentons successivement les ISI et les IPI. Pour chacune de ces deux familles de métaheuristiques, nous tentons, dans un premier temps, de définir un modèle ; puis, nous présentons les différentes stratégies de quelques métaheuristiques dont l'efficacité est reconnue. Enfin, nous décrivons différents modèles de parallélisation de ces méthodes.

1.1 Les métaheuristiques à solution unique

1.1.1 Approche du modèle ISI

Les méthodes itératives d'amélioration d'une solution unique reposent toutes sur un même algorithme de base, appelé recherche de voisinage (voir Fig. 1.1). La recherche de voisinage commence avec une solution initiale arbitraire, puis l'améliore pas à pas en choisissant, comme nouvelle solution courante, une solution dans son voisinage. Nous avons identifié un certain nombre de critères qui permettent de définir et différencier les recherches ISI ; ces critères sont les suivants :

- la mémoire de l'ISI ;
- le choix de la solution initiale ;
- la génération des solutions candidates ;
- la sélection d'un candidat comme nouvelle solution courante ;

– le critère d'arrêt du processus itératif.

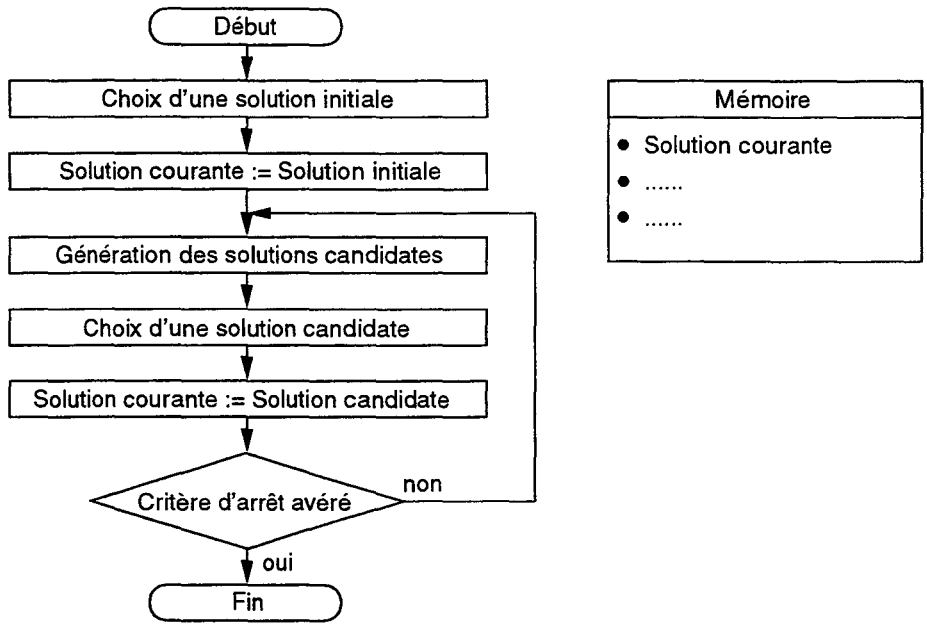


FIG. 1.1 - *La recherche de voisinage: amélioration d'une solution unique (ISI).*

Nous détaillons, ci-dessous, ces différents critères pour le modèle ISI en les illustrant par des exemples concrets.

1.1.1.1 La mémoire

Nous appelons mémoire d'une recherche ISI, l'ensemble des données qui sont mémorisées d'une itération à l'autre. La mémoire contient les données qui évoluent au cours des itérations comme, par exemple, la solution courante qui appartient toujours à la mémoire. La taille de la mémoire varie beaucoup en fonction de la méthode. Pour la recherche locale, elle est réduite à son minimum (la solution courante) ; pour la recherche tabou, elle peut être très grande si on désire retenir beaucoup d'informations sur le chemin déjà parcouru. Par exemple, Battiti et Tecchiolli utilisent une mémoire de très grande taille pour leur recherche tabou réactive [Battiti et al.94]. Ils sont contraints de mettre en œuvre des techniques de compression pour réduire la taille physique de cette mémoire afin que leur algorithme puisse fonctionner sur un ordinateur.

1.1.1.2 La solution initiale

Le plus souvent, la solution initiale est générée aléatoirement. Cependant, la solution initiale d'une ISI peut avoir une autre origine. Dans certaines méthodes hybrides, la solution initiale est produite par une autre heuristique. Souvent, on utilise une heuristique gloutonne pour générer la solution initiale ; car les heuristiques gloutonnes ont l'avantage d'être constructives (donc rapides). En procédant ainsi, l'ISI démarre avec une solution de meilleure qualité par rapport à une solution aléatoire. Par exemple, dans la méthode GRASP [Feo et al.91], qui alterne une méthode de descente (ISI) et une méthode gloutonne, la solution de départ de la méthode de descente est fournie par l'heuristique gloutonne.

1.1.1.3 La génération des candidats

Dans cette phase de la recherche de voisinage, il s'agit de produire l'ensemble des solutions sur lequel agira la sélection pour choisir la prochaine solution courante. Une relation de voisinage étant définie sur l'ensemble des solutions, l'ensemble des candidats peut être constitué de tout ou partie du voisinage immédiat de la solution courante. Le nombre de voisins retenus comme candidats varie en fonction de la nature de la méthode ISI. En outre, le nombre de candidats est fortement lié, pour des contraintes d'efficacité, à l'effort de calcul nécessaire à l'obtention de tous les candidats. Si l'obtention d'un voisin est très coûteuse, il peut être plus efficace de n'admettre comme candidats qu'une portion du voisinage immédiat de la solution courante plutôt que l'ensemble des voisins. Par ailleurs, on n'est pas obligé de choisir des candidats uniquement parmi les plus proches voisins. On peut générer des candidats un peu plus éloignés, mais pas trop, car les candidats doivent rester dans la région de la solution courante pour respecter l'esprit de l'heuristique ISI (exploitation). Cependant, pour certaines méthodes ISI incorporant des techniques de forte diversification, des candidats très éloignés sont générés en s'appuyant sur des éléments de la mémoire de l'ISI. C'est le cas, par exemple, pour la diversification basée sur la mémoire de fréquence dans certaines méthodes tabous [Glover et al.92]. La génération des candidats dans l'heuristique de recherche à voisinage variable (VNS – *Variable Neighborhood Search*) est très particulière [Hansen et al.99] : l'ensemble des solutions est muni de plusieurs opérateurs de voisinage ; les candidats sont pris dans plusieurs voisinages de la solution courante qui sont en concurrence.

1.1.1.4 La sélection d'un candidat

L'opération de sélection choisit la prochaine solution courante parmi les candidats. Généralement, le candidat du meilleur coût est choisi ; mais il existe d'autres critères de choix. Par exemple, dans le recuit simulé [Kirkpatrick et al.83], un candidat est choisi aléatoirement en fonction de sa qualité et du paramètre de tem-

pérature. Par ailleurs, la sélection se réfère parfois à la mémoire, comme dans la recherche tabou [Glover89a], dans laquelle on écarte les candidats tabous. Souvent, par souci d'efficacité et afin de limiter le nombre de candidats générés, la sélection est combinée avec la génération selon l'algorithme suivant : générer un sous-ensemble des candidats potentiels; tenter de sélectionner un candidat dans ce sous-ensemble; reprendre s'il n'y a pas eu de sélection (voir Fig. 1.2). Cette technique est mise en oeuvre dans les descentes stochastiques et les descentes de type premier-meilleur (voir section 1.1.2) pour lesquelles on génère un seul candidat à la fois jusqu'à ce qu'il soit meilleur que la solution courante et la remplace.

```
     Début
a.   Générer partiellement les candidats ;
     Tenter de sélectionner un candidat ;
     Si un candidat est sélectionné
          alors aller en b. ;
          sinon aller en a. ;
b.   Fin
```

FIG. 1.2 - *Combinaison de la génération et de la sélection des candidats.*

1.1.1.5 Le critère d'arrêt

Les critères d'arrêt sont variés. Par exemple, l'algorithme peut s'arrêter après un temps d'exécution maximum donné. Cependant, on préfère souvent employer un critère d'arrêt qui correspond à un état d'avancement propre à l'heuristique, c'est-à-dire à un état de la mémoire de l'ISI. En effet, en introduisant dans la mémoire le nombre d'itérations effectuées, on peut lancer une ISI pour un nombre d'itérations prédéterminé; ce critère d'arrêt est celui que nous avons rencontré le plus souvent à travers notre étude des heuristiques itératives. Cependant, on trouve aussi des critères d'arrêt plus sophistiqués. Par exemple, certaines recherches tabous comptent (et mémorisent) le nombre d'itérations durant lesquelles aucune amélioration de qualité n'est produite par rapport à la meilleure solution trouvée depuis le début de la recherche; elles arrêtent la recherche quand un maximum, qui est un paramètre de la méthode, est atteint. Un autre critère d'arrêt est celui du recuit simulé pour lequel un nombre stocké dans la mémoire (appelé « la température ») décroît au fil des itérations; le recuit s'arrête quand la température est nulle.

1.1.2 Les méthodes de descente

Les méthodes de descente sont sans doute les méthodes d'optimisation itératives les plus anciennes [Papadimitriou76] [Papadimitriou et al.82]. Elle doivent leur succès à leur rapidité d'exécution et à leur simplicité. En fait, elles sont l'expression la plus simple du modèle ISI dans laquelle la mémoire est réduite à la seule solution courante. Les méthodes de descente tirent leur nom du fait qu'à chaque pas elles progressent vers une solution voisine de qualité meilleure[1]. La descente s'arrête lorsque tous les voisins (candidats) sont moins bons que solution courante ; c'est à dire lorsqu'un optimum (local *a priori*) est atteint (voir Fig. 1.3). Johnson *et al.* ont étudié la complexité des méthodes de descente, mais leurs conclusions sont difficilement exploitables en pratique [Johnson et al.88]. Expérimentalement, on constate que les descentes s'arrêtent plutôt rapidement, en un temps polynomial de petit ordre [Papadimitriou et al.82] [Lin et al.73].

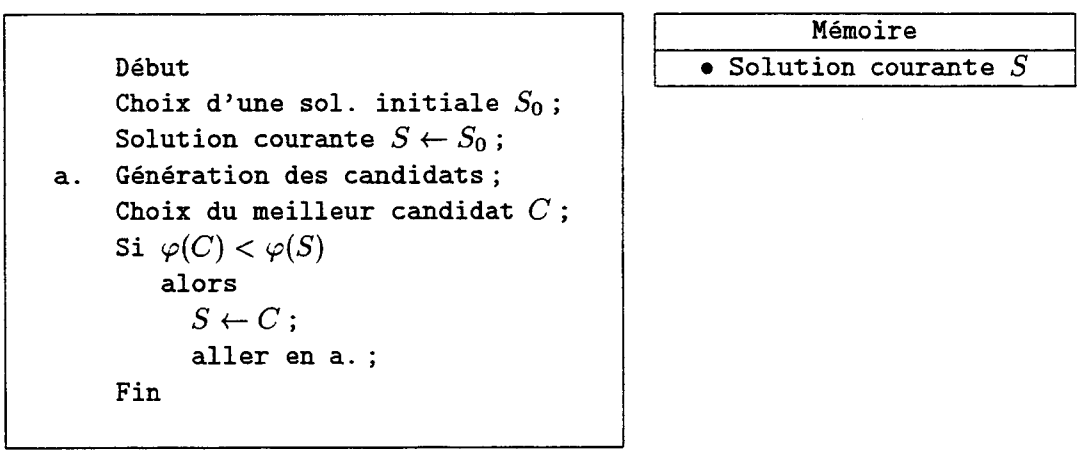


FIG. 1.3 - *Méthode de descente générique.*

On distingue principalement trois types de descentes en fonction de la stratégie utilisée pour la génération et la sélection des candidats :

- la descente déterministe : à chaque pas, la totalité des candidats est générée et seul le meilleur des candidats peut devenir la nouvelle solution courante. Cette ISI est aussi appelée descente stricte ou descente selon la plus forte pente (SDW - *Steepest Descent Walk*). On l'emploie plutôt lorsque le nombre de candidats est peu élevé et/ou lorsque l'évaluation des candidats est relativement rapide ;
- la descente stochastique : ici, la nouvelle solution courante est choisie de façon stochastique. Les candidats sont visités dans un ordre aléatoire et le premier

1. Vers le bas pour un problème de minimisation. Les méthodes de descente sont parfois appelées méthodes de montée (*Hill Climbing*) pour des problèmes de maximisation.

candidat de meilleure qualité que la solution courante est retenu. Contrairement à la SDW, cette descente n'est pas déterministe. Elle permet, par exemple, d'obtenir des résultats différents à partir d'une même solution initiale, générée par une heuristique gloutonne. En outre, le choix aléatoire du candidat permet d'éviter un plongeon systématique vers un attracteur décevant[2] ;

- la descente vers le premier meilleur: cette méthode est semblable à la descente stochastique mais elle est déterministe. La différence majeure est qu'ici l'ordre de visite des candidats est prédéterminé. Cette méthode, tout comme la descente stochastique présente l'avantage d'un coût de calcul plus faible car il n'est pas nécessaire de visiter l'ensemble des candidats à chaque itération. Cependant, il est souvent difficile de justifier l'ordre de visite des candidats qui est choisi. Pour le QAP, Heider propose d'utiliser un ordre cyclique. Pour numéroter les voisins de la solution courante, il numérote les transpositions qui génèrent le voisinage. À la première itération, l'ordre de visite est 1, 2, ..., $\frac{n(n-1)}{2}$; à l'itération suivante, c'est 2, 3, ..., $\frac{n(n-1)}{2}$, 1 ; puis 3, ..., $\frac{n(n-1)}{2}$, 1, 2 ; et ainsi de suite.

On peut s'interroger sur l'influence de la qualité de la solution initiale sur celle de l'optimum atteint au final. Dans son étude expérimentale sur le QAP, Bruijs suggère qu'il n'y aurait que peu d'intérêt à démarrer la descente avec une solution de bonne qualité [Bruijs84]. Pour notre part, il nous semble qu'une solution de bonne qualité, dans une vallée de bonne qualité de l'espace de recherche, a une forte probabilité de conduire, par une méthode de descente, à un optimum de meilleure qualité que celle d'un autre optimum issu d'une descente ayant pour origine une solution aléatoirement choisie. C'est d'ailleurs par ce principe que nous justifions la supériorité de performance des méthodes hybrides IPI/ISI, IPI explorant l'espace pour trouver de bonnes vallées, l'ISI explorant le fond de ces vallées.

Indépendamment du choix de la solution de départ, la qualité des solutions produites par les descentes est variable. En partie I, au chapitre 2, nous montrons, dans l'étude des paysages du QAP, la variation de la qualité des optima locaux fournis par des descentes SDW en fonction de la nature de l'instance. Cependant, d'une manière générale, la qualité des optima issus des méthodes de descente est plutôt insatisfaisante. C'est pourquoi ces techniques d'optimisation ont été complétées pour produire des méthodes plus performantes, mais aussi plus sophistiquées. Certaines de ces méthodes conservent la propriété de généralité des heuristiques de descente, c'est à dire qu'elles restent applicables à de nombreux problèmes d'optimisation. Ainsi sont nées d'autres recherches ISI comme le recuit simulé et la recherche tabou que nous présentons ci-après. Une autre approche pour améliorer les méthodes de descente, orientée par le problème à résoudre, a produit des heuristiques spécifiques. Dans cette catégorie, on peut citer, pour leur très bonnes performances, les méthodes

2. On peut imaginer qu'un chemin en pente douce descende plus bas qu'un autre chemin plus pentu.

conçues par Kernighan et Lin pour le voyageur de commerce [Lin et al.73] et pour le partitionnement de graphes [Kernighan et al.70].

1.1.3 Le recuit simulé

L'utilisation du recuit simulé, comme technique d'optimisation remonte au début des années 80. Cependant, l'algorithme était déjà utilisé 30 ans auparavant, connu sous le nom d'algorithme de Metropolis, pour simuler le refroidissement des matériaux selon un procédé appelé le recuit [Metropolis et al.53]. En 1983, Kirpatrick, Gelatt et Vecchi traduisent l'algorithme de Metropolis, du domaine de la physique vers le domaine de l'optimisation combinatoire et conçoivent une nouvelle métaheuristique itérative qu'ils appellent le recuit simulé (SA– *Simulated Annealing*) [Kirkpatrick et al.83]. En 1985, tout à fait indépendamment, Cerny, en Tchécoslovaquie, invente une technique très semblable pour la résolution du problème du voyageur de commerce [Cerny85].

Dans son fonctionnement (voir Fig. 1.4), la stratégie du SA ressemble à une méthode de descente stochastique. Cependant, la mémoire du SA est plus riche. En plus de la solution courante, elle contient la meilleure solution visitée depuis le début de la recherche et des valeurs qui contrôlent l'avancement de l'heuristique (température et numéro d'itération de palier). Contrairement à la descente stochastique, le SA permet, sous certaines conditions, la sélection d'un candidat d'une qualité inférieure à celle de la solution courante. En effet, à chaque itération, on visite un candidat choisi au hasard parmi tous les autres. Si ce candidat est de qualité supérieure à la solution courante alors il la remplace ; sinon, il se peut qu'il la remplace avec une probabilité P, dont l'expression ci-dessous, est directement importée de la physique statistique.

$$P = e^{-\frac{\Delta}{T}}$$

Cette probabilité P est fonction de Δ, la qualité relative du candidat C par rapport à la solution courante S : $\Delta = \varphi(C) - \varphi(S)$. Plus le candidat est mauvais par rapport à la solution courante, moins il a de chance d'être sélectionné. La probabilité de sélection est aussi liée à un facteur T, appelé température par analogie au procédé de recuit en physique : plus la température est élevée, plus la probabilité de sélection est élevée. Au cours de l'avancement de l'algorithme de recuit simulé, la température décroît par paliers. Généralement, le SA se termine avec des températures basses, qui n'autorisent qu'exceptionnellement la sélection de candidats de moins bonne qualité que celle de la solution courante ; le recuit se comporte alors, dans sa phase finale, comme une descente stochastique. Le lecteur désireux de plus de détails, théoriques ou pratiques, pourra consulter [vL et al.88] et [Aarts et al.89].

La suite de températures et la longueur des paliers (nombre d'itérations par palier) constitue les paramètres du recuit simulé. La température permet de contrôler le caractère explorateur ou exploiteur du recuit. En effet, quand la température est

```
     Début
     Choix solution initiale S_0
     Solution courante S ← S_0
     Meilleure solution M ← S
     Température courante T ← T_0
a.   Itération de palier I ← 1
     Tant que I ≤ K
        Si φ(S) < φ(M)
           alors
             M ← S
        Génération d'un candidat C
        Δ = φ(C) − φ(S)
        Si Δ < 0
           alors
             S ← C
           sinon
             probabilité P ← e^(−Δ/T)
             S ←(P) C
        I ← I + 1
     Si Suivant(T) existe
        alors
          T ← Suivant(T)
          aller en a.
     Fin
```

Mémoire
• Solution courante S
• Meilleure solution M
• Température courante T
• Itération de palier I

Paramètrage
• Suite de températures T_i
• Palier de températures K

FIG. 1.4 - *Méthode générique du recuit simulé.*

basse, le SA se comporte comme une descente et possède un fort pouvoir d'exploitation. À l'opposé, lorsque la température est haute, le SA accepte presque tous les candidats et parcourt l'espace de recherche comme une marche aléatoire ; il se comporte alors en explorateur. Au cours de la recherche, si le SA trouve une solution de bonne qualité dans une vallée de bonne qualité, alors la probabilité de quitter cette vallée est faible car il faudrait, pour cela, passer par des solutions de qualités fort différentes de la solution courante, qui ont peu de chances d'être sélectionnées. D'une manière imagée, on peut voir la température comme la hauteur des barrières (collines ou crêtes) que peut franchir la recherche. Dans le schéma d'exploitation classique du recuit, la température est haute au commencement, ce qui permet de franchir les barrières à travers l'espace de recherche pour trouver une vallée de bonne qualité ; puis, la température décroît pour enfermer la recherche dans cette vallée et l'exploiter, à la manière d'une descente sur la fin. La longueur de palier contrôle, elle

aussi, l'exploration : de longs paliers permettent à la recherche de parcourir l'espace longuement alors que des paliers courts réduisent l'exploration. Cependant, on utilisent généralement des paliers de longueur suffisante pour atteindre un « équilibre », c'est à dire une certaine stabilité dans la qualité de la solution courante. Parfois, le recuit simulé générique est décrit avec des paliers de longueurs variables qui se terminent quand un équilibre est détecté.

Quand on considère le cheminement du SA à travers le paysage de l'instance à résoudre, on comprend mieux pourquoi les paramètres du recuit doivent être adaptés à l'allure du paysage, à la nature de l'instance. En effet, une température trop élevée peut faire sortir la recherche d'une vallée prometteuse ; au contraire, une température trop basse piège trop vite le recuit dans un bassin de qualité médiocre. Ainsi, la métaphore des paysages permet de mieux comprendre pourquoi il n'existe pas de consensus pour le paramétrage du SA et pourquoi il faut faire beaucoup d'essais et de nombreux réglages pour obtenir de bons résultats avec le recuit simulé [Burkard et al.83b]. À cause de cette grande sensibilité du paramétrage, de nombreuses variantes ou améliorations ont été proposées. Parmi ces variantes du recuit, les plus étudiées sont relatives : à la vitesse de refroidissement (linéaire, logarithmique, ...), au choix du voisinage et à la loi de probabilité de la sélection. Une synthèse de l'application du recuit simulé et de ses variantes sur des problèmes classiques tels que la coloration de graphe, le partionnement de graphe ou le voyageur de commerce est disponible dans [Johnson et al.89] [Johnson et al.91] [Johnson et al.92]. La méthode du recuit simulé a été très étudiée, en théorie et en pratique. C'est une heuristique très répandue, dans de nombreux domaines d'application. Depuis son apparition, le SA a été appliqué au QAP [Burkard et al.83b] [Wilhelm et al.87] [Connolly90] [Laursen93] en utilisant, presque toujours, une approche classique sur le voisinage de la transposition. À chaque fois, les auteurs ont rencontré la même difficulté de paramétrage du SA et proposent des solutions. Cependant, il semblerait que le manque de robustesse du recuit simulé soit un handicap qui fait qu'on lui préfère d'autres techniques comme, par exemple, la recherche tabou.

1.1.4 La recherche tabou

On doit l'intention de la recherche tabou (TS – *Tabu search*) à Glover, à la fin des années 80 [Glover89a] [Glover89b]. La recherche TS est une métaheuristique performante qui s'applique à de très nombreux problèmes d'optimisation [Glover et al.93]. Comme le recuit simulé, la recherche tabou est une ISI dont la mémoire contient la solution courante, la meilleure solution visitée depuis le début de la recherche et une valeur qui contrôle l'avancement de la recherche. Le plus souvent, l'avancement de la TS est contrôlé par le nombre d'itérations effectuées ; ce qui permet d'arrêter la recherche après un nombre prédéterminé d'itérations. Cependant, la mémoire de la TS est plus riche que celle du recuit simulé. En effet, l'originalité de la TS vient de ce qu'elle maintient dans sa mémoire, en plus des éléments que nous venons de citer, un

historique de la recherche appelé la liste tabou. Contrairement au recuit simulé, la recherche tabou est une méthode déterministe[3] (voir Fig. 1.5). Au commencement, la TS se comporte comme une méthode de descente SDW jusqu'à ce qu'un optimum (local) soit atteint. Quand elle se trouve bloquée sur un optimum, la recherche tabou se poursuit en sélectionnant un candidat de moins bonne qualité que celle de la solution courante. Cette stratégie permet de sortir de l'optimum; mais, la TS, qui se comporte comme une descente en dehors des optima, pourrait retourner, dès l'itération suivante, à l'optimum qu'elle vient de quitter. Grâce à la liste tabou qui maintient un historique de la recherche, ce cyclage indésirable peut être évité. Pour empêcher les cycles, la TS, se référant à sa liste tabou, ne sélectionne pas les candidats qui sont proches d'une solution visitée récemment : ces candidats sont appelés des candidats tabous. Cependant, si un candidat tabou est d'une grande qualité, il est tout de même sélectionné selon un mécanisme appelé l'aspiration.

Dans la recherche tabou, le rôle original de la mémoire de l'ISI est de préserver des informations concernant les caractéristiques des solutions courantes successives qui permettent de déterminer le caractère tabou des candidats. L'ensemble de ces caractéristiques est conservé dans la liste tabou. Souvent, cette liste, qui est mise à jour à chaque itération, est une file du type FIFO[4] d'une taille l prédéterminée qui est un paramètre de la recherche. Cette liste permet donc de stocker un historique concernant les l dernières solutions courantes. Cette longueur l est un paramètre sensible de la TS et doit être adaptée à la nature de l'instance [SK90]. La taille de la liste tabou contrôle le pouvoir d'exploration de la recherche. En effet, quand la liste est courte, la recherche a un rôle d'exploitation. Comme l'historique est court, rien n'empêche la recherche de rester dans la région de l'espace de recherche où elle se trouve et de l'exploiter. Cependant, quand la liste est courte, la recherche TS a tendance à cycler, et par conséquent, il est inutile d'effectuer un grand nombre d'itérations et seule une portion très restreinte de l'espace de recherche est parcourue. Au contraire, quand la liste tabou est longue, la recherche sort rapidement de la région où elle se trouve et explore l'espace de recherche. Une recherche TS dotée d'une grande liste tabou parvient à trouver des régions intéressantes de l'espace de recherche mais ne peut pas les exploiter car elle les traverse sans s'y attarder. Comme toujours, l'idéal est de trouver le juste équilibre entre l'exploration et l'exploitation, c'est à dire trouver la bonne taille de la liste tabou. Cependant, on s'est aperçu, en pratique, qu'une taille de liste fixée pour toute la recherche n'est pas satisfaisante. En effet, comme pour le recuit simulé, il vaut mieux commencer avec un comportement explorateur (grande liste) pour trouver une région intéressante puis l'exploiter (petite liste). Cependant le recuit simulé tient compte de la qualité des solutions, et plus une vallée est profonde, plus il a de chance d'y être piégé pour l'exploiter ensuite. Mais, à l'inverse, la TS ne prend pas en compte la qualité des solutions et ne rend pas

3. On trouve certaines TS pour lesquelles la longueur de la liste tabou et/ou le génération des candidats est stochastique.

4. *First in, first out* : premier entré, premier sorti.

```
     Début
     Choix d'une sol. initiale S0
     Solution courante S ← S0
     Meilleure solution M ← S
     K ← 0
a.   Si K ≥ N
        alors
          aller en c.
     K ← K + 1
     Mise à jour de la liste tabou
     Si φ(S) < φ(M)
        alors
          M ← S
     Générer les candidats C
b.   C ← meilleur candidat de C
     Si φ(C) < φ(S)
      sinon si C n'est pas tabou
      sinon si C avère l'aspiration
        alors
          S ← C
        sinon
          Ôter C de C
          aller en b.
     aller en a.
c.   Fin
```

Mémoire
• Solution courante S
• Liste tabou
• Meilleure solution M
• Itération courante K

Paramétrage
• Nombre d'itérations N
• Taille de liste tabou
• Critère d'aspiration

FIG. 1.5 - *Méthode de recherche tabou générique.*

compte d'une vallée plus profonde que les autres. Ainsi, la stratégie qui consisterait à diminuer la longueur de la liste tabou progressivement, comme pour la température du recuit simulé, n'a pas de sens et n'apporte rien d'intéressant pour la TS car elle aboutirait à limiter la recherche dans une région quelconque de l'espace. Il faut donc imaginer d'autres stratégies à longueurs de listes variables.

On doit une des toutes premières applications de la recherche tabou au QAP à Skorin-Kopov, en 1990 [SK90]. Sa TS utilise une liste tabou de taille prédéfinie et fixe tout au long de la recherche. Un candidat est déclaré tabou si la transposition qui permet de l'obtenir à partir de la solution courante a été employée récemment pour générer la solution courante: la liste tabou contient donc des transpositions.

Selon l'auteur, cette TS est difficile à paramétrer et, de ce fait, manque de robustesse. En 1991, Taillard propose la recherche tabou robuste (RoTS) [Taillard91]. Dans la RoTS, la longueur de la liste tabou varie aléatoirement après un certain nombre d'itérations entre deux bornes qui sont des paramètres de la méthode. Pour le QAP, le statut tabou dans la RoTS change par rapport à la TS de Skorin-Kapov : un candidat est tabou si les deux éléments échangés par la transposition qui le définit sont à nouveau placés sur des emplacements qu'ils occupaient récemment. Sur le QAP la RoTS donne de bons résultats sur de nombreuses instances avec les mêmes paramètres, d'où son nom de robuste. En 1994, Battiti et Tecchioli, pour améliorer la robustesse de la recherche tabou, proposent d'ajuster la taille de la liste tabou dynamiquement en fonction de l'état de la recherche et conçoivent la recherche tabou réactive (ReTS) [Battiti et al.94]. Au fil des itérations, la taille de la liste est progressivement réduite, plutôt lentement ; mais, quand la recherche rencontre une solution déjà visitée (indicateur de cycle) alors la liste est agrandie[5], pour sortir de la région en cours d'exploitation. Cependant cette stratégie n'est pas toujours suffisante pour sortir de la région courante, notamment quand la recherche est confrontée à ce que les auteurs appellent un attracteur chaotique[6]. Le ReTS comporte donc un mécanisme qui diversifie fortement la recherche lorsque la même solution est visitée trop souvent[7] au cours des dernières itérations. Taillard établit une comparaison des performances des différentes recherches tabou sur des instances du QAP de différentes natures [Taillard95]. L'auteur montre qu'aucune méthode ne surpasse les autres sur une large gamme d'instances. Ainsi, bien que les TS aient gagné en robustesse, il faut tout de même adapter l'heuristique à la nature de l'instance.

Au jour d'aujourd'hui, il existe de très nombreuses variantes de recherches tabou impliquant des techniques plus ou moins sophistiquées pour intensifier ou diversifier la recherche [Glover et al.92] [Glover94]. D'une manière générale, une recherche tabou, même dans une version simple, se montre efficace sur de nombreux problèmes d'optimisation combinatoire. Pour le JSP (*Job shop Scheduling Problem*) la TS montre une réelle supériorité par rapport à d'autres techniques d'optimisation [Aarts et al.94] [Taillard94] ; pour le QAP, Battiti et Tecchioli comparent la ReTS au recuit simulé et obtiennent de meilleurs résultats avec leur recherche tabou [Battiti et al.94]. Cette supériorité de la recherche tabou provient sans doute du fait qu'elle est une métaheuristique plutôt robuste qui demande moins de réglages fins que, par exemple, le recuit simulé.

5. La taille de la liste est multipliée par une constante qui est un paramètre de la méthode.

6. La recherche est confrontée à un attracteur chaotique lorsque le chemin boucle dans une même région mais ne se recoupe pas

7. La ReTS utilise un seuil qui est un paramètre de la méthode.

1.1.5 Parallélisation des méthodes ISI

Ces dernières années, beaucoup d'efforts de recherche ont été consentis à la mise au point de métaheuristiques parallèles. Le parallélisme, par la force de calcul qu'il apporte, permet d'augmenter l'efficacité d'une heuristique ; il autorise également de s'attaquer à des instances de plus grande taille. Nous traitons ici de la parallélisation des ISI dont le but est de réduire le temps d'exécution. D'autres aspects de la parallélisation des métaheuristiques relatifs à l'hybridation et à la co-évolution, impliquant le partage du domaine de recherche ou la coopération entre processeurs, sont présentés à la partie III. Nous décrivons ici deux techniques de parallélisation des ISI : la distribution de la génération des candidats et l'approche multidépart. Le lecteur trouvera plus de détails dans [Verhoeven et al.95] et plus spécialement [Crainic98] [Voss93] pour la recherche tabou, et [Greening90] pour le recuit simulé.

1.1.5.1 Distribution de la génération des candidats

Il s'agit de distribuer (sur les processeurs) la génération des solutions candidates. C'est une parallélisation dite de « bas niveau », c'est à dire que la parallélisation ne modifie pas le comportement de la méthode. Les résultats produits par une ISI parallèle de bas niveau sont identiques à ceux que fournit une recherche séquentielle ; mais le temps d'exécution est généralement plus court. C'est une parallélisation du type maître/esclaves dans laquelle le maître est chargé du séquencement de l'ISI (voir fig. 1.6). Au commencement de la génération (évaluation) des candidats, il distribue la solution courante sur les différents processeurs qui engendrent chacun une portion de l'ensemble des candidats. Puis, les candidats sont regroupés sur le processeur maître qui exécute la fin de l'itération. Ici, la mémoire de l'ISI n'est pas distribuée sur les processeurs esclaves car la partie parallèle commence et termine dans la même itération. Ce type de parallélisation permet d'accélérer la génération des candidats lorsque cette dernière représente un calcul lourd, c'est à dire lorsque l'évaluation d'un candidat est très coûteuse et/ou lorsqu'il y a un grand nombre de candidats à évaluer. Cependant, il faut s'assurer que les données transmises aux esclaves ne soient pas trop volumineuses car le surcoût de communication et le goulet d'étranglement qui en résulte réduisent considérablement l'accélération de cette parallélisation.

Notons que ce modèle de parallélisation ne s'applique pas naturellement au recuit simulé, car, généralement, dans le SA on n'évalue qu'un seul candidat par itération. De même, le gain de la parallélisation de bas niveau pour une descente de type premier meilleur dépend fortement du nombre moyen de voisins qu'il faut consulter pour en trouver un qui soit meilleur que la solution courante. En conséquence, ce type de parallélisation a été employé pour des ISI dans lesquelles l'ensemble du voisinage de la solution courante est évalué : la SDW et la recherche tabou. Pour la résolution du QAP par une recherche tabou, des implantations de ce modèle paral-

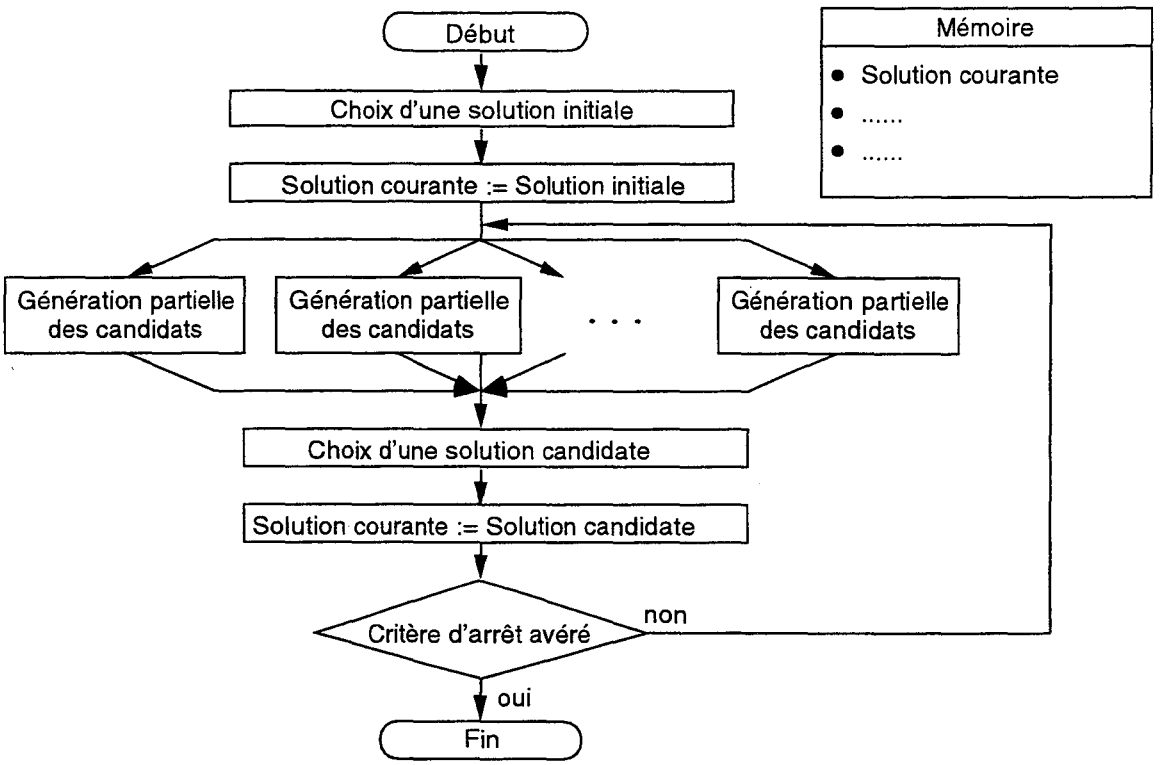


FIG. 1.6 - *ISI: parallélisation de la génération des candidats.*

lèle ont été réalisées, entre autres, sur une architecture massivement parallèles CM-2 [Chakrapani et al.93b] et sur un anneau de transputers [Taillard93b]. Pour leur recherche tabou sur la CM-2, Chakrapani et Skorin-Kapov utilisent n^2 esclaves pour évaluer le voisinage de la solution courante et obtiennent des accélérations presque linéaires [Chakrapani et al.93b]. L'implantation de Taillard sur un anneau de 10 transputers est un peu différente: il n'y a pas de processeur maître. En fait, tous les transputers jouent chacun le rôle du maître et de l'esclave. Ils exécutent tous la même recherche tabou séquentielle à l'exception de la génération des candidats. En effet, chaque processeur est en charge de l'évaluation d'une portion bien déterminée du voisinage de la solution courante. Après l'evaluation de sa portion de voisinage, chaque processeur diffuse aux autres son meilleur candidat. Tous les processeurs se trouvent alors en possession du même ensemble de candidats et effectuent le même choix comme s'il y avait un seul maître. Outre le QAP, la distribution de l'évaluation du voisinage pour la recherche tabou a été mise en œuvre pour résoudre le problème du voyageur de commerce (TSP) [Chakrapani et al.93a] et le problème de tournées de véhicules (VRP – *Vehicle Routing Problem*) [Garcia et al.94].

1.1.5.2 Parallélisation multidépart indépendante

Ce type de parallélisation n'a de parallèle que le nom. En effet, il s'agit de lancer simultanément plusieurs ISI (identiques ou non) sans aucun échange d'information entre les recherches ; si bien que le résultat produit serait identique si les ISI étaient exécutées en séquence. Cependant, cette méthode, souvent performante, permet une implantation très aisée sur un calculateur multiprocesseur, ce qui lui vaut d'être qualifié d'heuristique parallèle (voir Fig. 1.7). Généralement, les ISI qui s'exécutent simultanément diffèrent, à plus forte raison pour des ISI déterministes, par leur solution initiale et leur paramétrage. Cette parallélisation, ne nécessitant pas de communication entre les processeurs (recherches indépendantes), est particulièrement bien adaptée aux architectures dépourvues de communication à haut débit, tel qu'un réseau de stations de travail.

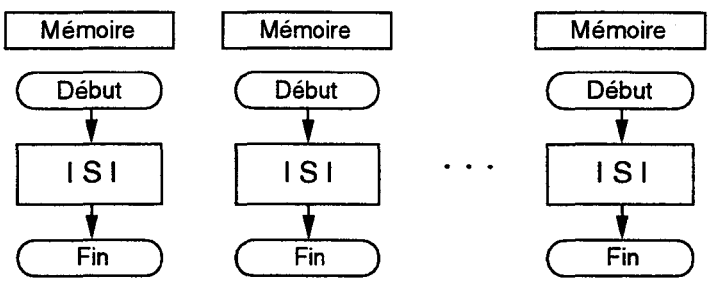


FIG. 1.7 - *ISI : parallélisation multidépart.*

Contrairement à la parallélisation de bas niveau qui ne modifie pas le comportement de la méthode séquentielle, la parallélisation *multistart* pose la question suivante. Est-il équivalent d'exécuter k marches ISI pendant un temps t ; ou bien 1 seule recherche pendant un temps $k \times t$? Plus formellement, il s'agit de savoir si la probabilité P_1 qu'une marche ISI trouve une solution d'une certaine qualité donnée est une fonction exponentielle du temps. Dans ce cas, la probabilite P_k d'obtenir une solution de qualité donnée avec k recherches pendant un temps t est égale à la probabilité de trouver une solution de même qualité en une seule marche pendant un temps $k \times t$ [Verhoeven et al.95].

$$\lambda \in \mathbb{R}^+ , \qquad P = 1 - e^{-\lambda t} \iff P_k(t) = P_1(k \times t)$$

Si c'est le cas, la parallélisation est très intéressante car on obtient alors une accélération linéaire[8]. Pour le QAP, quelques études ont été menées pour caractériser expérimentalement la distribution de la probabilité de trouver une solution (sub)optimale. Pour la RoTS, Taillard trouve une bonne corrélation entre la probabilité de trouver une solution optimale et une distribution exponentielle [Taillard91]. Battiti et

8. Le gain dû à la parallélisation croît linéairement avec le nombre de processeurs.

Tecchiolli [Battiti et al.92] ont observé que la probabilité de trouver une solution (sub)optimale avec une recherche tabou suit une distribution linéaire à condition de commencer la TS avec un optimum local. Ils précisent que la parallélisation *multistart* est efficace si le temps nécessaire à l'obtention du premier optimum local est relativement faible vis à vis du temps total de la marche TS. D'une manière générale, la parallélisation multidépart d'une ISI produit une heuristique efficace pour le QAP. Maniezzo *et al.* compare 8 métaheuristiques sur le QAP et concluent à la grande efficacité de la descente SDW *multistart* [Maniezzo et al.94].

Pour notre part, nous pensons que l'efficacité de la parallélisation *multistart* sur le QAP dépend de la nature des instances à résoudre. En effet, cette parallélisation ajoute un facteur de diversification aux recherches locales démarrant des recherches à partir de solutions réparties dans l'espace de recherche. Par exemple, si le paysage est une vallée profonde alors cette parallélisation n'apporte rien car chaque marche TS descend au fond de cette vallée après une longue marche. À l'inverse, si le paysage est constitué de multiples bassins, les marches TS sont courtes et restent piégées dans les bassins. Il est alors intéressant de lancer des marches dans différentes régions de l'espace afin d'augmenter les chances de trouver un bassin profond. Nous avons expérimenté une recherche tabou multidépart pour le QAP ; cette expérience est décrite au chapitre 2.

1.2 Les métaheuristiques à population de solutions

1.2.1 Caractérisation des IPI

Les méthodes que nous présentons dans la suite de cette section (algorithme génétique, recherche par dispersion et système de fourmis) ont un certain nombre de points communs que nous identifions ici pour donner une caractérisation de ces méthodes itératives à population[9] de solutions (IPI – *Iterative Population Improvement*). Le principe général des méthodes IPI est d'améliorer, itération après itération, une population de solutions en procédant à des combinaisons de ces solutions pour produire de nouvelles solutions qui sont alors ré-injectées dans la population en remplacement d'autres solutions. La figure 1.8 montre le séquencement d'une méthode IPI générique dont les étapes sont détaillées ci-après. Tout comme pour les ISI, nous définissons la mémoire d'une heuristique IPI comme l'information qui est conservée d'une itération à l'autre. Dans presque toutes les méthodes à population de solutions dont nous avons connaissance, la mémoire se limite à la population de

9. Nous entendons par population, un groupe de solutions dans lequel il est possible qu'une ou plusieurs solutions soient présentes plusieurs fois. Dans ce sens, population s'oppose à ensemble dans lequel il ne peut y avoir de doublon, par définition.

solutions et au compteur d'itérations qui contrôle l'arrêt de la recherche.

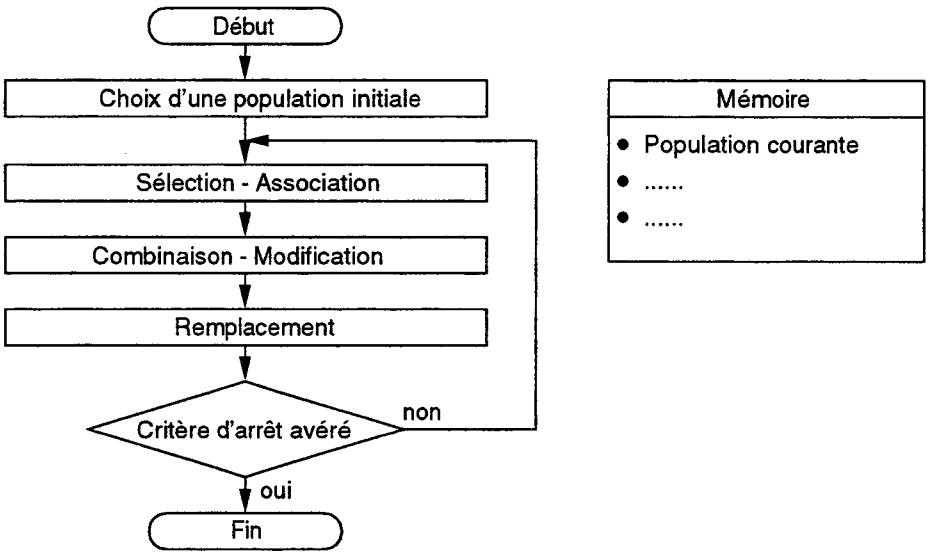


FIG. 1.8 - *Améliorations successives d'une population de solutions (IPI).*

Une heuristique IPI commence par le choix d'une population initiale. Souvent, cette population est générée aléatoirement selon une distribution uniforme ou est produite par une méthode optimisant la diversité. En effet, la population initiale doit être bien diversifiée afin qu'elle représente une bonne couverture de l'espace de recherche. Grâce à cette grande diversité de la population initiale, les méthodes IPI sont naturellement exploratrices. La taille de la population, bien qu'elle puisse varier, est très souvent fixe durant le déroulement de l'IPI ; elle constitue un paramètre de la méthode. La taille de la population conditionne directement le temps d'exécution de la méthode : plus il y a de solutions, plus il y a de traitements à effectuer.

Le processus itératif de l'heuristique à population de solutions commence par la phase de sélection. Il s'agit de choisir, dans la population courante, les solutions qui vont intervenir dans la génération de nouvelles solutions qui constitueront la population de l'itération suivante. La sélection crée une population intermédiaire, constituée des solutions sélectionnées en un ou plusieurs exemplaires (clonage). Le choix d'une solution et son nombre de clones sont généralement contrôlés par la qualité de la solution, d'une manière déterministe ou probabiliste. En général, meilleure est une solution, plus elle a de chance d'être sélectionnée et plus elle est répliquée. Vient ensuite l'association ; la population obtenue par la sélection est divisée en sous-populations (souvent disjointes) qui constituent chacune une unité sur laquelle opérera la combinaison indépendamment des autres sous-populations. Dans l'algorithme génétique, par exemple, les solutions sélectionnées sont associées par paires (ou plus) ; puis, chaque paire est soumise aux opérateurs de reproduction. Dans le

système de fourmis, l'association est triviale, car toutes les solutions sélectionnées sont associées en une seule population.

Après avoir sélectionné puis associé les solutions, viennent les phases de combinaison et de transformation qui produisent de nouvelles solutions. Dans la combinaison, chaque sous-population obtenue par l'association est traitée indépendamment, en parallèle. Pour chaque sous-population, l'opérateur de combinaison décompose les solutions qu'elle contient en différents attributs qu'il recombine pour construire de nouvelles solutions. Ainsi, une nouvelle solution est construite à partir des différents attributs contenus dans la sous-population traitée. Dans le système de fourmis, par exemple, les solutions sont décomposées en attributs, et seuls les attributs les plus fréquents dans la population sont recombinés pour former de nouvelles solutions. La phase de transformation suit la combinaison. Chaque nouvelle solution créée y est transformée indépendamment des autres. Généralement, sont appliqués à cette étape, des opérateurs de projection et/ou d'optimisation. La projection est utilisée quand la recombinaison a produit des solutions non réalisables. Ces dernières sont alors modifiées pour les rendre réalisables. L'optimisation (souvent une ISI) raffine la solution nouvellement créée. Lorsque la méthode IPI n'intègre pas d'opérateur d'optimisation dans la phase de transformation, cette dernière apparaît comme un point de connection avec une ISI pour obtenir une méthode hybride comme nous le développons dans la partie III. À titre d'illustration, la transformation de l'algorithme génétique, met en œuvre un opérateur de mutation qui altère légèrement les nouvelles solutions.

La dernière étape du processus itératif d'une IPI consiste en l'incorporation des nouvelles solutions dans la population courante, c'est la phase de remplacement. Les nouvelles solutions, après avoir été transformées, sont ajoutées à la population courante en remplacement (total ou partiel) des anciennes solutions. Généralement, les meilleures solutions remplacent les plus mauvaises ; il en résulte une amélioration de la population. Lorsque la nouvelle population, ainsi créée, n'est constituée que de nouvelles solutions, on parle de méthode IPI générationnelle. Après le remplacement, si l'algorithme n'est pas arrêté, une nouvelle itération est effectuée.

On retrouve, pour les IPI, des critères d'arrêt semblables à ceux des ISI. En effet, la méthode peut être stoppée après qu'un certain nombre d'itérations ait été effectué ; ou bien, après qu'un certain nombre d'itérations n'ait produit aucune amélioration. Cependant, on considère aussi un autre critère d'arrêt, spécifique aux IPI, appelé convergence. On dit qu'une IPI a convergé, lorsque presque toutes les solutions de la population sont identiques. Quand la convergence est atteinte, la combinaison ne peut plus produire de solutions originales et l'IPI ne peut plus produire d'amélioration. C'est pourquoi la convergence est naturellement utilisée comme critère d'arrêt. Cependant, la solution vers laquelle la méthode converge est souvent trouvée bien avant la convergence ; par conséquent, il est judicieux d'arrêter le processus itératif avant la convergence, après un nombre d'itérations déterminé expérimentalement, par exemple.

1.2.2 L'algorithme génétique

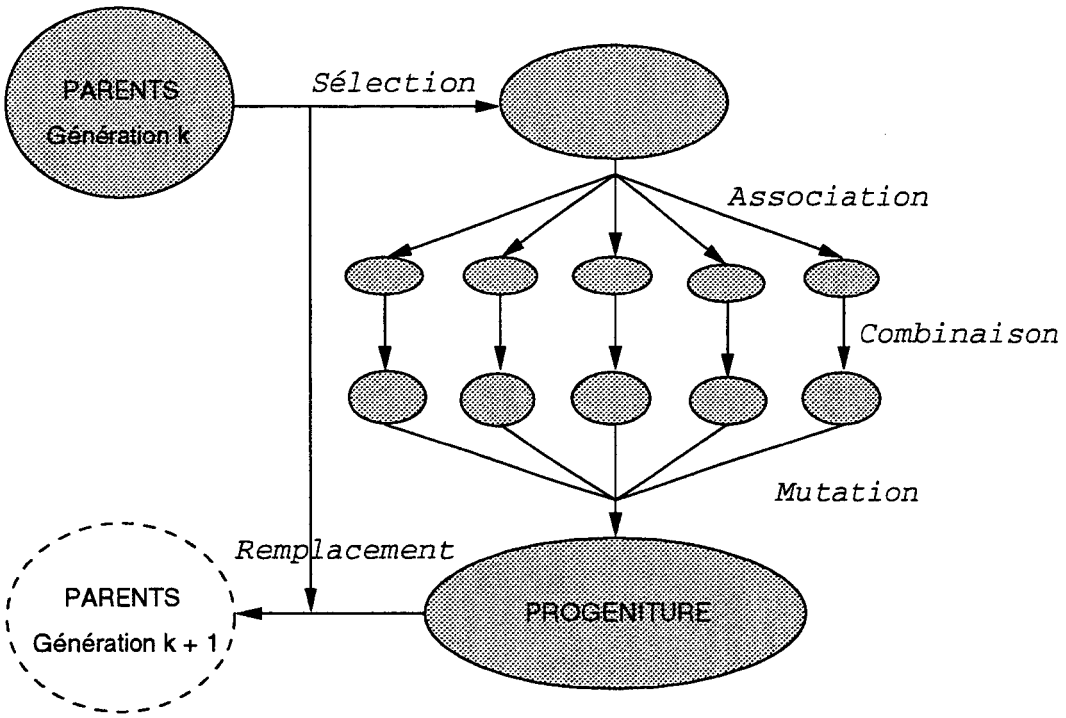


FIG. 1.9 - *Schéma du fonctionnement de l'algorithme génétique.*

L'algorithme génétique (GA – *Genetic Algorithm*) est inspiré des mécanismes d'évolution des populations d'organismes biologiques. Dans la nature, selon les principes de sélection naturelle décrits par Charles Darwin, les êtres vivants sont en compétition pour se nourrir comme pour se reproduire. Ainsi, au fil des générations, les organismes « les plus forts » (les mieux adaptés) se nourrissent et se reproduisent mieux et finissent par s'imposer en nombre dans la population. Au milieu des années 1970, John Holland dégage de son étude des processus d'adaptation naturelle une théorie de l'adaptation [Holland75]. Il s'appuie sur cette théorie pour construire des systèmes artificiels capables d'adaptation: c'est la naissance de l'algorithmique génétique. Aujourd'hui, les GA démontrent leur efficacité dans la résolution des problèmes dynamiques et des problèmes fortement complexes [Goldberg89]. On les retrouve également dans le domaine de l'optimisation combinatoire, parfois associés à d'autres heuristiques. Pour exposer les principes fondamentaux des GA (voir Fig. 1.9), nous nous sommes principalement inspirés de l'Algorithme Génétique Canonique (AGC) défini par J. Holland pour ses travaux théoriques [Beasley et al.93a, Beasley et al.93b, Whitley93].

Non seulement la méthode mais aussi la terminologie sont empruntées à la biologie. Ainsi, un GA est une méthode IPI dont les solutions sont appelées des individus

(ou chromosomes, ou phénotypes), les itérations sont appelées des générations et la qualité d'une solution est appelée sa *fitness*. Dans l'AGC, le génotype (codage des individus) est une chaîne de bits. Un individu est donc une suite de valeurs binaires. Un génotype de longueur L permet donc de coder 2^L valeurs différentes. Cependant, le codage des individus est un facteur prépondérant de l'efficacité d'un GA. Ainsi, pour de nombreux problèmes, le codage basé sur l'alphabet binaire n'est pas efficace ; donc, on utilise un codage mieux adapté, spécifique au problème. Pour que le GA soit efficace, la population initiale (souvent générée aléatoirement) doit présenter une grande diversité de phénotypes afin d'augmenter la probabilité d'avoir dans la population les caractères (allèles) d'un individu de bonne qualité. Par conséquent, la taille (nombre d'individus) de la population est un paramètre déterminant pour un GA, comme pour les autres IPI d'ailleurs. D'une manière générale, la sélection est un mécanisme stochastique. Une fonction d'évaluation, dénommée fonction *fitness*, donne le niveau d'adaptation d'un individu ; ce niveau est encore appelé la *fitness* ou la performance de l'individu. La sélection détermine la probabilité de reproduction d'un individu en fonction de sa *fitness*. Un individu obtient une probabilité de reproduction d'autant plus grande qu'il est bien adapté. Une fois les probabilités établies, les candidats à la reproduction sont tirés aléatoirement pour constituer la population des individus sélectionnés. Dans L'AGC original, la probabilité de reproduction est une fonction linéaire de la *fitness*. C'est la méthode de la roulette, appelée ainsi car on sélectionne les candidats à la reproduction à l'aide d'une roulette de casino sur laquelle chaque individu est représenté par un secteur dont l'angle est proportionnel à sa *fitness*.

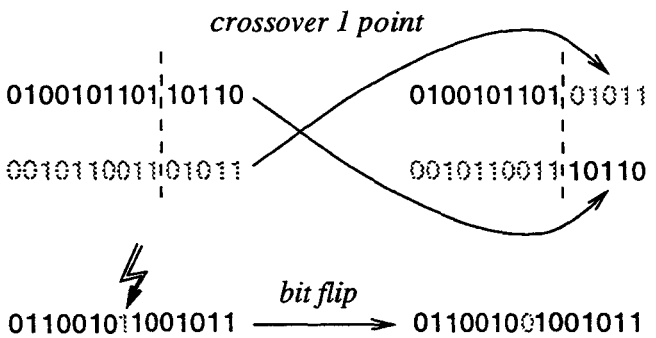


FIG. 1.10 - *Les opérateur de reproduction dans l'AGC.*

Dans un GA, la reproduction regroupe les phases d'association, de combinaison, et de transformation. Les opérateurs de reproduction, la recombinaison et la mutation opèrent sur les individus sélectionnés pour constituer la progéniture. La recombinaison correspond à la reproduction sexuée du monde naturel. Ainsi, pour la recombinaison, les individus sont associés par paires. On peut imaginer de nombreuses façons de grouper les individus en paires ; cependant, J. Holland préconise l'asso-

ciation d'un bon individu avec un autre choisi au hasard (uniformément). Quand les individus sont appariés, chaque paire peut être soumise à la recombinaison selon une certaine probabilité qui est un paramètre du GA. La recombinaison transforme deux individus en deux autres individus. Parmi les opérateurs de recombinaison fréquemment utilisés, on trouve les *crossovers* dont le principe de combinaison est l'échange de certaines fraction de phénotype. Pour le *crossover* 1 point, par exemple, un point de coupe est défini au hasard ; puis chaque phénotype est coupé ; et les parties coupées sont échangées (voir Fig. 1.10). La mutation, généralement appliquée après la recombinaison, pendant la phase de transformation, affecte légèrement le phénotype des individus. Elle peut modifier chaque individus selon une probabilité qui est un paramètre du GA, souvent de l'ordre 1 %. Pour l'AGC, la mutation, quand elle s'applique, change la valeur d'un bit (*flip*), choisi au hasard dans le phénotype de l'individu (voir Fig. 1.10). Le remplacement termine le cycle du GA pour produire une nouvelle génération de parents. L'AGC est un algorithme générationnel : la nouvelle génération occulte complètement les parents. Cependant, d'autres mécanismes de remplacement, basés sur le maintien dans la population de certains parents, ont montré parfois plus d'efficacité. Pour les GA k-élitistes, la technique de remplacement préserve les k individus les plus performants. Pour la méthode stable (*steady-state*), seuls un ou deux individus sont remplacés à chaque génération.

Dans les applications du GA au problème d'affectation quadratique, les solutions sont généralement représentées par des permutations sur lesquelles agissent des opérateurs de reproduction spécifiques. On utilise, par exemple, le croisement PMX [Grefenstette87] ou le *crossover* uniforme de Tate et Smith [Tate et al.95]. Pour la résolution du QAP, l'algorithme génétique n'est pas très performant, notamment sur les instances peu structurées [Brown et al.89] [Maniezzo et al.94]. En effet, le GA, s'il parvient à caractériser une région intéressante de l'espace de recherche, manque d'intensification. Les opérateurs de reproduction ne parviennent pas à raffiner suffisamment les bonnes solutions [Bachelet et al.96]. C'est pourquoi, l'algorithme génétique est souvent associé à une heuristique exploiteuse, une ISI, pour gagner en robustesse. Cependant, Tate et Smith proposent un algorithme génétique robuste pour le QAP. Cette méthode met en concurrence le croisement et la mutation qui est appliquée avec un taux élevé. Les auteurs relatent de bons résultats sur de petites instances structurées [Tate et al.95].

1.2.3 La recherche par dispersion

La recherche par dispersion (SS – *Scatter Search*) est une heuristique évolutionniste qui date d'une vingtaine d'années (voir Fig. 1.11) [Glover77]. Aujourd'hui, la SS est répertoriée dans une classe d'heuristiques appelée *path relinking* [Glover98]. Dans une SS, on utilise trois opérateurs : un opérateur de dispersion, un opérateur de recombinaison et un opérateur d'optimisation. L'opérateur de dispersion est utilisé pour générer une population initiale bien dispersée dans l'espace de recherche ; on

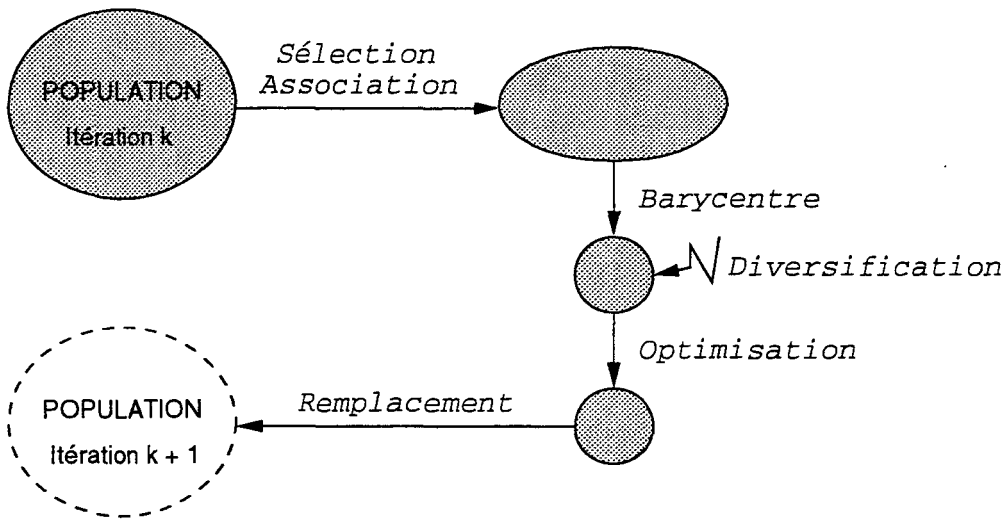


FIG. 1.11 - *Schéma du fonctionnement de la recherche par dispersion.*

l'emploie aussi, après un certain nombre d'itérations, pour contrarier la convergence de la méthode en restaurant un bon niveau de diversité dans la population courante. Pour commencer, la population initiale est obtenue en deux temps. D'abord une phase de diversification à partir d'une population de solutions « graines » ; puis, une phase d'optimisation des solutions obtenues ; l'opérateur est généralement une ISI rapide[10], une méthode de descente par exemple. Dans la phase de sélection, un petit nombre de bonnes solutions sont sélectionnées et sont associées en une seule sous-population pour être recombinées. La sélection permet de contrôler le pouvoir d'intensification de la recherche. En effet, pour intensifier la recherche, on sélectionne des solutions très proche, alors que pour explorer on choisit des solutions différentes. Dans de nombreuses mises en œuvre, la recherche par dispersion utilise, comme opérateur de combinaison, le barycentre des solutions pondérées par leur qualité. Cependant, tout autre opérateur qui parvient à regrouper les caractéristiques valorisantes des différentes solutions convient ; l'essentiel, comme souvent dans les méthode IPI, est d'intégrer un opérateur de combinaison bien adapté au problème à résoudre. Durant la phase de transformation, les nouvelles solutions sont rendue réalisables si nécessaire puis sont optimisées. Le remplacement est élitiste : les meilleures solutions de la population courante sont préservées pour qu'elles puissent être combinées à nouveau. Cependant, cette stratégie intensificatrice entraîne une forte convergence. Par conséquent, il est souvent nécessaire, pour rétablir l'équilibre

10. Qui s'exécute rapidement.

entre l'intensification et la diversification, d'introduire des mécanismes de diversification supplémentaires. Cette diversification s'insère alors entre la phase de combinaison et la phase de remplacement. Ainsi, de nouvelles solutions, obtenues par diversification à partir des solutions créées par la combinaison, sont intégrées à la population courante.

En 1996, Cung *et al.* ont proposé une recherche par dispersion pour le QAP [Cung et al.96]. Pour la recombinaison, ils utilisent une combinaison linéaire d'un petit nombre de solutions choisies au hasard parmi les meilleures de la population. Pour la projection, ils ont conçu une heuristique spécifique rapide ; cette heuristique est aussi utilisée pour introduire de la diversité. Pour l'optimisation, leur SS met en œuvre une recherche tabou simple. Les résultats obtenus sur des instances de natures variées montrent que cette heuristique est performante. Cependant, le fait d'utiliser une recherche tabou comme opérateur d'optimisation fait de cette SS une recherche sophistiquée. Ainsi, à nos yeux, cette heuristique ressemble plus à une méthode hybride IPI/ISI qu'à une simple recherche à population de solutions. Ainsi, la SS de Cung *et al.* parvient à un bon équilibre exploration / exploitation, qui explique sa performance.

1.2.4 Le système de fourmis

Le système de fourmis (AS – *Ants System*) est une méthode d'optimisation proposée par Dorigo en 1991 dans le cadre de sa thèse de doctorat [Colorni et al.91] [Dorigo et al.91] [Dorigo92]. Cette métaheuristique est inspirée du comportement des colonies de fourmis réelles pour optimiser le trajet du nid à une source de nourriture. Dans la nature, les fourmis déposent le long de leur trajet des substances odorantes (phéromones) qui marquent leur passage. Statistiquement, dans un temps donné, un trajet plus court reçoit plus de phéromones qu'un trajet plus long car plus de fourmis l'empruntent. Ainsi, certains chemins deviennent plus marqués que d'autres. Or les fourmis choisissent plus favorablement le trajet le plus marqué ; en conséquence le principe de marquage s'amplifie[11]. De plus, la phéromone s'évapore ; ainsi, à terme, seul le plus court chemin exploré demeure marqué.

Depuis l'apparition des AS comme technique d'optimisation, de nombreuses variantes ont été proposées, plus ou moins proches du comportement des fourmis réelles. L'AS que nous présentons, ci-après, bien qu'il s'éloigne un peu du mécanisme naturel apparaît plus performant que d'autres AS qui sont plus proches du comportement réel (voir Fig. 1.12). Dans cet AS, une fourmi déposera d'autant plus de phéromones que le trajet sera meilleur[12]. Ce type particulier d'AS est une IPI, dont la population courante rassemble les différents trajets (solutions) en cours d'améliora-

11. Ce type de mécanisme est dit autocatalyseur.

12. Ce qui suppose que le chemin soit parcouru avant que la phéromone soit déposée, ce qui n'est pas le cas réel.

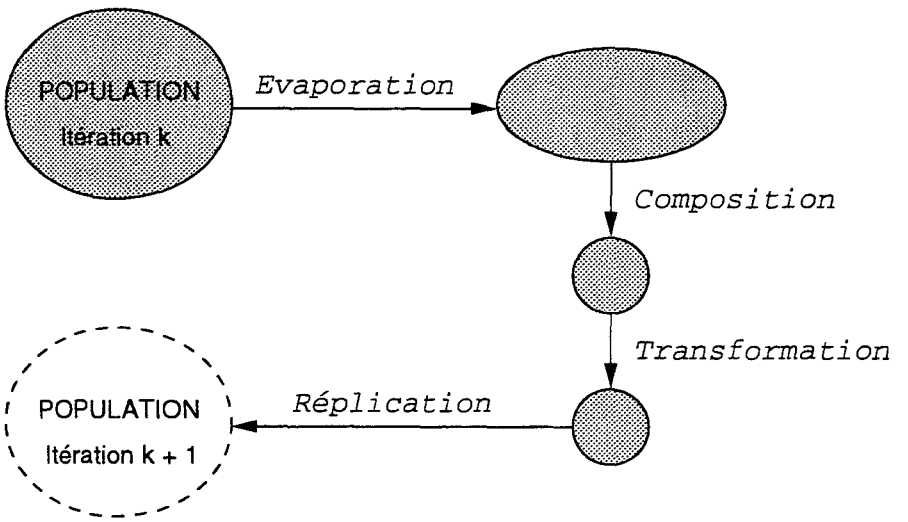


FIG. 1.12 - *Schéma du fonctionnement du système de fourmis.*

tion, et dont l'algorithme figure une colonie de fourmis. Une solution étant composée de différents attributs, comme un trajet est composé de tronçons, la quantité de phéromones sur une portion de trajet correspond au nombre d'occurences d'un attribut dans la population. Ainsi, pour que la méthode soit efficace, la population initiale doit présenter le maximum d'attributs possibles en proportion égale. D'une façon imagée, la population initiale est composée de trajets composés d'une grande diversité de tronçons. Cette propriété donne, au départ de la méthode, une probabilité de sélection uniforme pour tous les attributs possibles.

La phase de sélection reproduit le phénomène naturel d'évaporation des phéromones. Ainsi, les solutions dont les attributs sont très peu fréquents dans la population ne sont pas sélectionnées. Autrement dit, les trajets composés de tronçons faiblement pourvus de phéromones ne sont pas sélectionnés pour composer de nouveaux trajets. Toutes les solutions sélectionnées sont associées. Lors de la combinaison, les attributs les plus fréquents dans la population sont extraits selon une loi probabiliste pour composer de nouvelles solutions. Cela imite le comportement des fourmis réelles choisissant plus favorablement un tronçon riche en phéromones. Ainsi, la combinaison regroupe, au sein d'une même solution, des attributs fortement marqués avec l'objectif de produire une solution de meilleure qualité. Souvent, le choix des attributs est également guidé par une heuristique gloutonne. Cette heuristique permet d'accélérer le démarrage de la méthode en fournissant rapidement des solutions de bonne qualité. Les nouvelles solutions composées sont alors trans-

formées ; elle sont rendues réalisables si nécessaire. Dans la phase de remplacement, les solutions obtenues sont répliquées ; elles sont clonées d'autant plus qu'elles sont bonnes, ce qui correspond à déposer d'autant plus de phéromones que le trajet est court. Enfin, les clones sont ajoutés à la population.

Le système de fourmis a été appliqué, avec succès, à de nombreux problèmes classiques : notamment le TSP, le JSP et le QAP [Dorigo et al.96] [Colorni et al.94] [Maniezzo et al.98]. Cependant, comme les autres IPI, le AS n'est pas très exploiteur ; il ne parvient pas à améliorer finement les solutions qu'il produit. C'est pourquoi, le système de fourmis a été amélioré en lui associant une recherche locale qui s'applique après la composition des nouvelles solutions. Pour le QAP, le système de fourmis a été hybridé avec une recherche tabou [Taillard et al.97] [Talbi et al.99b]. Malgré l'hybridation avec une recherche tabou, Taillard et Gambardella constatent que leur algorithme manque de robustesse et demeure plus performant sur les instances structurées du QAP.

1.2.5 Parallélisation des méthodes IPI

La parallélisation des IPI est fortement inspirée des travaux de parallélisation des algorithmes génétiques qui sont, de loin, les méthodes IPI les plus étudiées [Talbi95] [Schwehm94] [Kapsalis et al.94]. Le principe de fonctionnement des IPI repose sur l'évolution d'une population. Dans les phases de sélection, d'association et de remplacement, l'IPI doit gérer la globalité de la population. C'est un handicap pour la parallélisation puisque la gestion d'une mémoire globale d'un grand volume (toute la population) requière beaucoup d'accès sur une architecture à mémoire distribuée, apportant un surcoût prohibitif. Ainsi, les différents modèles de parallélisation des IPI reposent sur les différentes façons de distribuer la population en sous-populations sur les processeurs, chaque sous-population évoluant plus ou moins indépendamment des autres.

1.2.5.1 La parallélisation centralisée

Dans les métaheuristiques à population de solutions, ce sont les phases de production de nouvelles solutions qui demandent le plus d'effort de calcul. Le modèle de parallélisation centralisée, de type maître/esclaves, permet de conserver la nature séquentielle de la méthode tout en distribuant les phases coûteuses (voir Fig. 1.13). Dans cette parallélisation, le processeur maître assure le déroulement de la méthode IPI et réalise les étapes de sélection, d'association et de remplacement, qui demandent une gestion globale de la population. Au commencement de la phase de combinaison, le maître distribue la population sur les processeurs esclaves. Chaque esclave travaille sur une sous-population de solutions et réalise les phases de combinaison puis de transformation. Après la transformation, les esclaves retournent au

maître les solutions qui viennent d'être créées.

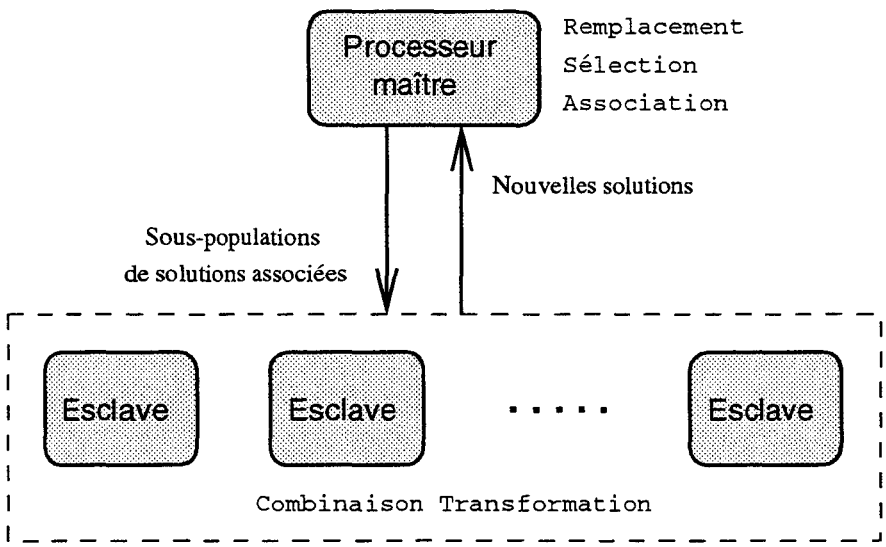


FIG. 1.13 - *IPI: parallélisation centralisée de type maître/esclaves.*

Cette parallélisation de bas niveau n'altère en rien le comportement de l'IPI qui fournit les mêmes résultats qu'une méthode séquentielle, mais plus rapidement. Elle offre une bonne accélération quand le temps de communication des solutions est faible par rapport au temps de production des solution nouvelles. Ainsi, cette parallélisation est bien adaptée pour une combinaison et/ou une transformation qui demandent un gros effort de calcul. Cependant, ce type de parallélisation, qui imite le comportement d'une IPI séquentielle, n'est plus très utilisé aujourd'hui car d'autres formes de parallélisation augmentent les performances de l'algorithme. Macfarlane et East comparent le modèle centralisé aux modèles parallèles insulaire et cellulaire [Macfarlane et al.90]. Leurs algorithmes génétiques parallèles sont implémentés sur une ferme de transputers. Ils sont évalués sur l'optimisation des fonctions de test de De Jong [Jong75]. Les auteurs concluent à la supériorité des modèles cellulaire et insulaire, en terme de qualité de solution, par rapport à l'algorithme génétique parallèle de bas niveau. Ils expliquent cette supériorité en précisant que les modèles insulaire et cellulaire préservent mieux la diversité de la population.

1.2.5.2 La parallélisation insulaire

La parallélisation insulaire, aussi appelée modèle parallèle en îles, tient son nom de sa ressemblance avec la manière dont évoluent certaines espèces d'oiseaux sur des îles relativement éloignées entre elles. Sur chaque île, une sous-population d'oiseaux évolue indépendamment des autres. Mais, de temps à autres, quelques oiseaux

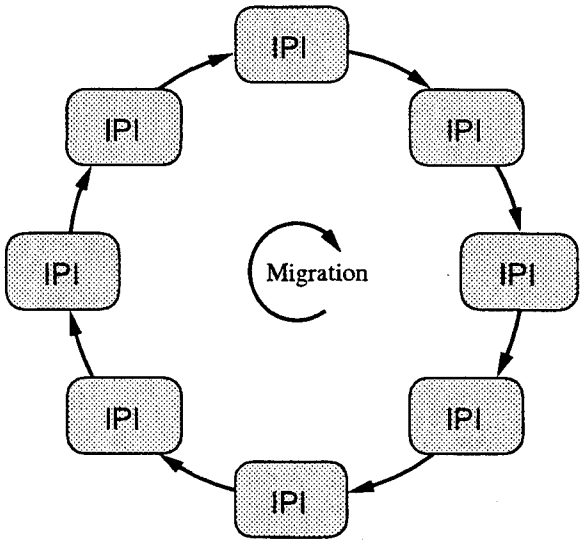


FIG. 1.14 - *Modèle de parallélisation IPI en îles sur un anneau de processeurs.*

migrent vers d'autres îles, mixant ainsi le matériel génétique des sous-populations. Ainsi, dans le modèle de parallélisation insulaire des IPI (voir Fig. 1.14), chaque processeur est en charge de l'évolution d'une sous-population ; il exécute toutes les phases de la méthode IPI, de la sélection au remplacement. Après un certain nombre d'itérations ou quand un certain degré de convergence est atteint (ou un autre critère), la migration est activée : le processeur envoie des solutions vers d'autres processeurs et en reçoit. Une fois de nouvelles solutions reçues et intégrées à la population courante, l'IPI reprend les itérations. Bien que la migration ne nécessite pas la synchronisation des processeurs, la plupart des implémentations réalisées à ce jour sont synchronisées pour la migration. L'implémentation standard consiste en l'exécution, sur chacun des processeurs de l'algorithme décrit à la figure 1.15.

De nombreux auteurs ont relaté leur expérience de la parallélisation insulaires des algorithmes génétiques, notamment Cohoon [Cohoon et al.87] et Tanese [Tanese87]. Comme ce modèle de parallélisation peut être considéré comme une méthode hybride, nous le présentons d'une manière plus détaillée, en tant que méthode hybride coévolutionniste, en partie III.

1.2.5.3 Le modèle parallèle cellulaire

Le modèle cellulaire de l'IPI est une approche totalement distribuée selon un parallélisme à granularité fine. La population est distribuée, à raison d'une[13] solution par processeur, sur une architecture partiellement connectée, par exemple une grille

13. Le modèle reste valable si chaque processeur est en charge d'une sous-population de solutions.

```
     -- sur chaque processeur --
     Début
     Choisir une sous-population
a.   Sélection - Association
     Combinaison - Transformation
     Remplacement
     Si critère d'interruption
        alors
          aller en b.
     aller en a.
b.   Envoi de solutions
     -- SYNCHRONISATION --
     Réception de solutions
     Intégration des solutions reçues
     Si Critère d'arrêt
        alors
          aller en c.
     aller en a.
c.   fin
```

FIG. 1.15 - *Séquencement du modèle de parallélisation en île sur un processeur.*

torique de processeurs (voir Fig. 1.16). C'est un modèle complètement synchrone dans lequel chaque processeur exécute le même algorithme (voir Fig. 1.17). En effet, chaque processeur exécute l'IPI sur une sous-population constituée de la solution qu'il détient et des solutions des processeurs qui lui sont connectés (ses voisins). Ainsi, il commence par envoyer sa solution à ses processeurs voisins, desquels il reçoit des solutions sur lesquelles il applique les étapes de l'IPI en terminant par le remplacement de sa solution.

Cette parallélisation, par rapport à d'autres modèles, se distingue par une sélection fortement locale, puisque cette dernière est limitée à un processeur et ses voisins. Cette localité de la sélection offre l'avantage d'amoindrir la vitesse de convergence de l'IPI, ce qui conduit à une meilleure exploration de l'espace de recherche, *a fortiori* si le nombre de processeurs est grand. Cette parallélisation à grain fin est bien adaptée aux architectures massivement parallèles telles que la *Maspar* MP ou *Connection Machine* CM-2. Quelques implémentations de GA insulaire ont produit de très bon résultats pour différents problèmes d'optimisation ; pour plus d'information, le lecteur consultera [Spiessens et al.91] [Collins et al.91] [Talbi et al.91]. Nous avons expérimenté ce modèle sur une architecture massivement parallèle (*Maspar*

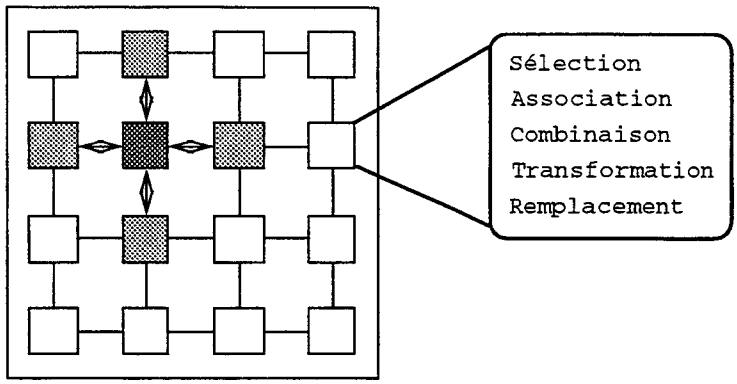


FIG. 1.16 - *Modèle cellulaire : IPI parallèle à grain fin.*

MP-1) dotée de 16000 processeurs, connectés en une grille torique de connexité 4, pour résoudre le QAP avec un GA (voir chapitre 2).

```
     -- sur chaque processeur --
     Début
     Choisir une solution initiale
a.   Envoi de la solution courante
     Réception de solutions voisines
     Intégration des solutions reçues
     Sélection - association
     Combinaison - Transformation
     Remplacement de la solution courante
     Si Critère d'arrêt
        alors
          aller en b.
     aller en a.
b.   fin
```

FIG. 1.17 - *Séquencement du modèle de parallélisation cellulaire.*

1.3 Conclusion

Dans ce chapitre, nous avons dressé un rapide état de l'art des métaheuristiques itératives et de leurs parallélisations. Notre approche distingue les méthodes à solution unique (ISI) des méthodes à population de solutions (IPI) : nous avons constaté que ces deux catégories ont des modèles de parallélisation différents ; nous montreront dans la suite qu'elles ont aussi des performances divergentes. Ainsi, ce chapitre est une synthèse de notions générales à propos des métaheuristiques itératives parallèles. Il est une introduction et un support à la présentation, au chapitre suivant, des métaheuristiques parallèles que nous avons réalisées pour résoudre le QAP.

Chapitre 2

Des métaheuristiques parallèles
pour le QAP

Dans ce chapitre, nous donnons les caractéristiques des implémentations de quelques métaheuristiques parallèles ISI et IPI, que nous avons réalisées pour résoudre le problème d'affectation quadratique. Nous avons utilisé différents type de machines parallèles : un ordinateur massivement parallèle, un cluster de 16 processeurs, et un réseau de stations de travail. Nous décrivons des expériences qui concernent un algorithme génétique, une méthode de descente et une recherche tabou dans des versions parallèles non hybridées. Nous spécifions les méthodes implémentées, les plates-formes et les environnements d'exécution, ainsi que les protocoles d'évaluation. Nous relatons l'observation de variations de performances des différentes heuristiques sur les différents types d'instances du QAP correspondant aux différents paysages identifiés au chapitre I:2.

2.1 Des GA massivement parallèles

Nous avons réalisé un algorithme génétique parallèle. Pour cela, nous avons choisi le modèle cellulaire (voir section 1.2.5.3). Par rapport aux autres, cette parallélisation, apporte une dimension spatiale à la méthode. En effet, les individus sont répartis dans l'espace que constitue le réseau de processeurs. Ainsi, un individu évolue par rapport à son entourage proche, ce qui réduit la pression de sélection et maintient la diversité. Un autre avantage de l'algorithme génétique cellulaire (CGA – *Cellular Genetic Algorithm*) concerne le contrôle de la convergence. En effet, le phénomène de convergence apparaît d'une manière locale en formant des régions convergées dans l'espace des processeurs. Il est alors possible d'intervenir localement dans ces régions, pour empêcher la convergence. Nous avons inséré cette technique interventionniste dans le CGA pour obtenir une méthode que nous appelons le CGA multidépart (MCGA

– *Multistart CGA*).

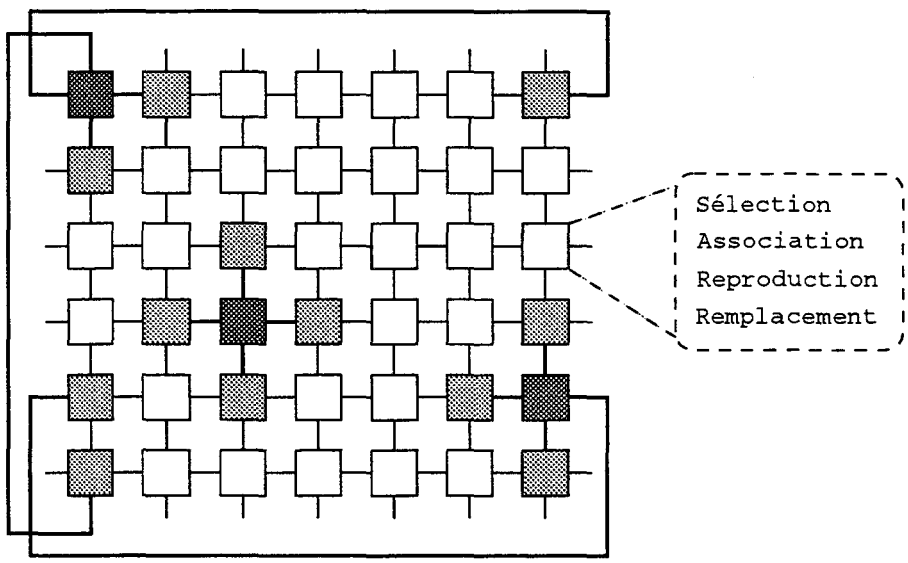


FIG. 2.1 - *Le CGA sur une grille torique de connexité 4. La figure montre plusieurs processeurs (gris foncé) et leurs 4 voisins respectifs (gris clair).*

2.1.1 L'algorithme parallèle cellulaire CGA

Nous avons conçu le CGA pour une grille de processeurs torique de connexité 4 (voir Fig. 2.1). Les paramètres du CGA sont l'ordre du réseau (les dimensions de la grille), les taux de recombinaison et de mutation, et le nombre d'itérations à exécuter. Le CGA se déroule comme suit. D'abord, la population initiale, choisie au hasard selon une distribution uniforme, est « mappée » sur la grille à raison d'un individu par processeur. Puis l'algorithme enchaîne ses itérations (voir Fig. 2.2). Sur chaque processeur, l'individu courant est associé avec le meilleur de ces quatre voisins directs (gauche, droite, haut, bas). Avec une probabilité égale au taux de recombinaison, le croisement est réalisé, produisant deux enfants. Pour le remplacement de la solution courante, le meilleur des enfants est mis en concurrence avec la solution courante et la solution résultant de la mutation de la solution courante. La mutation est opérée en fonction du taux de mutation. Pour la résolution du QAP, l'opérateur de recombinaison est le *crossover* PMX que nous décrivons par la suite à la section 2.1.2 ; l'opérateur de mutation est la transposition décrite en I:2.2.1.1.

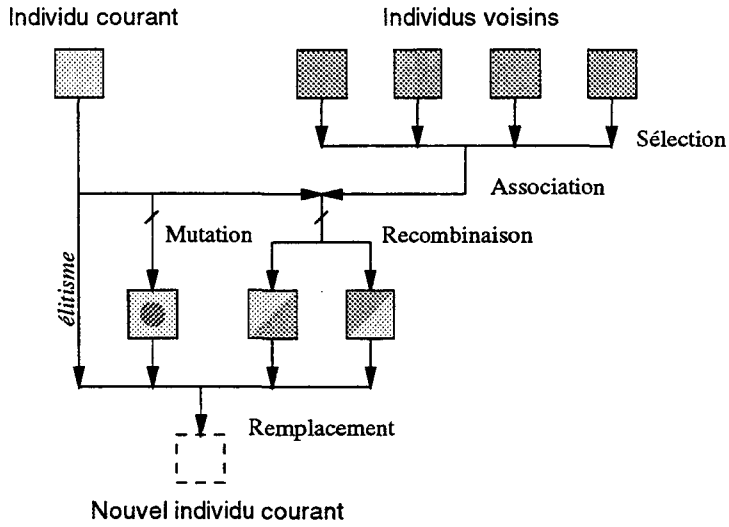


FIG. 2.2 - *Déroulement d'une itération (sur un processeur) dans le CGA.*

2.1.2 La recombinaison de type PMX

Pour coder les solutions du problème d'affectation quadratique, on utilise, presque toujours des permutations car c'est un codage très naturel. Pour employer ce codage dans un algorithme génétique, il faut utiliser des opérateurs spécifiques. Le PMX (*Partially Mapped Crossover*) est une recombinaison adaptée au croisement de deux permutations [Goldberg et al.85] [Grefenstette87]. Pour nos GA pour le QAP, nous avons conçu un *crossover* de type PMX, c'est un *crossover* deux points avec réparation des solutions obtenues pour retrouver deux nouvelles permutations. Le PMX que nous avons implémenté s'exprime comme suit : soient p et q, deux permutations sur $[1,n]$. Soient a et b, pris au hasard dans $[1,n]$ selon une loi uniforme et tel que $a < b$. Soient (p',q') le couple de permutations issues du PMX de (p,q) avec a et b pour points de coupe. On obtient p' de la façon suivante[1]. D'abord, on croise les deux permutations comme dans un *crossover* deux points classique ; on a :

$$p'(i) = \begin{cases} p'(i) = q(i),\ i \in [1,a] \\ p'(i) = p(i),\ i \in\ ]a,b] \\ p'(i) = q(i),\ i \in\ ]b,n] \end{cases}$$

À ce stade, p' n'est pas nécessairement une permutation. On doit parfois « réparer » p' pour qu'il devienne une permutation. Pour ce faire, on applique la méthode suivante qui élimine les doublons :

$$\text{tant que } \exists i,\ i \notin\ ]a,b],\ \exists j,\ j \in\ ]a,b]\ /\ p'(i) = p'(j) \text{ alors } p'(i) \leftarrow q(j)$$

1. On obtient q' d'une manière similaire, en remplaçant p par q et p' par q'.

Par exemple :

$$p = (1,2,3,4,5) \qquad q = (4,3,5,1,2)$$

les points de coupes étant $a = 2$ et $b = 4$ on obtient p', dans un premier temps, en croisant p et q :

$$\begin{array}{lcccccccc} p = & 1 & 2 & | & 3 & 4 & | & 5 \\ q = & 4 & 3 & | & 5 & 1 & | & 2 \end{array} \longrightarrow \quad p' = \quad 4 \quad 3 \quad | \quad 3 \quad 4 \quad | \quad 2$$

À ce stade, p' contient des doublons qu'il faut éliminer pour obtenir une permutation. En effet, $p'(1) = p'(4)$ et $p'(2) = p'(3)$; en suivant notre méthode, on remplace $p'(1)$ par $q(4)$ et $p'(2)$ par $q(3)$. On obtient alors une permutation :

$$p' = \quad 1 \quad 5 \quad | \quad 3 \quad 4 \quad | \quad 2$$

2.1.3 MCGA : un CGA *multistart*

Le CGA multidépart est très semblable au CGA mis à part que l'algorithme est interrompu plusieurs fois, après un certain nombre k d'itérations, pour une intervention qui limite la convergence. k est un paramètre de la méthode qui s'ajoute aux paramètres du CGA. Lors d'un arrêt (après k itérations), sur chaque processeur, la solution courante est comparée à celle de l'arrêt précédent. Si l'amélioration, en terme de *fitness*, n'est pas satisfaisante, alors la solution courante est remplacée par une nouvelle solution générée aléatoirement. Pour estimer s'il y a eu une progression de la *fitness* suffisante sur un processeur, nous utilisons un seuil A (par exemple $A = 1\ \%$), qui est un paramètre du MCGA. La solution courante s est remplacée dans le cas où $\varphi(s) < (1+A)\,\varphi(s')$, s' étant la solution courante du précédent arrêt, k itérations plus tôt. Cette intervention permet de limiter la convergence qui apparaît partiellement, dans certaines zones du réseau de processeurs. Quand une zone converge, les individus deviennent semblables et la zone devient stérile car la diversité des phénotypes décroît. Cependant, en même temps, l'algorithme reste productif dans d'autres zones du réseau de processeurs et il n'est pas souhaitable d'arrêter la méthode à cause de cette convergence partielle. Alors, plutôt que de laisser « tourner » les processeurs inutilement dans une zone ayant convergée, on injecte de nouvelles solutions aléatoires. De plus, les nouvelles solutions injectées apportent un matériel génétique neuf, source de diversité, qui ne peut être que profitable à la performance de l'heuristique.

Afin d'observer le phénomène de convergence locale, nous avons réalisé un outil de visualisation de l'évolution des algorithmes génétiques (Xgenetic[2]). Grâce à cet

2. Cet outil a été développé par G. Delmond et F. Diers (encadrés par V. Bachelet et E-G. Talbi) au cours de leur formation du DESS Génie Informatique, Intelligence Artificielle, et Génie Logiciel de l'Université Lille I, en 1996.

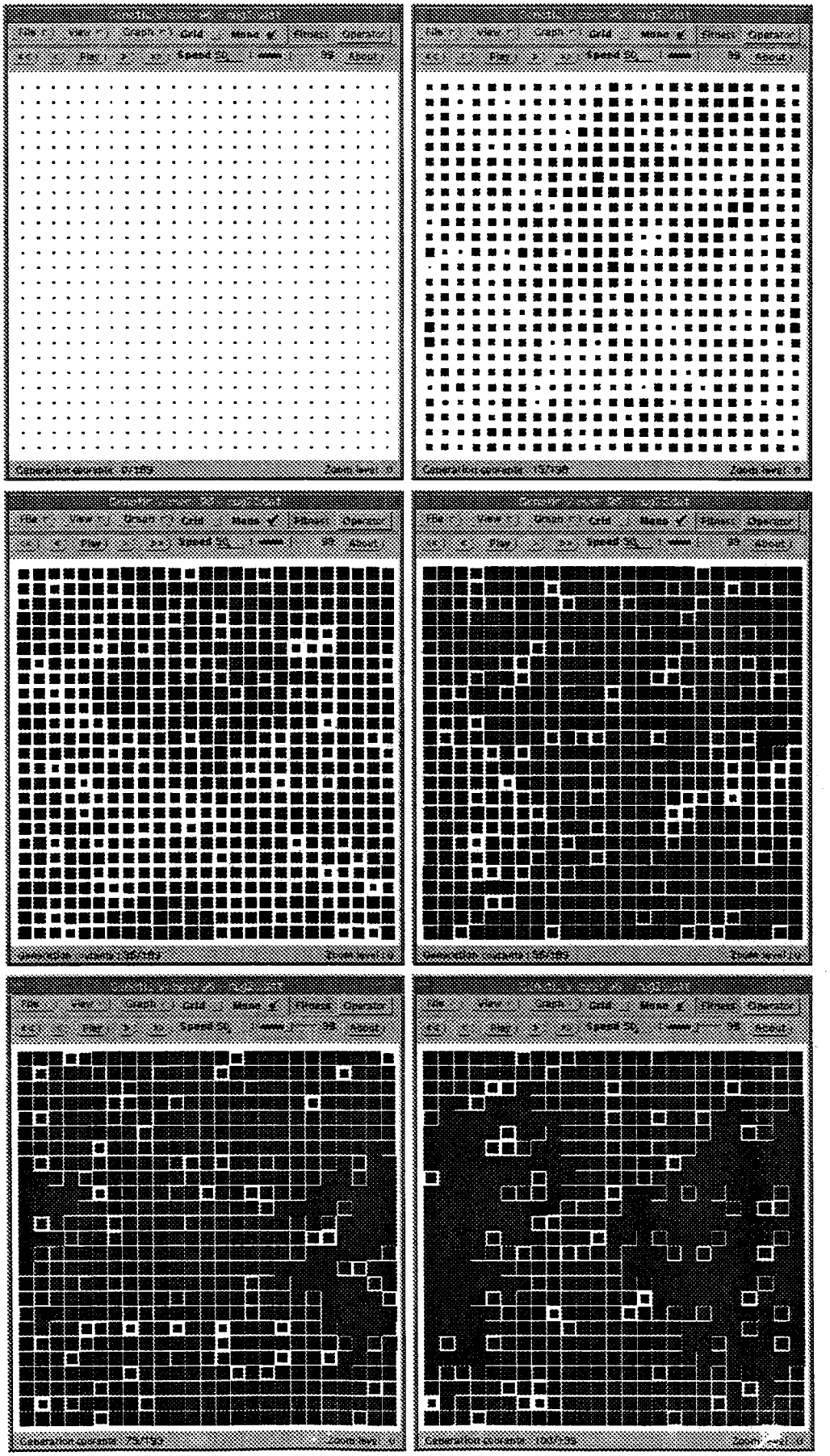


FIG. 2.3 - *Dans les algorithmes génétiques cellulaires, la convergence apparaît localement puis s'étend.*

outil, il est possible d'observer la convergence locale, puis les effets de l'injection de solutions nouvelles. La figure 2.3 montre quelques images de outil `Xgenetic` : les 6 images, vues de gauche à droite, puis de haut en bas, montrent l'évolution de l'algorithme `CGA` pour l'instance Nug20, sur un réseau de 25×25 cellules (processeurs). Dans `Xgenetic`, plus la fitness d'un individu est grande, plus le carré noir (sur la grille) qui représente l'individu est grand. Les images correspondent respectivement aux générations : 0, 15, 35, 55, 75, et 100. On observe que la convergence apparaît localement (taches sombres) puis s'étend (zones sombres plus grandes).

2.1.4 La plate-forme et l'environnement d'exécution

Nous avons implémenté les `CGA` et `MCGA` sur une machine massivement parallèle `DEC-Maspar-MP-1` que nous décrivons ci-après. Nous avons, également implémenté les `CGA` et `MCGA` sur une ferme de 16 stations de travail `DEC-Alpha` connectées par un réseau à haut débit ; cette ferme est présentée à la section 2.2.1.

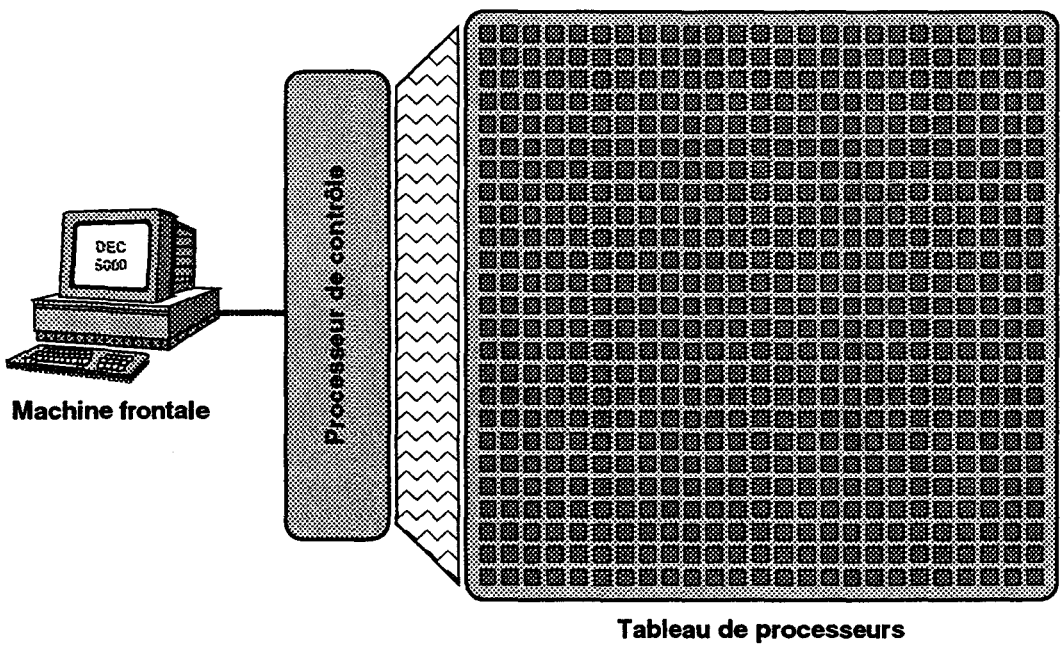


FIG. 2.4 - *Architecture de la machine massivement parallèle DEC-Maspar MP1. La Maspar contient un réseau torique (connexité 8) de 16 K processeurs de traitement qui sont pilotés par une même unité de contrôle (architecture SIMD). L'interface avec l'utilisateur est réalisée au travers d'une machine frontale qui est un DEC 5000.*

La `Maspar-MP1` est une machine de la classe `SIMD` (*Single Instruction, Multiple Data*) : un seul flot d'instructions pour tous les processeurs, et plusieurs flots de don-

nées, un flot par processeur. C'est une machine fortement synchrone où une seule unité de traitement contrôle simultanément tous les processeurs de traitement (voir Fig. 2.4). Ainsi, tous les processeurs exécutent simultanément la même instruction machine. L'architecture de la Maspar-MP1 est caractérisé par un réseau de processeurs formant une grille torique de connexité 8. Ce réseau contient 16 K processeurs 4 bits cadencés à 8 Mhz. Une DecStation 5000 sert de machine frontale et permet le lien entre l'utilisateur et la machine parallèle. La Maspar est une machine parallèle puissante car elle contient de nombreux processeurs mais elle ne permet l'exécution que d'algorithmes spécifiques, adaptés à son architecture en grille. De plus, c'est une machine mono-utilisateur ; cela impose la gestion d'un planning de réservation.

L'architecture de la Maspar-MP1 est particulièrement bien adaptée au GA cellulaire puisque c'est une grille de processeurs de connexité 8 dans laquelle il est facile de plonger une grille de connexité 4. Pour développer les applications CGA et MCGA, nous avons utilisé le langage MPL (*Maspar Parallel Application Language*) spécifique à la machine Maspar [Mpl93]. À raison d'un individu par processeur, soit 16384 solutions, cette architecture permet de traiter des populations de grande taille, mais la très grande taille de la population n'est pas nécessairement un facteur d'efficacité comme le présentons par la suite.

2.1.5 Évaluation des métaheuristiques CGA et MCGA

Nous avons réalisé l'évaluation des heuristiques CGA et MCGA sur un *cluster* de 16 processeurs Alpha. Sur cette machine, le modèle d'exécution parallèle que nous avons utilisé est du type SPMD[3], l'unité de programme correspondant à la gestion d'une cellule. Le modèle d'exécution parallèle SPMD n'impose pas de synchronisation au niveau des instructions du programme. Cependant, pour les CGA et MCGA, à chaque itération, il y a une synchronisation locale des programmes d'une cellule et de ces cellules voisines puisqu'une cellule doit connaître ses voisins. Les algorithmes ont été développés en C sur la machine virtuelle PVM (*Parallel Virtual Machine*) [Pvm94]. PVM est un outil qui permet de créer une machine parallèle (virtuelle) unique à partir de stations de travail, de PC, ou de super-ordinateurs. La bibliothèque de communication de PVM permet la réalisation d'applications parallèles par échanges de messages. Nous avons utilisé PVM, car cet outil est devenu un standard. Le fait que PVM soit répandu, apporte une grande portabilité à nos programmes, contrairement à nos développements sur la Maspar.

Pour l'évaluation de cette implémentation, la grille torique est d'ordre 16 (soit une population de 256 individus), chaque processeur étant en charge d'une colonne de 16 individus (voir Fig. 2.5). Nous avons expérimenté les CGA et MCGA sur des populations plus grandes sans obtenir de résultats meilleurs. De plus, en augmentant la taille de la population, on augmente beaucoup le coût de communication qui

3. *Single Program, Multiple Data* : un programme unique, des données multiples.

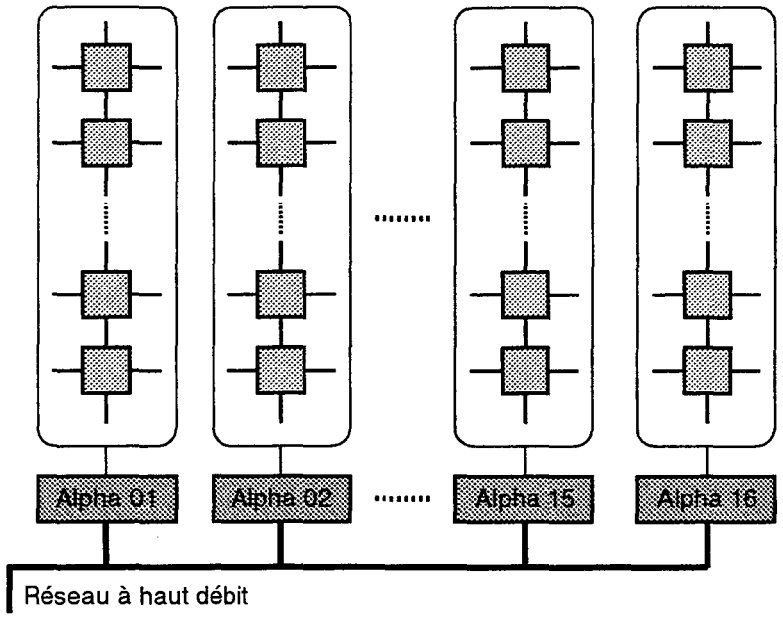


FIG. 2.5 - *Répartition des cellules des algorithmes CGA et MCGA sur le cluster de 16 processeurs Alpha à raison d'une colonne par processeur.*

devient plus important que le temps de calcul malgré le réseau à haut débit. Par exemple, pour une population de 16384 individus (comme sur la Maspar), à chaque itération, chaque processeur (une colonne d'individus) reçoit 256 solutions alors qu'il ne fait que 128 recombinaisons dans le pire des cas.

Pour toutes les instances traitées, nous avons utilisé les mêmes taux de reproduction (60 %) et de mutation (10 %). Ces paramètres ont été choisis expérimentalement après de nombreux essais pour les ajuster. Le nombre d'itérations effectuées correspond au nombre de générations nécessaires à la convergence de la population pour le CGA ; ce nombre est une valeur moyenne que nous avons déterminée expérimentalement. Le tableau 2.1 donne les résultats que nous avons obtenus sur quelques instances issues de la QAPlib. Nous remarquons que le MCGA est meilleur que le CGA pour la plupart des instances. Le MCGA obtient les meilleures solutions connues pour toutes les instances de taille inférieure à 30. Cependant, les résultats produits pour des instances de plus grande taille sont plutôt médiocres comparés à d'autres heuristiques simples, comme nous le montrons par la suite.

Par ailleurs, on constate que les GA ont une pauvre performance sur l'instance Esc128 alors que cette dernière est facilement résolue par une descente *multistart* comme nous le montrons par la suite. À l'inverse, l'instance Nug30 est résolue par le MCGA alors que la descente *multistart* n'y parvient pas. Ces phénomènes contradictoires s'expliquent en observant la nature de ces deux instances : Esc128 est une

Instance	Type	φ(QAPlib)	Meilleure		Moyenne		Nbr. d'itér.
			CGA	MCGA	CGA	MCGA	
Esc128	I	64	12.5000	**9.3750**	20.6250	**17.8125**	400
Els19	II	17 212 548	**0.0000**	**0.0000**	0.9673	**0.5078**	100
Nug20	-	2 570	1.3230	**0.0000**	2.8405	**2.0623**	200
Nug30	II	6 124	1.8289	**0.0000**	3.1189	**2.1228**	300
Esc32	-	2	**0.0000**	**0.0000**	**0.0000**	**0.0000**	30
Ste36a	-	9 526	4.9969	**1.5746**	10.0672	**6.4875**	350
Sko64	III	48 498	2.1609	**1.8681**	2.8063	**2.6265**	600
Sko90	-	115 534	**1.4593**	1.5026	2.2305	**2.0271**	1500
Sko100a	III	152 002	1.3618	**1.1974**	**2.0934**	2.1164	1500

TAB. 2.1 - *Évaluation des algorithmes génétiques cellulaires CGA et MCGA pour 10 exécutions. Les résultats sont présentés sous la forme* $100 \times \frac{\varphi(s)-\varphi(QAPlib)}{\varphi(QAPlib)}$ *où s est la solution fournie par la métaheuristique.*

instance uniforme de type I alors que Nug30 est une instance structurée de type II. Les GA sont plus performants sur les instances structurées (type II), car la diversité de la population initiale et celle engendrée par la reproduction produisent une bonne exploration de l'espace de recherche. Cette exploration permet d'isoler une ou plusieurs régions de bonne qualité dans lesquelles la population converge en les exploitant. En l'absence de structure (type I), l'algorithme génétique ne peut privilégier aucune région de l'espace de recherche, car toutes les régions ont la même qualité moyenne. Aussi la recombinaison n'est pas fructueuse et l'algorithme stagne. Nous reviendrons, par la suite, sur ces différentes performances des métaheuristiques en fonction de la nature des instances ; pour le moment, nous pouvons récapituler la performances des GA en fonction de la nature de l'instance à l'aide du tableau suivant :

	Type I	Type II	Type III
GA	−	+	−

2.2 Une Descente sur un *cluster* d'Alpha

Nous avons implémenté une descente de type SDW (voir 1.1.2) selon une parallélisation multidépart (voir section 1.1.5.2) sur le *cluster* 16-Alpha-farm que nous décrivons ci-après. Ce type de parallélisation permet d'exécuter rapidement un grand nombre de descentes indépendantes. Comme pour les GA, nous avons développé cette application sur la machine virtuelle PVM en langage C.

2.2.1 Le *cluster* d'Alpha

On appelle *cluster* un ensemble de machines puissantes connectées par un réseau performant. Le *cluster* que nous avons utilisé est composé de 16 processeurs Alpha 150 Mhz, muni chacun de 64 MO de mémoire. Les 16 machines sont connectées entre elles par un réseau optique d'un débit de 1 Giga bit. L'interface entre l'utilisateur et le cluster est réalisé au travers du réseau Ethernet (voir Fig. 2.6). Ce cluster est une machine parallèle très polyvalente car il permet l'exécution de programmes parallèles de toute granularité ; les hautes performances des processeurs et du réseau permettent même le parallélisme à grains fins pour lequel l'architecture cluster n'est pas adaptée *a priori*. L'alpha-farm a une architecture de classe MIMD (Multiple Instruction Multiple Data) : tous les processeurs peuvent exécuter des programmes différents manipulant des données différentes. Au contraire de la machine maspar (décrite à la section 2.1.4), la ferme d'Alpha est une machine multi-utilisateur et supporte des environnements de développement standard (C/PVM/OSF pour nous). C'est autant d'avantages qui nous ont fait préférer la ferme d'alpha à la Maspar pour nos études, la Maspar étant d'une utilisation moins « souple ».

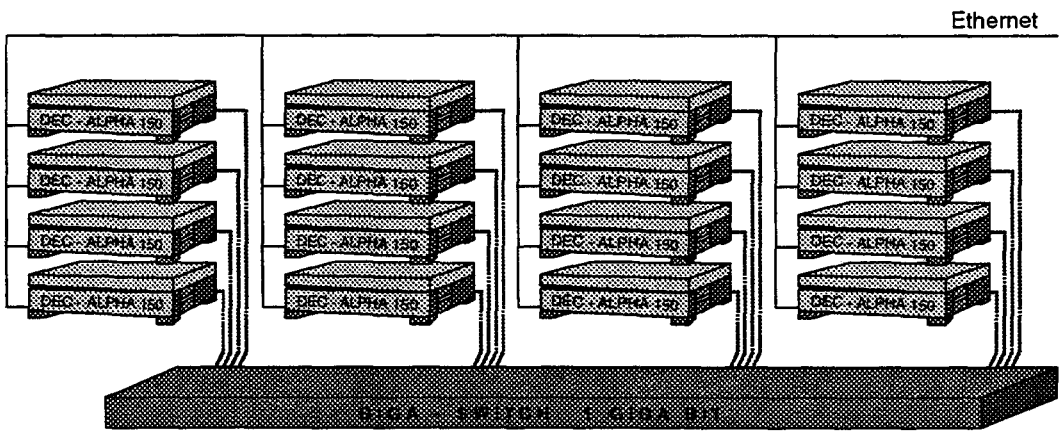


FIG. 2.6 - *Architecture du cluster* **16-alpha-farm** *composé de 16 processeurs DEC 150 connectés par un réseau optique à haut débit. L'interface avec l'utilisateur est réalisée au travers d'un réseau Ethernet.*

2.2.2 Évaluation incrémentale du voisinage

Dans les méthodes ISI que nous avons implémentées pour résoudre la QAP, la descente SDW *multistart* et la TS *multistart*, nous utilisons le même voisinage basé sur l'opérateur de transposition (voir section I:2.2.1.1). Le voisinage d'une solution, ainsi défini, s'évalue au premier abord en $O(n^4)$. En effet, il s'agit de calculer pour

chacun des $\frac{n(n-1)}{2}$ voisins (soit $O(n^2)$ voisins), la double somme $\sum_{j=1}^{n} d_{ij}.c_{\pi(i)\pi(j)}$ (en $O(n^2)$). Cependant, lorsque l'évaluation du voisinage est itérée, comme dans une ISI, le calcul peut s'effectuer de façon incrémentale; ce qui réduit considérablement le coût de l'évaluation car chaque voisin est évalué en temps constant. On parvient ainsi à évaluer tout le voisinage de la solution courante en $O(n^2)$.

Dans le cas où les matrices de distances $(d_{ij})_n$ et de flux $(c_{kl})_n$ sont symétriques à diagonales principales nulles, le calcul incrémental s'exprime comme suit [Taillard91]. Soit p, la solution courante. On note p_{rs} la solution obtenue par l'échange des emplacements des objets r et s dans p, $\Delta_p(rs)$ la variation de la fonction objectif:

$$\begin{aligned}\Delta_p(rs) &= \varphi(p_{rs}) - \varphi(p) \\ &= 2\sum_{i=1, i\neq r,s}^{n} (d_{is} - d_{ir})(c_{p(i)p(s)} - c_{p(i)p(r)})\end{aligned}$$

Soit $q = p_{rs}$, on exprime $\Delta_q(uv)$ de façon incrémentale comme suit:

$$\Delta_q(uv) = \Delta_p(uv) + 2(d_{ru} - d_{rv} + d_{sv} - d_{su})(c_{q(r)q(v)} - c_{q(r)q(u)} - c_{q(s)q(v)} + c_{q(s)q(u)})$$

Ainsi, pour calculer un voisin de la solution courante résultant de l'échange des éléments u et v, il suffit d'évaluer $\Delta_q(uv) - \Delta_p(uv)$ qui se calcule en temps constant. Donc, à chaque itération, sauf la première, l'ensemble du voisinage est évalué en $O(n^2)$. À la première itération, il faut évaluer $\Delta_p(rs)$ pour chaque voisin. Puisque $\Delta_p(rs)$ se calcule en $O(n)$, l'évaluation du voisinage à la première itération s'effectue en $O(n^3)$.

Dans le cas de matrices non symétriques, l'expression du calcul incrémental est un peu plus complexe, mais on obtient les mêmes conclusions. Ainsi, pour le QAP, dans les méthodes ISI qui évaluent la globalité du voisinage, on peut toujours calculer les voisins de manière incrémentale en $O(n^2)$. Le calcul incrémental du voisinage permet donc de changer la complexité de $O(n^4)$ à $O(n^2)$, ce qui accélère considérablement les algorithmes.

2.2.3 Évaluation de la SDW *multistart*

Nous avons effectué les évaluations (voir Tab. 2.2) de cette descente parallèle avec 1000 SDW s'exécutant simultanément (environ 60 par processeur Alpha) [Bachelet95]. Pour chaque descente, la solution initiale est obtenue par tirage aléatoire uniforme. Chaque descente s'arrête sur le premier optimum qu'elle rencontre. D'une manière globale, on constate que la descente multiple est une heuristique performante puisque le plus mauvais résultat obtenu (instance Chr25a) sur les 27 instances traitées n'est que de 10 % au dessus de la meilleure solution connue; pour plus de 20 instances, nous obtenons des résultats inférieurs ou de l'ordre de 1 %.

La dernière colonne du tableau 2.2 montre l'écart entre la qualité moyenne des solutions produites par la SDW *multistart* et celle de la meilleure solution trouvée.

Instance	Type	φ(QAPlib)	Meilleure	Moyenne	Écart
Chr25a	I	3 796	10.5901	57.6368	47.0467
Tai25a	I	1 167 256	1.9400	5.2042	3.2642
Lipa50a	-	62 093	1.0774	1.3098	0.2324
Tai60a	I	7 208 572	2.9489	4.7134	1.7644
Tai64c	-	1 855 928	**0.0000**	0.4281	0.4281
Tai80a	I	13 557 864	2.4960	3.7576	1.2616
Lipa90a	I	360 630	0.6713	0.7749	0.1036
Tai100a	I	21 125 314	2.4470	3.4055	0.9585
Esc128	I	64	**0.0000**	13.2313	13.2313
Tai256c	I	44 759 294	0.2155	0.4275	0.2120
Had20	II	6 922	**0.0000**	1.1904	1.1904
Bur26d	II	3 821 225	0.0006	0.4731	0.4725
Nug20	-	2 570	**0.0000**	n.c.	n.c.
Nug30	II	6 124	0.8491	4.1976	3.3485
Els19	II	17 212 548	**0.0000**	26.8930	26.8930
Ste36c	II	8 239 110	1.1667	9.4458	8.2791
Tai35b	II	283 315 445	0.1987	8.5871	8.3884
Tai40b	II	637 250 948	0.1338	10.4842	10.3505
Sko42	-	15 812	0.4364	n.c.	n.c.
Tai60b	III	608 215 054	0.4039	8.1244	7.7204
Sko64	III	48 498	1.1423	2.7114	1.5691
Tai80b	III	818 415 043	1.4407	6.1551	4.7144
Sko81	III	90 998	0.9121	2.2683	1.3562
Wil100	III	273 038	0.3853	1.0873	0.7020
Sko100a	III	152 002	0.7236	2.0921	1.3684
Tai100b	III	1 185 996 137	0.9110	5.3459	4.4349
Tho150	III	8 133 484	0.8969	2.4247	1.5278

TAB. 2.2 - *Évaluation de la SDW multidépart pour 1000 descentes. Les résultats sont présentés sous la forme* $100 \times \frac{\varphi(s)-\varphi(QAPlib)}{\varphi(QAPlib)}$ *où s est la solution fournie par la métaheuristique.*

Cet écart donne une mesure de l'intérêt de relancer la méthode. En effet, quand la meilleure solution trouvée a une qualité proche de la qualité moyenne, cela signifie que la recherche *multistart* a trouvé de nombreuses fois des solutions de cette qualité. Alors, on peut réduire le nombre de descentes, l'intérêt du redémarrage faible. Au contraire, si la meilleure solution est d'une qualité nettement supérieure à la moyenne, cela signifie qu'il y a peu de solutions de cette qualité dans la population des optima obtenus ; il est résulte que le redémarrage est profitable. En observant cet écart, on constate qu'il est plus faible pour les instances de type I (0.10 pour `Lipa90a`, 0.21 pour `Tai256c`, 0.23 pour`Lipa50a`) que pour les instances de type II et III (26.9 pour `Els19`, 8.39 pour `Tai35b`, 1.53 pour `Tho150`). On peut expliquer ces résultats en s'appuyant sur l'analyse des paysages. Les instances de type I ont un paysage plutôt uniforme et la diversité apportée par un grand nombre de solutions de départ des descentes n'est pas profitable. À l'opposé, les instances de type II et III sont plus structurées, leurs paysages sont plus diversifiés, comportant des vallée de profondeurs différentes. La quantité et la diversité des solutions de départ offrent une plus grande probabilité d'atteindre une vallée profonde ; c'est pourquoi la descente *multistart* est efficace. On déduit de ce comportement de la descente *multistart* que la descente `SDW` (sans redémarrage) est plus efficace sur les instances non structurée ; cette propriété de la descente peut être résumée par le tableau suivant :

	Type I	Type II	Type III
SDW	+	−	−

2.3 Une TS avec parallélisme adaptatif

Pour l'étude des recherches tabous parallèles, nous avons orienté nos travaux vers l'approche multidépart. Nous avons concentré notre étude sur les méthodes parallèles sans coopération. Quelques études ont porté sur des recherches tabous multiples coopérantes ; mais elles n'ont pas montré leur supériorité par rapport aux recherches tabous multiples indépendantes [Crainic et al.93]. De plus, le multi-tabou sans coopération suit un modèle plus simple ; c'est pourquoi nous l'avons choisi. Nous avons implémenté des TS parallèles sur différentes plates-formes : sur `C/PVM/Alpha-farm` et sur `MARS` (*Multi-user Adaptive Resource Scheduler*), un environnement de programmation parallèle adaptative [Talbi et al.99a]. Nous ne présentons pas ici l'implémentation sur la plate-forme `C/PVM/Alpha-farm` car elle est semblable à celle de la descente *multistart* que nous avons décrite plus haut. Dans cette section, nous détaillons les caractéristiques de la recherche tabou que nous avons réalisée et quelques principes généraux du système `MARS`.

2.3.1 MARS : un environnement de programmation parallèle adaptative

MARS est un environnement de programmation parallèle sur architectures hétérogènes multi-utilisateurs [Talbi et al.99a] [Talbi et al.97] [Talbi et al.96]. À l'origine de MARS, est posé le constat que de plus en plus d'institutions (universités, laboratoire, ...) possèdent un réseau de stations de travail (NOW – *Network Of Workstations*) qui sont sous-utilisées. En effet, les concepteurs de MARS ont montré que les stations de leur laboratoire sont disponibles pendant 83 % du temps en moyenne (étude réalisée en semaine sur une cinquantaine de stations durant 24 heures). Ce temps de non utilisation des machines représente une ressource de calcul importante mais difficilement exploitable car éparpillée dans le temps et l'espace. Le système MARS permet d'exploiter cette sous-utilisation des machines. Pour ce faire, il observe la disponibilité des stations ; il fournit une tâche à une machine qui devient disponible (phase de déploiement) et ôte son travail à une machine qui devient occupée (phase de repli). MARS est qualifié d'adaptatif car il reconfigure dynamiquement l'ensemble des processeurs qui supportent l'application parallèle, en fonction de l'état de charge des stations. De plus, pour éviter de gêner « son » propriétaire en consommant les ressources de « sa » machine, les critères de disponibilité utilisés par MARS tiennent compte du caractère personnel des stations de travail. En effet, lorsqu'un utilisateur travaille sur sa machine (utilisation du clavier ou de la souris, forte charge de la station), elle est déclarée non disponible et, par conséquent, n'est pas utilisée par le système MARS.

Une application parallèle développée sur l'environnement MARS est calquée sur un modèle maître/esclaves (SPMD). Le processus maître assure la gestion des esclaves : chaque esclave reçoit son travail du maître, s'exécute, puis retourne son résultat au maître. Durant l'exécution de l'application, le processus maître tourne sur une seule machine alors que les esclaves sont distribués dynamiquement par le système MARS sur la plate-forme d'exécution. MARS déplie (crée de nouveaux processus esclaves) et replie (retire des esclaves) l'application selon que les machines sont disponibles ou non. Sur les machines disponibles, des esclaves s'exécutent, les autres tâches esclaves étant en attente sur le processus maître. Quand une machine du système devient disponible, le maître lui envoie un esclave, c'est-à-dire une structure de donnée qui représente la tâche esclave et son point d'avancement, puis le processus esclave s'exécute. Quand une machine n'est plus disponible, parce qu'elle devient chargée ou qu'elle est réquisitionnée par son propriétaire, alors le processus esclave qui s'y exécutait est interrompu, puis la structure de donnée qui définit la tâche esclave est retournée au processus maître. La reconfiguration dynamique du système MARS est illustrée par la figure 2.8 relative à la recherche tabou parallèle que nous présentons à la section 2.3.3.

Lors de longues exécutions sur une plate-forme composée de nombreuses stations de travail, les pannes matérielles et logicielles sont fréquentes : si la probabilité d'oc-

currence d'une panne est p pour une machine, il devient $k \times p$ pour un NOW de k machines, sans considérer les défaillances liées au réseau lui-même. Les concepteurs de MARS ont constaté qu'une panne survient en moyenne toutes les 3 heures sur le réseau de leur laboratoire. Pour se prémunir des pannes, le système MARS dispose d'un mécanisme de sauvegarde/restauration. Ce mécanisme consiste en la sauvegarde périodique de l'état de l'application sur un support permanent[4]. Dès qu'une panne est détectée, l'application parallèle est restaurée à partir de la dernière sauvegarde ; on évite ainsi de relancer l'application depuis son commencement. Dans MARS, la gestion de la sauvegarde/restauration est automatique, elle ne nécessite pas l'intervention du programmeur. Quand le système opère une sauvegarde, le processus maître diffuse une requête de collecte des résultats partiels auprès des esclaves. À la réception, chaque esclave retourne son état d'avancement au maître. Quand le maître a réceptionné les états de tous les esclaves, il crée un processus miroir qui est chargé de la sauvegarde pendant que l'exécution de l'application se poursuit en parallèle. La restauration consiste alors à reconstruire l'état du maître à partir de la sauvegarde. Ce système de tolérance aux pannes est assez peu coûteux : ses auteurs ont constaté un surcoût moyen d'environ 1.5 % du temps d'exécution total de l'application parallèle.

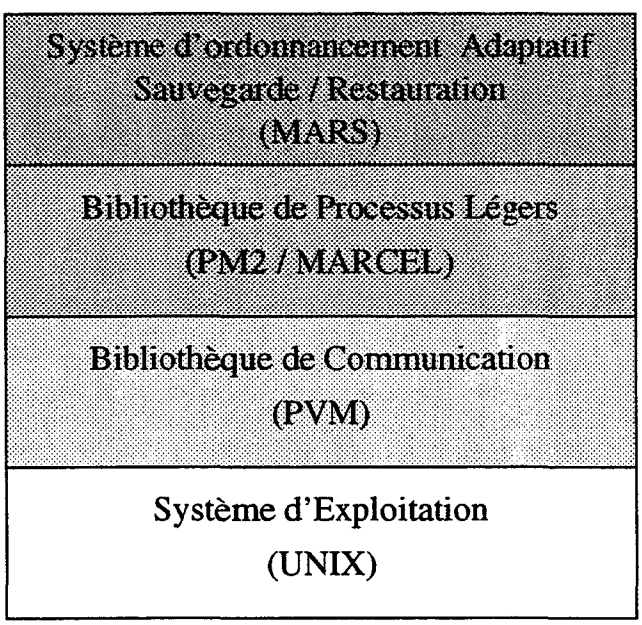


FIG. 2.7 - *Le système MARS est réalisé, au niveau utilisateur, sur le système d'exploitation UNIX. MARS utilise une bibliothèque de processus légers et une bibliothèque de communication par échange de messages.*

4. Un support non sensible aux pannes, un disque par exemple.

Le système MARS a été développé au niveau utilisateur sur le système d'exploitation UNIX (voir Fig. 2.7). Il utilise la bibliothèque de communication par échange de message PVM et son modèle d'exécution s'appuie sur le système préemptif PM2 (*Parallel Multithreaded Machine*) à base de processus légers. Cette structure en couche confère à MARS une grande portabilité car elle ne demande pas de modifier le noyau d'UNIX pour son installation ; de plus, il n'est pas nécessaire d'être utilisateur root. À titre d'exemple, dans notre étude, nous avons installé MARS sur les systèmes d'exploitation hétérogènes suivants : SunOS, OSF, Solaris et Linux.

2.3.2 Notre TS séquentielle

La méthode avec tabous que nous avons mise en place est un tabou simple : il n'intègre pas de technique d'intensification ou de diversification sophistiquée. L'opérateur de voisinage est la transposition. Pour calculer le voisinage, nous utilisons l'évaluation incrémentale décrite à la section 2.2.2.

Dans le voisinage de la solution courante, une solution est tabou si elle est obtenue par une transposition qui a été appliquée durant les l dernières itérations, où l est la longueur de la liste tabou. La liste tabou que nous utilisons est donc une file[5] de longueur l où défilent les transpositions successivement appliquées aux solutions courantes. La longueur de la liste tabou est choisie aléatoirement au démarrage de la TS, entre $\frac{n}{2}$ et $\frac{3n}{2}$; puis, elle demeure inchangée tout au long de la recherche. La fonction d'aspiration que nous utilisons est classique, elle s'applique lorsqu'un mouvement conduit à une solution meilleure que la meilleure solution rencontrée depuis le début de la recherche.

La recherche commence sur une solution initiale choisie aléatoirement dans l'espace de recherche selon une loi uniforme. Puis, elle se poursuit pendant un certain nombre d'itérations. Ce nombre d'itérations est le seul paramètre de notre méthode TS.

2.3.3 La TS multidépart sur MARS

La recherche tabou parallèle multidépart (MTS – *Multistart Tabu Search*) a été développée sur la plate-forme MARS selon un modèle maître/esclaves. Dans l'implémentation de notre TS parallèle, les processus esclaves font chacun une recherche tabou séquentielle. Une TS, à quelque stade qu'elle soit (à commencer, en cours ou achevé), est représentée par sa mémoire ISI qui contient la solution courante, la liste tabou, la meilleure solution trouvée et le nombre d'itérations effectuées (voir Fig. 2.8). Durant l'exécution, MARS déplie et/ou replie la MTS en créant de nouveaux processus TS et/ou en retirant des esclaves TS, en fonction des disponibilité

5. *FIFO: First In First Out.*

de la plate-forme d'exécution. Quand une machine du système devient disponible, le maître lui envoie une TS, c'est à dire la mémoire de la TS ; puis, la TS s'exécute. Quand une machine n'est plus disponible, alors la TS qui s'y exécute termine l'itération courante ; puis, sa mémoire est retournée au processus maître.

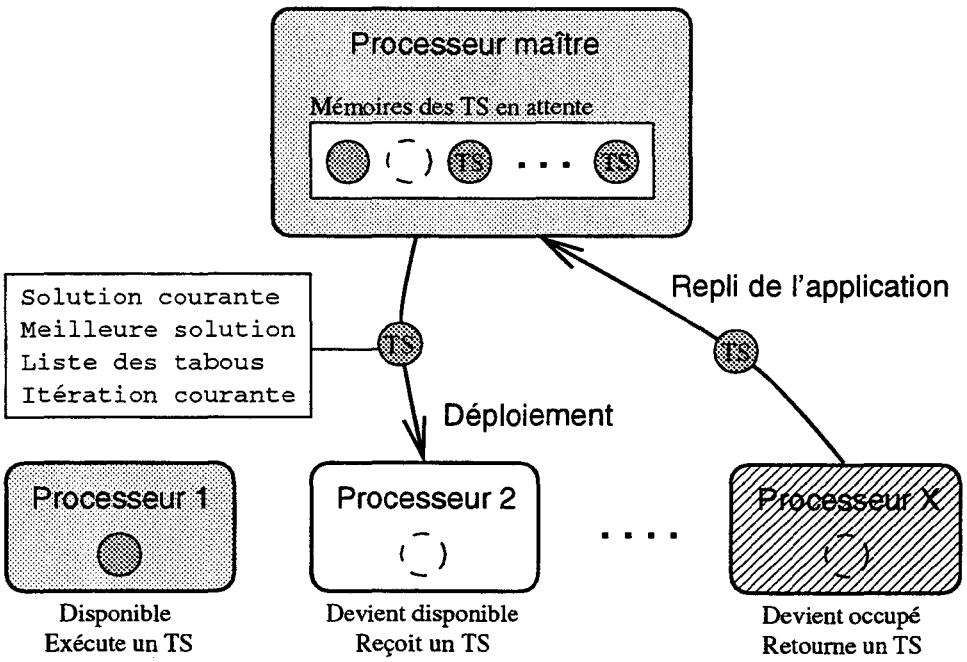


FIG. 2.8 - *MTS : recherche tabou parallèle multistart sur MARS.*

2.3.4 Évaluation de la TS multidépart

Pour l'évaluation de notre TS *multistart*, nous avons utilisé un NOW composé de machines hétérogènes : plus de 150 stations de travail (PC-Linux, Sun4-SunOS, Alpha-OSF, Sun-Solaris) et un *cluster* composée de 16 processeurs DEC-Alpha (voir Tab. 2.9). Ces machines sont, pour la plupart d'entre elles, installées dans les bureaux, les laboratoires et les salles de travaux pratiques de l'université. Ainsi, ces machines étant quotidiennement utilisées par les étudiants et les chercheurs, on comprend bien l'intérêt de la plate-forme de parallélisme adaptatif, qui permet d'accéder à la puissance de calcul que constitue ce réseau de station de travail sans perturber l'utilisateur qui est devant sa console.

À l'analyse des résultats, on remarque que la MTS est une méthode efficace puisqu'elle trouve la meilleure solution connue pour plus de la moitié des 12 instances traitées, même avec 50 n itérations seulement (voir Tab. 2.3). On note que l'augmentation du nombre d'itérations de 50 n à 1000 n ne produit pas la même amélioration

Instance	Type	φ (QAPlib)	Meilleure		Amélio.
			50 n itér.	10^3 n itér.	%
Tai25a	I	1 167 256	**0.7361**	**0.7361**	0.0000
Tai64c	I	1 855 928	**0.0000**	**0.0000**	0.0000
Tai100a	I	21 125 314	0.9461	**0.7229**	23.5949
Tai256c	I	44 759 294	0.2988	**0.2359**	21.0581
Els19	II	17 212 548	**0.0000**	**0.0000**	0.0000
Bur26d	II	3 821 225	**0.0000**	**0.0000**	0.0000
Nug30	II	6 124	**0.0000**	**0.0000**	0.0000
Tai35b	II	283 315 445	**0.0000**	**0.0000**	0.0000
Ste36c	II	8 239 110	**0.0000**	**0.0000**	0.0000
Sko64	III	48 498	0.0948	**0.0000**	100.0000
Sko100a	III	152 002	0.1211	**0.0579**	52.1739
Tai100b	III	1 185 996 137	0.5455	**0.3191**	41.5033

TAB. 2.3 - *Évaluation du MTS pour 1 exécution avec 100 TS (équivalent à 100 exécutions du TS séquentiel). Les résultats sont présentés sous la forme* $100 \times \frac{\varphi(s)-\varphi(QAPlib)}{\varphi(QAPlib)}$ *où s est la solution fournie par la métaheuristique.*

en fonction du type d'instances. Sur les instances de type I, on n'obtient qu'une légère amélioration : 21 % pour `Tai256c`, 0 % pour `Tai25a`, 24 % pour `Tai100a`. À l'inverse, sur les instances de type III, le gain est fort : 100 % pour `Sko64`, 52 % pour `Sko100a` et 42 % pour `Tai100b`. Ces résultats sont cohérents avec la nature des paysages des différentes instances. En effet, pour le type I, le paysage est plat et rugueux et de courtes marches suffisent. Pour les instances de type III, plus structurées, l'heuristique parvient à exploiter la structure de l'instance et de longues marches sont profitables.

	Type I	Type II	Type III
TS	+	+	−
TS (longues marches)	+	+	+

2.4 Conclusion

Dans ce chapitre, nous avons décrit l'étude de quelques métaheuristiques parallèles `ISI` (`SDW`, `TS`) ou `IPI` (`CGA`, `MCGA`) appliquées au `QAP`. Pour cette étude, nous avons mis en œuvre différentes classes de parallélisme (`SIMD`, `SPMD`) et différents environnements et plates-formes d'exécution (`MPL/Maspar`, `PVM/cluster`, `MARS/NOW`).

En ce qui concerne les environnements et les plates-formes d'exécution, notre préférence revient à `MARS/NOW` car c'est un environnement très souple d'utilisation.

Contrairement à MPL/Maspar, il ne nécessite pas de réservation car il utilise les ressources inexploitées du NOW, sans gêner les utilisateur des stations. En fait, grâce à son caractère adaptatif, MARS s'adapte aux disponibilités du NOW pour obtenir toujours le maximum de ressources de calcul que le réseau de stations peut fournir. Comparé à PVM/Alpha-farm, le système MARS est très avantageux car il dispose d'un mécanisme de sauvegarde/restauration qui le rend tolérant aux pannes si fréquentes lors de longues exécutions. Ainsi, MARS est le système le plus performant et le plus agréable que nous avons utilisé ; nous l'avons donc choisi pour supporter l'évaluation des métaheuristiques parallèles hybrides (voir partie III).

Sur le plan des évaluations, nous avons constaté, dans ce chapitre, des variations de performances des différentes métaheuristiques en fonction de la nature des instances à résoudre (voir tableau récapitulatif ci-dessous). Les métaheuristiques à solution unique ISI (pour de courtes marches) sont plus efficaces sur les instances uniformes (type I) alors que les méthodes à population IPI, au contraire, sont adaptées aux instances fortement structurées (type II). Les méthodes ISI affinent très localement la solution courante dans leur voisinage immédiat et atteignent le fond d'une vallée ; elles sont bien adaptées aux instances uniformes qui ont un paysage plat et rugueux car il suffit d'atteindre le fond d'une vallée pour obtenir une solution de bonne qualité. À l'inverse, les métaheuristiques IPI, grâce à la diversité de la population initiale et grâce à la recombinaison parviennent à caractériser une région de qualité dans l'espace de recherche ; elles sont adaptées aux instances très structurées dont le paysage est formé de quelques vallées profondes, car il suffit de trouver une vallée profonde pour obtenir une solution de qualité. En ce qui concerne les instances de nature intermédiaire (type III), nous avons constaté que de longues marches de TS sont efficaces. Cependant, ces longues marches sont coûteuses ; le rapport entre la qualité du résultat et l'effort de calcul nécessaire à son obtention n'est pas satisfaisant, surtout pour des instances de grande taille. Une autre approche consiste à utiliser, pour résoudre des instances dont le paysage est intermédiaire, des méthodes intermédiaire entre les ISI et les IPI : ce sont les métaheuristiques hybrides ISI/IPI que nous présentons, avec d'autres méthodes hybrides, à la partie III.

	Type I	Type II	Type III
GA	−	+	−
SDW	+	−	−
TS	+	+	−
TS (longues marches)	+	+	+

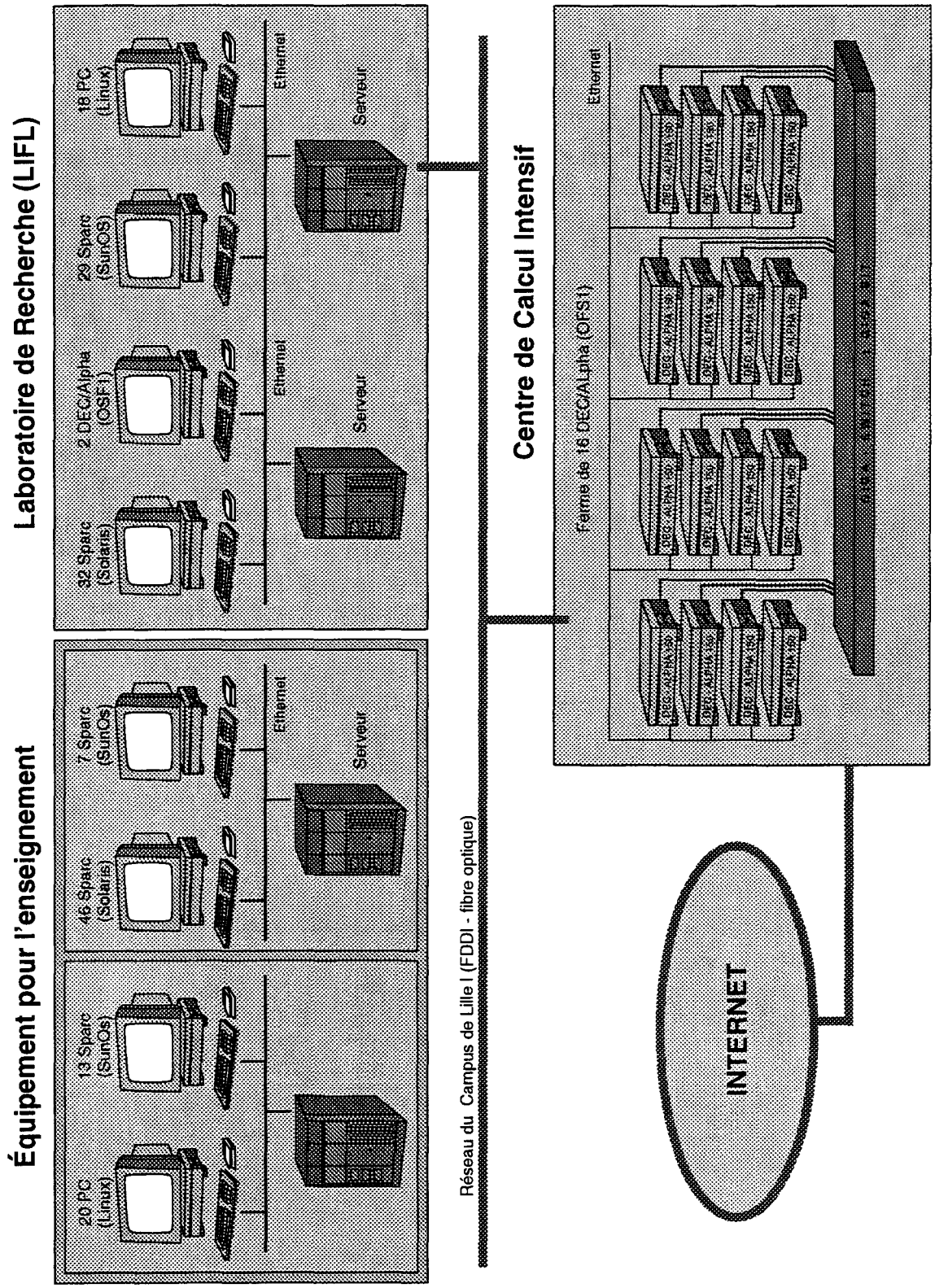


FIG. 2.9 - *La plate-forme d'évaluation que nous avons utilisée au travers de* **MARS** *est un* **NOW** *composé de plus de 150 stations de travail hétérogènes (PC-Linux, Sun4-SunOS, Alpha-OSF, Sun-Solaris) et d'un cluster de 16 processeurs* **DEC-Alpha**.

Troisième partie

Hybridation de métaheuristiques et coévolution

Cette partie est composée de deux chapitres. Le chapitre 1 est un état de l'art synthétique des métaheuristiques hybrides sous la forme d'une taxinomie originale. Dans le chapitre 2, nous présentons les méthodes hybrides coévolutionnistes que nous avons réalisées pour résoudre efficacement le QAP.

Les travaux décrits dans cette partie ont fait l'objet d'une présentation à la conférence internationale INFORMS et d'une publication dans la revue Calculateur Parallèles [Bachelet et al.98c] [Bachelet et al.98b].

Chapitre 1

Taxinomie des métaheuristiques hybrides

Durant ces dernières années, de nombreuses recherches sur l'hybridation de métaheuristiques ont été menées dans le domaine de l'optimisation combinatoire. Aujourd'hui, les méthodes hybrides obtiennent les meilleurs résultats sur la plupart des problèmes, qu'ils soient académiques ou pratiques (issus du monde réel). Cette efficacité résulte de combinaisons de méthodes de recherche classiques, ISI ou IPI ; cependant, on trouve, parmi les méthodes qui ont été proposées, une grande diversité de forme d'hybridation, plus ou moins complexes. Dans le but d'y voir plus clair, nous proposons une classification qui permet de comparer les méthodes hybrides suivant des critères qualitatifs, entre elles. Par ailleurs, nous avons doté notre classification d'une grammaire qui caractérise le classement d'une recherche hybride par une expression concise.

Parmi les taxinomies existantes, dans diverses domaines, on trouve des modèles de classification hiérarchiques et des modèles de classification à plat. La taxinomie que nous proposons, ici, est composée des deux modèles. C'est une taxinomie à deux composantes, l'une issue d'une classification hiérarchique et l'autre issue d'une classification à plat. La figure 1.1 représente la structure de cette taxinomie à deux clefs.

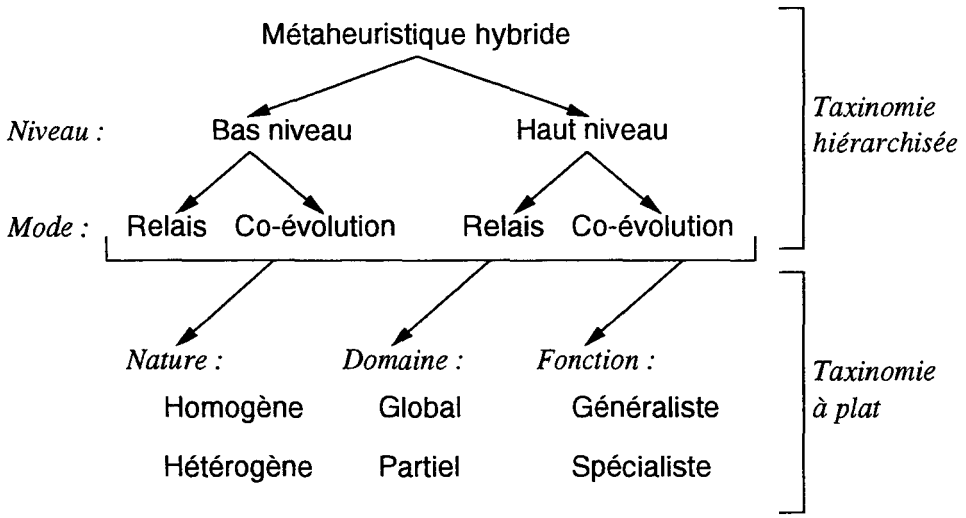


FIG. 1.1 - *Structure de notre taxinomie des recherches hybrides*

1.1 La classification hiérarchique

1.1.1 Les propriétés discriminantes

1.1.1.1 Niveau de l'hybridation

Le premier pas dans la classification sépare différents niveaux d'hybridation. On distingue les hybridations de bas niveau et les hybridations de haut niveau. La figure 1.2 illustre la notion de niveau pour l'hybridation de deux métaheuristiques ; cependant, parfois plus de deux méthodes sont hybridées.

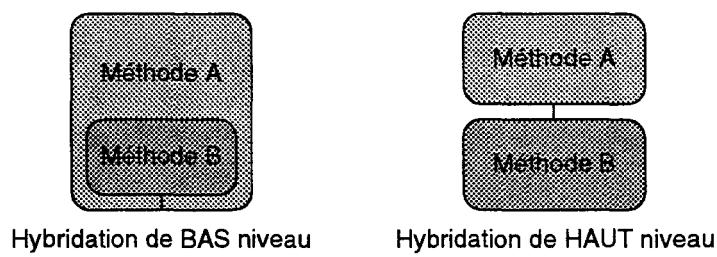


FIG. 1.2 - *Le niveau d'hybridation. Dans l'hybridation de bas niveau, la méthode est modifiée, ici la méthode B devient un élément fonctionnel de A. Pour l'hybridation de haut niveau, les méthodes A et B ne sont pas modifiées.*

L'hybridation de bas niveau modifie les éléments fonctionnels qui constituent une

méthode d'optimisation. Dans cette classe d'hybrides, une fonction interne d'une métaheuristique est remplacée par une (autre) métaheuristique.

Au contraire, l'hybridation de haut niveau conserve l'intégrité des méthodes qu'elle lie. Il n'y a pas de relation directe entre les mécanismes internes des métaheuristiques hybridées.

1.1.1.2 Mode de l'hybridation

Le mode de l'hybridation distingue les hybrides en mode relais des hybrides en mode coévolution.

En mode relais, les heuristiques hybridées opèrent les unes après les autres dans un ordre prédéterminé. Hormis la première, chaque méthode impliquée dans l'hybridation reçoit en entrée le résultat produit par la précédente, à la manière d'un pipeline.

La classe des recherches hybrides en mode coévolution intègre les modèles d'optimisation coopérative où plusieurs agents coopèrent en parallèle. Chaque méthode hybridée (agent) conduit une recherche dans un espace de solutions donné.

1.1.2 Quatre classes d'hybrides

Selon la classification hiérarchique, nous obtenons, à partir des deux propriétés discriminantes que sont le niveau et le mode de l'hybride, quatre classes d'hybrides détaillées ci-dessous. Nous aurions pu développer le classement hiérarchique en fonction d'autres discriminants (ceux de la classification à plat) mais nous préférons l'arrêter là, pour éviter d'augmenter le nombre de classes. Par ailleurs, ce classement en 4 catégories apparaît déjà dans d'autres classifications et nous n'avons pas voulu faire une taxinomie trop réformatrice mais plutôt apporter de nouveaux discriminants (nature, domaine et fonction) pour compléter la classification qui semble s'imposer en standard dans la communauté de l'optimisation combinatoire.

1.1.2.1 La classe LRH – *Low-level Relay Hybrid*

La classe des hybrides de bas niveau en mode relais (LRH) regroupe les hybrides constitués d'une méthode ISI dans laquelle est insérée une (autre) métaheuristique.

O. Martin et S. Otto ont réalisé une hybridation de type LRH qui combine un recuit simulé (SA – *Simulated Annealing*) et une méthode de descente déterministe (SDW – *Steepest Descend Walk*) pour résoudre le problème du voyageur de commerce (TSP – *Traveling Salesman Problem*) [Martin et al.92]. Le principe général de leur algorithme hybride est de limiter la recherche du recuit simulé à l'espace des optima locaux par l'intégration de la SDW au SA. L'hybride se déroule comme suit (voir

Fig. 1.3). On part d'une solution courante qui est un optimum local, on la perturbe jusqu'à obtenir une solution intermédiaire suffisamment éloignée dans l'espace de recherche. On effectue ainsi un saut dans l'espace de recherche que les auteurs appellent le *kick*. Puis, cette solution intermédiaire est optimisée par une méthode de descente jusqu'à l'obtention d'un optimum local qui devient un candidat du recuit simulé. Enfin, si le candidat est rejeté, on revient à la solution courante ; s'il est accepté, il devient la nouvelle solution courante. On se retrouve ainsi avec une solution courante qui est un optimum local qui diffère du précédent de façon significative.

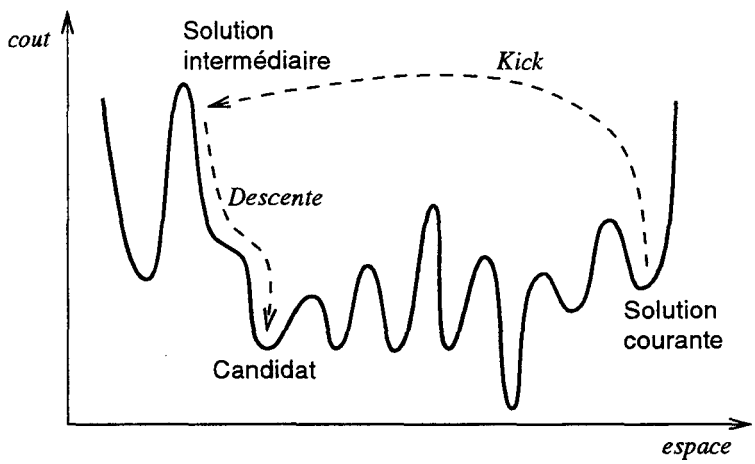


FIG. 1.3 - *Un exemple d'hybridation LRH dans lequel un recuit simulé intègre une recherche locale. La figure donne une représentation schématique de la fonction coût (ligne continue) et montre les étapes de la génération d'un nouveau candidat du recuit simulé (flèches en pointillé).*

Ainsi, cette métaheuristique hybride permet de sauter par dessus un grand nombre de barrières de coût en un seul pas. Cependant, cette technique d'hybridation peut facilement conduire à une recherche qui cycle si on ne prend pas garde au choix du *kick* et de la méthode de descente. En effet, on conçoit aisément que si le *kick* et la descente sont basés sur le même opérateur de voisinage, alors la descente risque de retourner à la solution courante. Par exemple, pour le TSP, une descente dans un voisinage 3-opt impose, pour que la méthode soit efficace (évite les cycles), l'utilisation d'un *kick* basé sur un opérateur k-change avec $k > 3$.

1.1.2.2 La classe LCH – *Low-level Co-evolutionary Hybrid*

Dans la conception d'une métaheuristique, on recherche l'équilibre entre deux objectifs contradictoires : l'exploration et l'exploitation. L'exploration est nécessaire, car elle apporte une information (partielle) de l'espace de recherche dans sa globalité

qui fournit une approximation de la qualité de l'optimum global. L'exploitation n'est pas moins importante, car une recherche autour de la solution courante tend à produire une meilleure solution.

Les métaheuristiques IPI (basées sur une population de solutions) sont plutôt performantes pour l'exploration de l'espace de recherche alors qu'elles le sont beaucoup moins pour l'exploitation des solutions qu'elles ont trouvées. C'est pourquoi, la plupart des heuristiques IPI classiques ont été LCH-hybridées avec des méthodes ISI (à solution unique) qui, elles, sont des heuristiques de recherche performantes en terme d'exploitation. Ainsi, l'hybride LCH réalise l'équilibre exploration/exploitation en hybridant deux algorithmes complémentaires, l'ISI optimisant localement, l'IPI optimisant d'une manière globale.

Par exemple, on peut utiliser un algorithme génétique pour l'optimisation globale et ajouter à ses opérateurs standard une méthode ISI (voir Fig. 1.4). Ainsi, au lieu d'utiliser un opérateur aveugle, c'est à dire qui ne tient pas compte de la *fitness* de l'individu, on prend un opérateur, dit lamarkien, qui est une heuristique locale, qui optimise l'individu et restitue au GA un individu amélioré. Les opérateurs standard des GA les plus souvent remplacés ou complétés sont la mutation et la recombinaison.

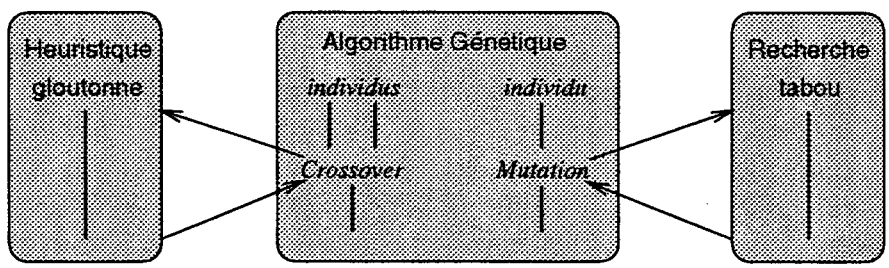


FIG. 1.4 - *Hybridation de bas niveau en mode coévolution. Ici, une recherche tabou et une heuristique gloutonne remplacent les opérateurs standard de l'algorithme génétique, respectivement la mutation et la recombinaison.*

Lamarkiens pour la mutation

Les opérateurs lamarkiens les plus souvent utilisés sont des méthodes de descente [Suh et al.87] [Ulder et al.90] [Jog et al.89], des recherches tabous [Fleurent et al.94] [Kim et al.95] [Thiel et al.94], ou des recuits simulés [Brown et al.89] [Chen et al.94] [Wang et al.95]. Dans de nombreuses études, les hybrides LCH ont donné des résultats bien meilleurs que d'autres méthodes sur des problèmes difficiles. De bons résultats ont été obtenus pour le problème de partitionnement de graphe (GPP – *Graph Partitioning Problem*) en hybridant un algorithme génétique et une recherche tabou [Fleurant et al.96]. Dans [Chu97], un hybride LCH est employé pour la résolution du

problème du sac à dos à contraintes multiples et pour le SCP (*Set Covering Problem*). Une recherche locale utilisant certaines spécificités du problème est incorporée dans les opérateurs génétiques. Dans [Hart94], les auteurs s'interrogent sur la meilleure utilisation des méthodes ISI dans les GA.

Lamarkiens pour la recombinaison

Les opérateurs de recombinaison classiques fonctionnent sans tenir compte d'aucune spécificité du problème. Cependant, certains opérateurs de recombinaison, dits heuristiques, intègrent des caractéristiques propres au problème à résoudre [Grefenstette87]. Des heuristiques gloutonnes intégrées comme opérateurs de recombinaison ont produit des résultats supérieurs à ceux du GA classique, pour la résolution des problèmes JSP (*Job-shop Scheduling Problem*), SCP et TSP entre autres.

Par ailleurs, l'hybridation coévolutionniste de bas niveau n'est pas réservée aux algorithmes génétiques ; l'hybridation LCH a également été expérimentée avec d'autres métaheuristiques IPI, l'objectif étant toujours d'intégrer une méthode ISI pour augmenter les performances de l'IPI. On peut citer : les systèmes de fourmis [Taillard et al.97] [Stutzle et al.97], la programmation génétique [Oreilly et al.95], et les recherches par dispersion [Cung et al.97].

1.1.2.3 La classe HRH – *High-level Relay Hybrid*

Dans les hybrides de haut niveau en mode relais, les heuristiques hybridées conservent leur intégrité. Les méthodes HRH-hybridées sont exécutées en séquence. Par exemple, on sait que les algorithmes à population (IPI) ne parviennent pas à ajuster finement les solutions proches des bons optima. Mais, à l'inverse, leur force réside dans la capacité de trouver rapidement des régions de bonne qualité, même pour des espaces de recherche très vastes ou très complexes. Une fois ces régions repérées, il peut être intéressant de poursuivre la recherche en affinant les solutions performantes qui s'y trouvent ; pour cela, on applique une recherche itérative à solution unique après l'IPI.

Considérons le cas des GA[1] (voir Fig.1.5). Expérimentalement, on constate, au bout d'un certain temps, que le GA converge : la population devient quasiment uniforme et la `fitness` n'augmente plus. On s'aperçoit qu'à partir de ce stade, le GA ne produit plus aucune amélioration. En fait, il s'est enfermé dans un bassin d'attraction duquel il a une probabilité presque nulle d'échapper dans un délai raisonnable. Ainsi, il est clair que si l'on désire encore progresser, l'exploitation de ce bassin d'attraction doit être réalisée par une heuristique d'un autre type que celui du GA. On utilise alors une recherche de type ISI car elles sont à même d'exploiter le bassin d'attraction trouvé par l'algorithme génétique. Par ailleurs, un hybride dans lequel

1. Les remarques qui suivent peuvent être étendues aux autres méthodes IPI.

une heuristique gloutonne est employée pour générer la population initiale du GA suit également le modèle HRH.

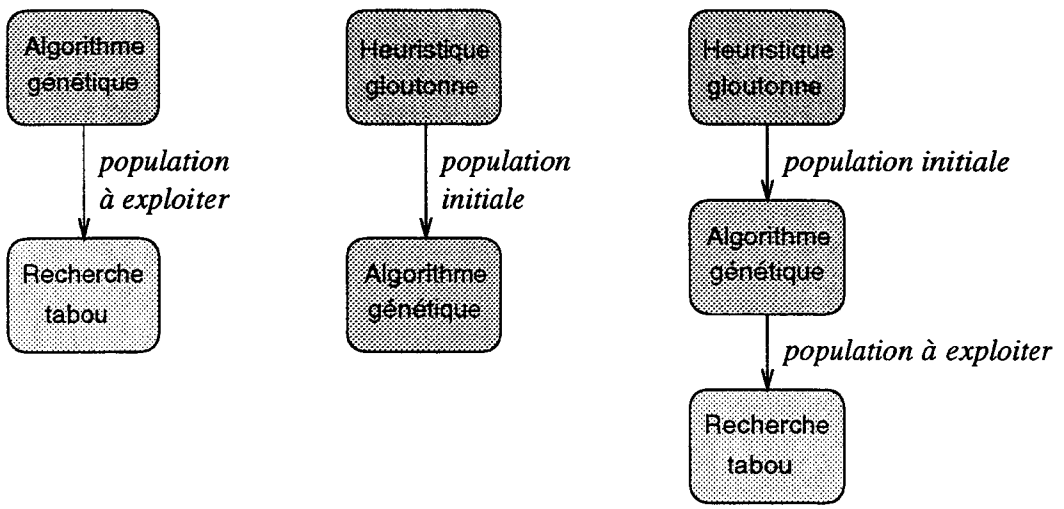


FIG. 1.5 - *Hybridation de haut niveau en mode relais. La figure montre 3 instances du schéma d'hybridation HRH. Ce modèle ne limite pas le nombre d'heuristiques pipelinées.*

De nombreux auteurs ont conçu des hybrides HRH à partir de méthodes IPI diverses. Dans [Mahfoud et al.95][Talbi et al.94], les auteurs ont utilisé, respectivement, un recuit simulé et une recherche tabou pour optimiser la population issue d'un GA. Dans [Nissen94], une méthode de descente améliore les solutions fournies par une heuristique ES (*Evolution Strategy*). Dans [Lin et al.91], l'algorithme proposé commence par un recuit simulé; il est suivi d'un algorithme génétique pour parfaire les solutions trouvées. Une étude expérimentale sur la résolution du GPP (*Graph partioning Problem*) avec un hybride HRH, impliquant un GA suivi d'un TS, montre que l'hybride fournit des solutions meilleures que l'algorithme génétique ou la recherche tabou seuls [Talbi et al.94].

1.1.2.4 La classe HCH – *High-level Co-evolutionary Hybrid*

Dans les hybridations de haut niveau coévolutionnistes, la structure interne des métaheuristique hybridées n'est pas modifiée. Ces dernières sont exécutées simultanément et coopèrent pour résoudre le problème. Intuitivement, on imagine facilement qu'un hybride HCH est au moins aussi performant que chacune des heuristiques qu'il lie. En fait, on constate expérimentalement, que l'hybride est souvent meilleur car chaque métaheuristique apporte sa « connaissance » du problème pour aider les autres méthodes.

Un exemple d'hybridation HCH est le modèle des algorithmes génétiques en îles.

Dans ce modèle, la population est partagée en petites sous-populations géographiquement éloignées. Un GA fait évoluer chaque sous-population et les individus peuvent migrer d'une sous-population à l'autre. Plusieurs paramètres configurent cet hybride : la topologie des connexions entre les sous-populations, le nombre d'individus qui migrent, la fréquence de la migration et la stratégie du remplacement.

Tanese propose un GA en îles où les sous-populations sont disposées dans un hypercube de dimension 4 [Tanese87]. Des migrations ont lieu périodiquement entre les sous-populations voisines sur l'hypercube (voir Fig. 1.6). Les individus à migrer sont choisis, de façon probabiliste, parmi les meilleurs individus des sous-populations ; puis, ils migrent et remplacent les individus les moins bons dans la population qui les accueille.

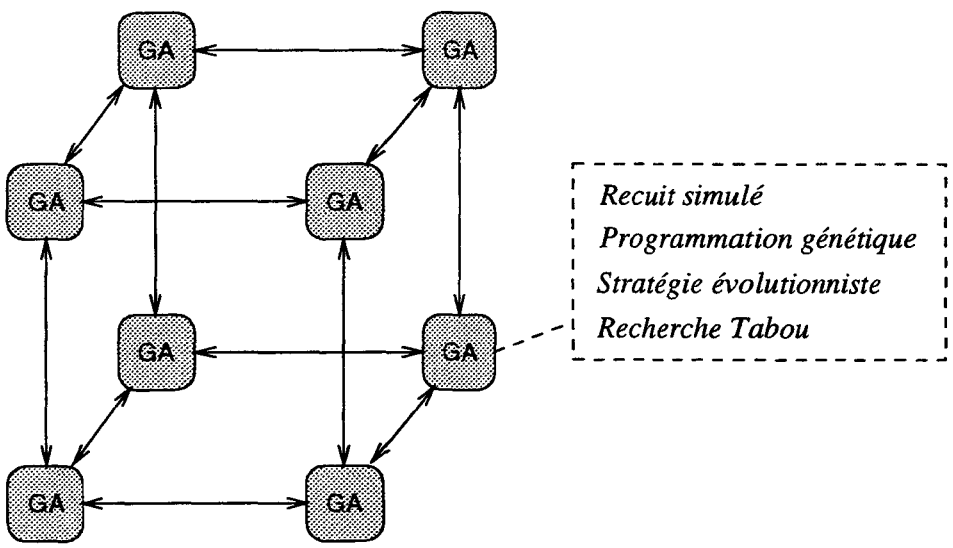


FIG. 1.6 - *Le modèle d'algorithme génétique en îles est un exemple d'hybridation de haut niveau coévolutionniste. Ce modèle a été aussi employé avec différentes topologies pour des recuits simulés, des programmations génétiques, des stratégies évolutionnistes et des recherches tabous.*

Cohoon, Hedge, Martin and Richards proposent un hybride HCH inspiré de la théorie des « équilibres ponctuels » [Cohoon et al.87]. Leur méthode est expérimentée sur une topologie en grille pour un problème de placement linéaire. Ils relatent que l'algorithme avec migration est plus performant que celui sans migration et le GA standard. Par la suite, cette étude a été poursuivie [Cohoon et al.90][Cohoon et al.91] pour l'étude d'un problème de conception VLSI (un partitionnement de graphe) sur un hypercube à 4 dimensions.

L'hybridation HCH s'applique aussi à d'autres métaheuristiques, telles que les recuits simulés [Falco et al.95], les programmations génétiques [Kosa et al.95], les

stratégies d'évolution [Voigt et al.90] et les recherches tabous [Falco et al.94].

1.2 La classification à plat

Les critères discriminants que nous avons retenus pour la classification à plat des métaheuristiques hybrides sont : l'homogénéité des méthodes hybridées, leurs domaines d'application et la nature (généraliste ou spécialiste) de leurs fonctions.

1.2.1 L'homogénéité des méthodes

Un hybride est dit homogène lorsque les méthodes hybridées sont les mêmes. Par exemple, le modèle d'algorithme génétique en îles est un hybride homogène ; toutes les méthodes hybridées sont des GA, mais chacune opère sur sa propre sous-population [Pettey et al.87]. En général, on utilise des paramétrages différents pour chaque heuristique hybridée[2]. Par exemple, un hybride avec des recherches tabous inclut plusieurs TS paramétrées par différentes solutions initiales, longueurs des listes tabous, ou autres [Voss93].

Dans un hybride hétérogène, différentes métaheuristiques sont hybridées (voir Fig. 1.7). Dans [Crainic et al.97], est présenté un hybride HCH, combinant un algorithme génétique et une recherche tabou, appliqué à la résolution d'un problème de conception de réseau. La population du GA est mise à jour, de façon asynchrone, par plusieurs recherches tabous. La méthode GRASP (*Greedy Randomized Adaptive Search Procedure*) peut être considérée comme un hybride HRH hétérogène itéré. En effet, GRASP alterne une heuristique gloutonne et une recherche locale. Cette méthode est dite adaptative, car l'heuristique gloutonne prend en compte les résultats obtenus à l'itération précédente [Feo et al.91] [Feo et al.94].

1.2.2 Le domaine d'application

Le domaine d'application des heuristiques hybridées permet de différencier deux classes d'hybridation : les hybridations globales et les hybridations partielles.

On appelle hybride global, un hybride pour lequel toutes les méthodes hybridées sont appliquées à l'espace de recherche dans sa totalité. L'objectif de ce type d'hybridation est d'explorer l'espace de recherche plus intensément. Tous les hybrides que nous avons cités ci-avant, sont des hybrides globaux car toutes les heuristiques résolvent le problème d'optimisation dans sa totalité. Par exemple, un hybride HCH

2. Les méthodes hybridées peuvent avoir un paramétrage identique ; mais, bien entendu, il n'y a aucun intérêt à utiliser plusieurs méthodes déterministes avec le même paramétrage car elles produiraient le même résultat.

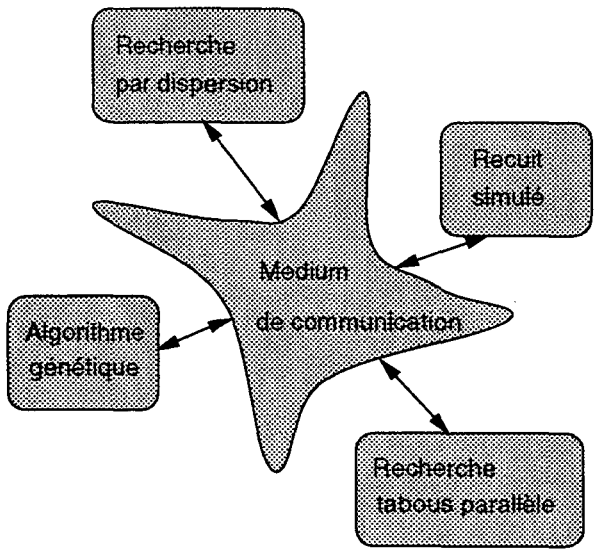


FIG. 1.7 - *Hybridation HCH hétérogène. Plusieurs algorithmes de recherche différents coopèrent pour résoudre le problème.*

global de recherche tabou est présenté dans [Crainic et al.93]. Chaque TS fait un certain nombre d'itérations, puis diffuse sa meilleure solution ; alors, la meilleure solution de tous les TS devient leur nouvelle solution initiale.

Pour l'hybridation partielle, le problème à résoudre est décomposé en sous-problèmes, ayant chacun un espace de recherche propre. Ainsi, chaque métaheuristique de l'hybride résout un sous-problème dans son propre espace de recherche. Cependant, il est généralement difficile de découper un problème en sous-problèmes indépendants sans lever quelques contraintes. Ainsi, pour que l'hybride partiel produise une solution réalisable, les méthodes hybridées doivent communiquer pour tenter de respecter les contraintes levées par la décomposition. Par exemple, ce type d'hybridation a été appliqué à un recuit simulé et à une recherche tabou [Taillard93a]. Dans [Husbands et al.90], une hybridation HCH homogène partielle est appliquée au JSP avec des GA. Chaque GA s'occupe d'une espèce qui représente l'ordonnancement d'une tâche. Ainsi, plusieurs GA coopèrent en tant que tâche. Le *médium* de communication regroupe les meilleurs individus de chaque GA et évalue l'ordonnancement global en se basant sur les meilleurs ordonnancements de tâches.

1.2.3 Hybrides généralistes ou spécialistes

Jusqu'à présent, nous n'avons cité que des hybrides généralistes, car toutes les heuristiques hybridées traitent le même problème d'optimisation. À l'inverse des hybrides généralistes, les hybrides spécialistes combinent des méthodes qui s'attaquent

à des problèmes différents. Par exemple, un hybride LCH spécialiste, combinant un algorithme génétique et un recuit simulé est présenté dans [Tsoi et al.95][Wong et al.94]. La phase de remplacement du GA est calquée sur le fonctionnement du SA. La probabilité de remplacement, P, est égale à la probabilité d'acceptation du SA :

$$P(\Delta E) = \frac{1}{1 + \exp(\Delta E/T)}$$

avec ΔE, la différence de *fitness* entre les nouveaux chromosomes et les chromosomes destinés à être remplacés ; T, le niveau de température de la génération courante. Dans [Mori et al.95] [Maekawa et al.96], les auteurs adaptent les concepts de température et d'entropie des recuits simulés à la phase de sélection d'un GA, afin de maintenir la diversité de la population.

Un autre cas d'hybridation HRH spécialiste est l'utilisation d'une heuristique pour améliorer une autre heuristique. Le principe est qu'une heuristique optimise le paramétrage de l'autre. Quelques études ont considéré ce principe : un GA pour optimiser un SA et des méthodes de bruitage [Krueger93], un GA pour des colonies de fourmis [Abbatista et al.95] et un GA pour un GA [Shahookar et al.90]. Dans [Shahookar et al.90], les taux de recombinaison, d'inversion et de mutation sont optimisés. Les individus du GA optimiseur sont des triplets qui codent ces trois taux. La *fitness* d'un triplet est calculée en exécutant l'autre GA (paramétré par ce triplet) ; la *fitness* du triplet est égale à la *fitness* de la meilleure solution obtenue.

1.3 Une grammaire taxinomique

Afin de représenter les différentes classes issues de notre taxinomie, nous proposons une notation concise et facile à déchiffrer :

- LRH(A_1 (A_2)) (nature, domaine, fonction)
- LCH(A_1 (A_2)) (nature, domaine, fonction)
- HRH(A_1 + A_2) (nature, domaine, fonction)
- HCH(A_1 , A_2) (nature, domaine, fonction)

Dans cette notation, les différents A_i représentent différentes métaheuristiques qui peuvent être, en autres :

- SA (*Simultated Annealing*) [Kirkpatrick et al.83] ;
- GA (*Genetic Algorithm*) [Holland75] ;
- ES (*Evolution Strategy*) [Rechenberg73] ;

- EP (*Evolutionary Programming*) ;
- GP (*Genetic Programming*) [Koza92] ;
- NN (*Neural Network*) ;
- DW (*Descent Walk*) [Papadimitriou et al.82] ;
- TS (*Tabu Search*) [Glover89a] ;
- GH (*Greedy Heuristic*) [Lawler76] ;
- AC (*Ant Colonies*) [Colorni et al.91] ;
- SS (*Scatter Search*) [Glover77] ;
- NM (*Noisy Method*) [Charon et al.93] ;
- CLP (*Constraint Logic Programming*) [Hentenryck89].

<métaheuristique_hybride> ::= <classe_hiérarchique> <classe_à_plat>
<classe_hiérarchique> ::= <LRH> | <LCH> | <HRH> | <HCH>
<LRH> ::= **LRH** (<métaheuristique> (<métaheuristique>))
<LCH> ::= **LCH** (<métaheuristique> (<métaheuristique>))
<HRH> ::= **HRH** (<métaheuristique> + <métaheuristique>)
<HCH> ::= **HCH** (<métaheuristique>)
<HCH> ::= **HCH** (<métaheuristique> , <métaheuristique>)
<classe_à_plat> ::= (<nature> , <domaine> , <fonction>)
<nature> ::= **homogène** | **hétérogène**
<domaine> ::= **global** | **partiel**
<fonction> ::= **généraliste** | **spécialiste**
<métaheuristique> ::= **DW** | **TS** | **SA** | **GA** | **ES** | **GP** | **NN**
<métaheuristique> ::= **GH** | **AC** | **SS** | **NM** | **CLP**
<métaheuristique> ::= <métaheuristique_hybride>

FIG. 1.8 - *Grammaire de la taxinomie des hybrides. Cette grammaire permet de décrire des schémas d'hybridation complexes.*

Cette notation (voir grammaire Fig. 1.8) permet de transcrire un grand nombre de schémas d'hybridation, des plus simples aux plus complexes. Par exemple, Boese *et al.* proposent une approche adaptative multidépart (AMS - *Adaptive Multi-Start*) qu'on peut représenter par le schéma HRH(LS + LCH(GA(DW))) [Boese et al.94].

D'abord, l'AMS génère, au hasard, un ensemble de solutions de départ ; puis, il exécute une descente (DW) sur chaque individu pour obtenir une population d'optima. Ensuite, l'AMS génère de nouvelles solutions de départ en combinant les caractéristiques de T meilleurs optima, T étant un paramètre de la méthode ; puis, une descente est appliquée à chaque solution pour donner une nouvelle population d'optima locaux qui seront combinés à leurs tours. l'AMS est ainsi itéré jusqu'à ce qu'un critère d'arrêt s'avère.

Dans son *PhD*, Levine propose un hybride pour le problème SPP (*Set Partioning Problem*) dont le schéma d'hybridation est `HCH( HRH( GH + LCH( GA( DW ))))`. De bons résultats ont été obtenus sur des problèmes réels de grande taille (planning d'équipages d'aviation civile). Le premier niveau de l'hybridation (HCH), est un modèle en îles de GA hybrides. La population de chaque algorithme génétique est générée par une heuristique gloutonne (hybridation HRH) et à chaque itération d'un GA, une descente optimise les solutions de la population (hybridation LCH). Un schéma `HRH( GH + LS + LCH( GA( GH + LS )))`, assez proche du précédent, est présenté dans [Freisleben et al.96] pour le problème du voyageur de commerce ; grâce à cette hybridation, l'algorithme génétique opère dans l'espace des optima locaux.

1.4 Conclusion

Dans ce chapitre, nous avons présenté une taxinomie des modèles d'hybridation des métaheuristiques de l'optimisation combinatoire. Ce classement, sans remettre en cause les taxinomies déjà existantes, les complète ; de plus, nous avons pris soin d'ouvrir suffisamment[3] notre classification pour qu'elle accueille de nouvelles formes d'hybridation. Par ailleurs, nous avons associé à cette classification, une notation concise et explicite qui permet de décrire de façon unique des schémas d'hybridations complexes.

3. Nous ne pouvons que l'espérer, pas l'assurer.

Chapitre 2

Métaheuristiques hybrides coévolutionnistes pour le QAP

Dans ce chapitre, nous présentons une étude expérimentale qui met en évidence l'importance de la diversification ou de l'intensification sur la performance de l'heuristique. Nous avons conçu une méthode hybride coévolutionniste spécialiste de haut niveau. Cette heuristique possède une configuration qui permet d'équilibrer la diversification et l'intensification. Nous relatons les résultats obtenus sur un large panel d'instances du QAP de différentes natures.

2.1 Parallélisme et coévolutionnisme

Classiquement, le parallélisme a été considéré comme un moyen d'augmenter la puissance de calcul en exécutant des tâches simultanément. Dans ce but, ont été développées des plates-formes SIMD, MIMD, réseaux ou fermes de stations de travail avec le sempiternel objectif de tirer parti du fait qu'exécuter des tâches en parallèle est plus rapide que les exécuter séquentiellement. Nous avons d'ailleurs tiré parti de ces architectures parallèles pour la résolution du QAP comme nous le décrivons au chapitre II:2. Cependant, une perspective complètement différente est apparue depuis quelques années, beaucoup moins technique, beaucoup plus conceptuelle. Selon cette approche, l'objectif n'est plus d'accélérer les traitements ; l'implantation est reléguée au second ordre, elle peut être aussi bien parallèle que séquentielle. L'intérêt majeur est d'avoir des activités qui coévoluent. En ce qui concerne l'hybridation des métaheuristiques, le modèle d'hybridation coévolutionniste de haut niveau (HCH) que nous décrivons au chapitre 1 correspond à cette approche (voir section 1.1.2.4).

Pour l'étude que nous relatons ici, nous avons tiré parti de ces deux approches du parallélisme : puissance de calcul et coévolution. Nous les avons combinées pour

concevoir des métaheuristiques parallèles hybrides coévolutionnistes HCH. L'objectif est ici de réaliser une méthode de recherche performante et robuste. Comme nous l'avons montré au chapitre II:2, à propos du QAP, certaines heuristiques manquent de diversification ou d'intensification. Aussi, plutôt que de tenter de corriger ces méthodes, au risque de les rendre plus complexes, nous avons choisi de faire coévoluer des agents simples dont les comportements sont connus.

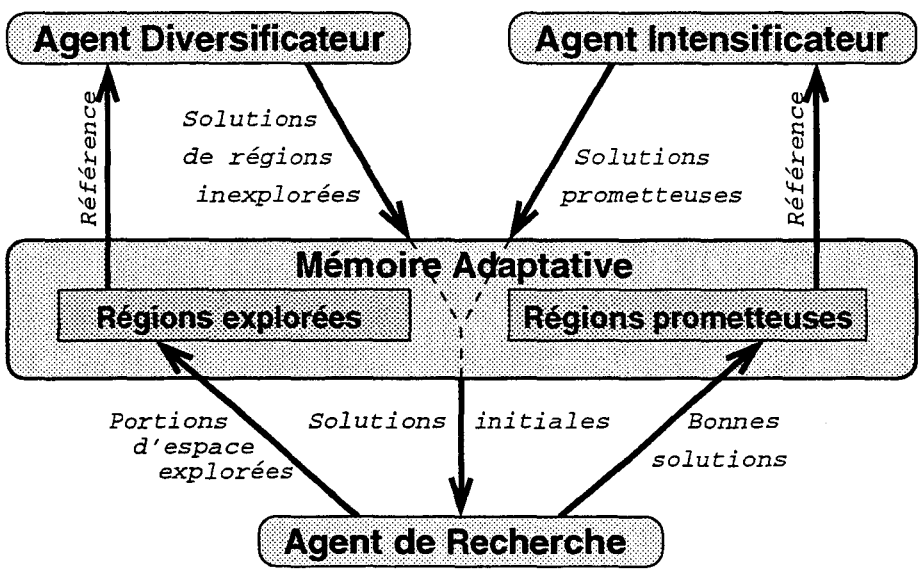


FIG. 2.1 - *Schéma fonctionnel des métaheuristiques hybrides coévolutionnistes que nous avons mis en œuvre.*

Une heuristique performante réalise un bon équilibre entre l'exploration de l'espace de recherche et l'exploitation des solutions déjà trouvées. Pour tenter d'atteindre ce juste équilibre, nous avons choisi de faire coopérer un agent de recherche de type ISI, un agent de diversification et un agent d'intensification (voir Fig. 2.1). Les agents communiquent au travers d'un organe passif de la méthode appelé la mémoire adaptative. Cette mémoire maintient un historique de la recherche; elle détient, notamment, des informations sur la portion de l'espace de recherche déjà visitée et sur la nature des bonnes solutions rencontrées. L'agent de recherche ISI reçoit les solutions de départ issues de régions non encore explorées en cas d'exploration ou de régions de bonne qualité en cas d'exploitation. Quand l'ISI termine une marche, elle renseigne la mémoire sur la portion d'espace qu'elle a explorée et sur la bonne solution qu'elle a trouvée. L'agent de diversification s'appuie sur la mémoire adaptative, qui contient une information sur la portion d'espace déjà visitée, pour produire une solution dans une région non encore visitée. L'agent d'intensification réfère à la mémoire adaptative, qui détient une information sur les régions de bonne qualité, pour fournir une solution dans l'une de ces régions.

Nous présentons dans la suite, deux instances du modèle hybride coévolutionniste décrit ci-avant. Le premier, appelé DGATS (*Diversifying Genetic Algorithm for Tabu Search*) implique deux agents uniquement : une méthode tabou parallèle pour la recherche et un algorithme génétique pour la diversification. L'autre, appelé AMTS (*Adaptive Multistart Tabu Search*), est la complétion de la métaheuristique coévolutionniste DGATS par un agent d'intensification qui accroît sa performance.

2.2 L'hybride coévolutionniste DGATS

Un algorithme génétique diversificateur pour la recherche tabou

La métaheuristique hybride DGATS que nous avons réalisée suit le modèle coévolutionniste décrit ci-dessus. Dans cette section, nous détaillons la mémoire adaptative et les différents agents impliqués. Pour chaque agent, nous donnons d'abord une description générale, en dehors de tout problème d'optimisation spécifique, puis, nous précisons les choix que nous avons adoptés pour adapter l'hybride à la résolution du QAP.

2.2.1 L'agent de recherche MTS

L'agent de recherche que nous avons choisi est l'heuristique tabou multiple (MTS) que nous avons décrite à la section II:2.3. La MTS est une méthode qui est déjà efficace à elle seule, car elle résout de manière (pseudo)optimale presque toutes les instances de petite taille que nous avons traitées (voir Tab. II:2.3). C'est une méthode simple que nous utilisons pour de courtes marches. Nous avons préféré la recherche TS à une descente SDW car elle est plus performante en qualité de solution, elle n'est pas beaucoup plus coûteuse, et on contrôle mieux la longueur de la marche et le temps d'exécution. En effet, dans la TS, on fixe le nombre d'itérations alors que la SDW s'arrête sur le premier optimum qu'elle rencontre.

Pour l'hybride, nous avons complété quelque peu la recherche tabou multiple. Nous avons augmenté la mémoire ISI de la TS pour qu'elle garde une information sur la portion de l'espace visité. Pour cela, nous utilisons une mémoire de fréquence. La mémoire de fréquence stocke des informations relatives à toutes les solutions successives le long de la marche de la recherche tabou. Elle mémorise la fréquence d'occurrence de certains événements pendant le recherche [Glover et al.92]. Soit H, l'ensemble des événements dont on relève le nombre d'occurences, on note F_e, la fréquence (le nombre d'occurences) de l'événement $e \in H$.

Pour le QAP, nous avons considéré les événements du type : « l'élément e_i est placé sur le site s_j ». La mémoire de fréquence (voir Fig. 2.2) est représentée par

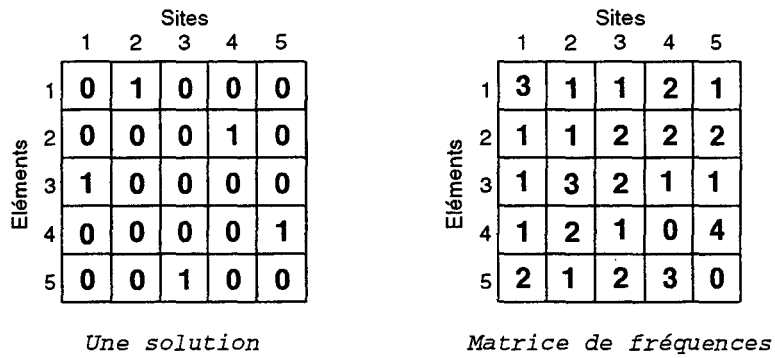


FIG. 2.2 - *Cette figure représente la solution (3, 1, 5, 2, 4) sous la forme matricielle ; la matrice de fréquence représentée correspond à la somme des solutions : (1, 3, 2, 5, 4), (2, 4, 1, 5, 3), (5, 2, 3, 1, 4), (4, 1, 5, 3, 2), (5, 4, 3, 2, 1), (1, 3, 5, 2, 4), (1, 3, 2, 5, 4), (3, 5, 4, 1, 2). Par exemple, la matrice de fréquence permet de savoir que l'élément 4 a été placé 4 fois sur le site 5 mais jamais sur le site 4.*

une matrice carrée d'ordre n, la taille de l'instance à résoudre. Cette matrice est obtenue en sommant les solutions courantes successives au long de la marche TS, ces solutions étant représentées sous forme matricielle (voir section I:1.2.1.1). On remarque que pour cette matrice de fréquence, la somme des coefficients sur une même ligne ou sur une même colonne est égale au nombre d'itérations effectuées par la marche. Cette remarque nous sera utile par la suite.

Au commencement de chaque marche TS, la mémoire de fréquence est initialisée à zéro. Durant la marche, elle est complétée avec les solutions courantes successives. Ainsi, à la fin de la recherche TS, La mémoire de fréquence représente la portion de l'espace de recherche explorée. À ce moment, la TS renseigne la mémoire adaptative en lui communiquant la meilleure solution trouvée et la mémoire de fréquence.

2.2.2 La mémoire adaptative du DGATS

La mémoire adaptative est l'organe de communication de la métaheuristique DGATS. Elle maintient un historique de la recherche de l'hybride. Elle est composée de deux parties qui servent pour la diversification par l'algorithme génétique : la mémoire de fréquence globale et l'ensemble des solutions de départ qui ont été fournies à la recherche tabou multiple (voir Fig. 2.3). La mémoire de fréquence globale correspond à la somme des matrices de fréquences locales fournies par les marches tabous. À chaque fois qu'une recherche tabou est achevée, la mémoire de fréquence globale est mise à jour en lui ajoutant la mémoire de fréquence locale. Pour le QAP, on obtient ainsi une matrice de fréquence globale dont la somme des coefficients sur une même ligne (ou sur une même colonne) est égale à la somme des

longueurs des marches déjà effectuées, égale au nombre total d'itérations effectuées par les recherches tabous. En plus de la mémoire de fréquence et de l'ensemble des solutions de départ, la mémoire adaptative contient la meilleure solution trouvée depuis le début de la recherche DGATS. Cette solution est mise à jour, si nécessaire, après l'achèvement d'une marche TS.

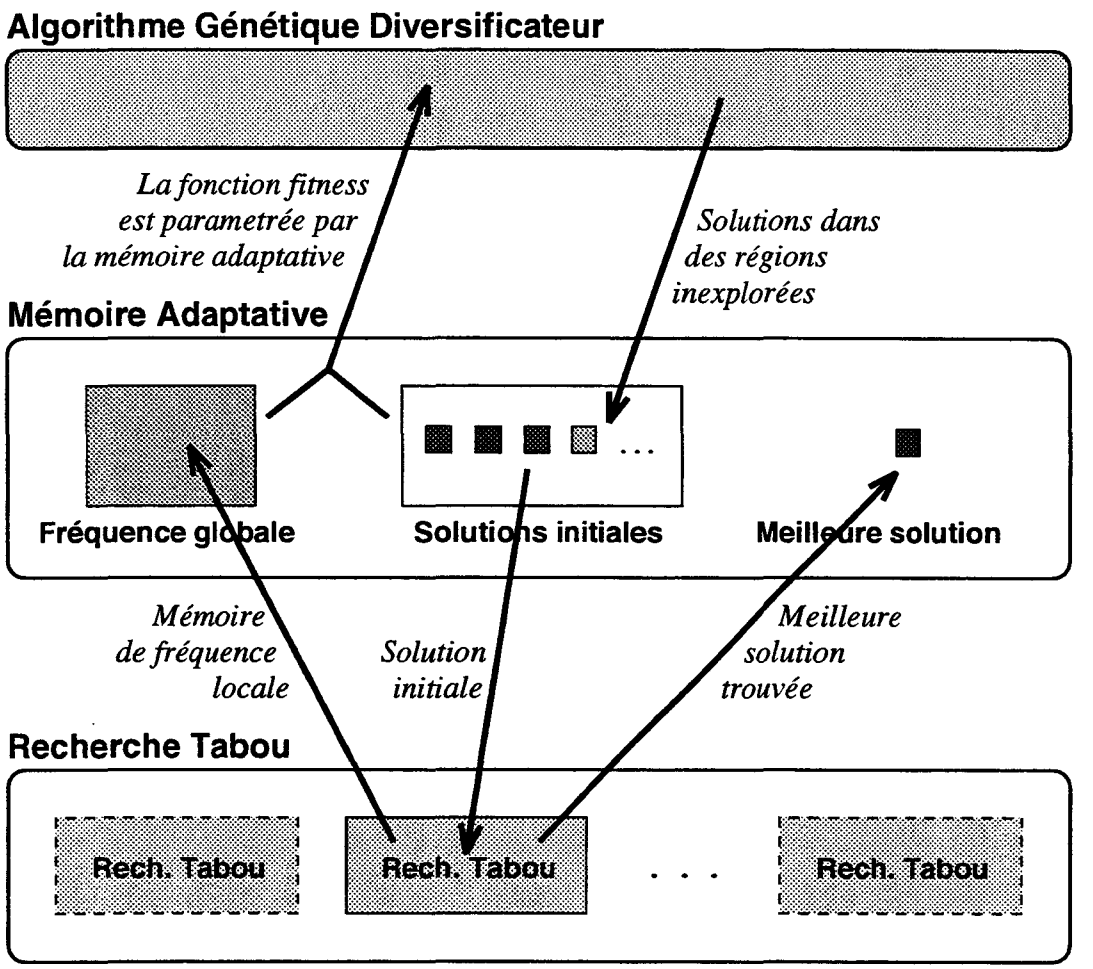


FIG. 2.3 - *Schéma fonctionnel de la métaheuristique coévolutionniste DGATS.*

2.2.3 Un GA diversificateur

L'agent de diversification que nous avons mis en place dans l'hybride DGATS est un algorithme génétique. Cependant, ce GA traite un problème d'optimisation complètement distinct de celui de la recherche tabou: selon notre taxinomie, le DGATS est une méthode hybride spécialiste. L'algorithme génétique est chargé de la

diversification en se référant à la mémoire adaptative. Son objectif est de fournir à la recherche tabou des solutions de départ dans des zones de l'espace de recherche peu explorées.

Plus précisément, l'algorithme génétique minimise la fonction objectif :

$$\varphi(u) = \alpha\, f(u) - g(u)$$

où α est un coefficient permettant d'équilibrer les influences de f et g. La fonction f est relative à la mémoire de fréquence globale ; la fonction g est relative à la mémoire de solutions initiales. u est une solution de la population de l'algorithme génétique. La fonction f se définit de la manière suivante :

$$f(u) = \sum_{e \in H} 1(e,u) F_e$$

où H est l'ensemble des événements considérés dans la mémoire de fréquence, F_e est la fréquence de l'événement e, $1(e,u)$ vaut 1 si l'événement e apparaît dans u, 0 sinon. La fonction g est définie comme suit :

$$g(u) = \min_{v \in D} (\mathrm{dist}(u,v))$$

où D est la mémoire de points de départ, $\mathrm{dist}(u,v)$ est la distance (dissimilarité) entre deux points u et v. Rappelons que cette distance doit être définie par rapport à l'opérateur utilisé dans les tabous, la distance entre deux points étant le nombre d'applications de l'opérateur nécessaires pour passer de u à v. Pour le QAP, nous utilisons donc la distance définie à la section I:2.2.1.3, basée sur l'opérateur de transposition. Cependant, le rôle de la fonction g étant d'évaluer la diversité d'une population, nous aurions pu utiliser la fonction d'entropie (voir section I:2.2.1.5) et réécrire g sous la forme :

$$g(u) = \mathrm{ent}(S \cup \{u\})$$

L'algorithme génétique diversificateur que nous avons utilisé se déroule comme suit. Au départ, la population initiale est générée uniformément. À chaque génération, les opérateurs de reproduction, recombinaison et mutation, sont appliqués en compétition sur le meilleur point de la population, le *crossover* utilisant un autre point tiré au hasard dans la population. Si un point, supérieur en qualité au meilleur point, est trouvé, alors il remplace le plus mauvais point de la population. Cet algorithme génétique, où au plus un point de la population est remplacé à chaque génération, suit le modèle *steady-state* défini par Whitley [Whitley88].

Pour la résolution du QAP, Nous avons choisi des opérateurs de reproduction classiques. Les opérateurs de mutation et de recombinaison sont, respectivement, la transposition et l'opérateur PMX (voir section II:2.1.2). Pour produire une nouvelle solution dans une région peu explorée, le GA optimise la fonction φ suivante :

$$\varphi(u) = \sum_{e \in H} 1(e,u)\, F_e - \min_{v \in D} (\mathrm{dist}(u,v))$$

En notant (F_{ij}) la matrice de fréquence globale, pour laquelle i et j décrivent respectivement les éléments et les emplacements, en notant $u(i)$ le site affecté à l'élément i pour la solution u, la fonction φ s'exprime par :

$$\varphi(u) = \frac{1}{N} \sum_{i=1}^{n} F_{u(i)i} - \min_{v \in D}(\text{dist}(u, v))$$

où N est le nombre total de solutions qui ont servi à constituer la matrice de fréquence globale[1]. La distance entre deux solutions u et v est calculée en temps linéaire (voir section II:2.2.1.3).

2.2.4 Évaluation de l'hybride parallèle DGATS

Afin de mettre en évidence le rôle de l'agent de diversification, nous avons comparé dans cette évaluation, l'heuristique DGATS avec une recherche tabou multidépart. La MTS est vue, ici, comme une méthode DGATS à laquelle on aurait enlevé l'algorithme génétique diversificateur. Les paramètres des deux métaheuristiques ont été choisis de manière à rendre les temps d'exécution semblables (sur des plates-formes matérielles identiques) : pour les heuristiques DGATS et MTS évaluées, le paramétrage de la recherche tabou est identique, notamment en ce qui concerne la longueur de marche. Ainsi, la comparaison des deux métaheuristiques parallèles est effectuée à coût de calcul constant, car l'effort de calcul nécessaire à l'agent de diversification est négligeable puisque extrêmement faible vis à vis de la puissance que requiert la recherche tabou seule. Pour l'évaluation de la DGATS, aussi bien que pour celle de la MTS, 1000 recherches tabous sont exécutées avec des marches de mêmes longueurs, plutôt courtes. Cependant, pour la DGATS, les solutions initiales proviennent de l'agent de diversification ; alors que, pour la MTS, les 1000 solutions initiales sont générées aléatoirement selon une distribution uniforme. Par ailleurs, pour la MTS, la mémoire adaptative comprend uniquement la meilleure solution trouvée par la recherche.

Pour le MTS, les 1000 recherches tabous sont exécutées simultanément. Pour la DGATS le séquencement des TS est tout autre. En effet, au départ, seule une partie des marches TS est exécutée pour fournir des résultats partiels et renseigner la mémoire adaptative. Ensuite, à partir de ces premiers résultats, le GA diversificateur produit de nouvelles solutions de départ pour relancer les recherches tabous. Plus précisément, l'hybride coévolutionniste se déroule comme suit : au départ, 100 recherches tabous sont lancées à partir de solutions initiales uniformément réparties dans l'espace de recherche. Puis, lorsqu'une recherche termine, elle renseigne la mémoire adaptative et une nouvelle recherche tabou est lancée avec une solution initiale fournie par l'agent diversificateur. Ce processus se termine lorsque les 1000 marches TS sont

1. Cette normalisation, la division par N, permet d'obtenir un résultat indépendant du nombre de solutions qui constitue la matrice de fréquence globale.

terminées. Ainsi, le GA fournit les 900 solutions initiales des derniers TS à partir de la mémoire de fréquence globale de la mémoire adaptative que renseigne les TS. La population du GA, de taille fixe, est constituée de n solutions qui sont, au démarrage de l'hybride, aléatoirement et uniformément choisies dans l'espace de recherche. Au cours de l'exécution de l'hybride DGATS, le GA diversificateur effectue n itérations du cycle *steady-state*, décrit ci-avant, pour fournir une nouvelle solution de départ.

Pour l'évaluation, les métaheuristiques MTS et DGATS ont été exécutées sur le système MARS selon un modèle maître / esclaves impliquant une bonne centaine de stations de travail de l'université, affectées usuellement aux chercheurs et aux étudiants. Nous décrivons plus précisément la plate-forme d'exécution à la section II:2.3.1. Lors de l'exécution, les différents agents et organes des métaheuristiques parallèles sont répartis comme suit. Le maître gère la mémoire adaptative et l'agent de diversification. Les esclaves, dont le nombre varie dynamiquement en fonction des disponibilités du système (sans toutefois excéder 100), effectuent les recherches tabous.

Instance	Type	φ(QAPlib)	Meilleure		Moyenne	
			MTS	DGATS	MTS	DGATS
Tai25a	I	1 167 256	**0.7361**	**0.7361**	**0.7361**	**0.7361**
Lipa50a	I	62 093	**0.0000**	**0.0000**	0.2963	**0.2754**
Tai100a	I	21 125 314	**0.9224**	0.9462	**1.0392**	1.0638
Tai256c	I	44 759 294	0.1775	**0.1749**	0.2236	**0.2179**
Els19	II	17 212 548	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Bur26d	II	3 821 255	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Nug30	II	6 124	**0.0000**	**0.0000**	**0.0000**	0.0065
Tai35b	II	283 315 445	**0.0000**	**0.0000**	0.0377	**0.0343**
Ste36c	II	8 239 110	**0.0000**	**0.0000**	**0.0746**	0.1033
Tai64c	-	1 855 928	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Sko64	III	48 498	**0.0701**	0.1443	**0.1938**	0.2182
Sko100a	III	152 002	0.2013	**0.1829**	0.2918	**0.2735**
Tai100b	III	1 185 996 137	**0.3824**	0.4780	0.6493	**0.6101**
Wil100	III	273 038	0.1055	**0.0894**	0.1921	**0.1854**
Tai150b	III	498 896 643	**0.8732**	1.0809	**1.2910**	1.3333
Tho150	III	8 133 484	**0.3490**	0.3981	0.4986	**0.4745**

TAB. 2.1 - *Résultats de l'évaluation comparative de la recherche tabou parallèle MTS et de l'heuristique coévolutionniste DGATS pour de courtes marches TS (5 n itérations) sur un panel d'instances de nature et de taille variées. Les résultats sont présentés sous la forme* $100 \times \frac{\varphi(s)-\varphi(QAPlib)}{\varphi(QAPlib)}$ *où s est la solution fournie par la métaheuristique. La colonne Meilleure montre la meilleure performance obtenue sur 10 exécutions, la colonne Moyenne présente la performance moyenne de ces exécutions. On constate qu'aucune des deux méthodes ne surpasse véritablement l'autre.*

2.2.5 Les résultats

Nous détaillons et commentons, ici, le résultat des expérimentations réalisées au cours de notre étude comparative des métaheuristiques parallèles MTS et DGATS à coûts de calcul égaux, visant à mettre en évidence l'impact de la diversification par le GA.

Instance	Type	φ(QAPlib)	Meilleure		Moyenne	
			MTS	DGATS	MTS	DGATS
Tai25a	I	1 167 256	**0.7361**	**0.7361**	**0.7361**	**0.7361**
lipa50a	I	62 093	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Els19	II	17 212 548	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Bur26d	II	3 821 255	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Nug30	II	6 124	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Tai35b	II	283 315 445	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Ste36c	II	8 239 110	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Sko64	III	48 498	**0.0000**	**0.0000**	0.0037	**0.0033**
Tai64c	-	1 855 928	**0.0000**	**0.0000**	**0.0000**	**0.0000**

TAB. 2.2 - *Résultats de l'évaluation comparative de la recherche tabou parallèle MTS et de l'heuristique coévolutionniste DGATS pour 100 n itérations par TS sur les instances de petite taille (moins de 100) que nous avons traitées. Les résultats sont présentés sous la forme* $100 \times \frac{\varphi(s)-\varphi(QAPlib)}{\varphi(QAPlib)}$ *où s est la solution fournie par la métaheuristique. La colonne Meilleure montre la meilleure performance obtenue sur 10 exécutions, la colonne Moyenne présente la performance moyenne de ces exécutions. On constate que pour la grande majorité des instances, la meilleure solution connue (QAPlib) est obtenue à chaque exécution, aussi bien pour la MTS que pour la DGATS.*

Nous avons réalisé une première évaluation avec de très courtes marches: 5 n itérations, où n est la taille de l'instance à résoudre (voir Tab. 2.1). On constate qu'aucune des deux méthodes ne surpasse véritablement l'autre. Ce résultat n'est pas surprenant pour les instances de type I car elles ne sont que très peu structurées et la diversification n'a donc que peu d'intérêt. Pour les instances de type II, les deux métaheuristiques trouvent la meilleure solution connue pour toutes les instances. En ce qui concerne les instances de type III, on observe que la MTS est supérieure pour la meilleure solution trouvée sur 10 exécutions, alors que, à l'inverse, la DGATS est plus performante en moyenne. En effet, sur 7 instances, la MTS surpasse 4 fois la DGATS pour la meilleure solution; et pour la moyenne, la DGATS surpasse 4 fois la MTS. De ce constat relatif aux instances de type III, nous déduisons que l'ajout de l'agent diversificateur sur la MTS, à effort de calcul égal, permet de gagner en exploration mais, en contrepartie, fait perdre en exploitation. En effet, on gagne en exploration car la DGATS fournit de plus nombreuses solutions de bonne qualité (supposées issues de plus de bassins); mais on perd en exploitation car la DGATS, comparée à la

MTS, trouve des solutions moins performantes. Ainsi, la DGATS parvient à trouver de nombreuses solutions de bonne qualité mais ne les exploite pas suffisamment. Pour y remédier, nous avons, dans une deuxième phase d'évaluation, allongé les marches des TS jusqu'à 100 n itérations. Comme nous l'avons observé à la section II:2.3.4, l'allongement des marches TS, est profitable pour les instances de type III.

Instance	Type	φ(QAPlib)	Meilleure		Moyenne	
			MTS	DGATS	MTS	DGATS
Tai100a	I	21 125 314	0.6867	**0.3960**	0.7834	**0.7347**
Tai256c	I	44 759 294	0.2359	**0.2279**	0.2708	**0.2391**
Sko100a	III	152 002	**0.0289**	**0.0289**	**0.0730**	0.0759
Tai100b	III	1 185 996 137	0.2445	**0.1518**	0.3969	**0.2817**
Wil100	III	273 038	0.0205	**0.0190**	**0.0352**	0.0395
Tai150b	III	498 896 643	0.9246	**0.8798**	1.1276	**1.1272**
Tho150	III	8 133 484	**0.0824**	**0.0824**	**0.1243**	0.1314

TAB. 2.3 - *Résultats de l'évaluation comparative de la recherche tabou parallèle MTS et de l'heuristique coévolutionniste DGATS pour 100 n itérations par TS sur les instances de grande taille (> 100) que nous avons traitées. Les résultats sont présentés sous la forme* $100 \times \frac{\varphi(s)-\varphi(QAPlib)}{\varphi(QAPlib)}$ *où s est la solution fournie par la métaheuristique. La colonne Meilleure montre la meilleure performance obtenue sur 10 exécutions (3 pour tai256c), la colonne Moyenne présente la performance moyenne de ces exécutions. On constate que la DGATS fournit une meilleure solution que la MTS sur toutes les instances.*

La seconde évaluation compare les métaheuristiques MTS et DGATS, à coûts de calcul égaux, pour des marches TS de 100 n itérations. Nous présentons les résultats obtenus en deux parties: d'une part les évaluations sur les instances de taille modeste (< 100), d'autre part les évaluations sur les instances de grande taille. En ce qui concerne les instances de taille modeste (voir Tab. 2.2), on constate que les deux méthodes sont très performantes car elles parviennent, pour 8 instances sur 9, à fournir la meilleure solution connue, cette performance étant obtenue à chaque exécution pour la grande majorité des instances. Ainsi, pour les instances de taille modeste, l'ajout de l'agent de diversification n'est pas utile, même si on observe une très légère supériorité de l'hybride DGATS sur la MTS pour l'instance Sko64 de type III. Pour les instances de grande taille, au contraire, on observe une très nette supériorité de la métaheuristique coévolutionniste DGATS (voir Tab. 2.3). En effet, la DGATS fournit une meilleure solution que la MTS sur toutes les grandes instances que nous avons testées. Néanmoins, au niveau absolu, les résultats obtenus par la métaheuristique coévolutionniste DGATS sur les grandes instances restent bien en deçà des meilleures solutions connues.

2.3 La métaheuristique AMTS

Bien que la DGATS soit une heuristique performante, nous nous sommes pas complètement satisfaits de ses performances, notamment sur les instances de grande taille. Nous pensons que la métaheuristique hybride DGATS gagnerait en performance si elle exploitait plus les solutions de bonne qualité qu'elle trouve. Pour aller dans cette direction, nous proposons la méthode hybride AMTS (*Adaptive Multistart Tabu Search*) : cette méthode reprend la DGATS dans laquelle on mémorise les solutions de bonne qualité et on intègre un agent d'intensification qui exploite ces solutions prometteuses.

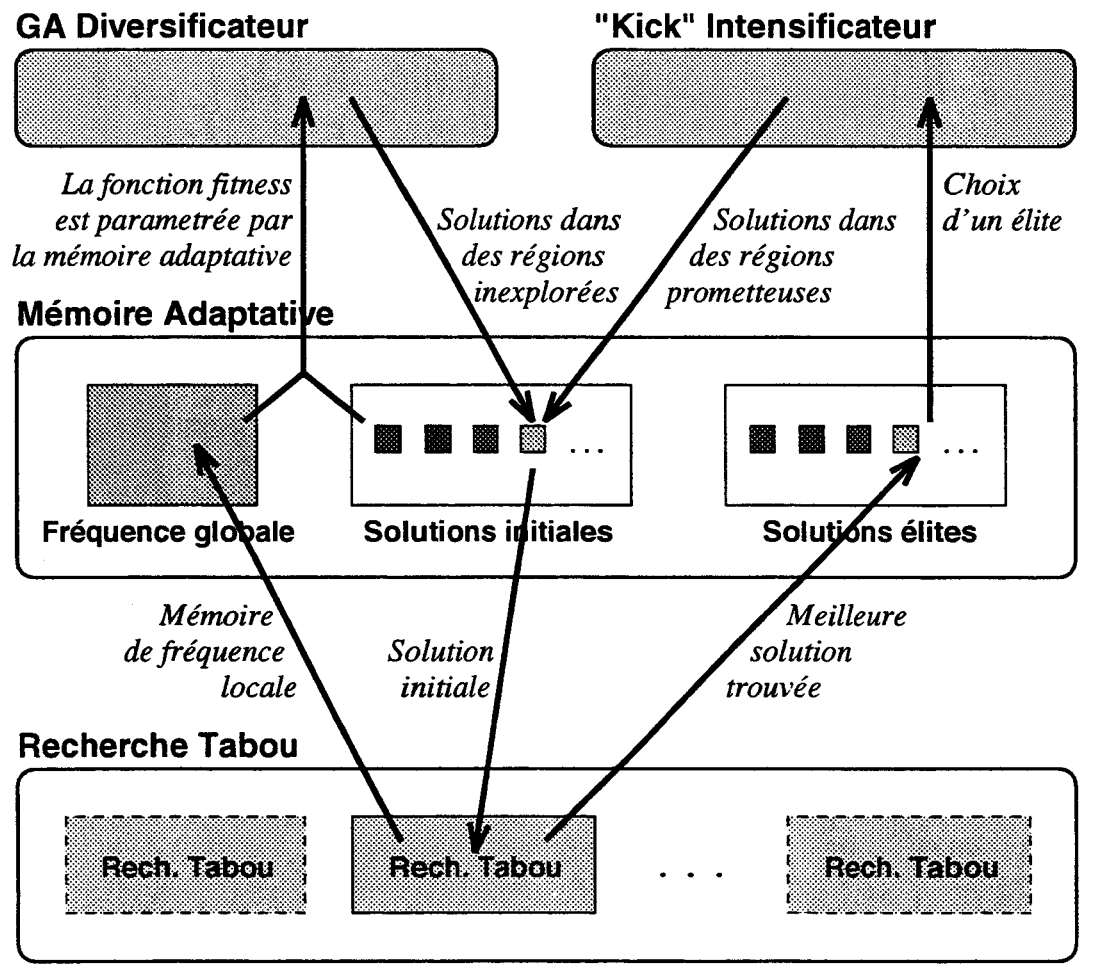


FIG. 2.4 - *Schéma fonctionnel de la métaheuristique coévolutionniste* **AMTS**.

2.3.1 La mémoire adaptative de l'AMTS

La mémoire adaptative de l'AMTS reprend celle de la DGATS presque à l'identique ; mais, au lieu de ne mémoriser que la meilleure solution trouvée, elle conserve un ensemble des meilleures solutions trouvées par les recherches TS. Cet ensemble de solutions élites constitue une représentation des régions de bonne qualité dans la portion d'espace explorée (voir fig. 2.4). La taille de cet ensemble de solutions élite est fixée et constitue un paramètre de la méthode.

On fait ici l'hypothèse qu'une solution de bonne qualité représente une région de bonne qualité. Cette hypothèse est avérée pour les instances structurée, car les bonnes solutions sont regroupées au fond des vallées. Le fonctionnement de cette mémoire d'élites est relativement simple. Quand une recherche tabou termine, elle communique sa meilleure solution trouvée à la mémoire adaptative. Si l'ensemble des élites n'est pas « plein » alors cette solution lui est ajoutée. Dans le cas contraire, quand l'ensemble des élites a atteint sa taille maximale, la nouvelle solution remplace la plus mauvaise solution élite si elle est de meilleure qualité. Dans tous les cas, une nouvelle solution ne sera pas intégrée à l'ensemble des élites s'il y a déjà au sein de ce dernier, un élite de même qualité. Cette précaution permet de préserver, à moindre coût, la diversité des solutions élites en limitant les redondances. On s'intéresse, dans ce mémoire, à une diversité de qualité de solution et non pas à une diversité de nature, car ce qu'on recherche, c'est une vallée plus profonde que les autres. Ainsi, en préservant la diversité des élites, la mémoire adaptative représente un plus grand nombre de régions prometteuses de différentes qualités.

2.3.2 L'agent intensificateur

L'agent intensificateur fournit des solutions initiales à la recherche tabou dans des zones prometteuses de l'espace de recherche à partir des solutions élites de la mémoire adaptative (voir fig. 2.4). Pour générer une nouvelle solution, l'agent intensificateur choisit une solution élite ; chaque élite ayant la même probabilité d'être choisie. Puis, il applique un opérateur de type « kick » sur la solution retenue. Le but de l'opérateur « kick » est de perturber la solution pour effectuer un saut dans l'espace de recherche (voir Fig. 2.5). Cependant, dans cet hybride, le rôle du « kick » est intensificateur, et il faut prendre garde à ne pas trop perturber la solution pour éviter de quitter la région prometteuse représentée par la solution élite. Si le « kick » modifiait trop la solution élite, on obtiendrait un opérateur de diversification aveugle, ce qui est à l'opposé du but recherché. Ici, on cherche à obtenir avec le « kick », une diversification limitée à une région prometteuse de l'espace de recherche. Pour résoudre le QAP, nous utilisons un opérateur « kick » qui consiste à appliquer un certain nombre de fois l'opérateur de transposition, ce nombre étant un paramètre de la métaheuristique hybride ; aussi il est souhaitable d'accorder ce nombre avec le diamètre des régions prometteuses.

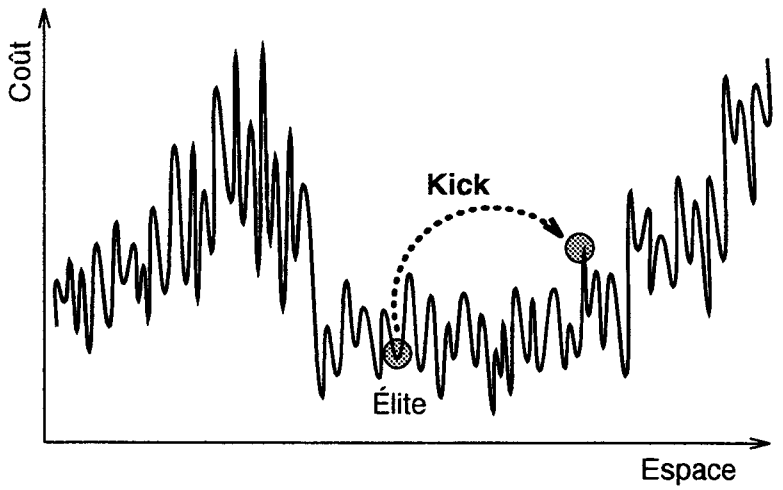


FIG. 2.5 - *Le « kick » effectue un saut dans l'espace de recherche. L'amplitude du saut ne doit pas être trop grande pour éviter de quitter la vallée de bonne qualité représentée par la solution élite.*

2.3.3 Évaluation de l'hybride AMTS

Dans la méthode coévolutionniste AMTS, la mémoire adaptative est dotée d'un ensemble de $\frac{n}{3}$ solutions élites qui servent à l'agent d'intensification, le « kick ». À chaque fois qu'une marche TS se termine, la meilleure solution trouvée durant cette marche est prise en compte dans l'ensemble des élites selon le protocole décrit précédemment. Dans l'AMTS, l'agent intensificateur et l'agent diversificateur sont impliqués ensemble pour fournir les 900 dernières solutions initiales de la recherche tabou. Ils interviennent, tour à tour, une fois sur deux, pour intensifier ou diversifier la recherche. Pour cette évaluation, le « kick » consiste en l'application de $\frac{n}{3}$ transpositions aléatoires.

Les évaluations ont été exécutées sur le système MARS selon une modèle maître/esclaves (voir section II:2.3.1). Le maître gère la mémoire adaptative et les agents d'intensification et de diversification. Les esclaves, dont le nombre varie dynamiquement en fonction des disponibilités du système d'exécution, effectuent les recherches tabous.

2.3.4 Les résultats

Dans l'évaluation de l'heuristique hybride AMTS, nous nous sommes attachés à mesurer l'influence de l'agent de diversification. Pour ce faire, nous avons comparé les performances l'AMTS et de l'hybride coévolutionniste AMTS. Puisque le MTS seul, parvient à résoudre à lui seul, les instances de petite taille, nous avons choisi d'évaluer les métaheuristiques hybrides sur de grandes instances ($n > 100$) uniquement.

Instance	Type	φ(QAPlib)	Meilleure		Moyenne	
			AMTS	DGATS	AMTS	DGATS
Tai100a	I	21 125 314	**0.3754**	0.3960	**0.5131**	0.7347
Tai256c	I	44 759 294	**0.1201**	0.2279	**0.1848**	0.2391
Sko100a	III	152 002	0.0434	**0.0289**	**0.0544**	0.0759
Tai100b	III	1 185 996 137	**0.0000**	0.1518	**0.1348**	0.2817
Wil100	III	273 038	**0.0029**	0.0190	**0.0085**	0.0395
Tai150b	III	498 896 643	**0.1903**	0.8798	**0.4393**	1.1272
Tho150	III	8 133 484	**0.0530**	0.0824	**0.0650**	0.1314

TAB. 2.4 - *Résultats de l'évaluation comparative des métaheuristiques hybrides coévolutionnistes AMTS et DGATS pour 100 n itérations par TS sur les instances de grande taille (> 100) que nous avons traitées. Les résultats sont présentés sous la forme* $100 \times \frac{\varphi(s)-\varphi(QAPlib)}{\varphi(QAPlib)}$ *où s est la solution fournie par la métaheuristique. La colonne Meilleure montre la meilleure performance obtenue sur 3 exécutions de AMTS, la colonne Moyenne présente la performance moyenne de ces exécutions. On constate que l'AMTS est supérieure à la DGATS sur la plupart des instances.*

La table 2.4 montre en vis à vis les résultats obtenus, à coûts de calcul égaux, par les deux métaheuristiques coévolutionnistes AMTS et DGATS sur des instances de grande taille. On constate que, pour presque toutes les instances, la méthode AMTS donne une meilleure solution. De plus, l'AMTS est en moyenne supérieure à la DGATS sur la totalité des instances. Ainsi, cette comparaison met en évidence la sensibilité de la performance d'une métaheuristique à l'équilibre entre l'exploration et l'exploitation. L'ajout, à la recherche DGATS, de l'agent d'intensification « kick » permet d'affiner cet équilibre et d'accroître la performance de la méthode d'une manière significative. Par ailleurs, les solutions fournies par la méthode AMTS sont de très bonne qualité ; elles avoisinent les meilleures solutions connues. Nous rappelons que ces résultats ont été obtenus avec un coût de calcul raisonnable correspondant à 100 000 n itérations de TS soit 100 000 $\frac{n(n-1)}{2}$ solutions évaluées. Par exemple, pour $n = 100$ (Tai100a, Wil100, ...), l'AMTS évalue 500 000 000 solutions et s'exécute en 4 heures environ selon la charge du réseau de stations.

2.4 Conclusion

Dans ce chapitre, nous avons proposé des métaheuristiques parallèles coévolutionnistes performantes dont la conception est directement inspirées des études que nous avons menées sur les paysages du QAP et sur les métaheuristiques itératives. En effet, la performance et la robustesse de ces méthodes coévolutionnistes provient d'un bon équilibre entre l'exploration de l'espace de recherche et l'exploitation des régions

prometteuses, en faisant coopérer des agents aux comportements complémentaires. Nous soulignons notamment la performance de la métaheuristique AMTS qui fournit des solutions de très bonne qualité dans un temps de calcul raisonnable. Ces bons résultats valident notre approche pour la résolution de problèmes combinatoires, tout au moins, en ce qui concerne le QAP.

Conclusion générale

Les travaux que nous avons réalisés et qui sont présentés dans ce mémoire, ont commencés en 1995. Cette année là, dans la communauté de l'optimisation combinatoire, la grande performance des métaheuristiques hybrides commençait à être reconnue. De très nombreuses méthodes hybrides avaient été expérimentées, très souvent avec succès, pour résoudre divers problèmes d'optimisation combinatoire. On constatait alors que la plupart des algorithmes hybrides performants mettaient en œuvre une recherche à solution unique (ISI) et une méthode à population de solution (IPI) (souvent un algorithme génétique intégrant une méthode de descente comme mutation). Cependant, peu de travaux avaient contribué à la justification de cette performance des méthodes hybrides. Dans ce contexte, nous nous sommes donné l'objectif de mieux comprendre le fonctionnement des métaheuristiques hybrides afin de concevoir des méthodes plus efficaces. Nous nous sommes aussi tournés vers les parallélisations de métaheuristiques et vers les plates-formes et les environnements distribués dans le but de limiter le temps d'exécution. Ainsi, nous avons conduit une étude sur les métaheuristiques parallèles hybrides, en prenant le QAP comme support d'application. Nous avons abordé cette étude sur trois fronts :

- l'étude de la structure intrinsèque des instances du QAP ;
- l'étude des métaheuristiques sur des environnements distribués ;
- l'étude des mécanismes d'hybridation et de coévolution.

Pour étudier la nature des instances du QAP, nous avons étudié leurs différents paysages. Nous avons utilisé un paysage construit par un opérateur de transposition. Nous avons choisi cet opérateur car il est presque exclusivement utilisé dans les recherches ISI appliquées au QAP. Le paysage ainsi construit est la représentation de l'univers dans lequel évoluent les recherches itératives. Nous n'avons pas retenu les approches globales du paysage car elles prennent en compte un très grand nombre de solutions de médiocre qualité qui ne sont pas visitées par les métaheuristiques. Nous avons adopté une démarche, plus proche des métaheuristiques, basée sur l'étude du comportement d'une méthode de descente. Nous avons sélectionné et proposé un certain nombre d'indicateurs statistiques qui font ressortir trois tendances principales dans la nature des instances : les instances de type I ont un paysage plat et

rugueux ; les instances de type II sont de petite taille et leur paysage comprend un regroupement central des optima locaux de bonne qualité qui constitue un massif ; les instances de type III sont plutôt grandes, leur paysage a une structure intermédiaire entre les instances des types I et II, il est formé de plusieurs massifs d'optima locaux éparpillés dans l'espace de recherche. Cette taxinomie en trois catégories, obtenue par calcul, rejoint d'autres classements qui ont été obtenus de manière empirique par la communauté œuvrant sur le QAP.

Un autre front d'étude concerne les métaheuristiques itératives. Nous avons conduit cette étude selon une approche qui distingue deux catégories : les méthodes ISI et les méthodes IPI. Pour chacune de ces catégories, nous avons proposé un modèle qui fédère les métaheuristiques les plus connues. Nous avons constaté que ces deux catégories ont des modèles de parallélisation différents. Nous avons décrit l'étude de quelques métaheuristiques parallèles ISI et IPI appliquées au QAP sur différents environnements et plates-formes d'exécution parallèle. Sur le plan des environnements d'exécution, il ressort de notre étude que le système MARS sur réseau de stations de travail (NOW) est un outil bien adapté à l'exécution de métaheuristiques itératives. En effet, grâce à son caractère adaptatif, MARS permet de toujours obtenir le maximum de ressources de calcul que le NOW peut produire. De plus, MARS est muni d'un mécanisme de sauvegarde/restauration qui le rend tolérant aux pannes si fréquentes lors de longues exécutions pour résoudre des instances de grande taille. En ce qui concerne l'évaluation des métaheuristiques, nous avons constaté que les ISI sont plus efficaces sur les instances uniformes (type I) et qu'à l'inverse, les IPI sont plus performantes sur les instances fortement structurées (type II). Nous avons justifié cette divergence de performance en montrant la nature exploiteuse des ISI et la nature exploratrice des IPI. Ces constatations nous ont amené à considérer des métaheuristiques hybrides ISI/IPI pour résoudre les instances de type III qui sont intermédiaires entre les instances de type I et II, en rétablissant par hybridation l'équilibre exploration/exploitation qui est un facteur de performance. Ainsi, d'un point de vue pragmatique, notre étude permet de choisir le type de métaheuristique à utiliser pour résoudre une instance. En effet, pour une instance inconnue, on détermine son type à l'aide des indicateurs que nous avons proposés. En fonction du type obtenu, on sait si une ISI ou une IPI peuvent fournir une solution de bonne qualité ou s'il faut utiliser une métaheuristique hybride.

Dans notre présentation des métaheuristiques hybrides, outre un état de l'art synthétique sous la forme d'une taxinomie originale, nous avons proposé une métaheuristique hybride qui combine les deux propriétés bénéfiques du parallélisme que sont la puissance de calcul et la coévolution. La puissance de calcul accélère les traitements ; la coévolution permet de faire coopérer des agents aux comportements complémentaires. Dans le but de concevoir une recherche qui profite d'un bon équilibre entre exploration et exploitation, l'hybride que nous proposons est basé sur la coévolution d'un agent de recherche ISI, d'un agent de diversification, et d'un agent d'intensification. Les trois agents coopèrent à travers un organe passif qui est

la mémoire adaptative. Cette mémoire contient des informations sur les zones de l'espace de recherche qui ont déjà été visitées et sur les zones prometteuses. Les agents de diversification et d'intensification, en se référant à la mémoire adaptative, fournissent des solutions de départ à l'agent de recherche. L'agent de diversification a un rôle explorateur en produisant des solutions dans des zones non encore explorées. L'agent d'intensification a un rôle exploiteur en fournissant de solutions dans des zones prometteuses. Nous avons appliqué ce modèle de métaheuristique coévolutionniste au QAP. Nous avons mis en œuvre une recherche tabou multiple comme agent de recherche, un algorithme génétique comme agent de diversification et un opérateur « kick » comme agent d'intensification. Grâce à cette métaheuristique hybride, nous avons égalé, pour de nombreuses instances du QAP, les meilleurs résultats connus.

Perspectives

Dans ce mémoire, nous présentons une démarche qui consiste à analyser la nature d'un problème (pour nous, le QAP) pour adapter une méthode de résolution. Il nous semble intéressant d'appliquer cette démarche à d'autres problèmes pour valider cette approche.

En ce qui concerne la métaphore des paysages, nous avons constaté qu'elle est un outil de description des instances qui fonctionne pour le QAP. Cependant, pour les paysages d'autres problèmes, il faudrait observer si les indicateurs de mesure du paysage que nous avons utilisés pour le QAP sont applicables, ou sinon en proposer d'autres. Actuellement, en collaboration avec des chercheurs de l'université de Tunis I, nous étudions la structure des paysages du problème de coloration de graphe.

Au niveau des plates-formes et environnements d'exécution, nous avons utiliser principalement le réseau de stations de travail de notre laboratoire et de son département d'enseignement. Il serait intéressant de s'orienter vers des réseaux de plus large échelle pour résoudre plus efficacement des instances plus grandes. Des systèmes de *métacomputing* pour l'optimisation ont vu le jour ces dernières années. Ils fédèrent la puissance de calcul de gros calculateurs répartis sur la planète.

Il serait aussi très attrayant d'appliquer les méthodes hybrides coévolutionnistes que nous avons proposées à d'autres problèmes combinatoires. Il nous semble que ces algorithmes hybrides coévolutionnistes devraient fonctionner efficacement car leur approche est générale. En effet, ils s'appuient sur la coévolution d'un agent de recherche, d'un agent d'intensification, et d'un agent de diversification afin d'obtenir un bon équilibre intensification/diversification. Les trois agents coopèrent à travers la mémoire adaptative qui détient des informations sur la portion d'espace déjà visitée et sur la nature des bonnes solutions.

Quatrième partie

Annexes

Annexe A

Le paysage du QAP
un recueil de graphiques

A.1 Préambule

Voici un recueil de graphiques regroupés par instances, classées de la plus petite à la plus grande. Ces graphiques résultent de l'étude statistique des paysages du QAP que nous présentons au chapitre I:2. Dans cette étude, 1000 solutions sont tirées au hasard de façon uniforme. Puis une marche SDW (selon la plus forte pente) est appliquée à chaque solution. On obtient ainsi 1000 optima locaux.

• Les graphiques dont le titre comporte la mention (random) correspondent à un ensemble de solutions initiales générées uniformément.

• Les graphiques ayant le mention (distant) ont été obtenues avec des solutions initiales dont les distances sont échelonnées par rapport à la meilleure solution connue.

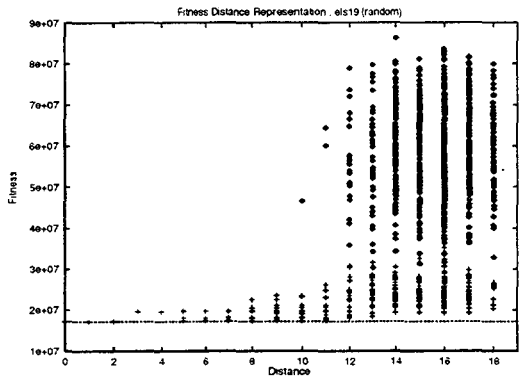
• Les graphiques intitulés *Fitness Distance Représentation* donnent une représentations des solutions en fonction de leur fitness et de leur distance par rapport à la meilleure solution connue (voir section I:2.3.2.3). On distingue deux nuages de solutions : en haut, les solutions initiales ; en bas, les optima locaux. La ligne en pointillé montre la fitness de la meilleure solution connue.

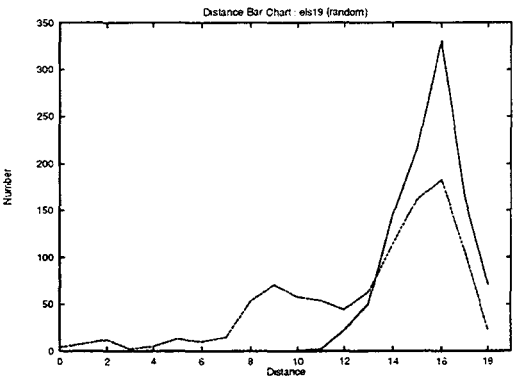
• Les graphiques intitulés *Distance Bar Chart* donnent le nombre de solutions en fonction de la distance à la meilleure solution connue (voir section I:2.3.2.3). La ligne pleine (resp. en pointillés) correspond aux solutions initiales (aux optima locaux).

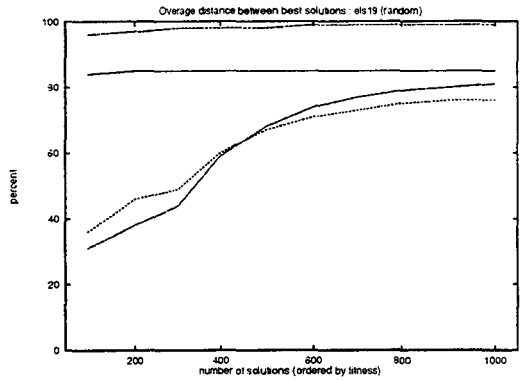
• Les graphiques intitulés *Average distance between the best solutions* montrent la dispersion des meilleures solutions en fonction de leur nombre (voir section I:2.3.2.5).

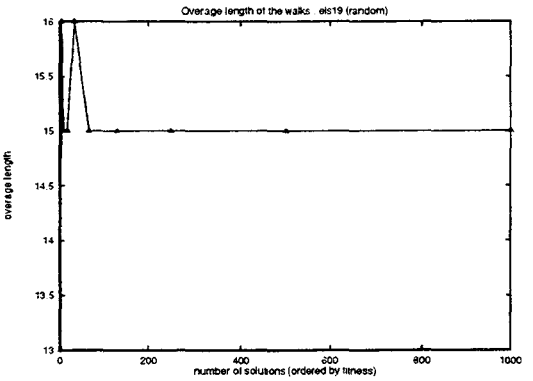
• Les graphiques intitulés *Average length of the walks* donnent la longueur de marche des descentes en fonction de la qualité de la solution trouvée.

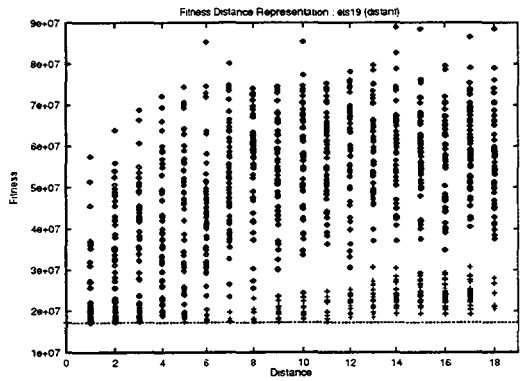
A.2 Instance Els19

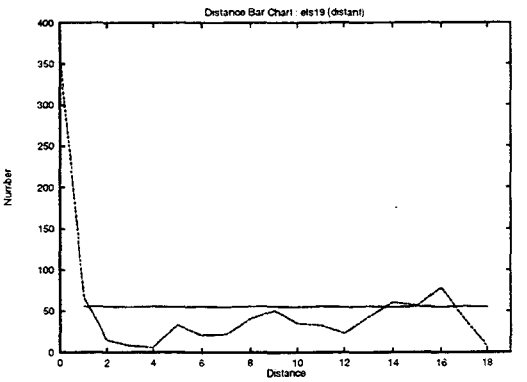




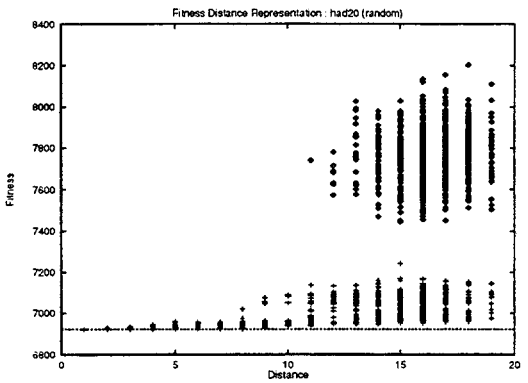


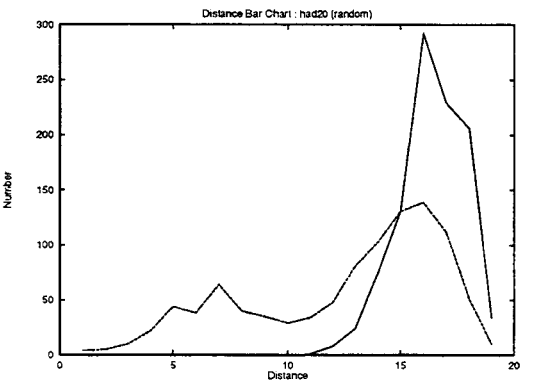


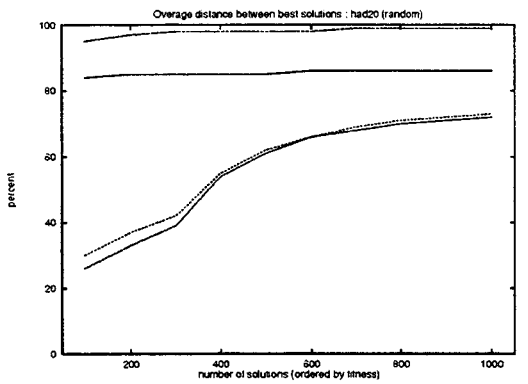


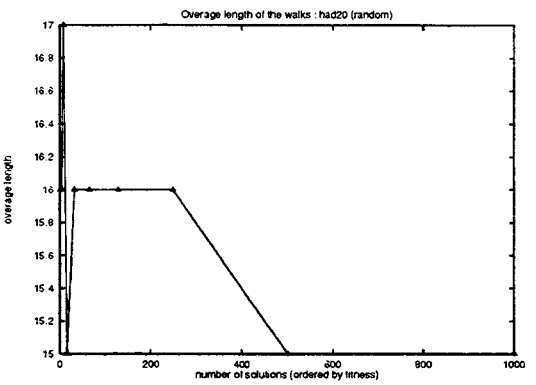


A.3 Instance Had20

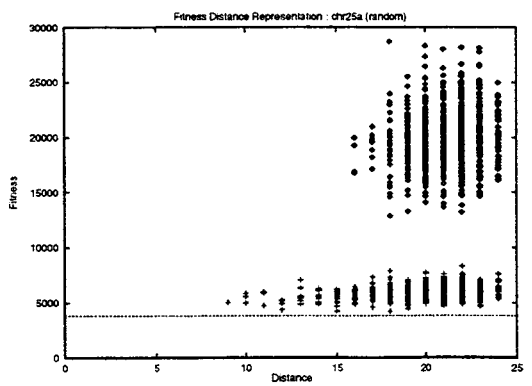


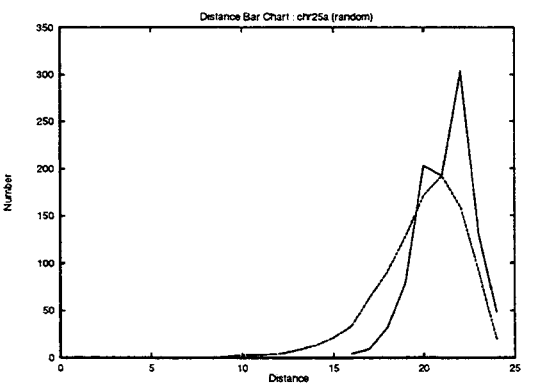


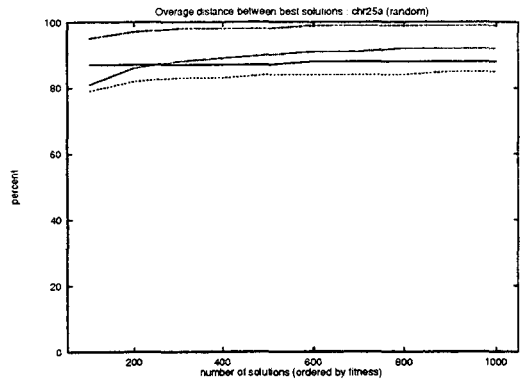


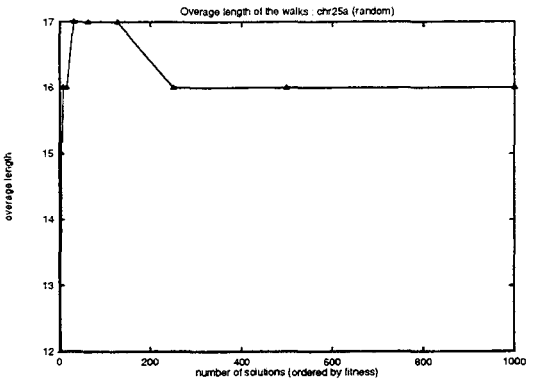


A.4 Instance Chr25a

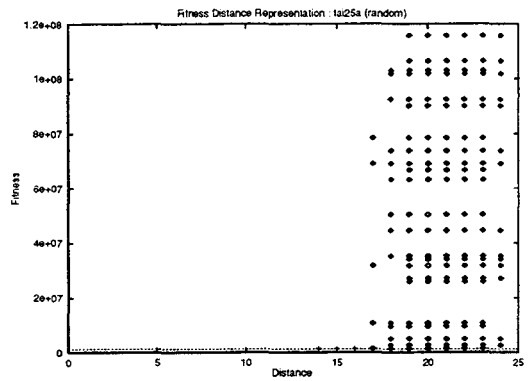


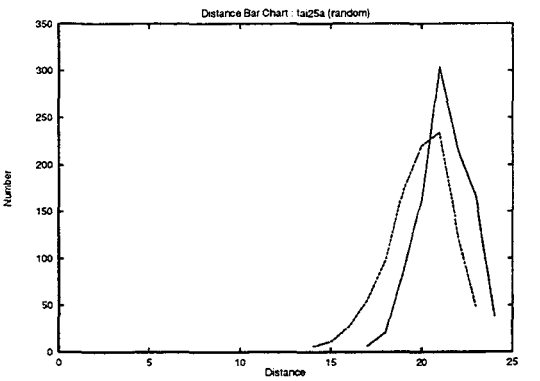


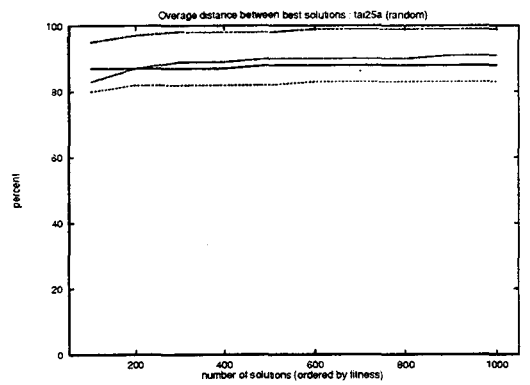


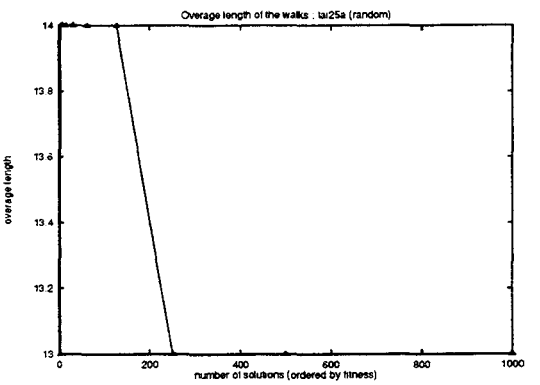


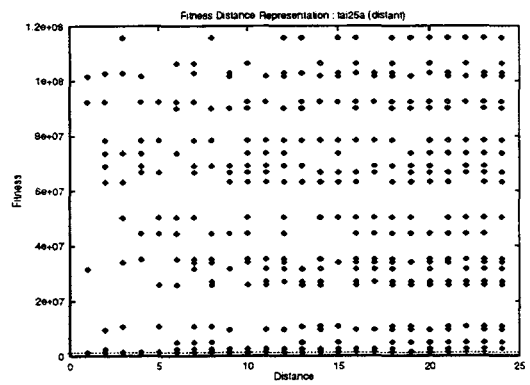
A.5 Instance Tai25a

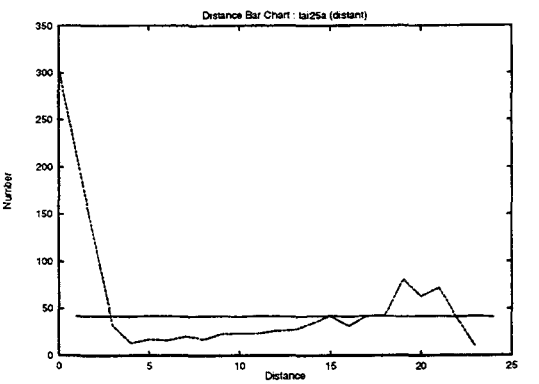




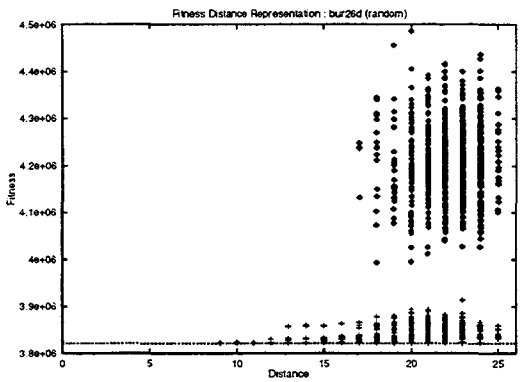


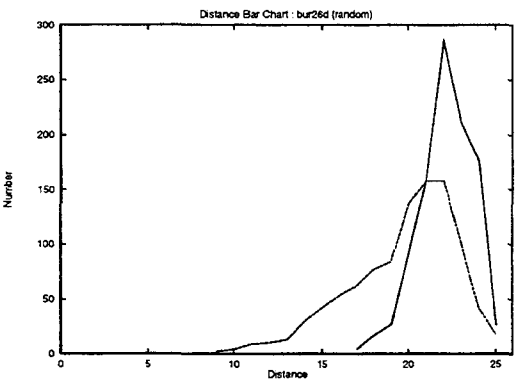


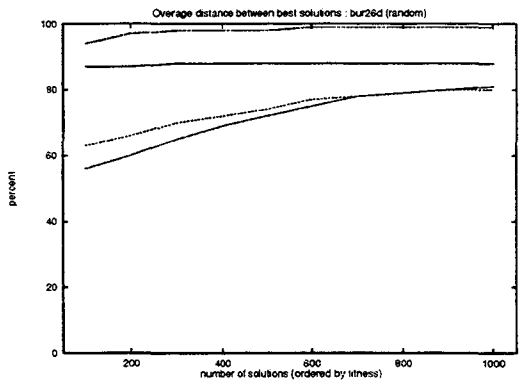


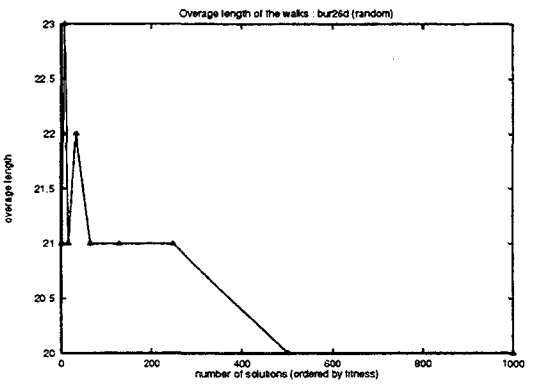


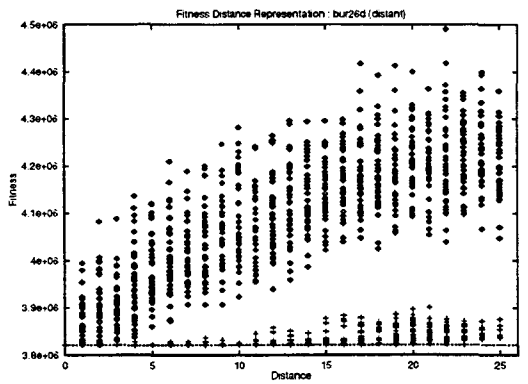
A.6 Instance Bur26d

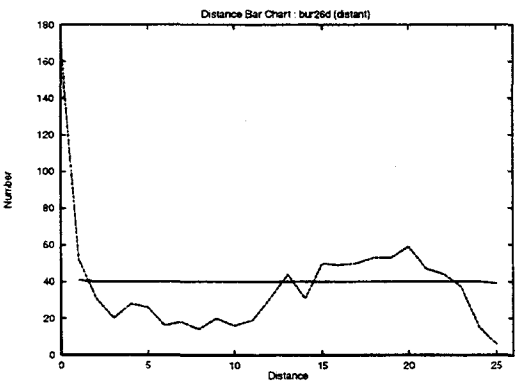




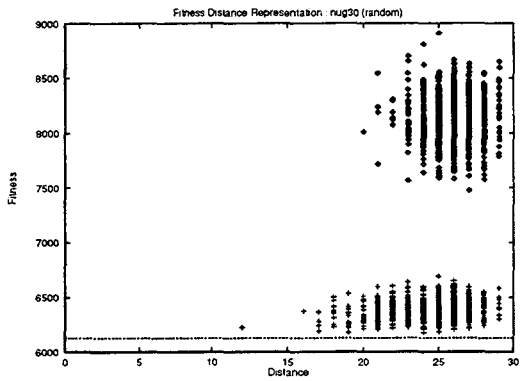


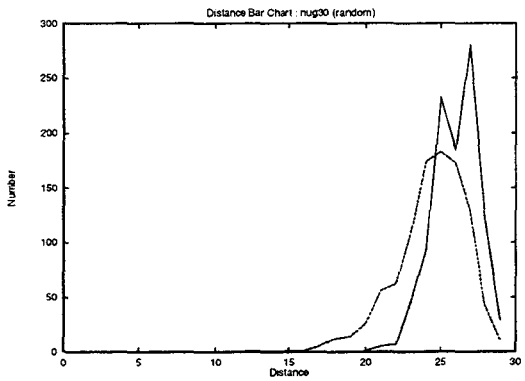


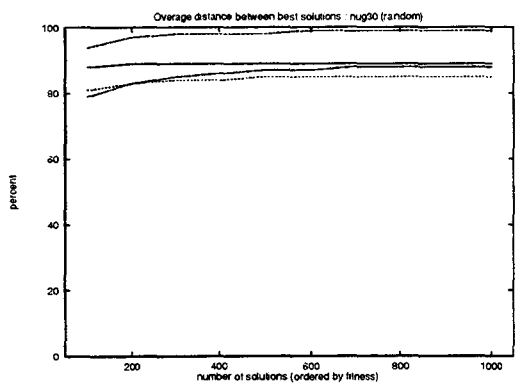


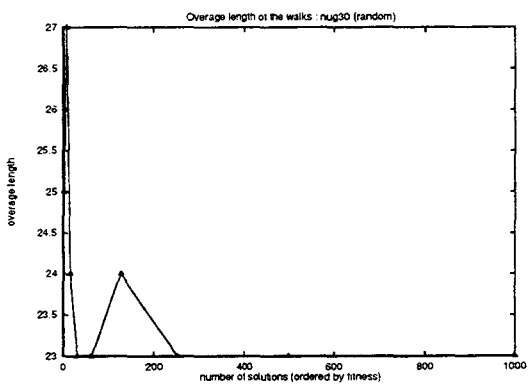


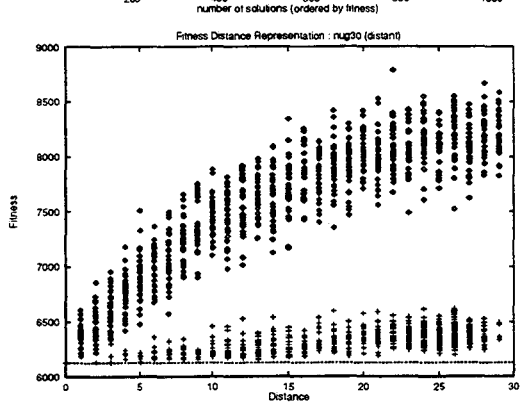
A.7 Instance Nug30

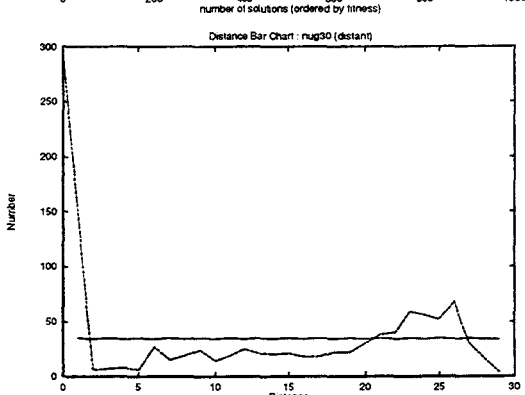




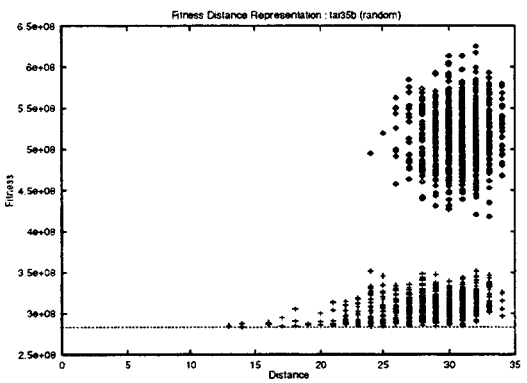


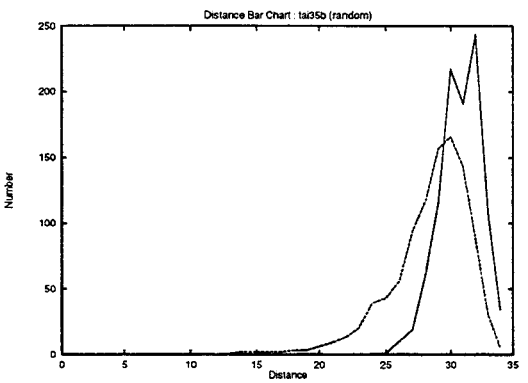


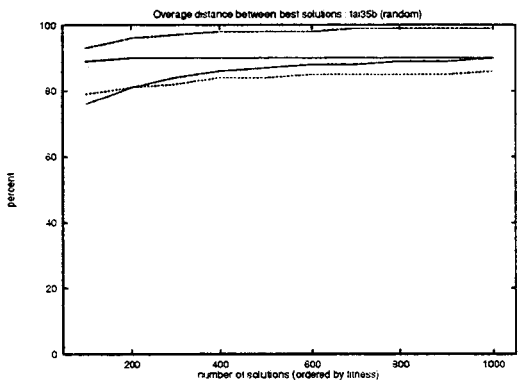


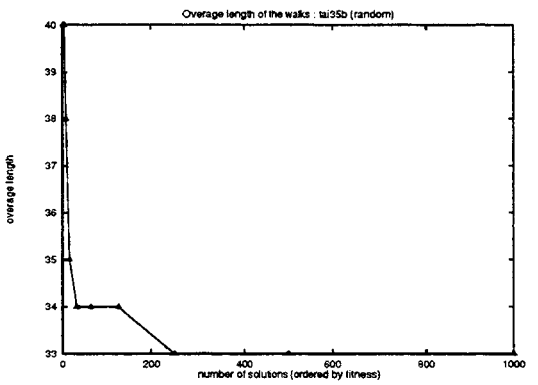


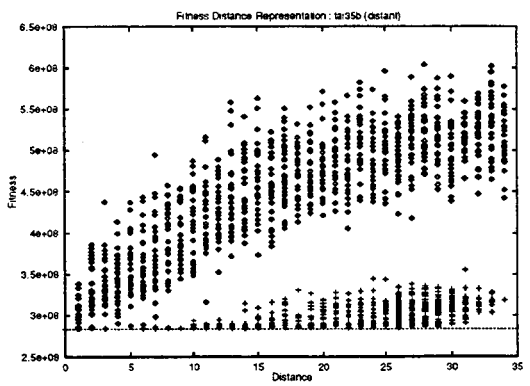
A.8 Instance Tai35b

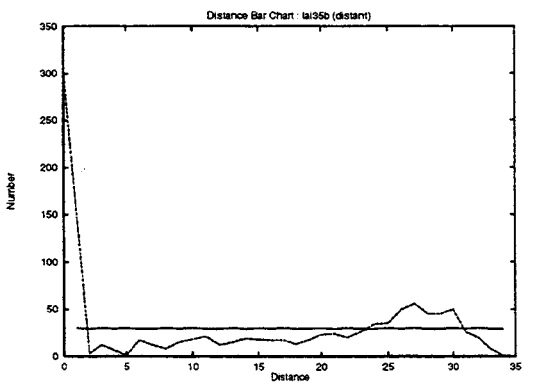




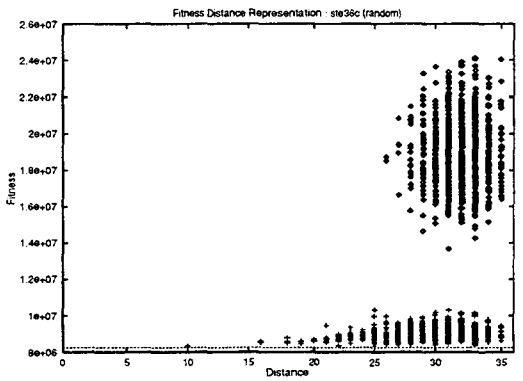


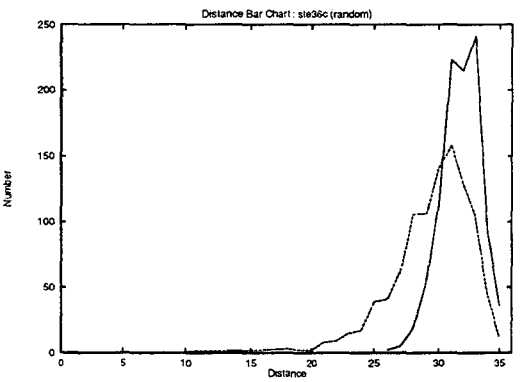


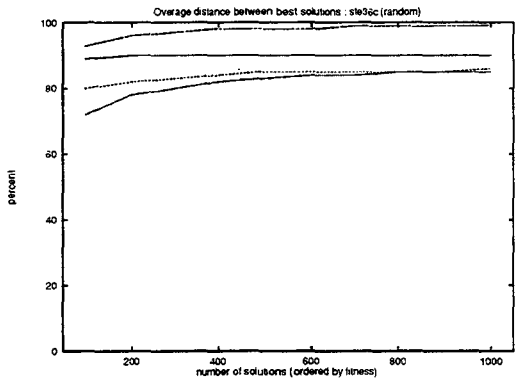


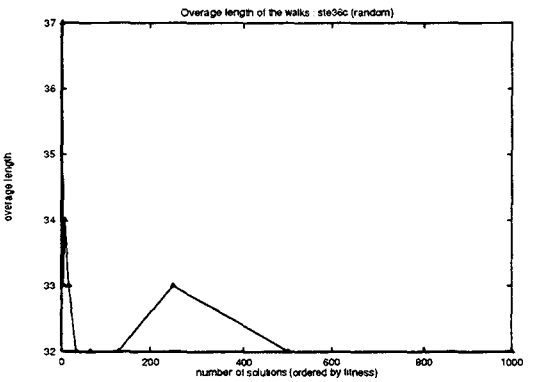


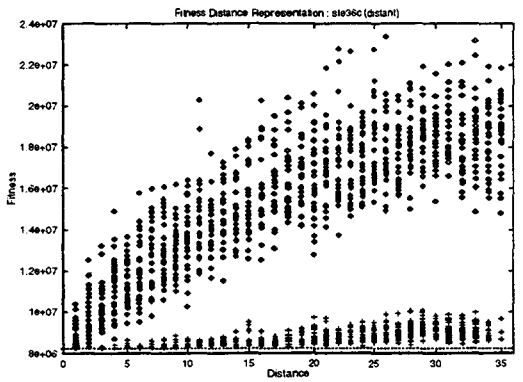
A.9 Instance Ste36c

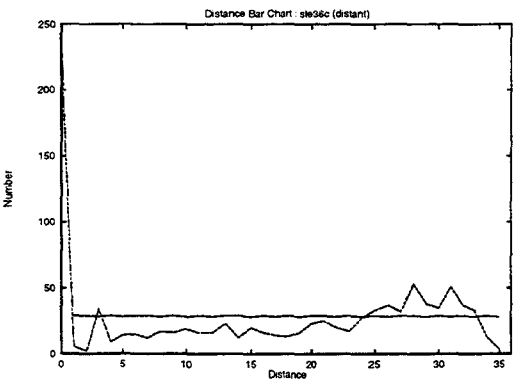




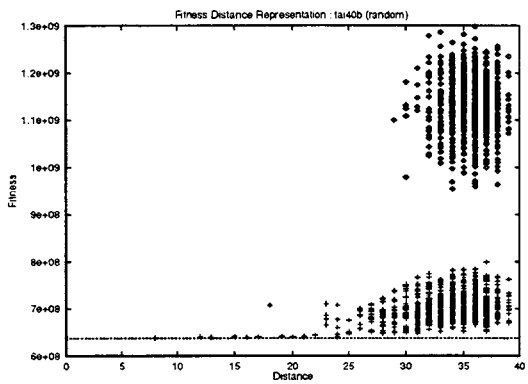


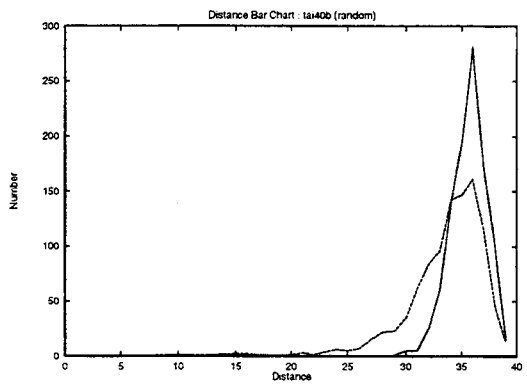


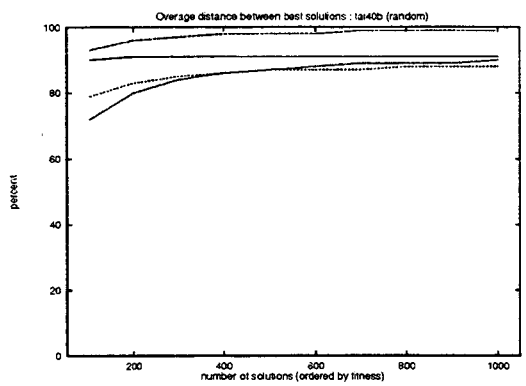


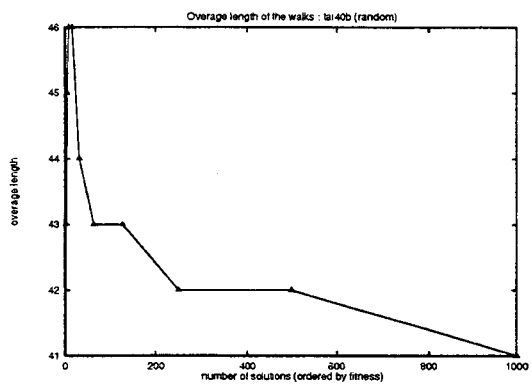


A.10 Instance Tai40b

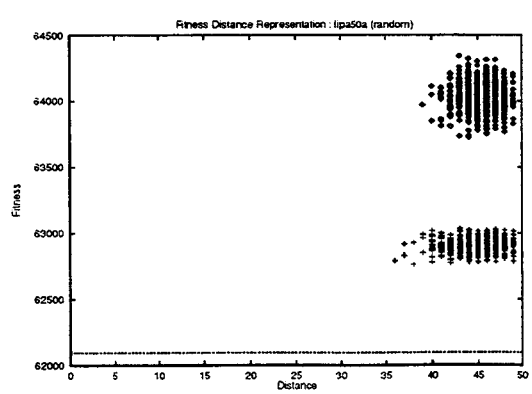


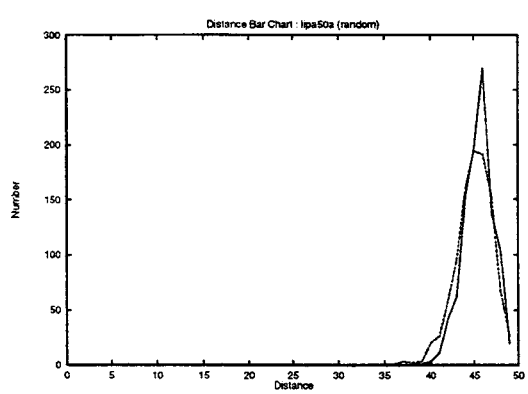


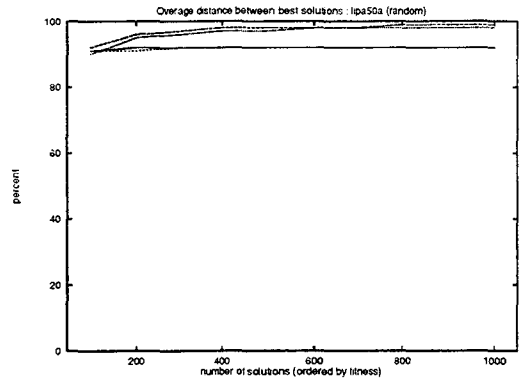


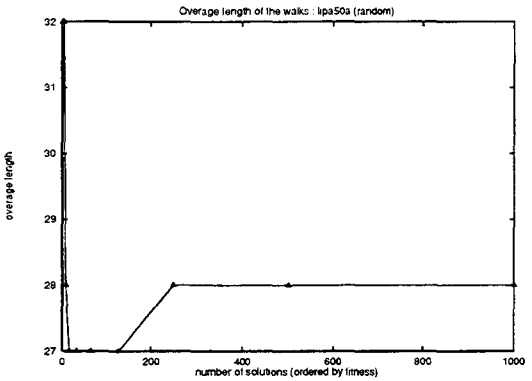


A.11 Instance Lipa50a

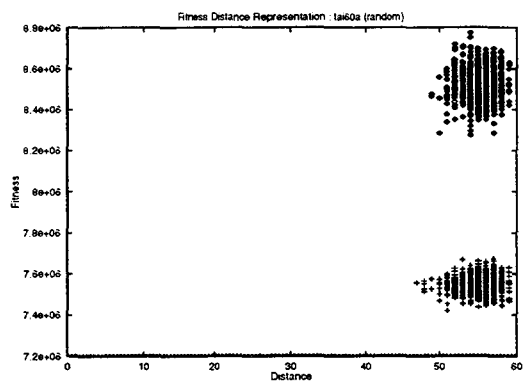


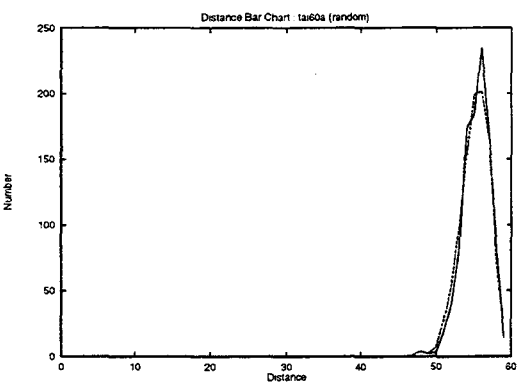


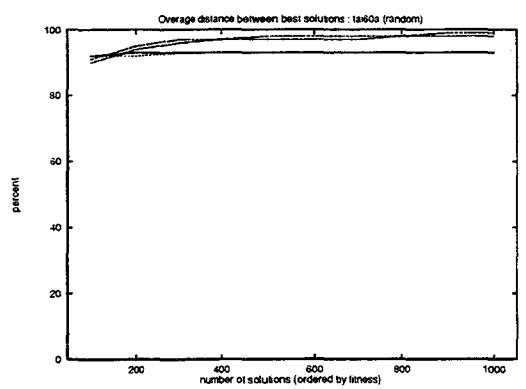


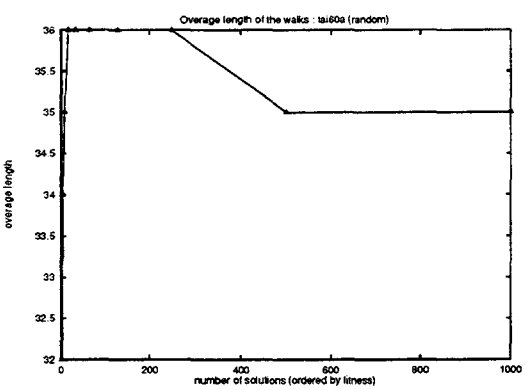


A.12 Instance Tai60a

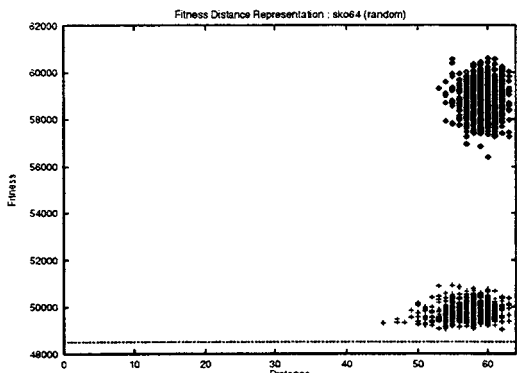


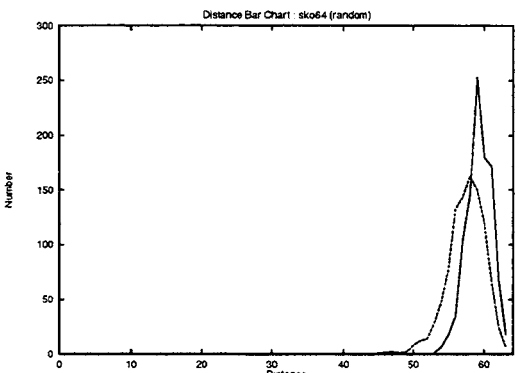


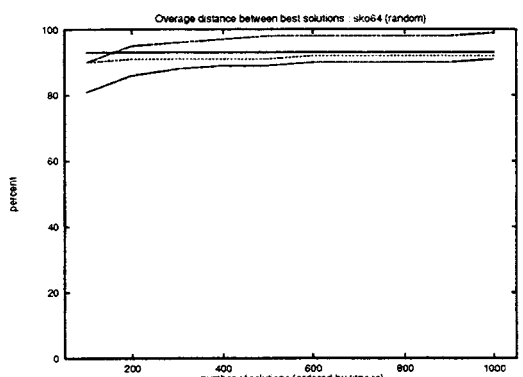


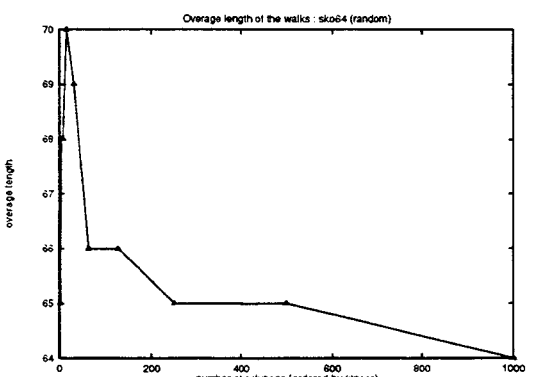


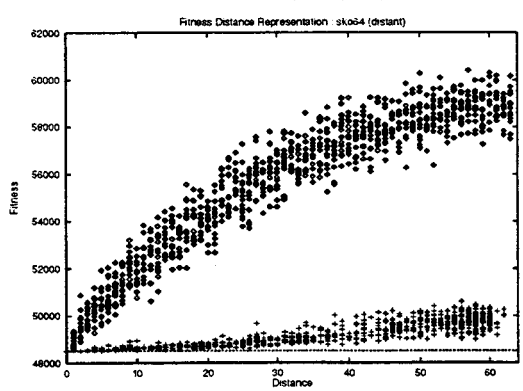
A.13 Instance Sko64

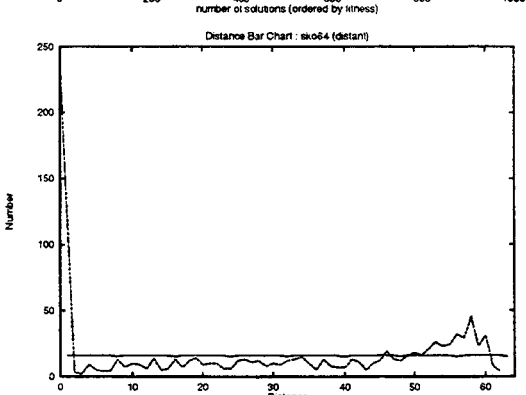




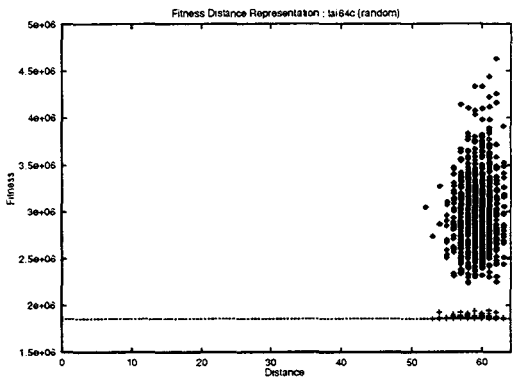


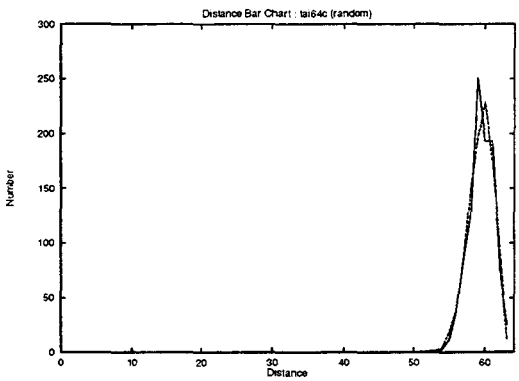


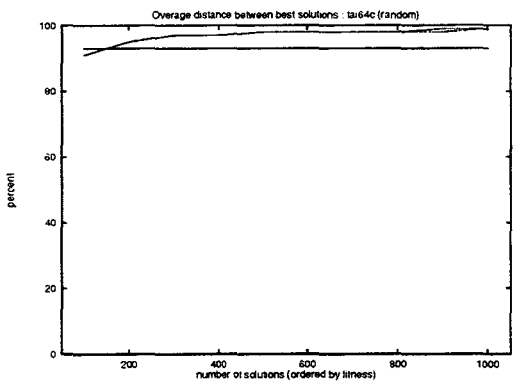


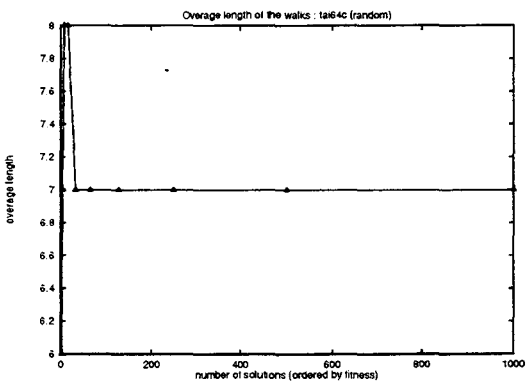


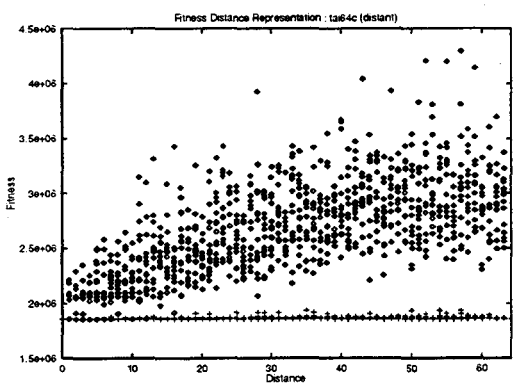
A.14 Instance Tai64c

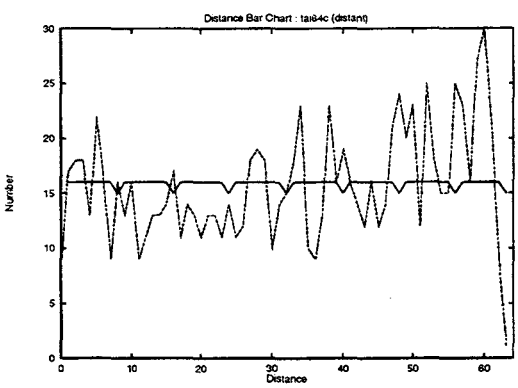




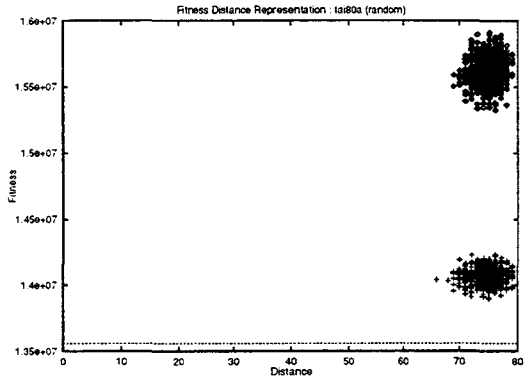


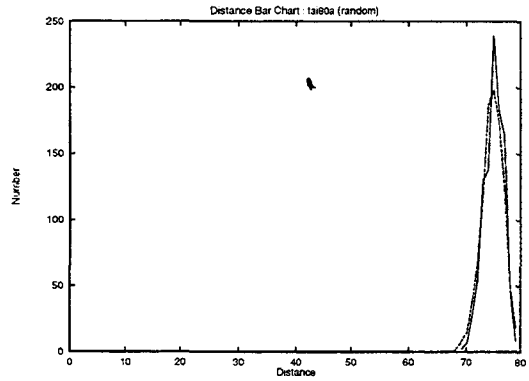


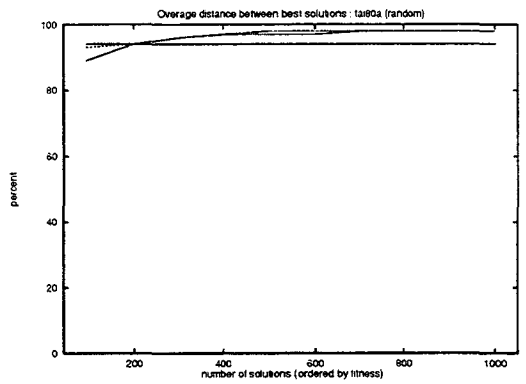


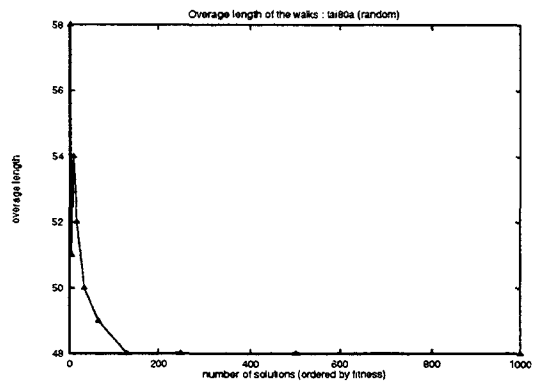


A.15 Instance Tai80a

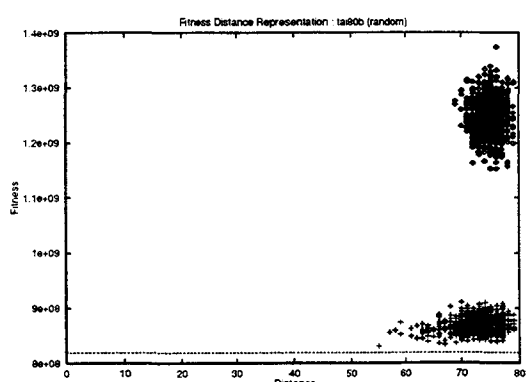


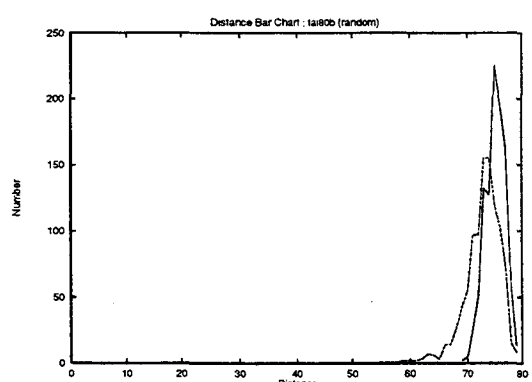


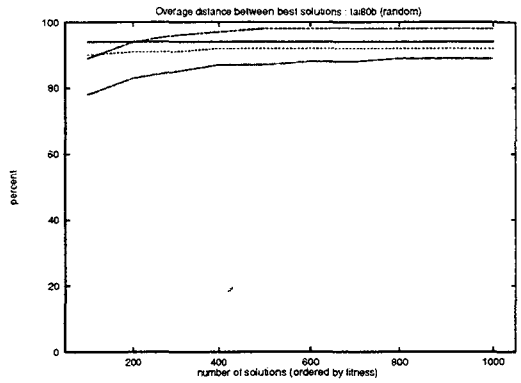


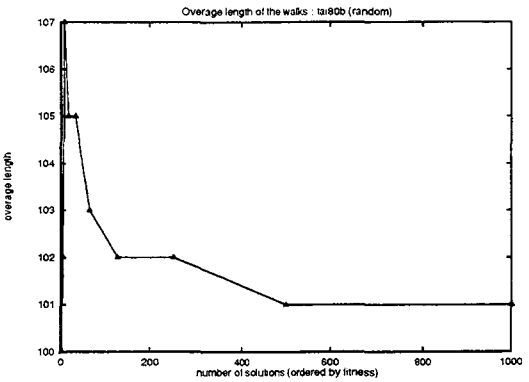


A.16 Instance Tai80b

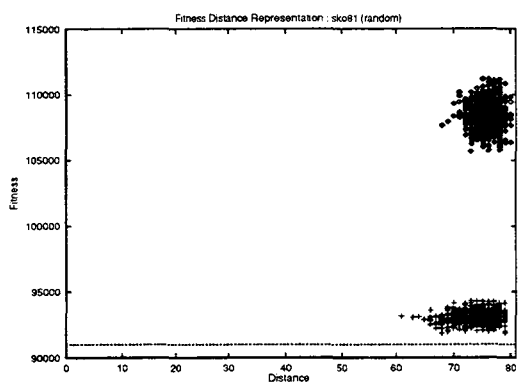


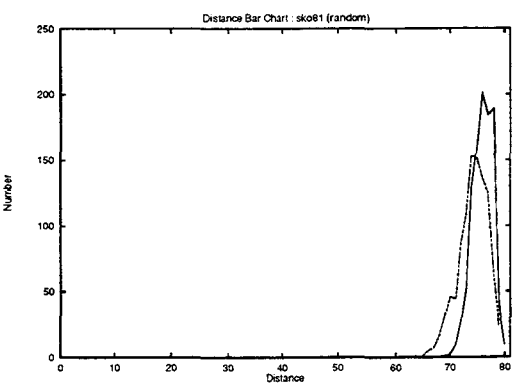


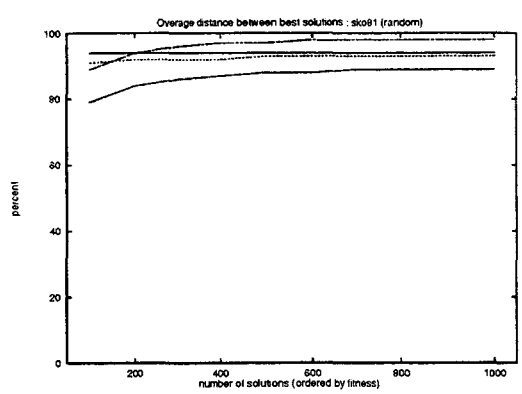


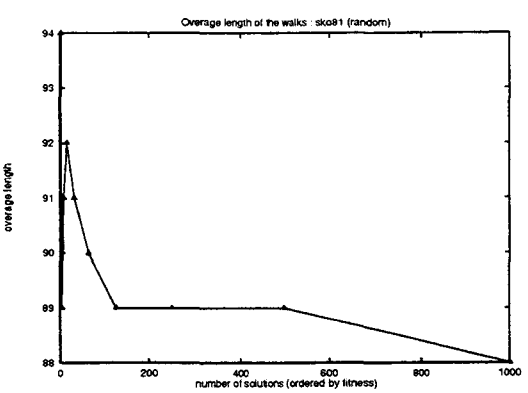


A.17 Instance Sko81

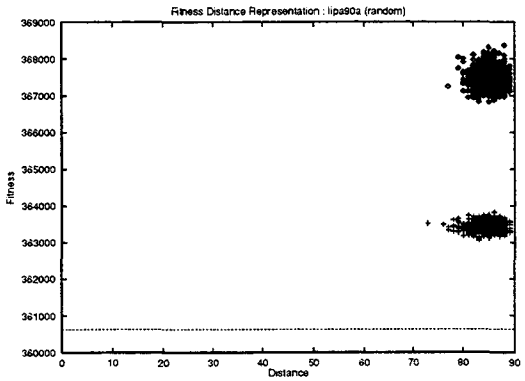


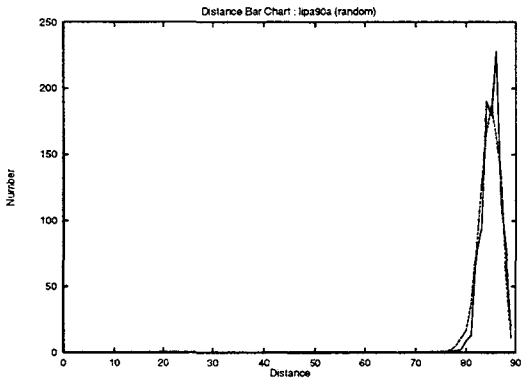


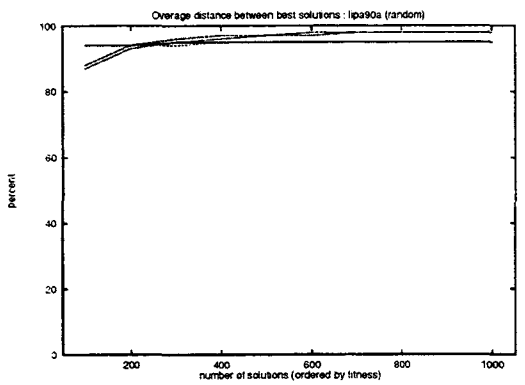


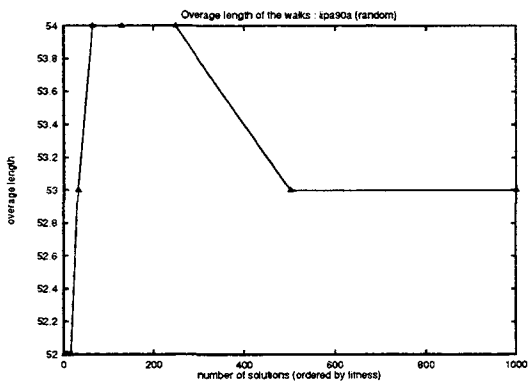


A.18 Instance Lipa90a

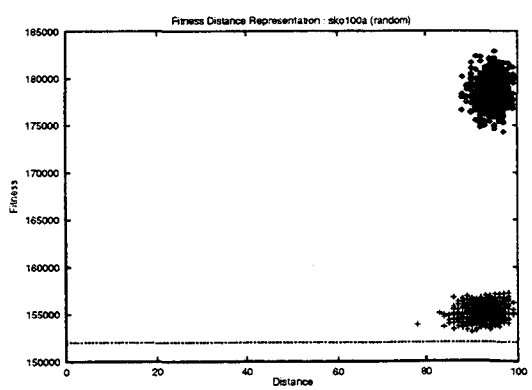


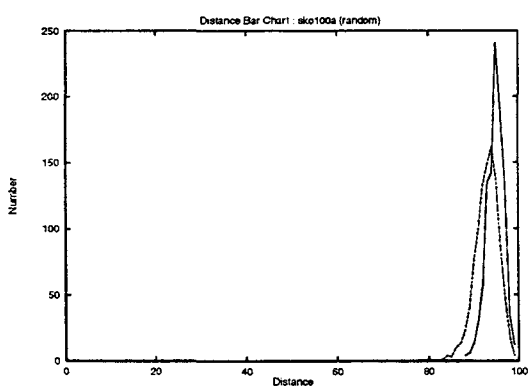


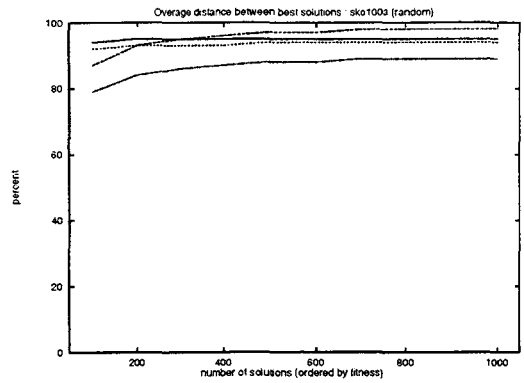


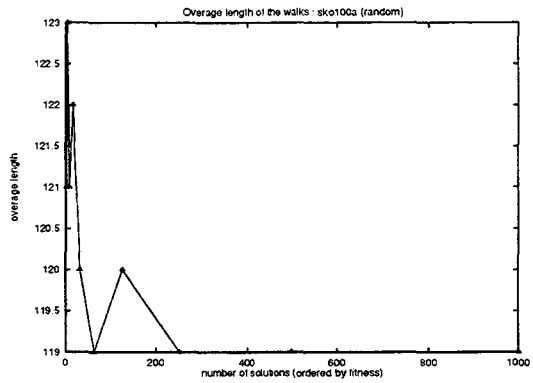


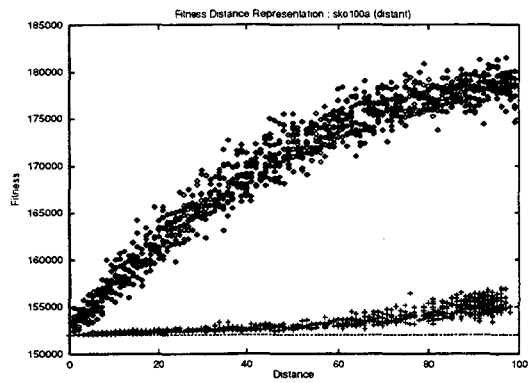
A.19 Instance Sko100a

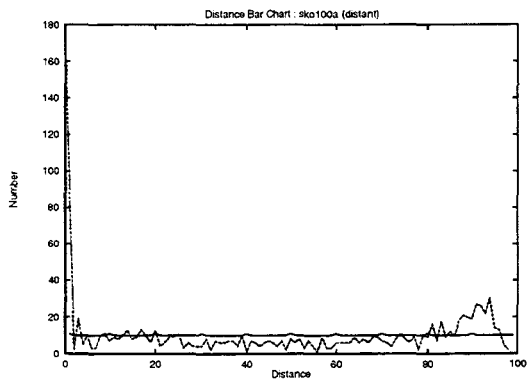




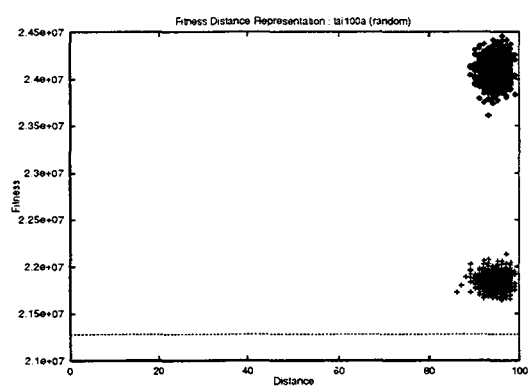


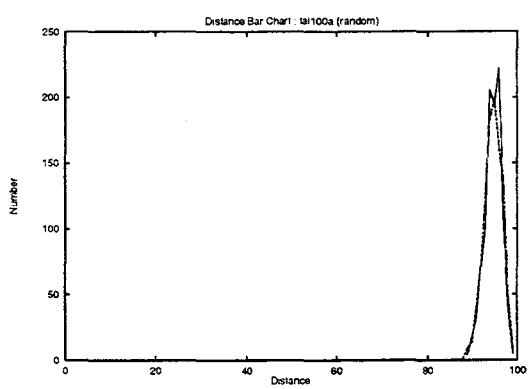


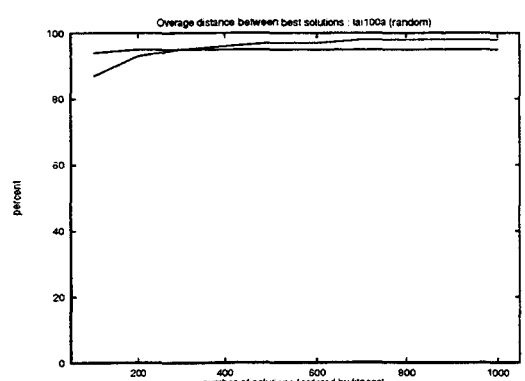


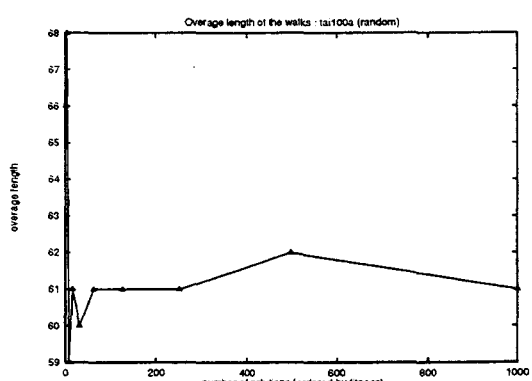


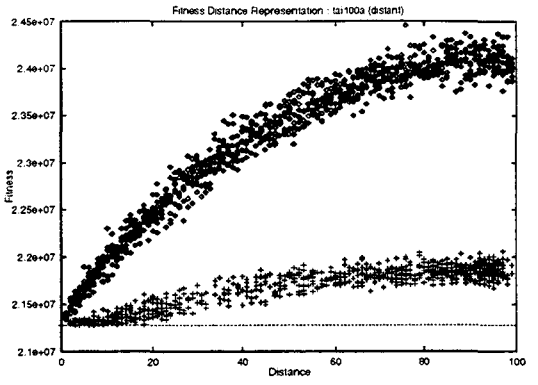
A.20 Instance Tai100a

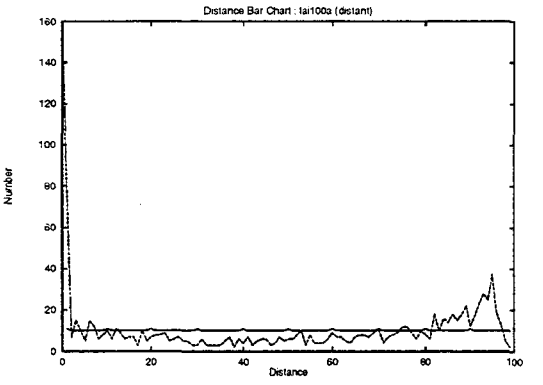




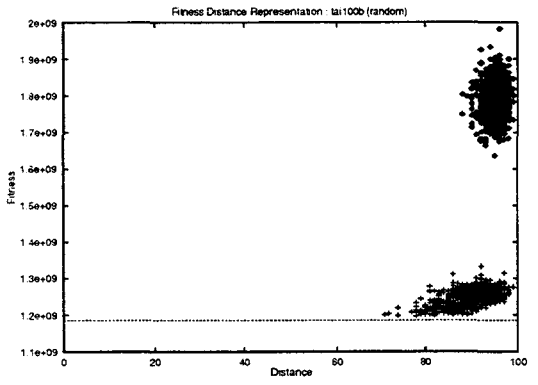


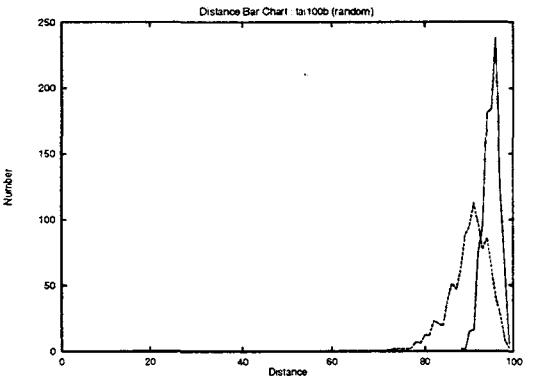


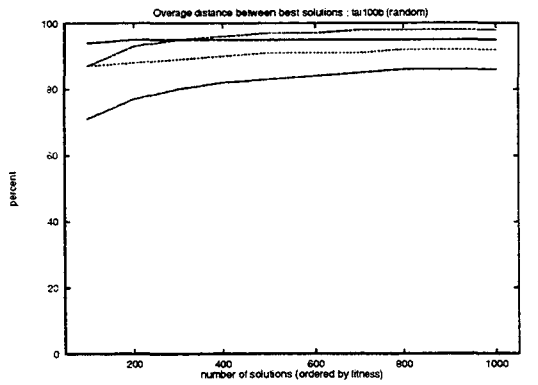


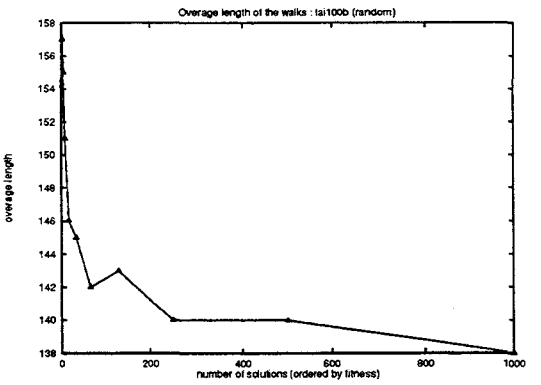


A.21 Instance Tai100b

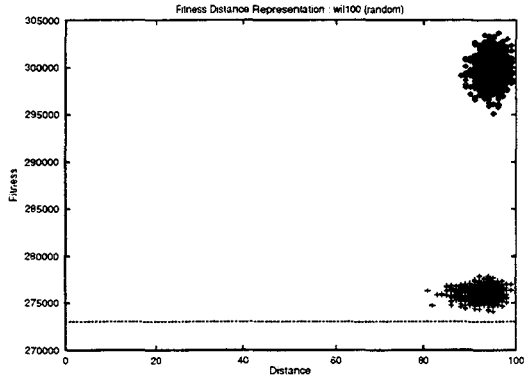


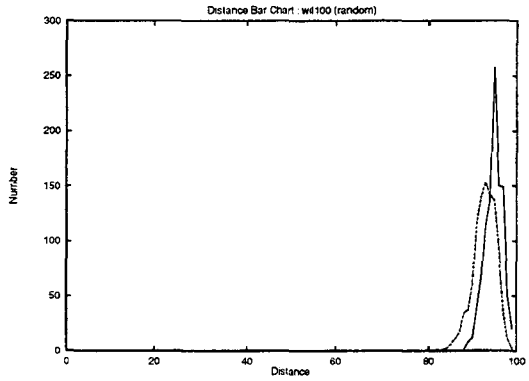


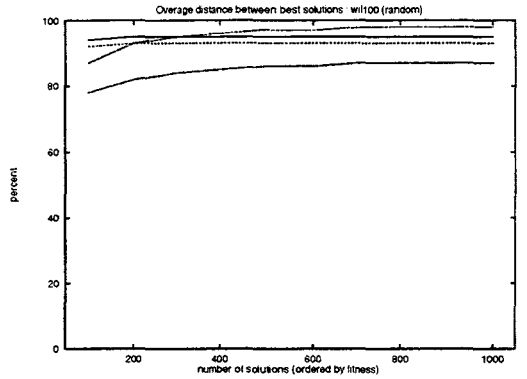


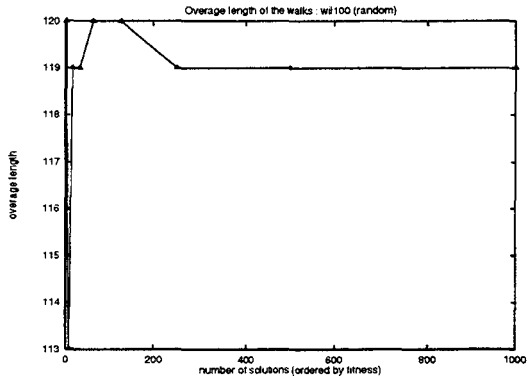


A.22 Instance Wil100

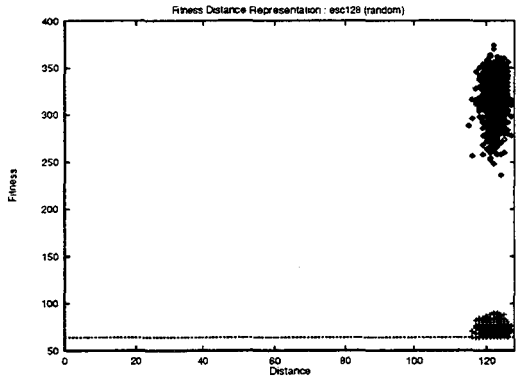


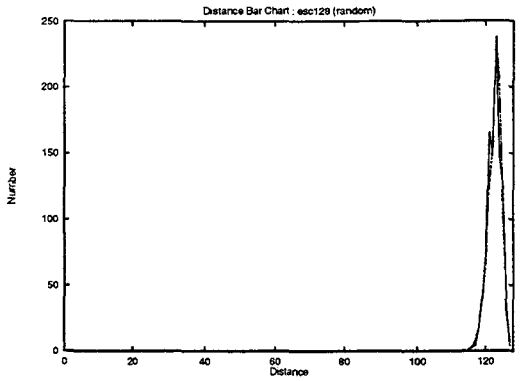


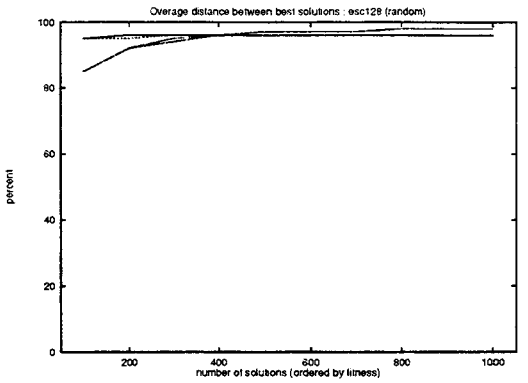


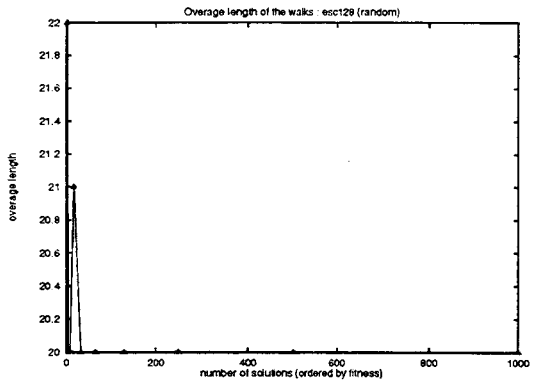


A.23 Instance Esc128

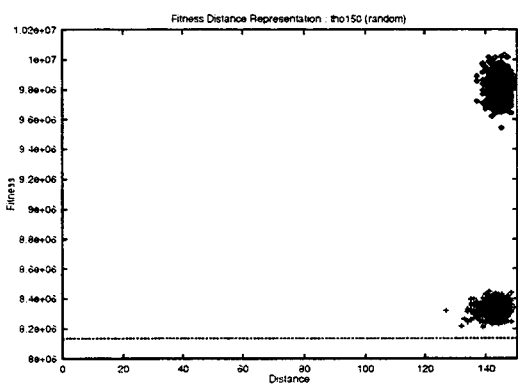


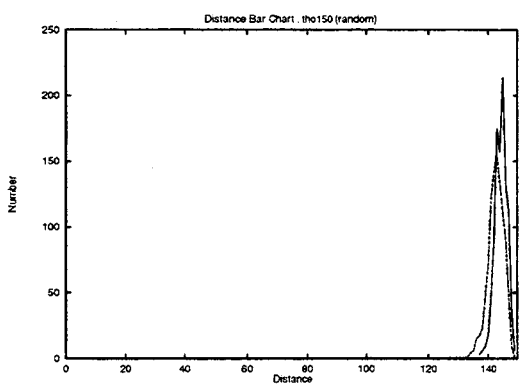


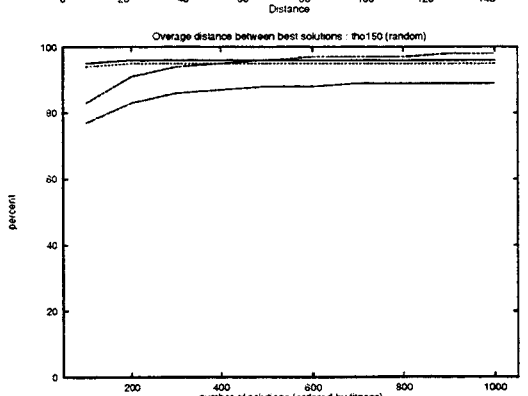


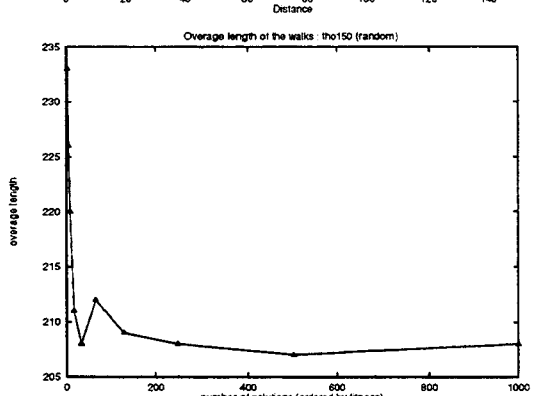


A.24 Instance Tho150

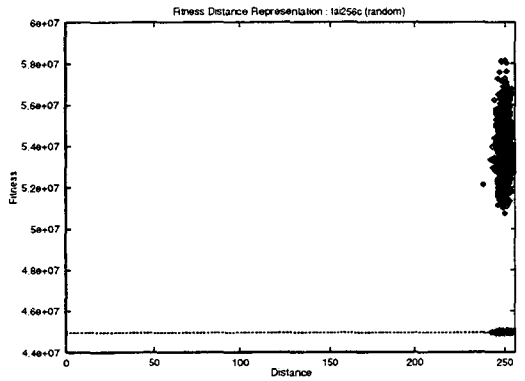


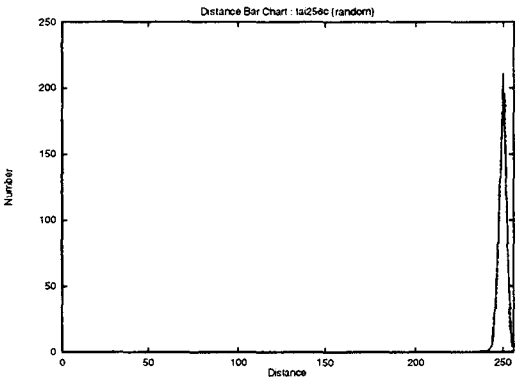


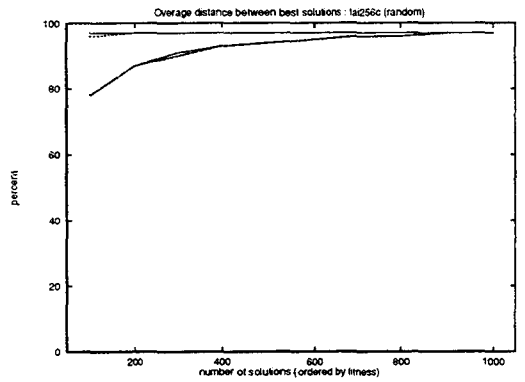


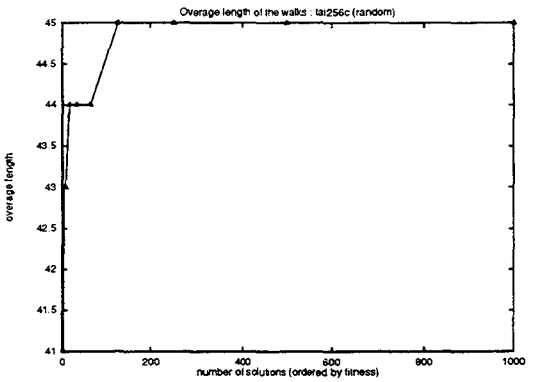


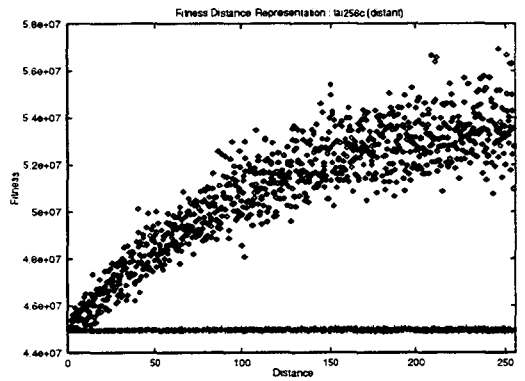
A.25 Instance Tai256c

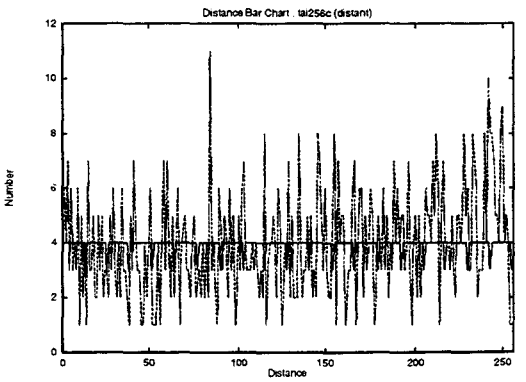












Bibliographie

[Aarts et al.89] Aarts (E.H.L.) et Korst (J.H.M.). – *Simulated annealing and Boltzman machines.* – Chichester, Wiley, 1989.

[Aarts et al.94] Aarts (E.H.L.), van Laarhoven (P.J.M.), Lenstra (J.K.) et Ulder (N.L.J.). – A computational study of local search algorithms for job shop scheduling. *ORSA Journal of Computing*, vol. 6, n° 2, 1994, pp. 118–125.

[Abbatista et al.95] Abbatista (F.), Abbatista (N.) et Caponetti (L.). – An evolutionary and cooperative agent model for optimization. *IEEE Int. Conf. on Evolutionary Computation ICEC'95*, pp. 668–671. – Perth, Australia, décembre 1995.

[Angel et al.96] Angel (E.) et Zissimopoulos (V.). – *On the ruggedness of the Quadratic Assignment Problem.* – Rapport technique n° LRI-96, Laboratoire de Recherche en Informatique, Paris, 1996.

[Armour et al.63] Armour (G.) et Buffa (E.). – Heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, vol. 9, 963, pp. 294–309.

[Bachelet et al.95] Bachelet (V.), Preux (P.) et Talbi (E-G.). – Parallel hybrid heuristics. *ICGA Workshop on Parallel Genetic Algorithms.* – Pittsburg, USA, juillet 1995.

[Bachelet et al.96] Bachelet (V.), Preux (P.) et Talbi (E-G.). – Parallel hybrid metaheuristics : application to the quadratic assignment problem. *Parallel Optimization Colloquium POC96*, pp. 233–242. – Versailles, France, mars 1996.

[Bachelet et al.97] Bachelet (V.), Preux (P.) et Talbi (E-G.). – The landscape of the quadratic assignment problem and local search methods. *The Tenth Meeting of the European Chapter on Combinatorial Optimization ECCOten.* – Puerto de la Cruz, Tenerife, Canary Islands, Spain, mai 1997.

[Bachelet et al.98a] Bachelet (V.), Geib (J-M.), Hafidi (Z.), Kebbal (D.) et Talbi (E-G.). – A platform for parallel optimization algorithms. *First*

meeting of the PAREO working group on Parallel Processing in Operations Research. – Versailles, France, juillet 1998.

[Bachelet et al.98b] Bachelet (V.), Hafidi (Z.), Preux (P.) et Talbi (E-G.). – Vers la co-opération de meta-heuristiques parallèles. *Calculateurs Parallèles, Réseaux et Systèmes répartis, Edition Hermes*, vol. 10, n° 2, avril 1998, pp. 211–223.

[Bachelet et al.98c] Bachelet (V.), Hafidi (Z.), Talbi (E-G.) et Preux (P.). – Diversifying tabu search by genetic algorithms. *INFORMS/CORS Spring Meeting.* – Montréal, Canada, avril 1998.

[Bachelet95] Bachelet (V.). – Heuristiques parallèles hybrides appliquées au problème d'affectation quadratique. – juin 1995. Rapport de D.E.A., Laboratoire d'Informatique Fondamentale de Lille.

[Battiti et al.92] Battiti (R.) et Tecchiolli (G.). – Parallel biased search for combinatorial optimization: Genetic algorithms and tabu. *Microprocessors and Microsystems*, vol. 16, 1992, pp. 351–367.

[Battiti et al.94] Battiti (R.) et Tecchiolli (G.). – The reactive tabu search. *ORSA Journal on Computing*, vol. 6, n° 2, 1994, pp. 126–140.

[Bazaraa et al.80] Bazaraa (M.) et Sherali (M.). – Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem. *Naval Research Logistics Quarterly*, vol. 27, 1980, pp. 29–41.

[Beasley et al.93a] Beasley (D.), Bull (D.R.) et Martin (R.R.). – An overview of genetic algorithms : Part 1, fundamentals. *University Computing*, vol. 15, n° 2, 1993, pp. 58–69.

[Beasley et al.93b] Beasley (D.), Bull (D.R.) et Martin (R.R.). – An overview of genetic algorithms: part 2, research topics. *University Computing*, vol. 15, n° 4, 1993, pp. 170–181.

[Boese et al.94] Boese (K.), Kahng (A.) et Muddu (S.). – A new adaptative multistart technique for combinatorial global optimizations. *Operations Research Letters*, vol. 16, n° 2, 1994, pp. 101–113.

[Boese96] Boese (K.D.). – *Models for iterative global optimisation.* – Los Angeles, Thèse de PhD, University of California, 1996.

[Brown et al.89] Brown (D.E.), Huntley (C.L.) et Spillane (A.R.). – A parallel genetic heuristic for the quadratic assignment problem. *Third Int. Conf. on Genetic Algorithms ICGA'89*, pp. 406–415. – Morgan Kauffmann, San Mateo, USA, juillet 1989.

[Bruijs84] Bruijs (P.A.). – On the quality of heuristic solution to a 19×19 quadratic assigment problem. *European Journal of Operational Research*, vol. 17, 1984, pp. 21–30.

[Buffa et al.64] Buffa (E.S.), Armour (G.C.) et Vollmann (T.E.). – Allocating facilities with CRAFT. *Harvard Business Review*, vol. 42, mars 1964, pp. 136–158.

[Burkard et al.77] Burkard (R.) et Offermann (J.). – Entwurf von schreibmashinentastaturen mittels quadratischer zuordnungsprobleme. *Zeitschrift für Operations Research*, vol. 21, 1977, pp. B121–B132.

[Burkard et al.83a] Burkard (R.) et Bonniger (T.). – A heuristic for the quadratic boolean programs with applications to the quadratic assignment problems. *European Journal of Operational Research*, vol. 13, 1983, pp. 374–386.

[Burkard et al.83b] Burkard (R.E.) et Rendl (F.). – A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, vol. 17, 1983, pp. 169–174.

[Burkard et al.91] Burkard (R.E.), Karisch (S.) et Rendl (F.). – Qaplib: A quadratic assignment problem library. *European Journal of Operational Research*, vol. 55, 1991, pp. 115–119.

[Cerny85] Cerny (V.). – Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, vol. 45, 1985, pp. 41–51.

[Chakrapani et al.93a] Chakrapani (J.) et Skorin-Kapov (J.). – Connection machine implementation of a tabu search algorithm for the traveling salesman problem. *Journal of COmputing and Information Technology*, vol. 1, n° 1, 1993, pp. 29–36.

[Chakrapani et al.93b] Chakrapani (J.) et Skorin-Kapov (J.). – Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, vol. 41, 1993, pp. 327–341.

[Charon et al.93] Charon (I.) et Hudry (O.). – The noising method: A new method for combinatorial optimization. *Operation Research Letters*, 1993.

[Chen et al.94] Chen (H.) et Flann (N.S.). – Parallel simulated annealing and genetic algorithms : a space of hybrid methods. *PPSN III Parallel problem solving from nature*, éd. par Davidor (Y.), Schwefel (H-P.) et Manner (R.). pp. 428–436. – Jerusalem, Israel, octobre 1994.

[Chu97] Chu (P.). – *A genetic algorithm approach for combinatorial optimization problems.* – Thèse de PhD, University of London, 1997.

[Cohoon et al.87] Cohoon (J.), Hedge (S.), Martin (W.) et Richards (D.). – Punctuated equilibria: A parallel genetic algorithm. *Second Int. Conf. on Genetic Algorithms.* MIT, pp. 148–154. – Cambridge, 1987.

[Cohoon et al.90] Cohoon (J.), Martin (W.) et Richards (D.). – Genetic algorithms and punctuated equilibria. *Parallel Problem Solving from nature*, éd. par Schwefel (H-P.) et Manner (R.). LNCS, pp. 134–141. – Dortmund, Germany, octobre 1990.

[Cohoon et al.91] Cohoon (J.), Martin (W.) et Richards (D.). – A multi-population genetic algorithm for solving the k-partition problem on hypercubes. *Fourth Int. Conf. on Genetic Algorithms*, éd. par Belew (R.) et Booker (L.). pp. 244–248. – San Mateo, CA, 1991.

[Collins et al.91] Collins (R.J.) et Jefferson (D.R.). – Selection in massively parallel genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms*, éd. par Belew (R.K.) et Booker (L.B.). pp. 249–256. – San Diego, 1991.

[Colorni et al.91] Colorni (A.), Dorigo (M.) et Maniezzo (V.). – Distributed optimization by ant colonies. *Proceedings of ECAL91 - European Conference on Artificial Life.* pp. 134–142. – Paris, France, 1991.

[Colorni et al.94] Colorni (A.), Dorigo (M.), Maniezzo (V.) et Trubian (M.). – Ant system for job-shop scheduling. *JORBEL Belgian Journal of Operations Research, Statistics, and Computer Science*, vol. 34, n° 1, 1994, pp. 39–53.

[Connolly90] Connolly (D.T.). – An improved annealing scheme for the qap. *European Journal of Operational Research*, vol. 46, 1990, pp. 93–100.

[Crainic et al.93] Crainic (T.D.), Toulouse (M.) et Gendreau (M.). – *Towards a taxonomy of parallel tabu search algorithms.* – Rapport technique n° CRT-933, Centre de Recherche sur les Transports, Université de Montreal, septembre 1993.

[Crainic et al.97] Crainic (T.), Nguyen (A.) et Gendreau (M.). – Cooperative multi-thread parallel tabu search with evolutionary adaptive memory. *2nd Int. Conf. on Metaheuristics.* – Sophia Antipolis, France, juillet 1997.

[Crainic98] Crainic (T.). – Méthodes parallèles de recherche avec tabous. *Calculateurs Parallèles, Réseaux et Systèmes répartis*, vol. 10, n° 2, avril 1998, pp. 171–193. – (french).

[Cung et al.96] Cung (V-D.), Mautor (T.), Michelon (P.) et Tavares (A.). – *A scatter search based approach for the quadratic assignment problem.* – Rapport technique n° 96-037, UVSQ, 78035 Versailles Cedex, France, PRISM, décembre 1996.

[Cung et al.97] Cung (V-D.), Mautor (T.), Michelon (P.) et Tavares (A.). – A scatter search based approach for the quadratic assignment problem. *IEEE Int. Conf. on Evolutionary Computation ICEC'97.* – Indianapolis, USA, avril 1997.

[David et al.73] David (A.) et Roucairol (C.). – Un algorithme de composition spatiale. *Notes méthodologiques en Architecture et Urbanisme*, vol. 2, 1973, pp. 69–87. – Centre MMI, Institut de l'Environnement.

[Dorigo et al.91] Dorigo (M.), Maniezzo (V.) et Colorni (A.). – *Ant system: an autocatalytic optimizing process.* – Rapport technique n° 91-016, Italy, Politecnico di Milano, 1991.

[Dorigo et al.96] Dorigo (M.), Manizzo (V.) et Colorni (A.). – The ant system: optimization by a colony of cooperating agents. *IEEE Transaction on Systems, Man, and Cybernetics-Part B*, vol. 26, n° 1, 1996, pp. 1–13.

[Dorigo92] Dorigo (M.). – *Optimization, learning and natural algorithms.* – Italy, Thèse de PhD, Politecnico di Milano, 1992.

[Duvivier et al.98] Duvivier (D.), Preux (P.), Fonlupt (C.), Robillard (D.) et Talbi (E-G.). – The fitness function and its impact on local search methods. *IEEE Conf. System, Man, and Cybernetics SMC'98.* pp. 2478–2483. – San Diego, USA, octobre 1998.

[Duvivier99] Duvivier (D.). – *Métaheuristiques hybrides pour le Job-Shop.* – Laboratoire d'Informatique du Littoral, Calais, Thèse de PhD, Université du Littoral – Côte d'Opale, (à soutenir en 1999).

[ela98] Çela (E.). – *The Quadratic Assignment Problem Theory and Algorithms.* – Kluwer Academic Publishers, 1998.

[Elshafei77] Elshafei (A.N.). – Hospital layout as a quadratic assignment problem. *Operational Research Quarterly*, vol. 28, n° 1, 1977, pp. 167–179.

[Falco et al.94] Falco (I. De), Balio (R. Del), Tarantino (E.) et Vaccaro (R.). – Improving search by incorporating evolution principles in parallel tabu search. *Int. Conf. on Machine Learning*, pp. 823–828. – 1994.

[Falco et al.95] Falco (I. De), Balio (R. Del) et Tarentino (E.). – An analysis of parallel heuristics for task allocation in multicomputers. *Computing*, vol. 3, n° 59, 1995.

[Feo et al.91] Feo (T.A.), Venkatraman (K.) et Bard (J.F.). – A grasp for a difficult single machine scheduling problem. *Computers and Operation Research*, vol. 18, 1991, pp. 635–643.

[Feo et al.94] Feo (T.A.), Resende (M.G.C.) et Smith (S.H.). – A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, vol. 42, 1994, pp. 860–878.

[Fleurant et al.96] Fleurant (C.) et Ferland (J.). – Genetic and hybrid algorithms for graph coloring. *Annals of Operation Research*, 1996.

[Fleurent et al.94] Fleurent (C.) et Ferland (A.). – Genetic hybrids for the quadratic assignment problem. *DIMACS Series in discrete Mathematics and Theoretical Computer Science*, vol. 16, 1994, pp. 173–188.

[Fonlupt et al.97] Fonlupt (C.), Robilard (D.), Preux (P.) et Talbi (E-G). – Fitness landscapes and performance of meta-heuristics. *2nd Int. Conf. on metaheuristics MIC97.* – Sophia-Antipolis, France, juillet 1997.

[Freisleben et al.96] Freisleben (B.) et Merz (P.). – A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. *2nd IEEE Conf. on Evolutionary Computation ICEC'96*, pp. 616–621. – mai 1996.

[Frenk et al.85] Frenk (J.), van Houweninge (M.) et Kan (A. Rinnoy). – Asymptotic properties of the quadratic assignment problem. *Mathematics of Operations Research*, vol. 10, 1985, pp. 100–116.

[Garcia et al.94] Garcia (B.L.), Potvin (J-Y.) et Rousseau (J.M.). – A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research*, vol. 21, n° 9, 1994, pp. 1025–1033.

[Garey et al.79] Garey (M.) et Johnson (D.). – *Computers and Intractability: A guide to the theory on NP-completeness.* – New York, W.H. Freeman and Co. Publishers, 1979.

[Geoffrion et al.76] Geoffrion (A.M.) et Graves (G.W.). – Scheduling parallel production lines with changeover costs: Practical application of the quadratic assignment problem/ lp approach. *Operations Research*, vol. 24, 1976, pp. 595–610.

[Gilmore62] Gilmore (P.). – Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics*, vol. 10, 1962, pp. 305–313.

[Glover et al.92] Glover (F.) et Laguna (M.). – *Modern Heuristic Techniques For Combinatorial Problems*, chap. 3, pp. 70–150. – Blackwell Scientific Publications, 1992.

[Glover et al.93] Glover (F.), Laguna (M.), Taillard (E.) et de Werra (D.). – Tabu search. *Annals of Operation Research*, vol. 41, 1993.

[Glover77] Glover (F.). – Heuristic for integer programming using surrogate constraints. *Decision Sciences*, vol. 8, 1977, pp. 156–166.

[Glover89a] Glover (F.). – Tabu search - part i. *ORSA Journal of Computing*, vol. 1, 1989, pp. 190–206.

[Glover89b] Glover (F.). – Tanu search - part ii. *ORSA Journal of Computing*, vol. 2, 1989, pp. 4–32.

[Glover94] Glover (F.). – *Tabu search fundamentals and uses.* – Rapport technique, University of Colorado Boulder, Graduate School of Business, may 1994.

[Glover98] Glover (F.). – Artificial evolution, lecture notes in computer science, 1998.

[Goldberg et al.85] Goldberg (D.) et Lingle (R.). – Alleles, loci, and the traveling salesman problem. *International Conference on Genetic Algorithms and their Applications.* – 1985.

[Goldberg89] Goldberg (D.E.). – *Genetic algorithms in search, optimization, and machine learning.* – Addison-Wesley, 1989.

[Greening90] Greening (D.). – Parallel simulated annealing techniques. *Phisica D*, vol. 42, 1990, pp. 293–306.

[Grefenstette87] Grefenstette (J.J.). – Incorporating problem specific knowledge into genetic algorithms. *Genetic algorithms and Simulated annealing*, éd. par Davis (L.). pp. 42–60. – San Mateo, CA, USA, 1987.

[Hansen et al.99] Hansen (P.) et Mladenović (N.). – An introduction to variable neighborhood search. *Metaheuristics: advances and trends in local search paradigms for optimization*, éd. par Voss, Martello, Osman et Roucairol, pp. 433–458. – Kluwer Academic Publishers, 1999.

[Hart94] Hart (W.). – *Adaptive global optimization with local search.* – San Diego, Thèse de PhD, University of California, 1994.

[Heffley77] Heffley (D.). – Assigning runners to a relay team. *Optimal Strategies in Sports*, 1977, pp. 169–171.

[Hentenryck89] Hentenryck (V.). – *Constraint satisfaction in logic programming.* – MIT Press, 1989.

[Herroellen et al.85] Herroellen (W.) et Gils (A. Van). – On the use of flow dominance in complexity measures for facility layout problems. *International Journal of Production Research*, vol. 23, 1985, pp. 97–108.

[Hertz et al.94] Hertz (A.), Jaumard (B.), Ribeiro (C.) et Formosinho (W.). – Local optima topology for the k-coloring problem. *Discrete Applied Mathematics*, vol. 29, n° 4–5, 1994, pp. 353–378.

[Hillier et al.66] Hillier (F.) et Connors (M.). – Quadratic assignment problem algorithms and the location od indivisible facilities. *Management Science*, vol. 13, 1966, pp. 42–57.

[Holland75] Holland (J.H.). – *Adaptation in natural and artificial systems.* – Ann Arbor, MI, USA, The University of Michigan Press, 1975.

[Husbands et al.90] Husbands (P.), Mill (F.) et Warrington (S.). – Genetic algorithms, production plan optimisation and scheduling. *Parallel Problem Solving from Nature*, éd. par Schwefel (H-P.) et Manner (R.). LNCS, pp. 80–84. – Dortmund, Germany, octobre 1990.

[Jog et al.89] Jog (P.), Suh (J.) et Gucht (D. Van). – The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. *3rd Int. Conf. Genetic Algorithms*, éd. par Kaufmann (Morgan). – 1989.

[Johnson et al.88] Johnson (D.S.), Papadimitriou (C.H.) et Yannakakis (M.). – How easy is local search. *Journal of computer and system siences*, vol. 37, 1988, pp. 79–100.

[Johnson et al.89] Johnson (D.S.), Aragon (C.R.), Geoch (L.A. Mc) et Schevon (C.). – Optimization by simulated annealing: An experimental evaluation, part i, graph partitioning. *Operations Research*, vol. 37, 1989, pp. 865–892.

[Johnson et al.91] Johnson (D.S.), Aragon (C.R.), Geoch (L.A. Mc) et Schevon (C.). – Optimization by simulated annealing: An experimental evaluation, part ii, graph coloring and number partitioning. *Operations Research*, vol. 39, 1991, pp. 378–406.

[Johnson et al.92] Johnson (D.S.), Aragon (C.R.), Geoch (L.A. Mc) et Schevon (C.). – Optimization by simulated annealing: An experimental evaluation, part iii, the traveling salesman problem. *Operations Research*, 1992, pp. 378–406.

[Jong75] Jong (K.A De). – *An analysis of the behavior of a class of genetic adaptive systems.* – Thèse de PhD, University of Michigan, 1975.

[Kapsalis et al.94] Kapsalis (A.), Smith (G.D.) et Rayward-Smith (V.J.). – A unified paradigm for parallel genetic algorithms. *Evolutionary computing*, éd. par Forgarty (T.C.). AISB Workshop, pp. 131–149. – Leeds, U.K., avril 1994.

[Kaufman et al.78] Kaufman (L.) et Broeckx (F.). – An algorithm for the quadratic assignment problem using benders' decomposition. *European Journal of Operational Research*, vol. 2, 1978, pp. 204–211.

[Kernighan et al.70] Kernighan (B.W.) et Lin (S.). – An efficient heuristic procedure for partioning graphs. *The Bell System Technical Journal*, 1970, pp. 291–307.

[Kim et al.95] Kim (H.), Hayashi (Y.) et Nara (K.). – The performance of hybridized algorithm of genetic algorithm, simulated annealing and tabu search for thermal unit maintenance scheduling. *2nd IEEE Conf. on Evolutionary Computation ICEC'95*, pp. 114–119. – Perth, Australia, décembre 1995.

[Kirkpatrick et al.83] Kirkpatrick (S.), Gelatt (C.D.) et Vecchi (M.P.). – Optimization by simulated annealing. *Science*, vol. 220, n° 4598, mai 1983, pp. 671–680.

[Kirkpatrick et al.85] Kirkpatrick (S.) et Toulouse (G.). – Configuration space analysys of travelling salesman problems. *J. Physique*, vol. 46, août 1985, pp. 1277–1292.

[Koopmans et al.57] Koopmans (T.C.) et Beckmann (M.J.). – Assignment problems and the location of economic activities. *Econometrica*, vol. 25, 1957, pp. 53–76.

[Kosa et al.95] Kosa (J.) et Andre (D.). – *Parallel genetic programming on a network of transputers.* – Rapport technique n° CS-TR-95-1542, Standford University, 1995.

[Koza92] Koza (J.). – *Genetic Programming.* – Cambridge, USA, MIT Press, 1992.

[Krueger93] Krueger (M.). – *Méthodes d'analyse d'algorithmes d'optimisation stochastiques à l'aide d'algorithmes génétiques.* – Thèse de PhD, École Nationale Supérieure des Télécommunications, 1993.

[Laporte et al.88] Laporte (G.) et Mercure (H.). – Balancing hydraulic turbine runners : a quadratic assignment problem. *European Journal of Operational Research*, vol. 35, 1988, pp. 578–381.

[Laursen93] Laursen (P.S.). – Simulated annealing for the qap - optimal tradeoff between simulation time and solution quality. *European Journal of Operational Research*, vol. 69, 1993, pp. 238–243.

[Lawler63] Lawler (E.). – The quadratic assignment problem. *Management Science*, vol. 9, 1963, pp. 586–599.

[Lawler76] Lawler (E.). – *Combinatorial optimization: Networks and matroids.* – Holt and Rinehart and Winston, 1976.

[Li et al.92] Li (Y.) et Pardalos (P.M.). – Generating quadratic assignment test problems with known optimal permutations. *Computational Optimization and Applications*, vol. 1, 1992, pp. 163–184.

[Lin et al.73] Lin (S.) et Kernighan (B.). – An effective heuristic for the traveling salesman problem. *Operation Research*, vol. 21, 1973, pp. 498–516.

[Lin et al.91] Lin (F.), Kao (C.) et Hsu (C.). – Incorporating genetic algorithms into simulated annealing. *Fourth Int. Symp. on AI*, pp. 290–297. – 1991.

[Macfarlane et al.90] Macfarlane (D.) et East (I.). – A investigation of several parallel genetic algorithms. *12th Occam User Group*, pp. 60–67. – Exeter, England, avril 1990.

[Maekawa et al.96] Maekawa (K.), Nori (N.), Tamaki (H.), Kita (H.) et Nishikawa (Y.). – A genetic solution for the traveling salesman problem by means of a termodynamycal selection rule. *2nd IEEE Conf. on Evolutionary Computation ICEC'96*, pp. 529–534. – mai 1996.

[Mahfoud et al.95] Mahfoud (S.) et Goldberg (D.). – Parallel recombinative simulated annealing: A genetic algorithm. *Parallel computing*, vol. 21, 1995, pp. 1–28.

[Maniezzo et al.94] Maniezzo (V.), Colorni (A.) et Dorigo (M.). – Algodesk : an experimental comparison of eigh evolutionary heuristics applied to the quadratic assignment problem. *EJOR European Journal of Operation Research*, 1994.

[Maniezzo et al.98] Maniezzo (V.) et Colorni (A.). – The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering*, 1998. – à paraître.

[Martin et al.92] Martin (O.), Otto (S.) et Felten (E.). – Large-step markov chains for the tsp: Incorporating local search heuristics. *Operation Research Letters*, vol. 11, 1992, pp. 219–224.

[Mautor93] Mautor (T.). – *Contribution à la résolution des problèmes d'implantation : algorithmes séquentiels et parallèlles pour l' affectation quadratique.* – Thèse de PhD, Université Pierre et Marie Curie PARIS VI, février 1993.

[Metropolis et al.53] Metropolis (N.), Rosenbluth (A.W.), Rosenbluth (M.N.), Teller (A.H.) et Teller (E.). – Equation of state calculation by fast computing machines. *Journal of Chem. Phys.*, vol. 21, 1953, pp. 1087–1091.

[Mori et al.95] Mori (N.), Yoshida (J.), Kita (H.) et Nishikawa (Y.). – A thermodynamical selection rule for the genetic algorithm. *IEEE Int. Conf. on Evolutionary Computation ICEC'95*, pp. 188–192. – Perth, Australia, décembre 1995.

[Mpl93] MPL. – *Maspar Parallel Application Language (MPL) Reference Manual*, mai 1993. Sofware Version 3.2, Document Part Number 9302-001, Revision A4.

[Nissen94] Nissen (V.). – Solving the quadratic assignment problem with clues from nature. *IEEE Transactions on Neural Networks*, vol. 5, n° 1, janvier 1994, pp. 66–72.

[Oreilly et al.95] O'Reilly (U-M.) et Oppacher (F.). – Hybridized crossover-based techniques for program discovery. *IEEE Int. Conf. on Evolutionary Computation ICEC'95*, pp. 573–578. – Perth, Australia, décembre 1995.

[Padberg et al.91] Padberg (M.) et Rinaldi (G.). – A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, vol. 33, 1991, pp. 60–100.

[Papadimitriou et al.82] Papadimitriou (C.) et Steiglitz (K.). – *Combinatorial Optimization: Algorithms and Complexity.* – Printice-Hall, 1982.

[Papadimitriou76] Papadimitriou (C.H.). – *The complexity of combinatorial optimization problems.* – Thèse de PhD, Princeton University, 1976.

[Pardalos et al.94] Pardalos (P.M.), Rendl (F.) et Wolkowicz (H.). – The quadratic assignment problem: A survey and recent developments. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 16, 1994, pp. 1–42.

[Pettey et al.87] Pettey (C.), Leuze (M.) et Grefenstette (J.). – A parallel genetic algorithm. *Second Int. Conf. on Genetic algorithms.* MIT, pp. 155–161. – Cambridge, juillet 1987.

[Preux et al.99] Preux (Ph.), Robillard (D.), Fonlupt (C.), Talbi (E-G.) et Bachelet (V.). – *Reaching summits is not wandering (or Getting insight into problem landscapes to go higher, faster).* – Rapport technique n° LIL-99-5, Calais, France, Laboratoire d'Informatique du Littoral, janvier 1999.

[Pvm94] PVM. – *Parallel Virtual Machine. A user's Guide and Tuturial for Networked Parallel Computing.* – Cambridge Massachussetts, London England, 1994.

[Rechenberg73] Rechenberg (I.). – *Evolutionstrategie: Optimierung technisher systeme nach prizipien der biologischen evolution.* – Stuttgart, Formann-Holzboog Verlag, 1973.

[Rhee88] Rhee (W.T.). – A note on asymptotic properties of the quadratic assignment problem. *Operations Research Letters*, vol. 7, n° 4, août 1988, pp. 197–200.

[Rhee91] Rhee (W.). – Stachastic analysis of the quadratic assignment problem. *Mathematics of Operations Research*, vol. 16, 1991, pp. 223–239.

[Sahni et al.76] Sahni (S.) et Gonzales (T.). – P-complete approximation problems. *Journal of the Association for Computing Machinery*, vol. 23, n° 3, juillet 1976, pp. 555–565.

[Schwehm94] Schwehm (M.). – Massively parallel genetic algorithms. *Massively parallel processing applications and development*, éd. par Dekker (L.), Smit (W.) et Zuidervaart (J.C.). pp. 505–512. – Elsevier Science B.V., 1994.

[Shahookar et al.90] Shahookar (K.) et Mazumber (P.). – A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Trans. on Computer-Aided Design*, vol. 9, n° 5, mai 1990, pp. 500–511.

[SK90] Skorin-Kapov (J.). – Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, vol. 2, n° 1, décembre 1990, pp. 33–45.

[Spiessens et al.91] Spiessens (P.) et Manderick (B.). – A massively parallel genetic algorithm: Implementation and first analysis. *Proceedings of the International Conference on Genetic Algorithms*, éd. par Belew (R.K.) et Booker (L.B.). pp. 279–285. – San Diego, 1991.

[Stadler95] Stadler (P.). – Towards a theory of landscapes. – 1995. 95-03–030 in http://www.santafe.edu/sfi/publications/95wplist.html.

[Steinberg61] Steinberg (L.). – The backboard wiring problem: A placement algorithm. *SIAM Review*, vol. 3, n° 1, janvier 1961, pp. 37–50.

[Stutzle et al.97] Stutzle (T.) et Hoos (H.). – The max-min ant system and local search for combinatorial optimization problems: Towards adaptive tools for global optimization. *2nd Int. Conf. on Metaheuristics.* – 1997.

[Suh et al.87] Suh (J.) et Gucht (D. Van). – Incorporating heuristic information into genetic search. *2nd Int. Conf. Genetic Algorithms.* pp. 100–107. – Lawrence Erlbaum Associates, 1987.

[Taillard et al.97] Taillard (E.D.) et Gambardella (L.M.). – An ant approach for structured quadratic assignment problems. *2nd Metaheuristics International Conference MIC'97*, éd. par Roucairol (C.), Osman (I.), Martello (S.) et Voss (S.). INRIA, pp. 217–222. – Sophia Antipolis, France, juillet 1997.

[Taillard et al.98] Taillard (E.D.), Gamberdella (L.M.), Gendreau (M.) et Potvin (J-Y.). – La programmation à mémoire adaptative ou l'évolution des algorithmes évolutifs. *Calculateurs parallèles*, vol. 10, n° 2, avril 1998, pp. 117–140.

[Taillard91] Taillard (E.). – Robust taboo search for the quadratic assignment problem. *Parallel Computing*, vol. 17, 1991, pp. 443–455.

[Taillard93a] Taillard (E.). – Parallel iterative search methods for vehicle routing problem. *Networks*, vol. 23, 1993, pp. 661–673.

[Taillard93b] Taillard (E.). – *Recherche itérative parallèle.* – Thèse de PhD, Ecole Polytechnique Federale de Lausanne, 1993.

[Taillard94] Taillard (E.D.). – Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal of computing*, vol. 6, n° 2, 1994, pp. 108–117.

[Taillard95] Taillard (E.). – Comparison of iterative searches for the quadratic assignment problem. *Location Science*, vol. 3, 1995, pp. 87–133.

[Talbi et al.91] Talbi (E-G.) et Bessière (P.). – A parallel genetic algorithm for the graph partitioning problem. *ACM International Conference on Supercomputing.* – Cologne, Germany, juin 1991.

[Talbi et al.94] Talbi (E-G.), Muntean (T.) et Samarandache (I.). – Hybridation des algorithmes génétiques avec la recherche tabou. *Evolution Artificielle EA94.* – Toulouse, France, septembre 1994.

[Talbi et al.96] Talbi (E-G.), Hafidi (Z.) et Geib (J-M.). – Mars: Adaptive scheduling of parallel applications in a multi-user heterogeneous environment. *2nd European School on Computer Science ESPPE'96*, pp. 119–122. – Alpe d'Huez, France, avril 1996.

[Talbi et al.97] Talbi (E-G.), Hafidi (Z.) et Geib (J-M.). – Parallel adaptative tabu search for large scale optimization problems. *2nd Metaheuristics International Conference MIC'97.* – Sophia Antipolis, France, juillet 1997.

[Talbi et al.99a] Talbi (E-G.), Geib (J-M.), Hafidi (Z.) et Kebbal (D.). – Mars: An adaptive parallel programming environment. *High Performance cluster computing*, éd. par Buyya (R.), chap. 4. – Prentice Hall PTR, 1999.

[Talbi et al.99b] Talbi (E-G.), Roux (O.), Fonlupt (C.) et Robillard (D.). – Parallel ant colonies for combinatorial optimization problems. *Lecture Notes in Computer Science*, éd. par Rolim (J.). BioSP3 Workshop on Viologically Inspired Solutions to Paralell Processsing Systems, in IEEE IPPS/SPDP'99 Int. Parallel Processing Symposium / Symposium on Parallel and Distributed Processing. – San Juan, Puerto Rico, USA, avril 1999.

[Talbi95] Talbi (E-G). – Algorithmes génétiques parallèles : Techniques et applications. *Parallélisme et applications irrégulières*, 1995, pp. 28–48. – C. Roucairol and al, Hermes, France.

[Tanese87] Tanese (R.). – Parallel genetic algorithms for a hypercube. *Second Int. Conf. on Genetic algorithms.* MIT, pp. 177–183. – Cambridge, juillet 1987.

[Tate et al.95] Tate (D.M.) et Smith (A.E.). – A gentic approach to the quadratic assignment problem. *Computers and Operations Research*, vol. 22, n° 1, 1995, pp. 73–83.

[Thiel et al.94] Thiel (J.) et Voss (S.). – Some experiences on solving multiconstraint zero-one knapsack problem with genetic algorithms. *INFOR.* – 1994.

[Tsoi et al.95] Tsoi (E.), Wong (K.) et Fung (C.). – Hybrid ga/sa algorithms for evaluating trade-off between economic cost and environmental impact in generation dispatch. *IEEE Int. Conf. on Evolutionary Computation ICEC'95*, pp. 132–137. – Perth, Australia, décembre 1995.

[Ulder et al.90] Ulder (N.), Aarts (E.), Bandelt (H-J), Laarhoven (P. Van) et Pesh (E.). – Genetic local search algorithms for the traveling salesman problem. *Parallel Problem Solving From Nature*, éd. par Schwefel (H-P) et Manner (R.). pp. 109–116. – Dortmund, Germany, octobre 1990.

[Vaessens et al.92] Vaessens (R.), Aarts (E.) et Lenstra (J.). – A local search template. *Parallel Problem Solving From Nature*, éd. par Männer (R.) et Manderick (B.). Université Libre de Bruxelles, pp. 67–76. – Belgique, sep 1992.

[Verhoeven et al.95] Verhoeven (M.G.A.) et Aarts (E.H.L.). – Parallel local search. *Journal of Heuristics*, vol. 1, n° 1, 1995, pp. 43–65.

[vL et al.88] van Laarhoven (P.J.M.) et Aarts (E.H.L.). – *Simulated Annealing: Theory and Applications.* – Dordrecht, Kluwer, 1988.

[Voigt et al.90] Voigt (H-M.), Born (J.) et Santibanez-Koref (I.). – Modelling and simulation of distributed evolutionary search processes for function optimization. *Parallel Problem Solving from Nature*, éd. par Schwefel (H-P.) et Manner (R.). LNCS, pp. 373–380. – Dortmund, Germany, octobre 1990.

[Vollman et al.66] Vollman (T.E.) et Buffa (E.S.). – The facilities layout problem in perspective. *Management Science*, vol. 12, n° 1, juin 1966, pp. 450–468.

[Voss93] Voss (S.). – *Network optimization problems*, chap. Tabu search: Application and prospects, pp. 333–353. – World Scientific, 1993.

[Wang et al.95] Wang (L.), Kao (C-Y.), Ouh-young (M.) et Chen (W-C.). – Molecular binding: A case study of the population-based annealing genetic algorithms. *IEEE Int. Conf. on Evolutionary Computation ICEC'95*, pp. 50–55. – Perth, Australia, décembre 1995.

[Weinberger90] Weinberger (E.). – Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, vol. 63, 1990, pp. 325–336.

[Weinberger91] Weinberger (E.). – Local properties of kauffman's n-k model: A tunably rugged energy landscape. *Physical Review A*, vol. 44, n° 10, novembre 1991, pp. 6399–6413.

[Whitehead et al.64] Whitehead (B.) et Elders (M.Z.). – An approach to the optimum layout of single-storey buildings. *Architect's Journal*, 1964, pp. 1373–1380.

[Whitley88] Whitley (D.). – GENITOR: A different genetic algorithm. *Proc. of the Rocky Mountain Conference on Artificial Intelligence.* – Denver, CO, USA, 1988.

[Whitley93] Whitley (D.). – *A genetic algorithm tutorial.* – Rapport technique n° CS-93-103, Colorado state university, novembre 1993.

[Wilhelm et al.87] Wilhelm (M.R.) et Ward (T.L.). – Solving quadratic assignment problems by simulated annealing. *IIE Transactions*, vol. 19, n° 1, mars 1987, pp. 107–119.

[Wolpert et al.95] Wolpert (D.H.) et Macready (W.G.). – *No Free Lunch Theorems for Search.* – Rapport technique n° SFI-TR-95-02-010, Santa Fe, NM, USA, The Santa Fe Institute, février 1995.

[Wong et al.94] Wong (K.) et Wong (Y.). – Genetic and genetic/simulated-annealing approaches to economic dispatch. *Gener. Transm. and Distrib.*, vol. 141, n° 5, septembre 1994, pp. 507–513.

[Wright32] Wright (S.). – The roles of mutation, inbreeding, crossbreeding and selection in evolution. *Procceding of the sixth International Congress*, pp. 356–366. – 1932.

Liste des notations

$\mathrm{Amp}\,P$	:	l'amplitude de P
$(c)_n$	:	matrice des coûts des flux
$\mathrm{card}\,P$	:	la cardinalité de P
$\mathrm{Dmm}(P)$	:	diamètre moyen relatif de P
Δ_{Amp}	:	la variation d'amplitude par SDW de U à O
Δ_{Dmm}	:	la variation de diamètre par SDW de U à O
Δ_{Ent}	:	la variation d'entropie par SDW de U à O
$(d)_n$	:	matrice des distances
d	:	une distance
$\mathrm{dd}\,(d)$	:	dominance de distance
$\mathrm{diam}\,(P)$	:	diamètre de P
$\mathrm{dist}(s,t)$	:	distance de s à t dans G
$\mathrm{dmm}\,(P)$	:	diamètre moyen de P
$\mathrm{ent}\,(P)$	:	entropie de P
$\mathrm{Ent}\,(P)$	:	entroppie relative de P
$\mathrm{fd}\,(c)$	:	dominance de flux
G	:	espace de recherche
$\mathrm{Gap}\,(O)$	:	écart relatif moyen des solutions de O
$\mathrm{Lmm}\,(P)$	:	moyenne des longueurs des marches SDW de U à O
$\mathrm{lmm}\,(P)$	:	moyenne des longueurs des marches SDW issues de P
n	:	taille de l'instance
$\mathrm{O}(n)$	:	de l'ordre de n
O	:	ensemble des optima locaux fournis par les SDW
φ	:	fonction coût
P	:	une population de solutions
π_*	:	un optimum global
Π_n	:	l'ensemble des permutations de taille n
ρ	:	fonction d'autocorrélation
$\mathcal{R}$	:	relation de voisinage
S	:	l'ensemble des solutions
s	:	une solution
$(s)_n$	:	matrice des coûts de placement
t	:	une solution
U	:	ensemble des solutions initiales
$V(s)$	:	voisinage immédiat de s
$V_k(s)$	:	voisinage de s à l'ordre k

Liste des abbréviations

AC	:	*Ant Colonies*
AGC	:	Algorithme Génétique Canonique
AMS	:	*Adaptive Multi-Start*
AMTS	:	*Adaptive Multistart Tabu Search*
AS	:	*Ants System*
CGA	:	*Cellular Genetic Algorithm*
CLP	:	*Constraint Logic Programming*
CRAFT	:	*Computerized Relative Allocation of Facilities Technique*
DGATS	:	*Diversifying Genetic Algorithm for Tabu Search*
DW	:	*Descent Walk*
EP	:	*Evolutionary Programming*
ES	:	*Evolution Strategy*
FIFO	:	*First In First Out*
GA	:	*Genetic Algorithm*
GH	:	*Greedy Heuristic*
GPP	:	*Graph Partitioning Problem*
GP	:	*Genetic Programming*
GRASP	:	*Greedy Randomized Adaptive Search Procedure*
HCH	:	*High level coevolutionary hybrid*
HRH	:	*High level Relay Hybrid*
IPI	:	*Iterated Population Improvement*
ISI	:	*Iterated Solution Improvement*
JSP	:	*Job-shop Scheduling*
LCH	:	*Low level Coevolutionary Hybrid*
LRH	:	*Low level Relay Hybrid*
LS	:	*Local Search*
MARS	:	*Multiuser Adaptive Ressource Scheduler*
MCGA	:	*Mulistart Cellular Genetic Algorithm*
MCP	:	*Maximum Clique Problem*
MIMD	:	*Multiple Instruction Multiple Data*
MPL	:	*Maspar Programming Language*
MTS	:	*Multistart Tabu Search*
NFL	:	*No Free Lunch*
NM	:	*Noisy Method*
NN	:	*Neural Network*
NOW	:	*Network Of Workstations*
PLS	:	*Polynomial-time Local Search*

PMX : *Partially Mapped Crossover*

PVM : *Parallel Virtual Machine*

QAPlib : Bibliothèque d'instances du **QAP**

QAP : *Quadratic Assignment Problem*

ReTS : *Reactive Tabu Search*

RoTS : *Robust Tabu Search*

SA : *Simultated Annealing*

SCP : *Set Covering Problem*

SDW : *Steepest Descend Walk*

SIMD : *Simple Instruction Multiple Data*

SPMD : *Simple Program Multiple Data*

SPP : *Set Partioning Problem*

SS : *Scatter Search*

TSP : *Traveling Salesman Problem*

TS : *Tabu Search*

VNS : *Variable Neighborhood Search*

VRP : *Vehicle Routing Problem*