RECHERCHES ITÉRATIVES DIRIGÉES PARALLÈLES

Thèse no 1153 présentée
au Département de Mathématiques de

l'École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences par

Éric Denis Taillard
ingénieur informaticien diplômé EPFL

acceptée su proposition du jury :

Prof. D. de Werra : rapporteur
Prof. J. Fonlupt : corapporteur
Prof. F. Glover : corapporteur
Prof. Th. Liebling : corapporteur
Dr. H. Schiltknecht : corapporteur

Lausanne, EPFL, 1993

TABLE

RÉSUMÉ

L'apparition de calculateurs de plus en plus performants a favorisé l'éclosion d'une multitude de méthodes heuristiques visant à trouver de bonnes solutions aux problèmes d'optimisation combinatoire. Parmi ces méthodes, celles qui sont basées sur la répétition de modifications locales apportées à une solution en vue de l'améliorer se sont révélées très prometteuses ; elles sont qualifiées de « recherches itératives » dans ce travail ; les processus de *recuit simulé*, les méthodes d'*acceptation à seuil* ou les *algorithmes génétiques* sont quelques unes des plus connues de ces recherches itératives ; elles sont présentées succinctement dans le chapitre 2.

Récemment a été proposé le principe qui s'est avéré très efficace en pratique de doter une recherche itérative de mémoire afin de la guider dans l'espace des solutions admissibles. Le chapitre 3 propose et étudie quelques techniques de direction des recherches itératives ; en particulier, il est montré que la méthode de direction la plus simple, qui interdit d'effectuer l'inverse d'une modification locale pendant quelques itérations, se révèle efficace en pratique bien qu'en théorie elle ne puisse garantir à la recherche de visiter une fois une solution optimale du problème considéré.

Naturellement, une recherche itérative dirigée ne se programme pas exactement de la même manière pour tout problème d'optimisation combinatoire ; comme il n'existe à l'heure actuelle aucun outil mathématique pour prédire l'efficacité théorique de ces recherches, la démarche adoptée dans ce travail consiste à mettre en évidence la manière dont elles peuvent être implantées simplement et efficacement sur quelques exemples particuliers de problèmes. Le chapitre 1 présente ces exemples de problèmes, à savoir

celui de la chaîne de traitement, celui de traitement à séquences fixées, celui d'affectation quadratique et enfin celui de distribution de biens ou d'élaboration de tournées de véhicules.

Si les recherches itératives dirigées présentent l'avantage appréciable d'être facilement implantables pour de nombreux problèmes, en particulier ceux tirés de cas pratiques, elles ont en revanche comme principal inconvénient d'être très coûteuses en temps de calcul. Comme l'issue la moins onéreuse pour augmenter les performances des calculateurs consiste à les doter de multiples processeurs, il est intéressant de montrer comment adapter une recherche itérative pour qu'elle fasse usage de plusieurs processeurs en parallèle ; le chapitre 4 présente quelques techniques de parallélisation des recherches itératives dirigées.

ABSTRACT

The increasing calculation power of computers has led to the appearance of many heuristic methods designed to find good solutions to combinatorial optimization problems. Among these methods, those based on the repetition of local modifications of a solution, seeking to improve the last, have been shown to be very promising ; they are called “ iterative searches ” in this work ; the processes of *simulated annealing*, the methods of *threshold accepting* and the *genetic algorithms* are some of the best known of these iterative searches ; they are succinctly presented in chapter 2.

A recent proposal has been the principle of endowing iterative searches with memory in order to guide them in the space of feasible solutions, and it turns out that this principle is very efficient in practice. Chapter 3 studies some techniques for guiding iterative searches ; in particular, the simplest guiding method, consisting in forbidding for some iterations to perform the reverse of a local modification, seems to be efficient in practice but in theory it cannot guarantee that the search will find an optimal solution of a considered problem.

Naturally, a guided iterative search cannot be programmed in the same way for all combinatorial optimization problems ; as no mathematical tools predicting the theoretical efficiency of these searches exist now, the way adopted in this work consists in showing how guided local searches can be easily and efficiently implemented on particular instances of problems. Chapter 1 presents these instances of problems ; these are the permutation flow shop sequencing, the job shop scheduling, the quadratic assignment and the vehicle routing problems.

Guided iterative searches have the advantage of being easy to implement for numerous problems, especially for those occurring in real life. However, they have as a major disadvantage very high computational times. As the less expensive way of increasing the performances of computers is to build them with many processors, it is interesting to present how to adapt an iterative search in such a way that it uses many processors in parallel ; chapter 4 presents some techniques to run an iterative search on a parallel computer.

REMERCIEMENTS

Je tiens tout d'abord à exprimer ma reconnaissance à mon directeur de thèse qui m'a initié à la recherche opérationnelle et qui a toujours su me conseiller tout en me laissant une grande liberté dans mes travaux.

Ensuite, ma gratitude va aux membres du jury qui ont aimablement accepté d'examiner cette dissertation, particulièrement à ceux dont le français n'était pas la langue maternelle.

Je n'oublie pas celle et ceux qui ont relu ce travail, relevant nombre de coquilles, et qui ont donc substantiellement contribué à l'amélioration sa présentation.

Enfin, je remercie mes collègues et amis pour les fructueuses discussions et suggestions qu'ils ont pu m'apporter et pour l'ambiance de travail agréable qu'ils ont su créer, ainsi que mes concurrents qui m'ont obligé à me surpasser, me contraignant à proposer des méthodes de plus en plus efficaces.

INTRODUCTION

Le titre de cette étude peut paraître quelque peu hermétique ; commençons donc par l'expliciter : les recherches itératives, qualifiées aussi de locales, sont de plus en plus étudiées actuellement et cet engouement est indéniablement dû à l'augmentation des performances des calculateurs. Ces techniques sont essentiellement appliquées à des problèmes d'optimisation discrète pour la résolution desquels on ne connaît pas d'algorithme efficace et dont les solutions sont modifiables localement.

À la base, il y a une idée simple : si l'on n'arrive pas à trouver facilement une très bonne solution du problème que l'on traite, il est en revanche souvent aisé d'en produire une de qualité quelconque ; on essaie ensuite d'effectuer sur cette solution un petit changement de manière à l'améliorer, ceci permettant d'obtenir une nouvelle solution que l'on va aussi essayer d'améliorer par une modification locale et ainsi de suite, d'où le qualificatif d'itératif.

Il n'est malheureusement pas possible pour tout problème de définir des modifications raisonnables de sorte que, à partir de n'importe quelle solution, il soit toujours possible d'obtenir une solution optimale ou même une bonne solution en n'utilisant que des modifications améliorantes ; par conséquent, avec un tel schéma, il est nécessaire de pouvoir détériorer une solution, même s'il existe parfois des modifications améliorantes, dans l'espoir d'arriver plus tard à une solution de qualité élevée. Pour qu'une recherche itérative puisse déterminer quelles modifications détériorantes doivent être réalisées, on doit lui donner des règles qui lui permettent de se diriger dans l'ensemble des solutions.

Un des buts du présent travail est précisément de dégager des règles de direction de recherches itératives.

Nous pensons que ces règles, dans un cadre général, dépendent fortement du problème que l'on cherche à résoudre. Par conséquent, nous commencerons par définir quelques problèmes variés dans le domaine de l'optimisation combinatoire et ensuite déterminerons des politiques de direction de recherches itératives adaptées à ceux-ci, pour enfin tirer des règles générales de direction qui devraient être applicables à un vaste ensemble de problèmes.

Naturellement, nous ne partons pas du néant dans le domaine des recherches itératives dirigées ; sous un titre reflétant mal à notre avis l'ensemble des concepts de direction, Glover énumère, dans un long article divisé en deux parties, [45] et [46], parues respectivement en 1989 et 1990 et constituant un développement d'un autre article, [44], paru en 1986, un ensemble de principes dont certains nous ont été très utiles dans l'élaboration d'une première approche de recherche itérative dirigée. En particulier, nous avons systématiquement utilisé un principe essentiel, bien que souvent insuffisant à lui seul pour réaliser une recherche efficace, basé sur l'interdiction de procéder à des modifications locales qui ne feraient qu'en annihiler d'autres réalisées au cours d'itérations récentes. Il est d'ailleurs à l'origine de la dénomination générique donnée par Glover à ce type de méthodes et qu'on peut traduire par « recherche avec interdictions ». Malheureusement, cette dénomination est souvent utilisée abusivement pour qualifier la plus simple des techniques de direction et on entend, on lit que telle méthode est plus performante que la recherche avec interdictions ; parler de recherches itératives dirigées permet d'éviter ce genre de comparaisons douteuses, si l'on suppose qu'il n'est pas possible de comparer une méthode à l'ensemble des recherches itératives dirigées.

Finalement, nous proposerons quelques techniques de parallélisation des recherches itératives, afin de pouvoir entre autres faire pleinement usage de la puissance de calcul considérable dont dispose un calculateur doté de plusieurs processeurs interconnectés. La parallélisation d'un processus intrinsèquement séquentiel constitué par une recherche itérative suppose une approche nouvelle de la problématique, si l'on fait abstraction des

techniques purement informatiques de parallélisation de codes (essentiellement basées sur la décomposition de boucles). En particulier, la décomposition de gros problèmes en sous-problèmes plus petits et indépendants permet à la fois d'inculquer de la connaissance à une recherche itérative dirigée et d'en accélérer le déroulement, même si l'on ne dispose pas de calculateurs distribués.

Ce travail constitue donc une recherche à la fois fondamentale, cherchant à répondre aux questions « comment peut-on en général diriger efficacement une recherche itérative et comment la paralléliser ? » et appliquée, puisque nous illustrerons sur des problèmes concrets les principes proposés, montrant au passage leur pertinence par l'obtention de solutions de qualité élevée. Nous choquerons sans doute le théoricien pur en utilisant le qualificatif de fondamental, car aucun théorème de base prédisant le comportement de recherches itératives dirigées n'a encore été démontré. Nous espérons donc que ce travail, au travers de nombreuses expériences numériques, lèvera un voile sur le fonctionnement et le rôle de quelques techniques de direction des recherches locales.

1. PROBLÈMES ÉTUDIÉS : NOTATIONS ET PROPRIÉTÉS

Dans ce chapitre, nous avons groupé la présentation des divers problèmes d'optimisation combinatoire dont nous discuterons la résolution par recherche itérative dirigée plus loin. Ce regroupement vise plusieurs buts : il nous permettra tout d'abord de définir chaque problème avec les notations utilisées pour le décrire, ensuite de donner quelques propriétés et parfois des conjectures que des recherches itératives dirigées nous ont permis d'émettre sur des exemples particuliers de problèmes, et enfin de présenter succinctement l'état actuel de la recherche sur ces problèmes.

1.1. PROBLÈME DE LA CHAÎNE DE TRAITEMENT

1.1.1. Description

Ce problème, connu sous le nom de « permutation flow shop » en anglais et noté $F|\ |C_{max}$ dans la classification de Lawler et al. [65], peut être décrit comme suit : n objets doivent passer sur une chaîne de traitement composée de m machines. Chaque objet doit subir m opérations effectuées successivement par les machines 1 à m. Connaissant le temps de travail t_{ij}, non négatif, de la $i^{\text{ième}}$ opération de l'objet j (c'est-à-dire le temps passé par l'objet j sur la machine i), il s'agit de trouver un ordre de passage des objets sur la chaîne, ordre qui minimise la longueur de l'intervalle de temps compris entre le début du traitement de la première opération sur la première machine et la fin du traitement de la dernière opération sur la dernière machine. On suppose que chaque machine ne peut effectuer qu'une opération à la fois, mais que le nombre d'objets en attente de traitement

devant une machine n'est pas limité ; un objet ne peut être traité que par une machine à la fois et lorsqu'une opération a débuté, elle ne peut s'interrompre ; l'ordre dans lequel les objets sont traités est le même pour toutes les machines.

Une solution à un exemple de problème de taille n peut se représenter sous la forme d'une permutation π des n objets, donc l'ensemble des solutions admissibles est de taille $n!$

1.1.2. Propriétés et conjecture

Nous nous sommes intéressé essentiellement aux propriétés d'un type de problème que l'on rencontre fréquemment dans les publications et où les temps de travail sont tirés aléatoirement. Pour mieux comprendre l'essence du problème de la chaîne de montage, nous avons généré 500 exemples de problèmes à $n = 9$ tâches et $m = 10$ machines, avec une distribution uniforme et entière des temps de travail entre 1 et 100, car, pour une telle taille, il est possible de visiter l'ensemble des solutions admissibles. On pourra ainsi se faire une idée de la difficulté du problème — et apporter des éléments de réponse à la question « est-il facile de trouver une solution à tant de pour cent de l'optimum ? » — et donc de l'efficacité des méthodes heuristiques considérées. Sur la figure 1, nous avons

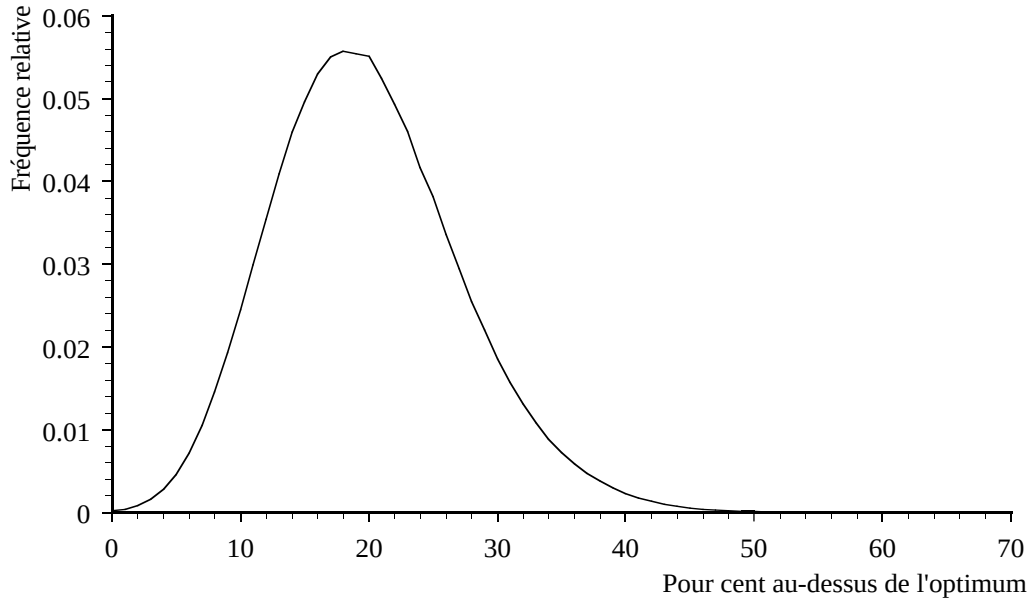

Figure 1 : Distribution de l'ensemble des valeurs de solutions.

représenté la fréquence à laquelle est apparue chaque valeur de solution, relativement à la valeur de la solution optimale. On observe que la valeur moyenne d'une solution tirée au hasard représente un temps de traitement situé environ à 20% au-dessus de la valeur de la meilleure solution ; une heuristique produisant des solutions à 10% ou plus au-dessus de la valeur optimale — comme c'est le cas des heuristiques présentées par Baker dans [5] ou par Dannenbring dans [25] — n'est donc certainement pas très futée… Par contre, une infime partie (0.02%) de l'ensemble des solutions a une valeur à moins de 1% de l'optimum ; il ne faudra donc pas s'étonner d'avoir de la peine à trouver une solution à une fraction de pour cent de l'optimum.

Une autre question qu'il faut se poser est relative à la fiabilité avec laquelle on peut comparer des heuristiques sur une série de problèmes générés aléatoirement : en d'autres termes, si les valeurs des solutions optimales diffèrent beaucoup l'une de l'autre pour une même taille de problème, et si de plus la répartition de ces valeurs n'est pas symétrique par rapport à une valeur moyenne, il faudra comparer les diverses méthodes mises en jeu sur un grand nombre de problèmes pour obtenir des résultats fiables. Pour les 500 problèmes tirés précédemment, nous voyons en figure 2 que la répartition des valeurs

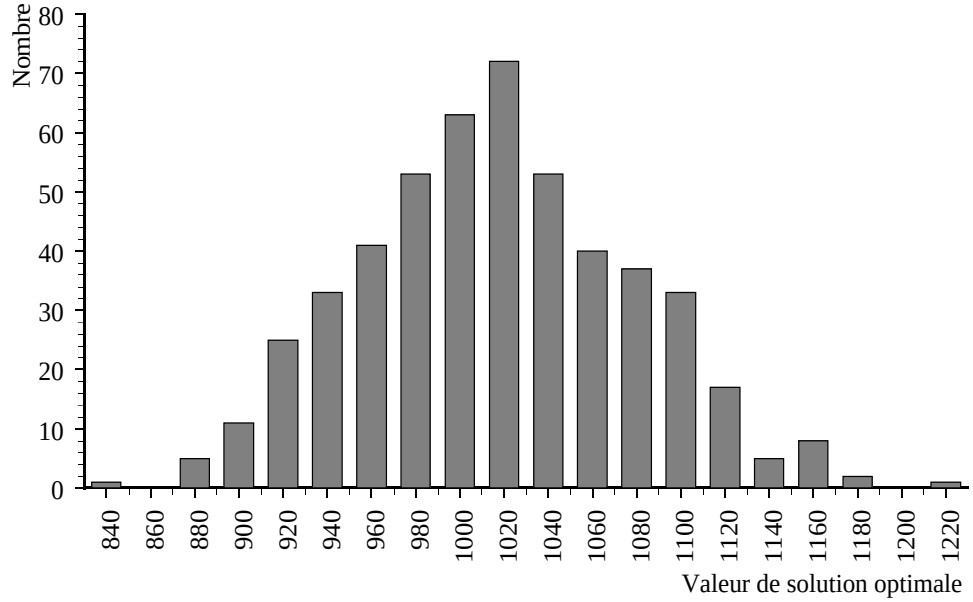

Figure 2 : Distribution des valeurs de solutions optimales.

optimales est plus ou moins symétrique et étalée sur des valeurs à 20% autour de la moyenne. Il devrait donc être possible de réaliser des tests d'heuristiques avec un relativement petit nombre d'exemples de problèmes.

Finalement, remarquons qu'il existe une borne inférieure b à la valeur de la solution optimale, borne facile à calculer et qui, pour le type de problèmes proposés dans [93], se rapproche de cette valeur d'autant plus que n est grand et m petit. Cette borne se justifie ainsi : notons T_i le temps de travail effectif de la machine i (i.e. $T_i = \sum_{k=1}^{n} t_{ik}$, $i = 1, \ldots, m$), d_i le temps minimal que cette machine doit attendre avant de traiter sa première tâche (i.e. $d_i = \min_j \sum_{k=1}^{i-1} t_{kj}$, $i = 2, \ldots, m$, $d_1 = 0$), et f_i le temps minimal s'écoulant entre la fin de son travail et la fin de toutes les opérations (i.e. $f_i = \min_j \sum_{k=i+1}^{m} t_{kj}$, $i = 2, \ldots, m-1$, $f_m = 0$) ; ainsi, pour tout i, l'étendue temporelle des travaux ne peut être inférieure à $d_i + T_i + f_i$. Par conséquent une borne inférieure de l'étendue temporelle des travaux est donnée par :

$$b = \max_{1 \le i \le m} d_i + T_i + f_i \qquad (1)$$

Conjecture 1 :

> Pour des problèmes dont les temps de travail t_{ij} sont entiers, tirés aléatoirement, uniformément entre deux bornes fixes, lorsque m est fixé, la probabilité que la différence entre la valeur d'une solution optimale et la borne inférieure b donnée par (1) soit nulle tend vers 1 lorsque n tend vers l'infini.

Nous motivons cette conjecture par le tableau 1, dans lequel nous donnons, pour des problèmes générés conformément à la conjecture 1 avec des temps de travail tirés entre 1 et 99, la proportion de problèmes pour lesquels nous avons pu trouver une solution (optimale) dont la valeur est précisément la borne inférieure b. À part pour les problèmes à 2 machines, cette proportion est probablement sous-estimée, puisqu'il est possible que la méthode itérative utilisée, [91], ne trouve pas une solution optimale.

Nous pensons qu'il est important que les temps de travail soient entiers et compris entre deux bornes fixes, car, plus l'écart entre ces bornes est grand, plus la proportion de problèmes pour lesquels nous arrivons à trouver une solution de valeur égale à la borne

Nombre de tâches	Nombre de machines	borne *b* atteinte (%)
20	2	93
50	2	97.8
100	2	99.1
200	2	99.6
500	2	99.9…
20	5	35
50	5	41
100	5	54
200	5	65
20	10	1
50	10	3
100	10	6
200	10	28

Tableau 1 : Proportion empirique des problèmes où la borne *b* est atteinte.

inférieure *b* est faible. Donc, il se pourrait que la conjecture soit fausse pour des temps de travail aléatoires réels. La fonction de répartition des temps de travail a aussi son importance ; la conjecture pourrait être fausse pour certaines fonctions, mais probablement pas pour des fonctions « régulières » comme une répartition normale. Dans le tableau 1, nous remarquons que lorsque *m* devient grand, la proportion de problèmes pour lesquels la borne *b* est atteinte diminue fortement. C'est pourquoi nous avons choisi de fixer *m* dans la conjecture.

1.1.3. État actuel de la recherche

Ce problème est facile à résoudre pour $m = 2$ (cf. Johnson, [58]), mais il est NP-difficile pour *m* quelconque (cf. Garey et Johnson, [39]) et ne peut alors être résolu exactement que pour des exemples de taille relativement petite. Beaucoup de méthodes heuristiques ont été développées pour tenter de trouver de bonnes solutions à ce problème ; on peut citer quelques méthodes de complexité $O(n \log(n) + nm)$ dont on pourra trouver la description dans Baker, [5] ou dans Dannenbring, [25], qui consistent à trier les objets selon divers critères avant de calculer la valeur de la solution. Elles fournissent des solutions de qualité médiocre (plus de 5 à 10% au-dessus de l'optimum, si *n* et *m* deviennent grands) mais

nécessitent des temps de calcul très courts. Plus récemment, Navaz et al. [71] ont présenté une méthode de complexité $O(n^2m)$ qui fournit des résultats sensiblement meilleurs que ceux des précédentes (solutions dont la valeur est de 2 à 4% au-dessus de l'optimum). Ensuite, plusieurs recherches itératives ont été implantées : tout d'abord SPIRIT, due à Widmer et Hertz [98] et quelques recuits simulés dûs à Ogbu et Smith, [75], [76] et à Osman et Potts, [78] dont les efficacités sont plus ou moins comparables. Enfin, l'approche de Reeves, [82], basée sur les méthodes génétiques, produit des résultats semblables aux algorithmes de recuit simulé, mais son auteur n'a pas réussi à améliorer une seule des meilleures solutions connues d'un jeu de 120 problèmes que nous proposons dans [93] et qui ont été trouvées par la recherche itérative dirigée présentée plus loin qui a fait l'objet d'une publication [91]. La recherche itérative dirigée de [91] constitue une nette amélioration de SPIRIT ; on peut donc la considérer comme une des méthodes fournissant les meilleures solutions au problème de la chaîne de traitement.

Dans la littérature, on trouve quelques exemples de problèmes de ce type, citons Carlier qui donne dans [14] huit petits problèmes avec leurs valeurs de solution optimale. Les 120 problèmes que nous proposons dans [93] forment un jeu dont la prétention est de lancer de nouveaux défis tant pour les méthodes de résolution exacte (les plus petits exemples, comportant 20 objets et 5 machines, peuvent être abordés par de telles méthodes) que pour les méthodes approchées (les meilleures solutions que nous donnons pour les plus gros exemples proposés — 500 objets et 20 machines — ont demandé plusieurs heures de calcul chacune).

1.2. PROBLÈME DE TRAITEMENT À SÉQUENCES FIXÉES

1.2.1. Description

Ce problème est connu sous le nom de « job shop » en anglais et il est noté $J||C_{max}$ dans la classification de Lawler et al. [65]. Comme pour le problème précédent, on cherche ici à minimiser la longueur de l'intervalle de temps séparant le début et la fin du traitement de n objets sur m machines. Chaque objet doit subir un certain nombre d'opérations, pas

forcément le même pour tous les objets. Nous notons N le nombre total d'opérations à effectuer. Une opération i est caractérisée par :

- o_i l'objet sur lequel elle doit être réalisée,
- m_i la machine sur laquelle elle doit avoir lieu,
- t_i sa durée,
- po_i (si elle existe) l'opération devant être réalisée immédiatement avant i sur l'objet o_i,
- so_i (si elle existe) l'opération succédant à i sur l'objet o_i.

Ainsi, la gamme opératoire de chaque objet varie selon l'objet (alors qu'elle était unique dans le problème précédent) ; par contre, les opérations d'un objet doivent être réalisées dans un ordre et sur une machine donnés, d'où la dénomination de problème de traitement à séquences fixées. On suppose à nouveau que chaque machine ne peut effectuer qu'une opération à la fois et que le nombre d'objets en attente de traitement devant une machine n'est pas limité. De même, un objet ne peut être traité que par une machine à la fois et les opérations ne peuvent pas être interrompues. Remarquons qu'un problème où toutes les gammes opératoires seraient composées de m opérations devant avoir lieu sur les machines 1, 2, …m constitue une généralisation du problème de la chaîne de traitement puisqu'un objet peut en dépasser un autre.

Une solution de ce problème consiste à donner pour chaque opération :

- pm_i (si elle existe) l'opération qui a lieu sur la machine m_i juste avant i,
- d_i le temps séparant le début des opérations du début de i.

De manière équivalente, une solution peut être donnée par :

- sm_i (si elle existe) l'opération qui a lieu sur m_i succédant à i,
- f_i le temps séparant la fin de i de la fin des opérations.

On peut donner une formulation de ce problème en termes de graphe : à chaque opération correspond un sommet ; deux sommets fictifs, 0 et $N + 1$, représentant respectivement le début et la fin du travail, sont ajoutés au graphe. Un arc (i, j) de longueur t_i relie les sommets-opérations i et j si $o_i = o_j$ et $po_j = i$. Au sommet-opération initial de chaque objet est connecté un arc de longueur 0 issu du sommet-opération 0 ; de même on relie le

sommet-opération final de chaque objet au sommet-opération $N + 1$ par un arc de longueur équivalente à la durée de l'opération finale de chaque objet. Les opérations devant s'effectuer sur une même machine sont reliées entre elles par une arête, formant par conséquent une clique.

Le problème consiste à choisir une orientation pour toutes les arêtes (en leur donnant une longueur égale au temps de travail de l'opération initiale et en ne créant pas de circuits) de manière à minimiser la longueur du plus long chemin de 0 à $N + 1$. La figure 3 illustre

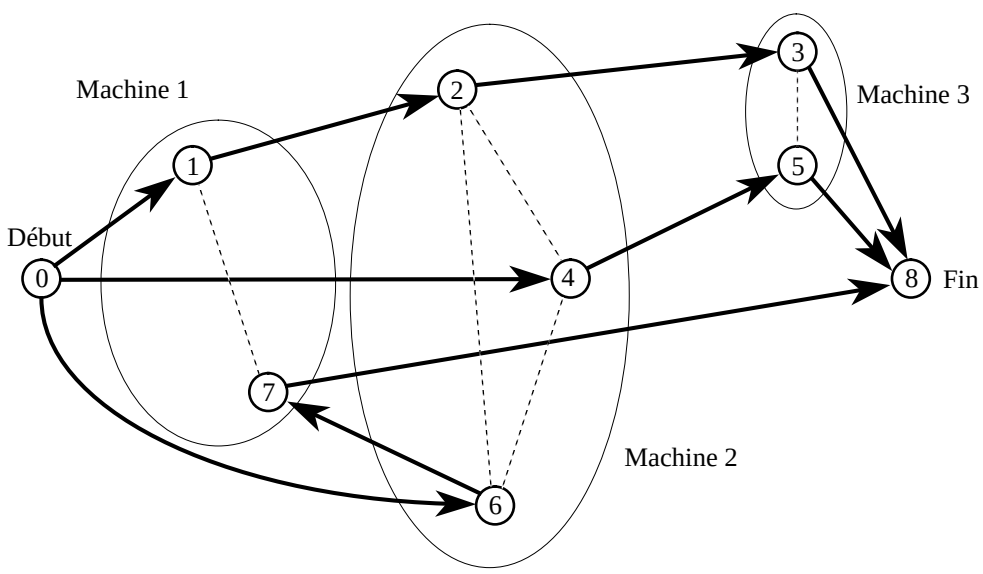

Figure 3 : Exemple de problème de traitement à séquences fixées.

le graphe obtenu pour un exemple à 3 objets et 3 machines où l'objet 1 doit subir les opérations 1, 2, 3 qui ont lieu sur les machines de mêmes numéros, l'objet 2 les opérations 4, 5 sur les machines 2, 3 et l'objet 3 les opérations 6, 7 effectuées par les machines 2, 1.

1.2.2. Propriétés et conjectures

Nous considérerons par la suite le type de problème le plus étudié dans la littérature où les gammes opératoires des objets sont tirées aléatoirement, ne sont pas en corrélation les unes avec les autres et sont constituées de m opérations — une par machine. Les temps de

travail sont entiers, tirés aléatoirement et uniformément entre deux bornes (1 et 99, généralement).

Ce type de problème admet une borne inférieure b à la valeur de la solution optimale similaire à celle de l'équation (1). Mais comme le temps d'inactivité minimal de chaque machine au début et à la fin des travaux tend vers 0 lorsque n devient grand, on peut la simplifier et écrire :

$$b = \max_i T_i = \max_i \sum_{k\,:m_k = i} t_k \qquad (2)$$

Conjecture 2 :

> Pour des problèmes où les objets doivent passer une fois sur toutes les machines mais dont les ordres de passage ne sont pas corrélés et dont les temps de travail sont entiers, tirés aléatoirement, uniformément entre deux bornes fixes, la probabilité que la différence entre la valeur d'une solution optimale et la borne inférieure b donnée par (2) soit nulle tend vers 1 lorsque $n/m \to \infty$.

Cette conjecture est motivée par le fait que nous avons toujours pu trouver une solution de valeur b (calculée comme dans (1) car n n'était pas toujours très grand) pour des milliers d'exemples de problèmes avec temps de travail compris entre 1 et 99 lorsque $n \geq 6 \cdot m$ ($m = 2 \ldots 20$) et par le tableau 2 où l'on voit que la proportion de

Nombre d'objets	Nombre de machines	borne b atteinte (%)
15	15	0
20	15	4
30	15	26
50	15	78
30	20	0
50	20	27
100	20	97

Tableau 2 : Proportion empirique des problèmes où la borne *b* est atteinte.

problèmes rencontrant cette borne est bien plus élevée à tailles égales que pour des problèmes de chaîne de traitement. Il faut remarquer que l'on n'impose plus que m soit fixé, mais simplement que $n/m \to \infty$.

Il semble que les problèmes les plus difficiles à résoudre sont ceux pour lesquels $n \cong m$. Par contre les problèmes où $n \gg m$ sont beaucoup plus faciles à résoudre, à tel point que la recherche itérative dirigée mise au point dans [92] et présentée plus loin, réussit à trouver les solutions optimales de ces problèmes en un temps qui semble polynomial en n et m, ce qui nous permet d'émettre la

Conjecture 3 :

> Il existe un algorithme polynomial en n et m qui trouve une solution optimale, avec probabilité tendant vers 1 lorsque $n/m \to \infty$, aux problèmes où les objets doivent passer une fois sur toutes les machines mais dont les ordres de passage ne sont pas corrélés et dont les temps de travail sont entiers, tirés aléatoirement, uniformément entre deux bornes fixes.

Remarquons que nous ne connaissons pas un tel algorithme : la méthode de déplacement du goulot d'étranglement, du moins dans son implantation la plus efficace connue à présent, due à Applegate et Cook [4], semble prendre un temps exponentiel en n (voir figure 15, page 61) alors que nous ne pouvons pas garantir l'obtention d'une solution optimale en un temps polynomial avec la méthode de [92], même si, pour les problèmes que nous avons tirés, cela semble être le cas.

1.2.3. État actuel de la recherche

Ce problème est NP-difficile (cf. Lawler et al. [65]) et il ne peut être résolu exactement que pour des exemples bien particuliers : lorsque n et m sont petits (< 15), ou lorsque $n \gg m$. Pour ce qui est des méthodes exactes, signalons celle de Carlier et Pinson [15], où l'on prouve pour la première fois qu'une solution d'un problème à 10 objets et 10 machines posé un quart de siècle plus tôt par Fisher et Thompson dans [36] était optimale, et celle de Brucker et al. [9], qui a sensiblement accéléré la méthode

d'énumération implicite de Carlier et Pinson [15] et qui a été capable de trouver la plupart des solutions optimales des 40 petits problèmes dûs à Lawrence [66]. Les tailles pour lesquelles il n'a pas été possible de prouver que les solutions étaient optimales sont : 15 objets 10 machines, 20 objets 10 machines, 15 objets 15 machines. On peut donc considérer que les problèmes comportant à peu près le même nombre d'objets que de machines peuvent être résolus avec satisfaction jusqu'à cent à deux cents opérations.

Pour des problèmes de plus grande taille, il est nécessaire de recourir à des méthodes heuristiques pour trouver de bonnes solutions en un temps raisonnable. Citons par ordre chronologique celle d'Adams et al. [2], dont l'idée consiste à trouver successivement le meilleur ordre possible des travaux à effectuer sur chaque machine constituant un goulot d'étranglement de la solution courante, c'est-à-dire sur une machine qui termine son travail en dernier, le recuit simulé de van Laarhoven et al. [62], la méthode d'Applegate et Cook [4], qui est une généralisation de [2] et enfin la technique d'insertion de Werner et Winkler [97]. Dans [92], nous comparons une recherche itérative dirigée, qui sera présentée plus loin, au recuit simulé de van Laarhoven et al. [62] et à la méthode d'Applegate et Cook [4]. Nous arrivons à la conclusion que notre méthode, pour tous les exemples de problèmes testés, surpasse nettement le recuit simulé de [62] ainsi que la méthode de déplacement du goulot d'étranglement de [4], au moins en ce qui concerne la qualité des solutions produites et souvent au niveau du temps de calcul nécessaire. Par contre, si elle réussit à trouver de très bonnes solutions en un temps très court pour de petits problèmes, notre méthode peut prendre plus de temps pour trouver une solution optimale que certaines méthodes exactes.

1.3. PROBLÈME D'AFFECTATION QUADRATIQUE

1.3.1. Description

Étant donné n objets et des flots f_{ij} entre l'objet i et l'objet j ($i, j = 1 \ldots n$), et n emplacements dont les distances d_{rs} entre les emplacements r et s sont connues ($r, s = 1 \ldots n$), il s'agit de placer les n objets sur les n emplacements de manière à

minimiser la somme des produits flots $\times$ distances. Mathématiquement, cela revient à chercher une permutation π dont la $i^{\text{ième}}$ composante $\pi(i)$ donne la place de l'objet i, permutation qui minimise $\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)}$.

Ce problème a de multiples applications pratiques ; les plus connues sont sans doute la répartition de bâtiments ou de services sur un campus universitaire (cf. Dickey et Hopkins [29]) ou dans un grand hôpital au Caire (cf. Elshafei [32]), le placement de modules logiques dans des circuits électroniques (cf. Nugent et al. [74], Skorin-Kapov [88] et Steinberg [90]), et le placement des touches de claviers de machines à écrire (cf. Burkard et Offermann [12]). La matrice des flots, dans ces exemples, représente respectivement, la fréquence à laquelle les personnes doivent se déplacer d'un bâtiment à un autre, le nombre de connexions devant être réalisées entre deux modules et la fréquence à laquelle deux lettres particulières apparaissent consécutivement dans une langue donnée. La matrice des distances a une signification évidente dans les deux premiers exemples ; dans le troisième, elle représente le temps séparant la frappe de deux touches. On trouvera une bibliographie plus complète et commentée dans Burkard [10] ou dans Finke et al. [35], par exemple.

1.3.2. Propriétés

Le problème d'affectation quadratique est NP-difficile et, à moins que P = NP, il n'existe pas de schéma d'approximation polynomial pour ce problème (cf. Sahni et Gonzalez [86]). Cependant, les problèmes générés aléatoirement (flots et distances tirés uniformément) jouissent de la propriété suivante : lorsque $n \to \infty$, la valeur d'une solution quelconque (même la plus mauvaise) tend vers la valeur d'une solution optimale (cf. Burkard et Finke [11]). Nous avons observé dans [94] que si la valeur d'une solution aléatoire est environ à 20% au-dessus de la valeur optimale pour les problèmes de taille 10, on est encore à plus de 10% pour les problèmes de taille 100. Pour des problèmes générés différemment, par exemple où les emplacements sont situés sur une grille et les flots générés aléatoirement, comme dans Nugent et al. [74] ou dans Skorin-Kapov [88], la différence avec l'optimum est encore plus grande. Il existe même des problèmes, par

exemple ceux de Steinberg [90], où la valeur d'une solution aléatoire est plus de quatre fois plus élevée que la valeur optimale.

1.3.3. État actuel de la recherche

Les méthodes exactes, par exemple celle de Roucairol [84], sont capables de traiter des problèmes jusqu'à une taille de 15 environ. Comme les tailles de problèmes pratiques dépassent souvent 20 ou 30, il n'est pas étonnant que moult méthodes heuristiques aient fleuri. Si l'on considère les plus efficaces mises au point à ce jour, force est de constater que la plupart sont des recherches itératives.

En premier lieu, des recuits simulés ont été développés (cf. Burkard et Rendl [13], Wilhelm et Ward [99] et Connolly [22]) qui ont rapidement été surpassés par des méthodes plus intelligentes et plus agressives. Tout d'abord celle de Skorin-Kapov [88] dont l'idée principale consiste à varier manuellement en cours de recherche les paramètres au vu des solutions produites ; cette recherche itérative dirigée, déjà plus efficace que le recuit simulé de Burkard et Rendl [13], a été supplantée par la nôtre [94] qui est à la fois plus rapide — la complexité d'un pas de la recherche a été diminuée de $O(n)$ — et plus robuste car de bonnes valeurs de paramètres peuvent être données a priori. Skorin-Kapov a ensuite automatisé les variations de paramètres de sa recherche [88] pour arriver à [89], une méthode à onze (!) phases qui semble assez performante, bien que non formellement comparée à la nôtre [94]. Citons encore la recherche itérative dirigée de Chakrapani et Skorin-Kapov [16], pour son adaptation à une machine massivement parallèle.

La recherche itérative dirigée que nous proposons et qui sera discutée plus loin est donc une des méthodes les plus efficaces à ce jour pour le problème d'affectation quadratique tout en étant exemplairement simple à mettre en œuvre — quelques dizaines de lignes de code Pascal, pas de paramètres à ajuster.

1.4. PROBLÈMES DE DISTRIBUTION DE BIENS

Nous étudierons deux variantes de ce problème, aussi connu sous le nom de problème d'élaboration de tournées de véhicules ; la première est académique et ne contient qu'un nombre extrêmement restreint de contraintes alors que la seconde est issue d'un cas pratique et comporte de ce fait plusieurs complications qui obligent à adapter une recherche itérative de manière fort différente.

1.4.1. Descriptions

Le problème académique, qui constitue une simplification extrême de problèmes réels peut être décrit comme suit : un ensemble non limité de véhicules pouvant transporter un volume V de marchandises doivent livrer n commandes de biens à partir d'un dépôt unique de manière à minimiser la distance totale qu'ils parcourent. Chaque commande i a un volume v_i ($i = 1, \ldots, n$) et on connaît les distances directes d_{ij} entre les lieux où doivent être livrées les commandes i et j ($i, j = 0, \ldots, n$, 0 représentant le dépôt). Les véhicules effectuent des tournées T_k, ($k = 1, 2, \ldots$), qui partent du dépôt et qui y reviennent. Une variante de ce problème impose de plus que la longueur des tournées soit bornée supérieurement par une valeur L donnée.

Pour le problème réel que nous avons traité, il y a deux types de véhicules : des camions et des remorques. Chaque véhicule est caractérisé par son volume et sa charge utile, par son coût d'utilisation au kilomètre ainsi que par le sous-ensemble de véhicules qu'il peut tirer (pour un camion) ou qui peut le tirer (pour une remorque). Si le volume utile d'un train routier est bien la somme des volumes utiles du camion et de la remorque, en revanche, la charge utile est limitée par le fait que le poids d'un train routier ne doit pas dépasser les 28 tonnes inscrites dans la loi suisse.

Les commandes sont passées par des magasins situés dans les cantons de Vaud et du Valais, chaque magasin pouvant en passer jusqu'à deux dont la somme peut dépasser la capacité d'un véhicule. Chaque magasin est caractérisé par une fenêtre de temps pendant laquelle il peut être desservi, par le sous-ensemble de camions qui peut l'atteindre (route de montagne), par le fait qu'il dispose de l'infrastructure nécessaire pour recevoir un train

routier et par le temps de manœuvre pour y accéder avec un camion, et, s'il y a lieu avec un train routier. Une commande est caractérisée par le magasin qui l'a émise, par son volume, par son poids et par un temps de déchargement.

Chaque véhicule ne peut effectuer qu'un parcours au plus. Un camion seul partira du dépôt — à 5 h. du matin au plus tôt —, fera un parcours circulaire, passant une fois par chaque magasin pour lequel il a une commande et reviendra au dépôt. La tournée d'un train routier peut contenir des magasins qui ne sont accessibles que par camion ; dans ce cas, la remorque devra être dételée et laissée pour déchargement à un magasin équipé pour recevoir des trains routiers, puis le camion seul ira livrer des commandes issues de magasins accessibles par camion uniquement, retournera au magasin où la remorque a été laissée et sera à nouveau attelée, avant de continuer la tournée.

Le but de ce problème réel est de minimiser les coûts de livraisons, ce qui ne va pas forcément de pair avec la minimisation du nombre de véhicules engagés ou de la distance totale parcourue. Par rapport au problème académique, ce problème contient de multiples difficultés supplémentaires, comme l'élaboration des tournées, puisqu'on a des fenêtres de temps et un mode de livraison particulier par train routier, ou l'affectation des véhicules à un ensemble de tournées construites au préalable.

1.4.2. Propriétés et conjecture

Ce problème est NP-difficile, ce qui se montre aisément par le fait que le problème de la partition d'un ensemble $\{a_1, a_2, \ldots, a_n\}$ de nombres entiers dont la somme est paire en deux sous-ensembles dont la somme des éléments est identique et vaut $V = \left(\sum_{k=1}^{n} a_k \right) / 2$, problème NP-complet (cf. Garey et Johnson [39]), peut se transformer en un problème de distribution comme suit : les n commandes ont un volume $v_i = a_i$ ($i = 1, \ldots, n$) et doivent être livrées en un même lieu, éloigné d'une distance $d > 0$ du dépôt à l'aide de véhicules de capacité V. Il existera une solution (optimale) de valeur $4d$ si, et seulement si le problème de bipartition a une solution.

Peut-être ne connaît-on que très peu de propriétés sur les problèmes de distribution à cause du nombre de paramètres nécessaires à la génération d'un exemple de problème

— distances, volumes demandés, volumes transportés. C'est pour cela que nous avons cherché des exemples de problèmes dont une solution, que nous conjecturons optimale, puisse être facilement trouvée. Ces problèmes sont générés sur une grille de la manière suivante : soient q et k, deux entiers positifs ; les $n = (2q \cdot k)^2$ commandes de volumes unitaires doivent être livrées en des lieux de coordonnées cartésiennes (i, j) $(i, j = -q \cdot k + 1,\ -q \cdot k + 2,\ \ldots,\ q \cdot k)$ par des véhicules de capacité $V = 2q$ à partir d'un dépôt situé en (0.5, 0.5). Pour ces problèmes, nous émettons la

Conjecture 4 :

> Pour $q = 1$, $q = 2$ et $q = 3$, pour tout k, entier supérieur à 1 il existe une solution optimale (l'ensemble des tournées) du problème généré sur une grille avec q et $k - 1$ qui est contenue dans une solution optimale du problème généré avec q et k.

Si cette conjecture est vraie, on peut donc construire successivement les solutions optimales pour des problèmes de plus en plus gros simplement en ajoutant chaque fois une « couche » de commandes d'épaisseur q autour de la grille précédente.

Cette conjecture est essentiellement motivée par le fait que les meilleures solutions que nous avons obtenues à l'aide d'une recherche itérative dirigée présentaient ces caractéristiques et que le bon sens ne nous permettait pas de penser qu'elles n'étaient pas optimales. Graphiquement, les solutions qui seraient optimales sont représentées en figure 4 pour $q = 2$ et $q = 3$. Sur cette figure, les trajets depuis et vers le dépôt ne sont pas dessinés par souci de clarté. Nous remarquons que les solutions supposées optimales ne contiennent que des tournées dont les lieux de livraison sont situés sur des rectangles.

Cette propriété, de même que la conjecture, ne sont certainement pas généralisables pour $q \geq 4$ car les meilleures solutions connues ne présentent pas ces caractéristiques, comme on peut le voir en figure 5 où nous avons représenté la meilleure solution connue pour $q = 4$ et $k = 1$ ainsi que la meilleure manière d'organiser les tournées dans une sous-grille de 8×8 points située dans un coin de la grille obtenue avec $q = 4$ et $k = 6$.

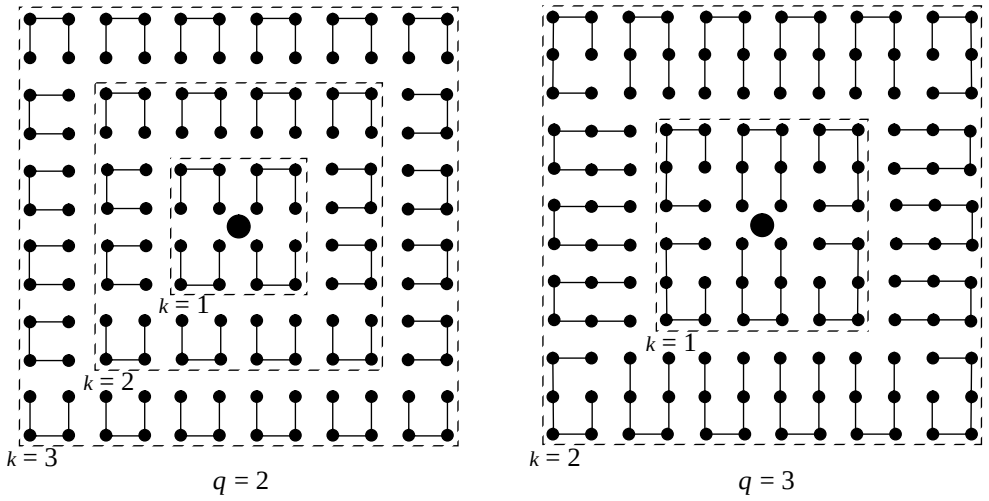

Figure 4 : Solutions probablement optimales sur des grilles avec $q = 2$ et $q = 3$.

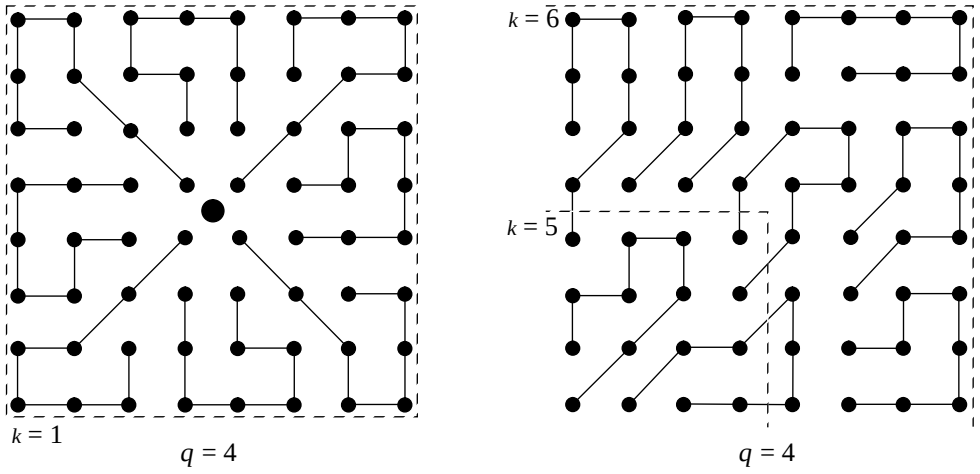

Figure 5 : Meilleures solutions connues sur des portions de grilles avec $q = 4$.

1.4.3. État actuel de la recherche

Les problèmes de distribution de biens sont étudiés depuis longtemps et de multiples méthodes exactes et heuristiques ont vu le jour. Pour les méthodes exactes, mentionnons

celles de Christofides et al. [19], [20], Laporte et al. [64] et Argarwal et al. [3] qui toutefois ne peuvent traiter que de petits exemples (50 commandes et 8 véhicules selon Christofides [17]). Il existe quatre types de méthodes heuristiques : les méthodes constructives qui élaborent les tournées des véhicules en ajoutant l'une après l'autre toutes les commandes (cf. Clarke et Wright [21], Desrochers et Verhoog [28], Dror et Levy [30], Gaskel [40], Mole et Jamerson [70], Nelson et al. [72], Passens [80] et finalement celle de Yellow [104]), celles qui procèdent en deux phases — groupement de villes puis résolution de problèmes de voyageur de commerce ou résolution d'un problème de voyageur de commerce puis décomposition en tournées admissibles — (cf. les méthodes de Christofides et al. [18], Fisher et Jaikumar [37], Gillett et Miller [43], Haimovich et Rinnooy Kan [49], Noon et al. [73], Wren [102] et Wren et Holliday [103]), celles basées sur une énumération implicite partielle et enfin les méthodes itératives (cf. Gendreau et al. [42], Osman [77], Pureza et França [81], la nôtre [95] et celle de Willard [100]). Pour des informations plus complètes, on pourra se référer à Bodin et al. [7], Christofides [17], Christofides et al. [18], Golden et Assad [48] et Laporte et Nobert [63].

De toutes ces méthodes heuristiques, les méthodes itératives produisent de loin les meilleures solutions, du moins si on les compare sur un jeu de 14 problèmes proposés par Christofides et al. dans [18]. Le recuit simulé et la recherche itérative dirigée d'Osman [77] n'ont réussi à trouver que 5 des meilleures solutions connues, la recherche itérative dirigée de Gendreau et al. [42] que 9 des meilleures solutions, alors que la technique de décomposition que nous proposons dans [95] et qui sera discutée plus loin, associée à une recherche itérative dirigée très simple, a réussi à trouver toutes les meilleures solutions connues, dont 5 n'ont jamais été égalées. Il est fort probable que la plupart de ces problèmes soient résolus de manière optimale (à part les 3 ou 4 plus gros).

En ce qui concerne le problème réel, très spécialisé, la littérature ne comporte que nos travaux, [85], qui seront présentés partiellement et commentés plus loin. Cette méthode permet de trouver en un temps raisonnable des solutions qui coûtent 10 à 20% de moins que les solutions adoptées par l'entreprise, tout en étant parfois mieux adaptées à ses besoins. Comme on ne peut supposer que cette dernière cherche à perdre de l'argent, on

en déduit que notre recherche itérative dirigée, [85], est donc une méthode tout à fait compétitive.

Pour conclure ce chapitre, nous relèverons que les recherches itératives dirigées discutées plus loin nous ont permis, par les solutions excellentes qu'elles produisent, de mieux connaître les problèmes abordés ici. Ainsi nous avons appris que c'est vraisemblablement une augmentation conjointe du nombre d'objets et de machines sur lesquelles ils doivent subir des opérations qui rend difficiles les problèmes de la chaîne de traitement et de traitement à séquences fixées, puisque les problèmes où le nombre de machines est fixé peuvent se résoudre de plus en plus rationnellement avec l'augmentation du nombre d'objets.

Pour le problème de distribution de biens, la connaissance de la solution optimale de problèmes de grande taille est une donnée qui permet de tester avec acuité l'efficacité de méthodes approchées ; même si nous n'avons pu prouver que les solutions présentées en figure 4 sont optimales, celles-ci nous suggèrent une décomposition du problème en couches concentriques ; par ailleurs, tant la preuve de la conjecture 4 que la découverte de solutions meilleures restent un défi.

2. RECHERCHES ITÉRATIVES

2.1. INTRODUCTION

Dans ce chapitre, nous présenterons succinctement les bases de quelques méthodes itératives couramment utilisées pour « résoudre » des problèmes d'optimisation combinatoire. Nous avons mis entre guillemets le mot « résoudre » pour signaler que les solutions produites par les méthodes itératives ne sont souvent pas optimales et qu'une connotation qualitative s'ajoute à la dimension temporelle habituelle du mot : un problème pourra être résolu plus ou moins bien, suivant la qualité de la solution trouvée, et en plus ou moins de temps. La comparaison de méthodes itératives devra donc se faire simultanément sur ces deux critères.

Les méthodes itératives sont souvent classées parmi les techniques d'intelligence artificielle. Cette classification est sans doute abusive, du moins pour certaines d'entre elles, guidées presque exclusivement par le hasard. Des auteurs taxent peut-être ces méthodes du qualificatif d'intelligent parce qu'elles produisent, après un nombre souvent impressionnant d'itérations durant lesquelles elles ont énuméré quantité de solutions médiocres voire mauvaises, une solution de bonne qualité qui aurait demandé un gros effort à un acteur humain.

Presque toutes ces méthodes cherchent à imiter un comportement de la nature : les méthodes génétiques considèrent les solutions d'un problème comme des chromosomes et font subir à celles-ci des opérations de croisement, de mutation et de sélection ; le recuit simulé considère la fonction à optimiser comme l'énergie d'un matériau en fusion, énergie que l'on cherchera à minimiser par décroissance très lente d'un paramètre

analogue à la température dans le processus physique réel ; les méthodes d'acceptation à seuil dérivent directement du recuit simulé ; finalement, les recherches itératives dirigées, présentées dans le chapitre suivant, tentent d'inculquer au processus itératif de la mémoire et des rudiments d'intelligence, ou plus exactement de connaissance sur le problème traité. Dans les sections suivantes, nous allons passer en revue les fondements de ces méthodes itératives qui constituent l'héritage des méthodes itératives dirigées.

2.2. ALGORITHMES GÉNÉTIQUES

Ce type d'algorithmes a été proposé il y a plus de quinze ans par Holland [57] ; on trouvera une collection de papiers et une bibliographie très complète dans Davis [26] et [27]. L'idée de base de ces algorithmes est de simuler les méthodes d'amélioration génétique ou de sélection naturelle. Très schématiquement, une manipulation génétique consiste à produire un chromosome fils à partir de deux chromosomes parents en substituant des gènes du premier chromosome parent à certains du second.

Ce processus se traduit, en termes d'optimisation combinatoire, par le mélange (ou croisement pour reprendre la terminologie génétique) de portions (l'analogue des gènes) de solutions (les chromosomes). Cela suppose que l'on dispose d'un ensemble (une population) de solutions en principe différentes ; cet ensemble verra son cardinal augmenter si certaines solutions ne sont pas éliminées. Pour conserver une population de taille raisonnable, généralement comprise entre une dizaine et une centaine de solutions, après un nombre donné de croisements, on éliminera certaines solutions aléatoirement, avec une plus grande probabilité pour les moins bonnes.

Durant le croisement de deux chromosomes, il peut se produire des mutations, c'est-à-dire des modifications aléatoires à l'intérieur des gènes. Ces mutations seront simulées dans le processus combinatoire afin de préserver la diversité de la population des solutions. De manière générique, les algorithmes génétiques peuvent être décrits comme suit :

Méthode génétique :

Données, initialisation :

Choisir p solutions initiales s_1^0, s_2^0, …, s_p^0 admissibles, la meilleure étant aussi notée s^*.

Choisir $C > 0$

$k := 0$

Itération à répéter tant qu'une condition d'arrêt n'est pas remplie :

$c := 0$

Augmentation de la population à répéter tant que $c < C$:

Choisir aléatoirement deux solutions s_i^k et s_j^k $(1 \le i, j \le p,\ s_i^k \ne s_j^k)$.

Incrémenter c d'une unité.

Croiser s_i^k avec s_j^k pour obtenir s_{p+c}^k

Faire subir une mutation aléatoire à s_{p+c}^k.

Si s_{p+c}^k est meilleure que s^*, poser $s^* := s_{p+c}^k$

Sélection :

Éliminer aléatoirement C solutions de la génération k, en créant ainsi une nouvelle génération de solutions s_1^{k+1}, s_2^{k+1}, …, s_p^{k+1}.

Incrémenter k d'une unité.

Résultat : s^*.

Le choix aléatoire de solutions répond à une distribution non uniforme qui privilégie l'élimination de solutions mauvaises et qui avantage la production de solutions à partir de parents correspondant à de bonnes solutions. Dans cet algorithme, nous n'avons pas spécifié de critères d'arrêt, c'est-à-dire le nombre d'itérations que l'algorithme effectuera, nombre qui influencera directement son temps d'exécution. Dans la pratique, c'est souvent une borne sur ce dernier qui détermine le critère d'arrêt car il est difficile de trouver d'autres critères logiques : dans la plupart des cas, pour des problèmes NP-difficiles, on ne connaît pas la valeur d'une solution optimale et aucune recherche itérative ne peut garantir de produire en un temps fini une solution optimale voire de qualité donnée pour certains problèmes. Par conséquent, à moins de pouvoir caractériser les solutions satisfaisantes, seule la patience de l'utilisateur d'une méthode itérative est déterminante pour le choix du critère d'arrêt. Notons que la patience est une notion floue qui dépend de

divers paramètres, comme par exemple le fait d'avoir trouvé des solutions meilleures que celles publiées dans la littérature…

Pour illustrer les concepts de reproduction, mutation et sélection, nous donnons brièvement une adaptation de méthode génétique au problème de la chaîne de traitement, adaptation due à Reeves [82]. Dans cette dernière, la population initiale est composée de permutations aléatoires. À chaque moment de la recherche, les solutions sont triées par ordre de qualité croissante. Le mécanisme de sélection choisit la solution s_i^k avec une probabilité $\frac{2i}{P(P+1)}$ et la solution s_j^k avec une probabilité $\frac{1}{P}$, où P est le cardinal de la population courante. Ensuite, on choisit aléatoirement, uniformément, un point de croisement entre les permutations s_i^k et s_j^k. Une permutation fille est obtenue en retenant les éléments d'un parent jusqu'au point de croisement, puis en insérant dans l'ordre chaque élément non encore inséré de l'autre parent. La figure 6 illustre le croisement de deux solutions.

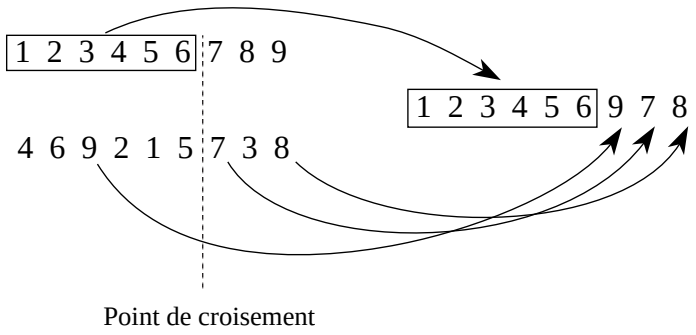

Figure 6 : Un croisement possible entre deux permutations.

Le mécanisme de mutation, illustré en figure 7 consiste à déplacer aléatoirement un élément de la permutation, lui-même choisi au hasard.

Il peut sembler surprenant que de tels mécanismes aléatoires puissent fournir de bonnes solutions, mais l'intuition qu'une partie importante de l'ensemble des bonnes solutions présente des séquences communes doit s'avérer correcte pour le problème de la chaîne de traitement. Ce qui est essentiel dans les algorithmes génétiques, c'est de déterminer

Figure 7 : Une mutation possible d'une permutation.

quelles structures doivent être conservées dans les bonnes solutions et comment concilier deux structures apparemment antagonistes ; c'est de cela que dépendra l'efficacité d'une adaptation. Le principal avantage des algorithmes génétiques est qu'ils sont pensés à la base pour être exécutés sur des ordinateurs distribués ; en effet, dans l'algorithme tel qu'il a été présenté, l'augmentation de la population, qui est généralement la partie la plus coûteuse en temps calcul, ne nécessite aucune synchronisation entre processus, il suffit que chacun d'eux ait à sa disposition en début d'itération la population initiale.

2.3. MÉTHODES DE DESCENTE

Le qualificatif de « descente » pour ce type de méthodes provient du fait que l'on suppose que l'optimisation de la fonction-objectif est une minimisation. Par la suite, on fera l'hypothèse que le problème à résoudre peut se formuler de la manière suivante :

$$\min_{s \in S} f(s) \tag{3}$$

Dans cette formulation, f dénote la fonction-objectif, s une solution admissible du problème et S l'ensemble des solutions admissibles. Pour être plus exhaustif, on aurait pu parler de méthodes d'amélioration. Ces méthodes, dans un cadre tout à fait général, sont vieilles comme le monde. En effet, que fait un acteur humain de nature sceptique lorsqu'on lui donne la solution d'un problème dont il n'est pas capable ou dont il n'a pas la patience de trouver une solution optimale ? Cet acteur va essayer de modifier légèrement la solution qu'on lui propose et de vérifier qu'il n'est pas possible de trouver de meilleures solutions en procédant à des changements locaux.

Formellement, cela suppose qu'il est possible de définir pour toute solution s de S un ensemble $N(s)$ de solutions, contenu lui aussi dans S et que l'on appellera ensemble des solutions voisines de s. Par exemple, pour le problème de la chaîne de traitement, l'ensemble $N(s)$ pourra être l'ensemble des solutions qu'il est possible d'obtenir en faisant subir une mutation à la solution s, ainsi qu'on l'a défini dans la section précédente. On pourra donc formuler l'algorithme générique des méthodes de descente comme suit :

Méthode de descente :

Donnée, initialisation :

s^* appartenant à S.

continue := vrai.

Itération à répéter tant que *continue* est vrai :

Générer un échantillon V de solutions de $N(s^*)$

Trouver s' dans V tel que $f(s') \leq f(s)$ pour tout s de V.

Si $f(s') < f(s^*)$ poser $s^* := s'$; sinon poser *continue* := faux.

Résultat : s^*

Il existe deux raisons de limiter la recherche d'une solution voisine de s^* à un sous-ensemble V de $N(s^*)$, mais remarquons au préalable que l'implantation la plus directe de cet algorithme est de prendre $V = N(s^*)$. La première raison est qu'il peut être très coûteux d'examiner toutes les solutions de $N(s^*)$; on peut, par exemple, commencer l'examen de $N(s^*)$ et choisir pour s' la première solution rencontrée meilleure que s^*. La seconde raison est qu'il est possible, en modifiant uniquement la condition d'arrêt et la définition de V, d'obtenir au choix l'algorithme générique d'un recuit simulé ou d'une recherche itérative dirigée.

Les méthodes d'amélioration produisent en général des solutions de qualité honorable en regard de leur temps d'exécution souvent rapide, bien que non borné polynomialement en théorie. Ainsi, nous avons observé dans [94], pour des problèmes d'affectation quadratique générés aléatoirement (distribution uniforme et non corrélée des distances et des flots), qu'une méthode de descente trouvera en moyenne des solutions à moins de 5% au-dessus de la meilleure solution connue pour des problèmes dont la taille dépasse 20,

mais qu'il existe des exemples de problèmes pour lesquels les solutions trouvées sont à plus de 40% au-dessus de l'optimum.

Pour un exemple de problème de chaîne de traitement à 50 objets et 20 machines, nous avons observé qu'en partant de 50'000 solutions initiales différentes tirées aléatoirement, nous sommes arrivés à 50'000 optima locaux différents après avoir appliqué une méthode de descente ; la valeur de ces derniers était au moins 2% au-dessus de la valeur d'une solution optimale. Naturellement, le temps d'exécution et la qualité des solutions produites dépendent énormément du choix de $N(s)$.

2.4. RECUIT SIMULÉ

Les méthodes de descente ne produisent pas de très bonnes solutions car on choisit en général un voisinage $N(s)$ de taille relativement modeste, ce qui implique que le premier optimum local dans lequel la méthode s'arrête, c'est-à-dire le premier s^* tel qu'il n'existe pas de s dans V avec $f(s^*) > f(s)$, est très rarement d'excellente qualité. Cette constatation dépend naturellement du problème traité, mais dans la plupart des problèmes d'optimisation combinatoire, il existe un très grand nombre d'optima locaux (et qui sont souvent de piètre qualité).

L'idée à la base du recuit simulé est d'autoriser le choix d'une solution dont la qualité décroît. Ce choix se fera aléatoirement à l'image du processus de recuit d'un matériau. Pour diminuer les tensions à l'intérieur d'un solide, on procède comme suit : tout d'abord on fait fondre le matériau, puis on le refroidit le plus lentement possible. S'il reste suffisamment longtemps à une température T, la probabilité qu'il soit dans un état d'énergie E est donnée par une distribution de Boltzman $c(T) \cdot e^{\frac{-E}{kT}}$ où $c(T)$ est un facteur de normalisation et k la constante de Boltzman. Ainsi, en refroidissant très lentement le liquide, on obtiendra un solide qui aura une énergie interne minimale. Pour simuler ce processus, Metropolis et al. [69] ont proposé une méthode de Monte Carlo qui génère une séquence d'états de la manière suivante : étant donné un état initial des particules du matériau, on considère un petit déplacement aléatoire d'une particule choisie

aléatoirement. Ce déplacement modifie de E l'énergie du système. Si $E < 0$ (amélioration), [69] proposent d'effectuer le déplacement, sinon, on ne l'effectue qu'avec une probabilité donnée par $e^{\frac{-\Delta E}{kT}}$.

Il a fallu plus de trente ans pour qu'on se rende compte que l'on pouvait adapter ceci à des problèmes d'optimisation combinatoire en remplaçant simplement la fonction-énergie par la fonction-objectif. C'est ce qu'ont fait, indépendamment, Černy [24] et Kirkpatrick et al. [60]. L'algorithme générique du recuit simulé diffère des méthodes de descente uniquement par le fait qu'il accepte avec une probabilité non nulle de se diriger vers une solution plus mauvaise. Il s'exprime de manière générique comme suit :

Recuit simulé :

Données, initialisation :

Une suite $T_0, T_1, T_2, \ldots$ de nombres réels positifs.

s_0 appartenant à S.

$k := 0$

Itération à répéter tant qu'un critère d'arrêt n'est pas rempli :

Choisir aléatoirement, uniformément, s' dans $N(s_k)$.

Si $f(s') \leq f(s_k)$, poser $s_{k+1} := s'$;

sinon, poser $s_{k+1} := s'$ avec probabilité $e^{\frac{f(s_k) - f(s')}{T_k}}$

ou $s_{k+1} := s_k$ avec la probabilité complémentaire.

Incrémenter k d'une unité.

Résultat : s_k

Une optimisation de cet algorithme consiste à prendre comme résultat le meilleur des s_i ($i = 0, \ldots, k$) au lieu de s_k. Nous voyons que, relativement aux méthodes de descente, l'échantillon V de solutions élues de $N(s_k)$ est réduit à un seul élément. Les températures $T_0, T_1, T_2, \ldots$ sont souvent choisies lorsqu'on effectue les itérations, et non a priori comme indiqué ci-dessus. On choisit généralement une suite de températures non croissantes ; une règle abondamment utilisée est de poser $T_{k+1} := \alpha T_k$, avec $0 < \alpha < 1$, α étant très proche de l'unité. La méthode du recuit simulé présente un avantage théorique : Hajek [50], puis dans une généralisation Faigle et Schrader [34], ont montré que, sous des

conditions irréalisables en pratique, la méthode converge presque sûrement vers une solution optimale du problème traité.

Ce type de méthode itérative produit en général de bien meilleures solutions que les méthodes de descente, mais au prix d'un temps de calcul beaucoup plus élevé.

2.5. ACCEPTATION À SEUIL

Les méthodes d'acceptation à seuil dérivent directement des méthodes de recuit simulé. Certains auteurs, en particulier Dueck et Scheuer [31], ont remarqué que la partie d'un recuit simulé qui pouvait être la plus coûteuse était de décider si une solution plus mauvaise devait être acceptée. L'idée des méthodes d'acceptation à seuil est d'accepter toutes les solutions qui dégradent la solution courante de moins d'un seuil donné pour chaque itération. On arrive donc à la formulation générique suivante :

Acceptation à seuil :

Données, initialisations :

Une suite S_0, S_1, S_2, … de nombres réels positifs.

s_0 appartenant à S.

$k := 0$

$s^* := s_0$

Itération à répéter tant qu'un critère d'arrêt n'est pas rempli :

Choisir aléatoirement, uniformément, s' dans $N(s_k)$.

Si $f(s') < f(s_k) + S_k$, poser $s_{k+1} := s'$; sinon poser $s_{k+1} := s_k$.

Si $f(s^*) > f(s_{k+1})$, poser $s^* := s_{k+1}$.

Incrémenter k d'une unité.

Résultat : s^*.

Comme pour le recuit simulé, la suite des seuils est généralement choisie non croissante, mais il est tout à fait envisageable d'élever la valeur d'un seuil, en particulier si l'on se trouve bloqué dans un minimum local.

Pour être complet dans cette énumération des méthodes itératives les plus efficaces et les plus couramment utilisées, nous devrions encore parler des méthodes itératives dirigées. Comme leur discussion est au cœur de ce travail et que nous présenterons notre point de vue sur ce sujet en faisant fi d'une multitude de stratégies proposées par Glover dans [45] et [46], certainement pertinentes mais qui n'ont pas encore prouvé leur efficacité en pratique ou dont nous n'avons pas envisagé l'implantation, nous ferons la présentation des méthodes itératives dirigées dans un chapitre séparé.

3. RECHERCHES ITÉRATIVES DIRIGÉES

Ce qu'il manque indéniablement aux méthodes itératives présentées ci-dessus, c'est un zeste d'intelligence. En effet, grande est la tentation d'inculquer à une recherche itérative une portion de notre bon sens, fût-elle infime, afin qu'elle ne soit pas dirigée que par le hasard et l'obnubilation d'une fonction-objectif à minimiser. Mettre au point une recherche itérative dirigée pose un double défi : premièrement, comme dans toute recherche itérative, il faut que le moteur de la recherche, c'est-à-dire le mécanisme d'évaluation de solutions voisines, soit efficace ; secondement, il s'agit de transmettre à la recherche des bribes de connaissance du problème traité, de manière à ce qu'elle ne visite pas l'espace des solutions au petit bonheur mais au contraire qu'elle soit dirigée dans cet espace au moyen de principes, qui peuvent être appelés, peut-être pompeusement, intelligents.

Un principe de base, émis pour la première fois par Glover [44], est de constituer un historique de la recherche itérative ou, ce qui est équivalent, de doter la recherche de mémoire. Dans ce chapitre, nous étudierons comment transposer ce concept de mémoire dans les cas pratiques que nous nous sommes proposé de traiter afin d'obtenir un moteur efficace, concept qu'on s'efforcera de généraliser pour pouvoir l'appliquer à un ensemble de problèmes aussi vaste que possible. Nous ne présenterons pas, dans leur foisonnement baroque, tous les concepts émis dans [45] et [46], ni ne reprendrons le nom de baptême donné par Glover à ce que nous nommons « recherche itérative dirigée », car ce nom ne fait référence qu'à l'interdiction de choisir certaines solutions du voisinage et nous pensons que ce mécanisme est à la fois insuffisant pour bien diriger la recherche, comme

nous le montrerons dans la deuxième partie de ce chapitre, et pourrait même s'avérer inutile pour certaines applications. Le lecteur intéressé par les méthodes itératives dirigées ne manquera pas d'étudier les articles écrits à ce sujet par Glover [45] et [46], et pourra aussi trouver des introductions plus succinctes dans Glover et al. [47], Hertz et al. [55] ou Hertz et de Werra [56].

La première partie de ce chapitre présentera un certain nombre de principes permettant de guider une recherche itérative. Parmi l'ensemble des principes proposés par divers auteurs, nous avons sélectionné ceux qui nous paraissaient les plus efficaces pour diriger une recherche. En particulier, nous insisterons sur une manière simple et efficace de mettre en œuvre les concepts de mémoire à court et à long terme, et nous analyserons le rôle de ces implantations de mémoires en étudiant leurs effets lorsqu'on fait varier la valeur des paramètres leur étant associés. En effet, une croyance (mystique ?) indiquait que 7 était la valeur qu'il fallait donner à un paramètre afin d'obtenir les meilleurs résultats possibles ; nous verrons que cette croyance est sans fondement et que d'autres valeurs peuvent donner de meilleurs résultats, en particulier, nous proposerons de tirer aléatoirement la valeur de certains paramètres et nous montrerons que cette politique semble plus robuste que le choix d'une valeur a priori.

La deuxième partie de ce chapitre rappellera d'abord succinctement un résultat de Faigle et Kern, [33], qui fait état de la convergence d'une recherche itérative dirigée probabiliste ; ensuite, nous montrerons qu'un mécanisme de mémoire à court terme, basé sur l'interdiction de visiter certaines solutions, et qui a été très largement utilisé en raison de sa grande efficacité pratique, s'avère insuffisant pour empêcher que la recherche ne visite cycliquement un ensemble restreint de solutions.

Nous proposerons, dans la troisième partie de ce chapitre, des méthodes originales de mise en œuvre de recherches itératives dirigées pour les problèmes exposés au chapitre 1.

3.1. NOTIONS DE BASE

Dans cette section, nous allons tout d'abord présenter des notations et des techniques qui ne sont pas spécifiques aux recherches itératives dirigées, mais qui jouent un rôle essentiel dans celles-ci : il s'agit de la modélisation du problème et de l'évaluation du voisinage. Ensuite, nous aborderons la partie propre aux recherches itératives dirigées, c'est-à-dire les moyens de diriger la recherche par l'usage de mémoires.

3.1.1. Mouvements

À la base de la plupart des recherches itératives, il y a la définition de l'ensemble $N(s)$ des solutions voisines de s, mais on aura intérêt à considérer, plutôt que l'ensemble $N(s)$, l'ensemble des modifications que l'on peut apporter à s. Nous appellerons « mouvement » une modification apportée à une solution. Ainsi, la mutation d'une solution du problème de la chaîne de traitement (voir figure 7, page 25) peut être considérée comme un mouvement, caractérisé par l'ancienne et la nouvelle place de l'élément de la permutation qui a été déplacé.

L'ensemble $N(s)$ des solutions voisines de la solution s s'exprimera comme l'ensemble des solutions admissibles que l'on peut obtenir en appliquant à la solution s un mouvement m appartenant à un ensemble de mouvements M. L'application de m à s sera noté $s \oplus m$ et on aura l'équivalence $N(s) \equiv \{ s \oplus m \,|\, m \in M, s \oplus m \in S \}$. L'avantage d'exprimer le voisinage en termes de mouvements est qu'on peut souvent caractériser l'ensemble M plus facilement. Ainsi, dans l'exemple d'une mutation d'une permutation, M sera caractérisé par l'ensemble des couples (ancienne place, nouvelle place), indépendamment de la solution courante. Dans certaines applications cependant, cette simplification peut aboutir à la définition de mouvements qui mèneraient à des solutions non admissibles et en toute généralité, on a $|N(s)| \leq |M|$, sans pour autant que $|M|$ soit beaucoup plus grand que $|N(s)|$. Pour une solution s peu contrainte on a typiquement $|N(s)| = |M|$.

Ceci nous amène à discuter de l'ensemble de définition S des solutions admissibles : en effet, il peut arriver que cet ensemble soit très « déchiqueté » c'est-à-dire que, sans la

définition d'un voisinage potentiellement très large, il ne soit pas possible de générer toutes les solutions admissibles ou plus précisément, qu'il ne soit pas possible d'arriver à une solution optimale en partant d'une solution quelconque. Dans ce cas, pour éviter que pour certaines solutions le voisinage soit gigantesque (et donc qu'une itération nécessite un temps de calcul prohibitif), alors que ce voisinage est réduit pour des solutions avec une majorité de contraintes saturées, on étend le domaine de définition des solutions admissibles tout en pénalisant les solutions violant des contraintes du problème initial. On modifiera donc le problème ainsi :

$$\min_{s \in S^{étendu}} (f(s) + p(s)) \qquad (4)$$

où $S \subset S^{étendu}$, $p(s) = 0$ pour $s \in S$, et $p(s) > 0$ si $s \notin S$. Costa et Hertz ont notamment proposé et fait usage dans [23], [52] et [53] de telles techniques de pénalisation dans des adaptations de recherches itératives dirigées pour la confection d'horaires scolaires où la diversité des contraintes à prendre en considération est impressionnante.

Nous remarquons ici que la modélisation du problème est loin d'être triviale et que le choix de la fonction à minimiser et de la fonction de pénalité pourra être délicat. En particulier, ces fonctions devront prendre un nombre de valeurs différentes aussi grand que possible sur leur domaine de définition pour que la recherche puisse être efficacement dirigée : comment pourrait-on en effet décider quel mouvement choisir lorsqu'il existe un grand nombre de solutions voisines à coût identique ? Pour y remédier, on choisira par exemple, pour la fonction de pénalité, un indicateur de l'importance des violations des contraintes plutôt qu'un compteur du nombre de contraintes violées.

3.1.2. Évaluation du voisinage

Pour que le moteur de la recherche soit efficace, il faut que le rapport entre la qualité ou l'opportunité des mouvements et le temps de calcul nécessaire à leur évaluation soit aussi élevé que possible. Si la qualité d'un type de mouvement ne peut être justifiée que par le bon sens et de manière empirique, l'évaluation du voisinage peut en revanche souvent être

considérablement accélérée et justifiée par des considérations algébriques : si l'on définit $\Delta(s, m) = f(s \oplus m) - f(s)$, on arrive dans bien des cas à simplifier l'expression $f(s \oplus m) - f(s)$ et évaluer ainsi rapidement la quantité $\Delta(s, m)$. Comme le mouvement m n'apporte qu'une modification locale de la solution, il est souvent possible d'évaluer $\Delta(s \oplus m, m')$ en fonction de $\Delta(s, m)$ et d'arriver à un examen très rapide de l'intégralité du voisinage en mémorisant les valeurs de $\Delta(s, m')$.

Il se peut que l'évaluation de $\Delta(s, m)$ soit très difficile et coûteuse à faire. Prenons l'exemple du problème de distribution de biens : une solution s peut être donnée comme une partition des biens à livrer en sous-ensembles dont les poids ne sont pas supérieurs à la capacité des véhicules. Calculer $f(s)$ suppose, pour chacun de ces sous-ensembles, de trouver un ordre optimal dans lequel on va délivrer les biens, ce qui est un problème difficile en soi ; donc, le calcul de $f(s)$, et par conséquent celui de $\Delta(s, m)$ ne peuvent être raisonnablement imaginés pour tout mouvement éligible (c'est-à-dire appartenant à M) ; il pourrait éventuellement l'être pour chaque mouvement élu (effectivement réalisé) mais, en pratique, on devra se contenter de calculer la vraie valeur de $f(s)$ pour un nombre très restreint de solutions. Dans le cas du problème de la chaîne de traitement à gamme opératoire variable, nous verrons que nous proposons une évaluation approximative de $\Delta(s, m)$, mais qui peut se faire en temps constant, plutôt qu'un calcul exact qui prendrait un temps proportionnel à la taille du problème.

En toute généralité, une recherche itérative ne considère pas forcément à chaque itération l'ensemble des solutions de $N(s)$ mais seulement un sous-ensemble que l'on a noté V car une manière d'accélérer l'examen du voisinage est de réduire sa taille ; nous verrons dans le paragraphe suivant que cette réduction pourra aussi avoir pour but de diriger la recherche.

Pour réduire le nombre de solutions éligibles de $N(s)$, une solution adoptée par certains auteurs est de tirer aléatoirement dans $N(s)$ un nombre de solutions bien plus petit que $|N(s)|$. Une autre idée, lorsqu'on a un voisinage donné par un ensemble statique M de mouvements, est de considérer une partition de M en sous-ensembles ; à chaque itération, un seul de ces sous-ensembles sera examiné ; enfin, en faisant l'hypothèse qu'un

mouvement de bonne qualité restera bon lorsque la solution n'a pas beaucoup changé, on pourra classer les mouvements selon leur évaluation à une itération donnée et ne considérer pendant les quelques itérations suivantes que les mouvements classés parmi les meilleurs. Naturellement, ce classement se dégrade au cours de la recherche et il convient de le rétablir en faisant périodiquement une évaluation complète du voisinage.

3.1.3. Direction de la recherche

La première idée qui vient à l'esprit, lorsqu'on veut faire usage de mémoire dans un processus itératif, est de voir si une solution candidate a déjà été visitée. Cependant, cette idée peut être difficile à mettre en pratique et de surcroît s'avérer peu efficace, voire mauvaise. En effet, cela suppose de mémoriser chaque solution que l'on a visitée et de tester à chaque itération et pour chaque solution éligible si cette dernière a déjà été énumérée. Ceci peut éventuellement se faire de manière efficace en utilisant des tables de hachage, comme l'ont suggéré Woodruff et Zemel dans [101], mais on ne peut échapper à une croissance de la place mémoire qui augmente linéairement avec le nombre d'itérations effectuées.

L'interdiction pure et simple de solutions peut mener à des absurdités : supposons que l'ensemble des solutions admissibles puisse être représenté par les points à coordonnées entières délimités par une surface dans le plan et que l'on puisse se déplacer de n'importe quelle solution admissible à n'importe quelle autre en effectuant des déplacements d'une unité de longueur. Dans ce cas, on peut facilement trouver des trajectoires qui déconnectent la solution courante d'une solution optimale ou qui bloquent la recherche itérative faute de solutions voisines admissibles, à cause de l'interdiction de passer par une solution déjà visitée (voir figure 8).

Cependant, l'usage d'une mémoire exacte, sans pour autant interdire inconditionnellement le retour à une solution déjà visitée, peut s'avérer utile dans certains cas [6], mais cet usage reste lourd, tant au niveau de l'implantation que de la place nécessaire à la mémorisation des solutions.

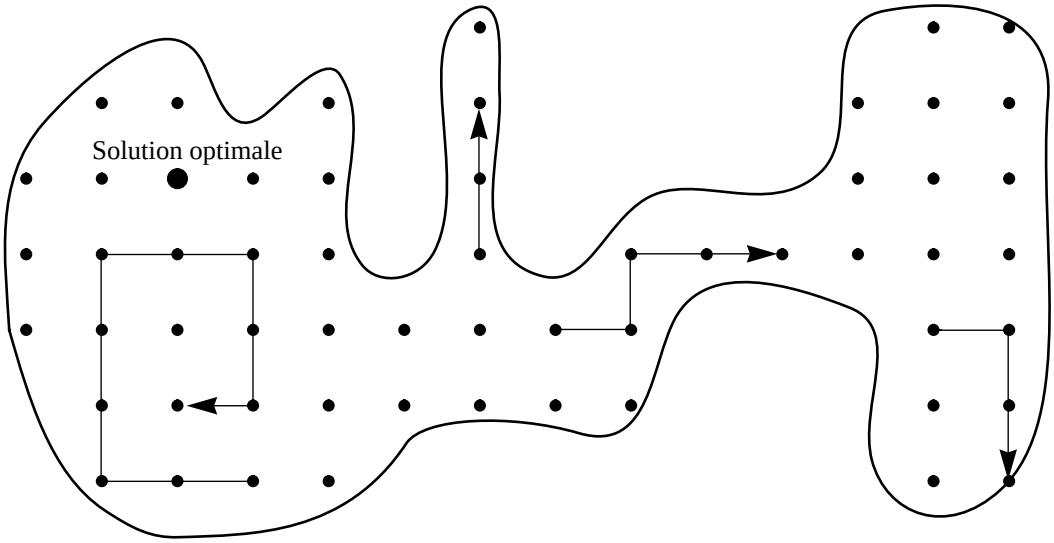

Figure 8 : Trajectoires bloquant la recherche ou la déconnectant de la solution optimale.

L'approche que nous nous proposons d'étudier, au contraire, se base sur des implantations beaucoup plus légères et faciles à mettre en œuvre, qui ont prouvé leur pertinence dans certaines applications, mais elles ne peuvent être justifiées au niveau théorique comme nous le verrons plus loin.

Comme il peut être inefficace de restreindre le voisinage $N(s)$ à des solutions non encore visitées, nous proposons de travailler plutôt au niveau de l'ensemble M des mouvements applicables à une solution. Cet ensemble est souvent de taille relativement modeste (typiquement de taille $O(n)$ ou $O(n^2)$ si n est la taille du problème) et doit posséder la caractéristique de pouvoir mener à une solution optimale en partant de n'importe quelle solution admissible. Dans un premier temps, pour simplifier notre propos, nous supposerons qu'il bénéficie encore de la propriété de réversibilité : il doit exister pour tout mouvement m applicable à une solution s un mouvement m^{-1} tel que $(s \oplus m) \oplus m^{-1} = s$. Comme il est stupide d'effectuer m^{-1} juste après avoir effectué m, on peut donc dans tous les cas limiter l'ensemble des mouvements applicables à $s \oplus m$ à ceux différents de m^{-1}. De plus, cela évite de visiter cycliquement s et $s \oplus m$ au cas où s serait un minimum local de la fonction relativement au voisinage choisi et où le meilleur

voisin de $s \oplus m$ serait précisément s. Ce type de méthode a été proposé pour la première fois par Hansen [51].

En généralisant cette technique de restriction du voisinage, c'est-à-dire en interdisant pendant plusieurs itérations d'effectuer l'inverse d'un mouvement qui vient d'être fait, on empêche d'autres cycles mettant en jeu un nombre plus important de solutions intermédiaires. On espère ainsi que la solution se soit suffisamment modifiée, lorsqu'on peut à nouveau effectuer l'inverse d'un mouvement, pour qu'il soit improbable — mais non impossible — de retourner à une solution déjà visitée. Si tel est le cas, on espère que l'ensemble des mouvements interdits a changé, donc que la trajectoire future de la recherche se modifiera. Le nombre de mouvements interdits reste assez restreint, car, si on suppose que M ne dépend pas de la solution courante, il est raisonnable de n'interdire qu'une fraction des mouvements de M. Ce type de mémoire sera donc une mémoire à court terme, portant typiquement sur quelques unités ou quelques dizaines d'itérations.

Pour la commodité du propos, nous avons supposé que nous mémorisions des mouvements inverses de ceux qui ont été effectués. Cependant, il n'est pas toujours possible ou évident de définir un mouvement inverse. Prenons l'exemple d'un problème où il s'agit de trouver une permutation optimale de n éléments (cela pourrait être le problème de la chaîne de traitement ou d'affectation quadratique). Un type de mouvement pouvant paraître raisonnable est de permuter les éléments i et j de la permutation ($1 \leq i < j \leq n$). Dans ce cas, l'ensemble M des mouvements applicables à une solution quelconque est donné par l'ensemble des couples (i, j). Mais, si l'on effectue par la suite le mouvement (i, k), l'interdiction de (i, j) nous empêchera de visiter certaines solutions sans pour autant prévenir des phénomènes de cyclage : en effet, les mouvements (i, j) (k, p) (i, p) (k, j) (k, i) (j, p) appliqués successivement ne modifient pas la solution. Ce n'est donc pas forcément le mouvement inverse qu'il faut éviter de faire trop rapidement, mais il faut plutôt interdire de reprendre certains attributs de ces mouvements. Dans l'exemple précédent, si l'on nomme $s(k)$ la place de l'élément k à une itération, ce n'est pas le mouvement (i, j) qu'il faut interdire lorsqu'on vient de l'effectuer, mais c'est, par exemple, de remettre simultanément l'élément $s(i)$ à la place i et l'élément $s(j)$ à la place j.

On se préservera ainsi au moins des cycles de longueur plus petite ou égale au nombre de mouvements interdits.

De manière générale, la mémoire à court terme interdira d'effectuer certains mouvements en stockant directement des mouvements ou indirectement des attributs de mouvement ou même des attributs de solution, voire des solutions dans certains cas. L'effet de cette mémoire, si l'on se représente le problème d'optimisation comme un paysage borné par un territoire qui définirait les solutions admissibles et où l'altitude correspondrait à la valeur de la fonction-objectif, est de visiter des vallées, (sans forcément se trouver toujours dans le fond de celle-ci du fait de l'interdiction de certains mouvements) et, au hasard de cols pas trop élevés, de passer dans une autre vallée.

Plus le nombre de mouvements interdits est élevé, plus on aura de chances de franchir des cols, mais moins on visitera de manière approfondie les vallées. Inversement, si les mouvements sont interdits pendant très peu d'itérations, les montagnes entourant les vallées auront peu de chances d'être franchies car il existera presque sûrement un mouvement autorisé qui mènera à une solution proche du fond de la vallée ; mais, en contrepartie, le fond de la première vallée visitée sera vraisemblablement trouvé.

Plus formellement, pour un nombre de mouvements interdits très petit, la recherche itérative aura tendance à visiter toujours les mêmes solutions. Si ce nombre augmente, la probabilité de rester prisonnier d'un ensemble très restreint de solutions diminue et, par conséquent, la probabilité de visiter plusieurs bonnes solutions augmente. Il ne faut cependant pas que le nombre de mouvements interdits soit trop grand car dans ce cas il devient très peu probable de trouver de bons optima locaux, faute de mouvements disponibles. La recherche est en quelque sorte dirigée par les rares mouvements autorisés plutôt que par la fonction-objectif. Notons au passage qu'un mouvement qui mène à une solution meilleure que toutes celles visitées par la recherche dans les itérations précédentes n'a aucune raison d'être interdit. Afin de ne pas manquer cette solution, on modifiera donc l'éventuel statut interdit de tels mouvements.

En figure 9, nous avons illustré ces phénomènes dans le cas du problème d'affectation quadratique : pour chacun des 3'000 problèmes de taille 12 tirés aléatoirement, nous

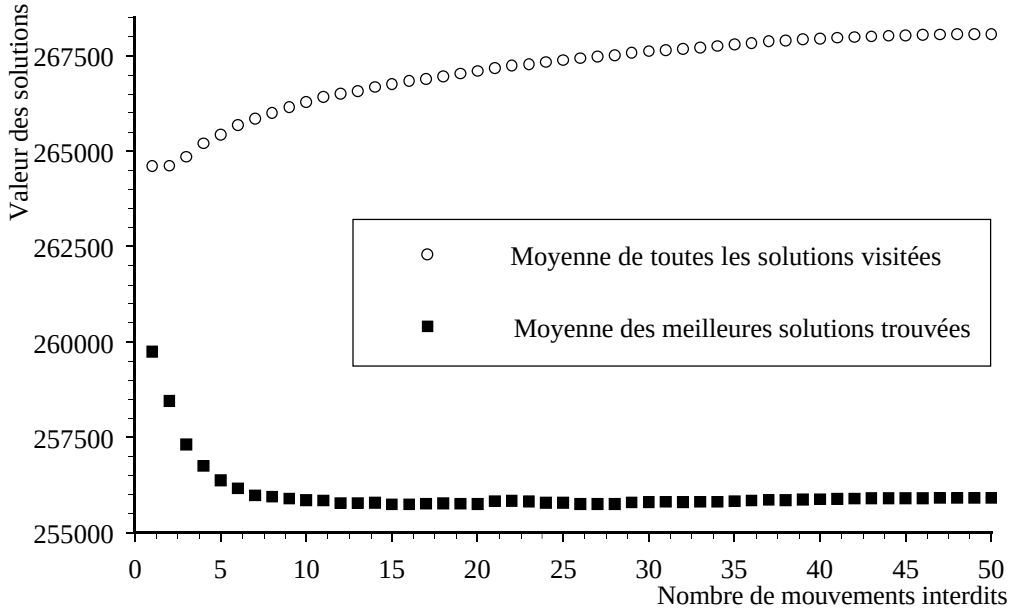

Figure 9 : Influence du nombre d'itérations pendant lesquelles on interdit les mouvements.

avons effectué 50 itérations d'une recherche itérative dirigée décrite en détail en section 3.3.3. Dans cette figure, nous donnons en fonction du nombre d'itérations durant lesquelles un mouvement inverse est interdit, les deux statistiques suivantes : premièrement la valeur moyenne de toutes les solutions visitées durant la recherche et secondement la valeur moyenne des meilleures solutions trouvées par chaque recherche. Nous remarquons que la première statistique croît avec le nombre de mouvements interdits, ce qui signifie que la qualité moyenne des solutions visitées baisse. Par contre, en ce qui concerne les meilleures solutions trouvées, leur qualité s'améliore avec l'augmentation du nombre de mouvements interdits, ce qui traduit le fait que l'on arrive de mieux en mieux à s'échapper d'optima locaux plus ou moins mauvais ; ensuite, leur qualité se détériore, mais avec une tendance beaucoup plus faiblement marquée. On en déduit que ce nombre doit être soigneusement choisi, en fonction du problème traité, de la taille du voisinage et du problème, du nombre total d'itérations effectuées, … Il est relativement aisé de déterminer l'ordre de grandeur à donner au nombre de mouvements interdits, mais la valeur optimale de ce nombre ne peut être trouvée autrement qu'en

essayant tous les nombres possibles. C'est pourquoi, pour bénéficier à la fois des avantages d'un petit nombre — pour visiter une vallée de manière approfondie — et d'un grand nombre de mouvements interdits — pour pouvoir s'échapper des vallées — on aura intérêt à modifier ce nombre au cours de la recherche. Plusieurs politiques peuvent être envisagées pour son choix : par exemple, il peut être tiré aléatoirement, à chaque itération ou après un certain nombre d'itérations, entre des bornes inférieures et supérieures, ces bornes étant souvent facilement déterminables ; il pourra aussi être augmenté ou diminué sur la base de critères récoltés durant la recherche, etc.

En reprenant l'exemple du problème d'affectation quadratique, nous avons représenté en figure 10 le nombre moyen d'itérations nécessaires à la résolution (obtention d'une

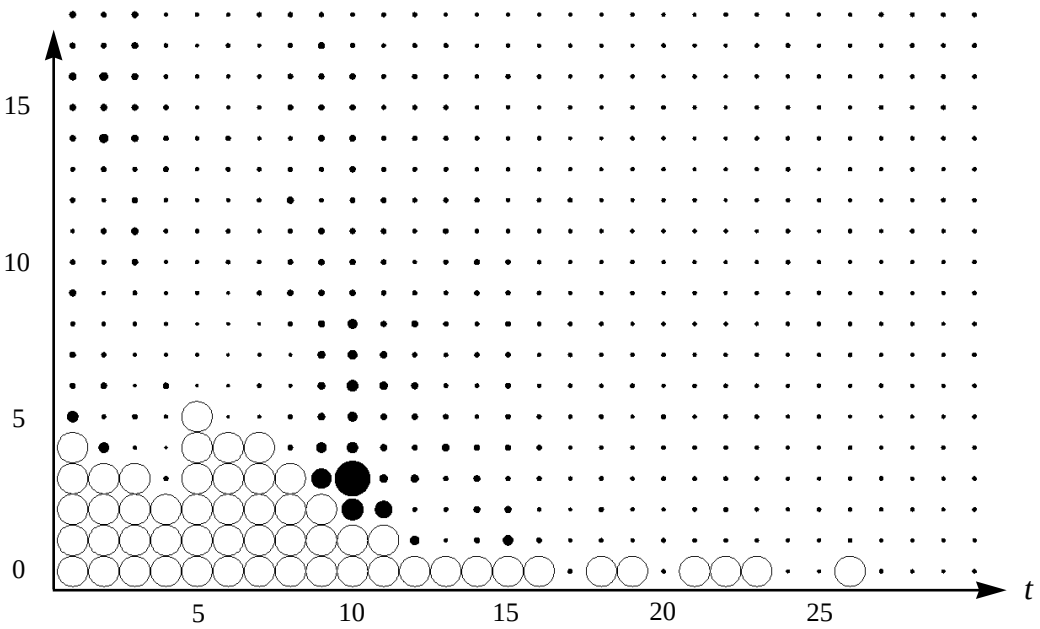

Figure 10 : Effet d'un tirage aléatoire du nombre d'itérations pendant lesquelles on interdit des mouvements (pour un problème particulier).

solution de valeur donnée) d'un problème de taille 12 généré aléatoirement lorsque la politique du nombre de mouvements interdits était de choisir ce nombre aléatoirement entre t et $t + \Delta$. Nous avons résolu ce problème 500 fois en partant de solutions initiales (permutations) aléatoires et nous avons placé dans cette figure pour chaque paire (t, Δ), un cercle dont le diamètre est proportionnel au nombre moyen d'itérations nécessaires à sa résolution. Un cercle vide indique qu'un cycle dans l'espace des solutions visitées est

apparu dans l'une au moins des résolutions. Nous remarquons que pour $\Delta = 0$, c'est-à-dire lorsque le nombre de mouvements interdits est fixe, des cycles peuvent apparaître même pour t relativement grand. Par contre, l'introduction d'un Δ positif, fût-il unitaire, permet de se prémunir beaucoup plus efficacement contre les cycles. Pour ce problème particulier et cet ensemble de résolutions, la meilleure paire (t, Δ) est $(5, 6)$, ce qui semble plaider en faveur de petites valeurs comme cela a été constaté en figure 9. Pour des raisons de robustesse, on préfère cependant choisir t plus grand, en espérant ne pas tomber sur une mystérieuse zone maudite comme $t = 10$, 11 ou 12 pour cet exemple.

Cette politique de choix aléatoire du nombre de mouvements interdits peut donc diriger la recherche automatiquement et de manière assez efficace vers de bonnes solutions. Notons qu'un tel mécanisme pourrait être qualifié de myope car il est dirigé principalement par la valeur de la fonction-objectif. Bien qu'il fournisse des résultats très encourageants vu sa simplicité, il ne peut être considéré comme un moyen intelligent de diriger la recherche mais doit plutôt être vu comme une base simple à implanter pour le moteur de la recherche.

D'autres mécanismes pour une mémoire à court terme ont été proposés dans la littérature : on peut citer Glover [45] qui propose de tirer aléatoirement le statut interdit/autorisé des mouvements, avec une probabilité évoluant en cours de recherche. Dans le cas où l'on étend le domaine des solutions admissibles tout en pénalisant les violations de contraintes, comme cela est suggéré dans l'équation (4), Gendreau et al. [42] ont imaginé de modifier les pénalités en cours de recherche : si, pendant les dernières itérations, une contrainte a été systématiquement violée, on augmente la pénalité qui lui est associée ; si au contraire elle n'a jamais été violée, on peut diminuer son poids. Cette technique est particulièrement bien adaptée au cas où l'on ne relaxe qu'une contrainte, car si plusieurs sont relaxées, on court le risque de ne visiter que des solutions non admissibles car les différents poids associés aux contraintes pourraient varier en opposition de phase, de sorte que les contraintes associées à des poids momentanément faibles seraient toujours violées.

3.1.4. Direction de la recherche à long terme

Dans le cas d'un voisinage formé par un ensemble statique de mouvements, c'est-à-dire lorsque cet ensemble ne dépend pas de la solution dans laquelle on se trouve, une statistique sur les mouvements choisis au cours de la recherche peut être d'une grande utilité : si des mouvements sont élus beaucoup plus fréquemment que d'autres, il y a lieu de supposer que la recherche a des difficultés à explorer des solutions de compositions variées, et qu'elle peut rester bloquée dans une « vallée ». On observe souvent dans la pratique des problèmes avec des vallées étendues, qui peuvent par conséquent être visitées en utilisant des mouvements de petite amplitude au niveau des différences en valeur absolue de la fonction-objectif ; si l'on utilise uniquement le mécanisme d'interdiction de mouvements inverses de ceux récemment effectués pour diriger la recherche, un nombre de mouvements interdits trop faible ne permet pas de s'échapper de certaines vallées ; nous avons vu qu'augmenter ce nombre de mouvements interdits a pour effet de forcer la recherche à rester souvent à flanc de coteau et même si la recherche change de vallée, elle peut ne pas réussir à trouver de bonnes solutions dans la nouvelle vallée à cause des mouvements qui lui sont interdits suite à la visite de la vallée précédente.

Pour pouvoir assurer une certaine diversité dans la recherche tout en n'interdisant pas trop de mouvements, nous proposons de pénaliser les mouvements fréquemment utilisés. On peut imaginer plusieurs politiques de pénalisation, comme par exemple l'interdiction d'effectuer les mouvements dont la fréquence d'occurrence dans la recherche dépasse un seuil donné, ou l'ajout d'une valeur proportionnelle à leur fréquence d'utilisation lors de l'évaluation des mouvements, ou encore l'obligation d'effectuer des mouvements non choisis pendant un grand nombre d'itérations. L'ajout d'une pénalité proportionnelle à la fréquence aura de plus un effet bénéfique pour les problèmes où la fonction-objectif ne prend qu'un petit nombre de valeurs, ce qui peut engendrer des équivalences gênantes pour la direction de la recherche entre les diverses évaluations des solutions voisines. En effet, la recherche aura alors tendance à choisir les mouvements les moins utilisés plutôt que d'élire un mouvement plus ou moins aléatoirement.

En figure 11, nous illustrons l'effet de la pénalisation des mouvements en ajoutant un facteur proportionnel à leur fréquence d'utilisation lors de leur évaluation. Pour cela, nous avons répété l'expérience que nous avons faite pour montrer l'influence du nombre

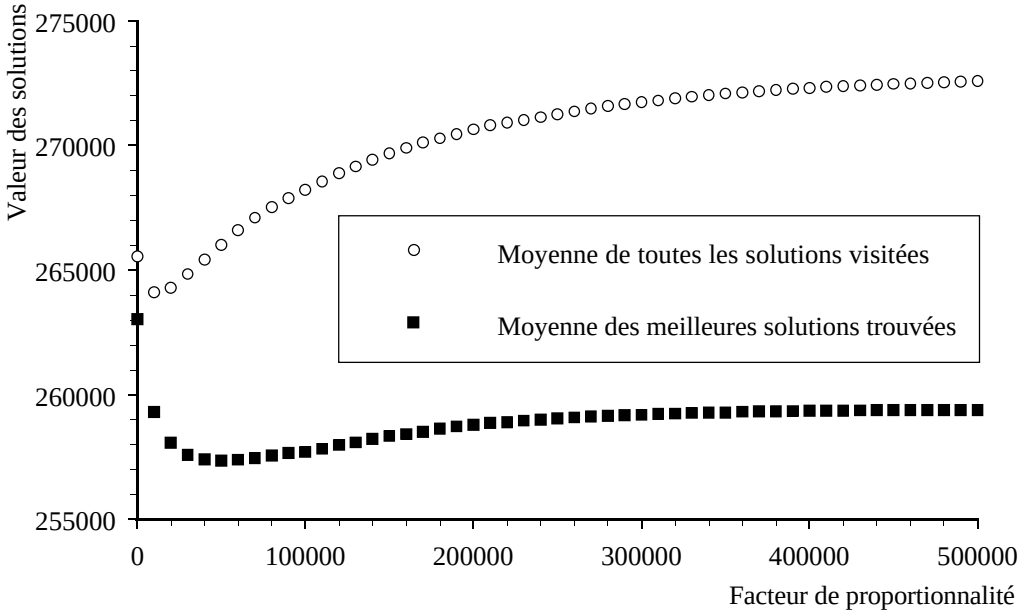

Figure 11 : Effet du coefficient de pénalisation sur les fréquences.

d'itérations pendant lesquelles on interdisait de faire le mouvement inverse d'un mouvement effectué (voir figure 9), mais cette fois en faisant varier uniquement le coefficient de pénalisation ; les mouvements sont donc pénalisés mais jamais interdits. Cette expérience porte à nouveau sur les 3'000 problèmes d'affectation quadratique de taille 12 générés aléatoirement. Nous avons représenté en figure 11 la moyenne des meilleures solutions trouvées après 50 itérations et la valeur moyenne de toutes les solutions visitées en fonction du coefficient de pénalisation. Nous remarquons que le comportement de ces deux statistiques est sensiblement le même que celui de la figure 9, mais que globalement les solutions générées sont moins bonnes que celles obtenues par l'usage d'une mémoire à court terme.

De manière analogue à ce qui se fait pour la mémoire à court terme, on peut généraliser cette mémoire à long terme pour des ensembles de mouvements non statiques, c'est-à-dire

où M dépend de s : on enregistre alors la fréquence à laquelle on a utilisé certaines caractéristiques de mouvements plutôt que les mouvements eux-mêmes. Nous noterons ici la similarité d'implantation de ces deux mémoires : l'une stocke l'itération à laquelle on peut à nouveau utiliser une caractéristique de mouvement alors que l'autre mémorise le nombre de fois que cette caractéristique a été utilisée dans les mouvements élus.

L'effet de la pénalisation sur les fréquences porte des fruits lors de longues recherches et peut donc être considéré comme une mémoire à long terme. Pour obtenir des implantations efficaces de recherches itératives dirigées, cette mémoire à long terme doit être utilisée en collaboration avec une mémoire à court terme. Cette collaboration peut être soit constante, en choisissant une fois pour toutes le nombre de mouvements interdits et le coefficient de pénalisation, soit variable, en alternant des phases de recherche où la mémoire à long terme a un rôle prépondérant et un rôle restreint. Ces phases auront pour but d'intensifier la recherche (lorsque le nombre de mouvements interdits est réduit et/ou la mémoire à long terme évoquée pour choisir des solutions aux caractéristiques proches des meilleures solutions énumérées par la recherche), ou de diversifier la recherche (lorsque le nombre de mouvements interdits est important et/ou la mémoire à long terme utilisée pour favoriser des solutions ou des mouvements aux caractéristiques rarement rencontrées).

Nous avons présenté ici certaines bases des recherches itératives dirigées. Nul ne doute que d'autres principes permettront à l'avenir d'aboutir à des méthodes plus efficaces et intelligentes, mais pour l'heure, n'oublions pas qu'améliorer une recherche itérative nécessite de son auteur qu'il détermine ce qu'elle fait d'idiot. Lorsque c'est possible, représenter graphiquement les solutions successivement visitées par la recherche stimulera efficacement l'esprit de l'implanteur et lui dictera, souvent de manière évidente, comment diriger sa recherche plus intelligemment. En effet, nous pensons que mettre au point une recherche itérative dirigée est un processus itératif : dans l'état actuel de nos connaissances, il nous semble très improbable de pouvoir élaborer une recherche itérative efficace du premier coup ; des ajustements, dépendant à la fois du type et de l'exemple de problème traité, devront très probablement avoir lieu ; nous avons uniquement décrit dans

cette section des principes qui devraient permettre à l'implanteur de se diriger vers une méthode efficace plus rapidement.

Nous verrons au travers des problèmes que nous nous sommes proposé de traiter comment ces concepts peuvent être mis en œuvre, mais auparavant, nous nous permettons quelques remarques sur la convergence des méthodes itératives dirigées.

3.2. REMARQUES SUR LA CONVERGENCE DES MÉTHODES ITÉRATIVES DIRIGÉES

3.2.1. Théorème de convergence

Faigle et Kern ont montré dans [33] qu'il était possible, à l'intérieur d'un recuit simulé, de modifier les probabilités de génération des solutions voisines et de modifier la valeur de la fonction-objectif pour certaines solutions, de manière à diriger le processus itératif tout en conservant la propriété de convergence globale du recuit simulé. Nous n'allons pas rappeler formellement les théorèmes de convergence contenus dans [33], mais nous donnerons ici leurs conséquences pratiques, c'est-à-dire comment on peut faire usage de mémoire pour diriger un recuit simulé.

Une première idée que l'on trouve dans [33] est de diminuer la probabilité de générer une solution avec certaines caractéristiques si l'on remarque, au cours de la recherche, que celles-ci ont rarement été rencontrées dans les bonnes solutions visitées au cours des itérations précédentes. Ce mécanisme peut donc être considéré comme une intensification de la recherche. Pour la diversifier, une autre idée est de combler progressivement les vallées dans lesquelles on se trouve en élevant la valeur de la fonction-objectif pour les solutions qui sont des optima locaux (mais non globaux). On modifiera ainsi la probabilité d'acceptation de telles solutions, ce qui est très important à basse température, car il devient très peu probable de s'en échapper.

Remarquons que ces théorèmes de convergence disent que la recherche itérative a une probabilité tendant vers 1 de se trouver dans une solution optimale après un temps infini. À notre avis, ces théorèmes sont trop forts : une recherche itérative n'a pas besoin de

converger vers une solution optimale pour être efficace. Au contraire, elle devrait visiter un ensemble de solutions d'autant plus grand qu'elle effectue un nombre élevé d'itérations, car il est important de passer *une fois* par une solution optimale. Aussi, l'idée de dégrader la qualité d'une solution déjà visitée nous séduit-elle et il devrait être possible par ce moyen de déduire un théorème qui établirait qu'une recherche infiniment itérative et déterministe passe au moins une fois par un optimum global (ou même par toutes les solutions admissibles du problème). Une première approche pourrait être d'ajouter à la valeur de la fonction-objectif initiale d'une solution une valeur proportionnelle au nombre de fois que cette solution a été visitée. Naturellement, ce type de pénalisation pourrait être affiné, mais il reste inapplicable en pratique car en plus d'un temps de calcul infini, un tel processus requerrait une mémoire de taille énorme et nous pensons que le chemin sera encore long avant d'arriver à une recherche itérative qui garantisse de trouver un optimum global en un temps borné par une fraction de la taille de l'ensemble des solutions admissibles.

3.2.2. Efficacité pratique — inefficacité théorique

Ce qui méritait d'être fait mais qui ne l'a jamais été, c'est de montrer que la recherche itérative dirigée dans sa forme la plus simple proposée par F. Glover dans [45] pouvait se révéler à la fois efficace en pratique et ne jamais passer par un optimum global pour certains problèmes mal conditionnés. Pour que ceci ait un sens, il est important de travailler avec une méthode efficace en pratique car il est inintéressant de démontrer qu'une méthode qui ne fonctionne pas peut ne jamais passer par un optimum global.

Tout d'abord, nous allons définir une recherche itérative dirigée élémentaire pour le problème de distribution de biens qui, bien que moins efficace que celle proposée dans le paragraphe 3.3.5, permet d'obtenir, si son unique paramètre est bien choisi, la meilleure solution connue d'un problème posé par Christofides et al. dans [18] et non encore résolu car de taille trop grande pour les méthodes exactes. Ensuite, nous allons montrer que cette recherche itérative, quelle que soit la valeur de son unique paramètre, ne passe jamais par l'optimum global d'un petit problème mal conditionné.

Pour définir une recherche itérative, la première chose à faire est de choisir une modélisation du problème et les mouvements applicables à une solution. Dans le cas du problème de distribution de biens, nous proposons d'autoriser des solutions où tous les biens ne sont pas livrés, mais en pénalisant de $2d$ un bien non livré qui devrait l'être à une distance d du dépôt (comme si l'on devait engager un véhicule uniquement pour le livrer). Le nombre maximum de véhicules engagés est limité par une valeur m (qui est aisément déterminable). Un mouvement consiste à enlever un bien i ($i = 1, \ldots n$) actuellement desservi par la tournée T_j ($j = 0, \ldots m$, la tournée T_0 représentant les biens non desservis) et à le livrer par la tournée T_k ($k = 0 \ldots m,\ k \neq j$). Lorsqu'on retire un bien i d'une tournée, on passera directement du lieu de livraison du bien précédant i à celui suivant i sur la tournée. On ajoute un bien sur une tournée là où le détour occasionné est le plus petit. Ensuite, nous devons définir comment la recherche est dirigée. Pour cela, nous avons choisi d'interdire pendant t itérations d'effectuer le mouvement inverse de celui qu'on vient de réaliser, c'est-à-dire, si l'on a enlevé le bien i de la tournée T_j pour l'insérer dans la tournée T_k, il sera interdit pendant t itérations de remettre i dans la tournée T_j (à moins que cela mène à une meilleure solution que la meilleure trouvée jusqu'ici). Comme solution initiale, nous avons choisi de ne rien livrer. Pour plus de précisions, le lecteur pourra se référer, notamment sur des détails d'implantation, à la méthode, plus élaborée il est vrai, décrite au paragraphe 3.3.5.

Malgré sa simplicité, cette procédure trouve assez rapidement des solutions de bonne qualité pour des problèmes classiques de la littérature lorsque le paramètre t est bien choisi. Par exemple, elle permet de trouver la meilleure solution connue d'un problème de taille 50 proposé par Christofides et al. dans [18]. Le petit exemple de problème mal conditionné est le suivant : on considère au maximum 2 véhicules de capacité 24 et 11 biens de volumes respectifs 3, 8, 8, 8, 3, 3, 3, 3, 3, 3, 3 à livrer en un seul lieu éloigné de d du dépôt. La solution optimale de ce problème consiste à livrer tous les biens de volume 3 par un véhicule et ceux de volume 8 par l'autre. On observe que la recherche itérative dirigée décrite ci-dessus, pour toute valeur du paramètre t, soit se retrouve périodiquement dans le même état (même solution courante et mêmes mouvements interdits pour un

nombre d'itérations identique), soit arrive à une solution où aucun mouvement autorisé n'existe, sans jamais passer par une solution optimale. En figure 12, nous avons

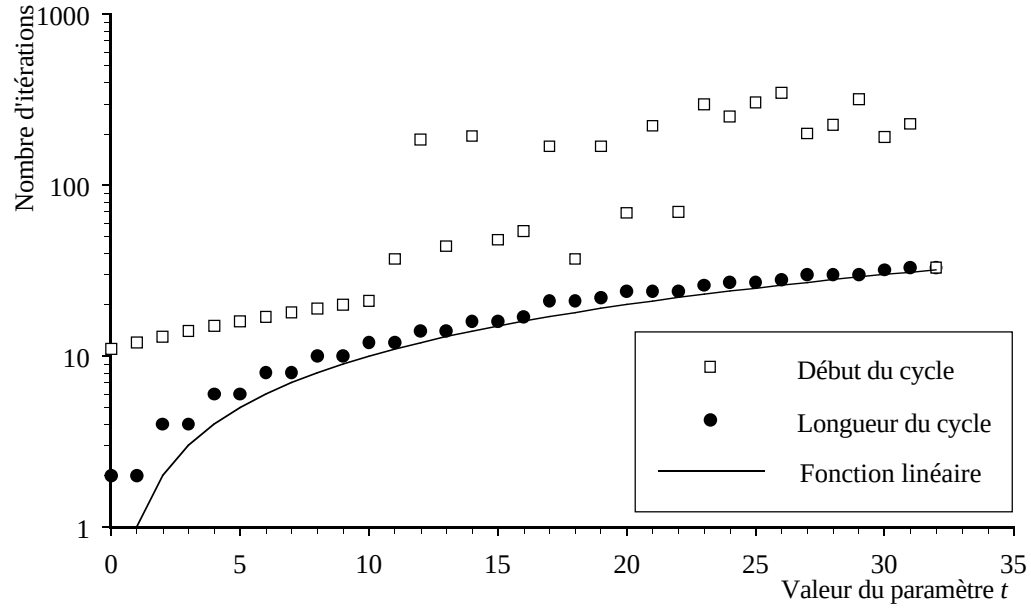

Figure 12 : Apparition de cycles en fonction du paramètre *t*, pour un problème mal conditionné.

représenté, en fonction du paramètre *t*, l'itération à partir de laquelle on entre dans un cycle et la longueur de ce dernier, c'est-à-dire le nombre de solutions intermédiaires que cette recherche itérative visite avant de se retrouver dans un état donné. Le nombre de solutions différentes, $|S^{étendu}|$, vaut 258 pour ce problème. Nous remarquons que parfois on peut visiter un nombre de solutions plus grand que $|S^{étendu}|$ lui-même avant de pénétrer dans un cycle. Cela explique peut-être pourquoi cette recherche se comporte relativement bien pour des problèmes mieux conditionnés. Nous voyons sur cette figure que la longueur des cycles est toujours légèrement plus grande que la valeur du paramètre *t*. Pour des valeurs de ce dernier dépassant 32, la recherche se bloque dans une solution, faute de solutions admissibles autorisées dans son voisinage.

3.3. APPLICATIONS

On trouvera dans cette section des applications originales de recherches itératives dirigées pour les problèmes que nous nous sommes proposé de traiter. Pour chacune de ces applications, nous commencerons par donner notre modélisation du problème, c'est-à-dire l'ensemble, éventuellement étendu, des solutions admissibles et la fonction-objectif. Puis, nous décrirons le type de mouvements que nous appliquons aux solutions du problème ainsi que la manière de les évaluer. Les moyens de diriger la recherche seront ensuite discutés et finalement, nous donnerons quelques résultats numériques et ferons des comparaisons avec d'autres méthodes itératives pour mettre en évidence l'originalité et l'efficacité de nos méthodes.

3.3.1. Problème de la chaîne de traitement

Modélisation et voisinage

Comme le problème consiste à trouver une permutation parmi toutes les permutations des éléments de 1 à n, la modélisation du problème reste très classique et colle parfaitement à sa formulation. Si π est une permutation, où $\pi(j)$ correspond au numéro de l'objet placé en $j^{\text{ième}}$ position sur la chaîne, la valeur de la fonction-objectif sera $f(\pi) = e^{\pi}_{mn}$, où e^{π}_{ij} est l'heure de fin de traitement au plus tôt sur la machine i de l'objet placé en $j^{\text{ième}}$ position et obéit aux relations (en supposant que l'on pose pour les valeurs fictives $e^{\pi}_{0j} = e^{\pi}_{i0} = 0$) :

$$e^{\pi}_{ij} = \max\{ e^{\pi}_{i-1j}, e^{\pi}_{ij-1} \} + t_{i(j)} \; (i = 1, \ldots, m, \; j = 1, \ldots, n) \qquad (5)$$

Le calcul de la fonction-objectif peut donc se faire en $O(n \cdot m)$.

Plusieurs types de mouvements sont envisageables : pour se restreindre à ceux pouvant être spécifiés par deux paramètres ou moins, on peut tout d'abord penser à l'inversion de deux objets placés consécutivement sur la chaîne, ensuite, à la transposition de deux objets distincts et enfin au déplacement d'un objet à une autre place. Le premier type engendre le voisinage le plus restreint puisque de taille $n - 1$. Le deuxième type définit un voisinage comprenant $\frac{n \cdot (n-1)}{2}$ mouvements et le troisième est de taille $n(n - 2) + 1$. Les capacités de ces divers types de voisinages à se diriger en peu d'itérations vers de bonnes

solutions sont très différentes ; plusieurs auteurs, Ogbu et Smith [75], [76], Osman et Potts [78], ainsi que nous-même dans [91], nous nous accordons à dire que le premier type est le plus mauvais et le troisième le meilleur. De plus, un argument décisif plaide en faveur de ce dernier type de mouvement : il est possible de calculer en $O(n^2m)$ la valeur de l'ensemble des solutions du voisinage avec l'algorithme décrit ci-dessous, alors que pour le second type, nous ne voyons pas comment faire cette évaluation mieux qu'en $O(n^3m)$.

Pour l'évaluation des mouvements, on suppose que l'on a calculé au préalable, à l'aide d'une formule similaire à (5), les valeurs q_{ij}^{π} qui donnent le temps minimum séparant le début de traitement de l'objet en $j^{\text{ième}}$ position sur la machine i dans la solution π de la fin des opérations. Dans l'algorithme qui suit, on notera $e_{ijk}^{\pi'}$ et $q_{ijk}^{\pi'}$ les valeurs analogues à e_{ij}^{π} et à q_{ij}^{π} lorsque l'objet qui était en $k^{\text{ième}}$ position a été retiré de la solution π. Remarquons que $e_{ijk}^{\pi'} = e_{ij}^{\pi}$ pour tout $j < k$ et $q_{ijk}^{\pi'} = q_{ij}^{\pi}$ pour tout $j > k$; il sera donc nécessaire de ne recalculer que les autres valeurs à l'aide de relations analogues à (5).

Algorithme calculant la valeur de tous les mouvements :

Pour $k = 1$ à n faire : {enlever l'objet en $k^{\text{ième}}$ position}
  Calculer $e_{ijk}^{\pi'}$ $(i = 1, \ldots, m,\ j = k + 1, \ldots, n)$
  Calculer $q_{ijk}^{\pi'}$ $(i = 1, \ldots, m,\ j = k - 1, \ldots, 1)$
  Pour $j = 1, \ldots, n,\ j \neq k$ faire : {insérer l'objet en $j^{\text{ième}}$ position}
    si $j < k$ poser $r_{ijk} = \max\{r_{i-1jk}, e_{ij-1}^{\pi}\} + t_{i(k)}$ $(i = 1, \ldots, m)$
    si $j > k$ poser $r_{ijk} = \max\{r_{i-1jk}, e_{ijk}^{\pi'}\} + t_{i(k)}$ $(i = 1, \ldots, m)$
    La valeur de la solution où l'objet en $k^{\text{ième}}$ position est déplacé en $j^{\text{ième}}$ position est donnée par :
      si $j < k$: $\max_i (r_{ijk} + q_{ijk}^{\pi'})$
      si $j > k$: $\max_i (r_{ijk} + q_{ij+1}^{\pi})$

Il est clair que cet algorithme fonctionne en $O(n^2m)$. Pour montrer qu'il calcule bien les valeurs attendues, il suffit de remarquer premièrement que r_{ijk} correspond à l'heure de début au plus tôt, sur la machine i, du traitement de l'objet précédemment placé en $k^{\text{ième}}$

position lorsqu'on le déplace à la $j^{\text{ième}}$ (on suppose $r_{0jk} = 0$) et secondement que pour tout j inférieur à n, on a : $f(\pi) = \max_i (e^{\pi}_{ij} + q^{\pi}_{ij+1})$.

Remarquons aussi que cet algorithme peut servir à déterminer en $O(nm)$ la meilleure place où l'on peut insérer un nouvel objet sur la chaîne de traitement. Nous avons donc pu améliorer, dans [91], de $O(n^3m)$ à $O(n^2m)$ une heuristique pour ce problème, décrite par Nawaz et al. dans [71], et dont le principe consiste à insérer, dans un certain ordre, les objets les uns après les autres, à la place où ils augmentent le moins le temps pendant lequel les machines sont inactives.

Direction de la recherche

Pour finir de spécifier notre recherche itérative, nous devons encore préciser comment elle est dirigée. Nous avons choisi d'interdire le retour à une solution dont la valeur a déjà été obtenue au cours des t dernières itérations. On prévient ainsi tout cycle de longueur t ou moins, mais, avec t fixé pendant toute la recherche, il n'est pas toujours possible d'obtenir les solutions optimales de petits problèmes dûs à Carlier [14] et de problèmes à 9 objets et 10 machines que nous avons générés aléatoirement. Par contre, en partant d'un t petit et en le faisant croître en cours de recherche (par exemple d'une unité toutes les 10 ou 20 itérations), il a toujours été possible de trouver les solutions optimales de ces problèmes.

Ce type d'interdiction de mouvements peut être implanté de manière efficace : soit M, un nombre entier, relativement grand, tel qu'il soit possible de mémoriser un tableau T de M entiers dans la machine sur laquelle on travaille. Si $f(s_k)$ est la valeur supposée entière de la solution s_k à l'itération k, ce qui n'est pas restrictif lorsqu'on travaille sur un calculateur, on mémorisera dans $T[f(s_k) \text{ modulo } M]$ la valeur $k + t$. Si une solution s' du voisinage potentiel de la solution à l'itération k' est telle que $T[f(s') \text{ modulo } M)] > k'$, s' ne sera plus considérée comme une solution éligible. Remarquons que ce type d'interdiction ne peut fonctionner que si la fonction-objectif a une grande étendue de valeurs, ce qui est le cas pour les problèmes que nous avons générés. La manière efficace décrite ci-dessus pour mémoriser les solutions interdites ne fait qu'approcher l'intention que nous avions d'interdire des solutions selon leur valeur puisque non seulement toute solution d'une

valeur donnée est interdite pendant t itérations, mais toutes celles qui ont cette valeur modulo M le sont aussi. Néanmoins, nous n'avons pas constaté de modifications du comportement de la recherche, si M est choisi suffisamment grand (> 10'000).

Si l'exemple de problème considéré a un éventail de valeurs de la fonction-objectif très restreint, nous conseillons un autre type d'interdiction qui fonctionne aussi bien que celui présenté ci-dessus, mais qui nécessite un ajustage plus délicat du paramètre t : on interdira de remettre un objet à une place d'où il a été déplacé durant les t dernières itérations (à moins, évidemment, que ce déplacement mène à une meilleure solution que la meilleure trouvée jusqu'à ce stade de la recherche). Ce type d'interdiction présente des lacunes en théorie car déplacer k fois de suite l'objet situé à la position i vers la position $i - k + 1$ ne change pas la solution ; en pratique cependant, ce type d'interdiction fonctionne relativement bien et peut aussi être implanté de manière à ce qu'il soit possible de tester en temps constant le statut d'un mouvement. Enfin, nous avons choisi une permutation aléatoire comme solution initiale.

Résultats numériques et comparaisons avec d'autres méthodes

Tout d'abord, nous présentons dans le tableau 3 une comparaison de diverses méthodes heuristiques rapides qui ont été proposées pour le problème de la chaîne de traitement : il s'agit des méthodes dues à Gupta, Johnson, Palmer et Campbell et al., (dont on peut

Nombre de problèmes générés		500	100	100	100	50	50
Nombre n d'objets		9	10	20	20	40	50
Nombre m de machines		10	10	10	20	10	10
Méthode	Complexité	Qualité					
Gupta	$n\log(n) + nm$	13.4	12.8	19.6	18.8	18.9	17.1
Johnson	$n\log(n) + nm$	10.9	11.8	16.7	16.8	17.3	16.3
Dannenbring	$n\log(n) + nm$	8.5	9.1	12.5	13.4	13.5	11.2
Palmer	$n\log(n) + nm$	8.3	9.0	13.3	12.5	10.9	10.7
Campbell et al.	$nm^2 + mn\log(n)$	4.5	5.2	9.7	8.6	9.9	9.3
Nawaz et al.	n^2m	2.1	2.2	3.9	3.8	2.6	2.1

Tableau 3 : Comparaison de la qualité de quelques heuristiques classiques.

trouver la description dans [5]), Dannenbring [25], et Nawaz et al. [71]. Pour faire cette

comparaison, nous avons indiqué dans ce tableau de combien de pour cent les solutions obtenues par ces heuristiques sont au-dessus de la meilleure solution trouvée au cours de 1’000 itérations par notre recherche itérative dirigée (ce nombre d’itérations est suffisant pour trouver les solutions optimales de problèmes à 9 objets et 10 machines que nous avons générés, comme on le verra en figure 13). Nous remarquons que certaines de ces méthodes fournissent des solutions très médiocres (on rappellera que pour les problèmes à 9 objets et 10 machines, une solution quelconque a en moyenne une valeur d’environ 20% au-dessus de celle de la solution optimale) et que l’heuristique due à Nawaz et al. se détache nettement du lot, avec des solutions tout de même toujours à plus de 2% au-dessus de l’optimum. Cela montre que notre recherche itérative peut trouver des solutions bien meilleures que celles fournies par ces méthodes constructives, au détriment du temps de calcul, car une itération de notre recherche est plus onéreuse que les quatre premières méthodes de ce tableau.

Néanmoins, notre recherche itérative est efficace, comme on peut le constater en figure 13 où l’on a représenté la proportion de problèmes résolus de façon optimale en fonction du nombre d’itérations. Nous voyons que quelques dizaines d’itérations permettent de trouver les solutions optimales d’un grand pourcentage des problèmes. Malheureusement, il existe un petit nombre de recherches qui ont besoin d’un nombre anormalement grand d'itérations pour trouver une solution optimale. Pour réaliser cette figure, nous avons considéré 200 problèmes, générés aléatoirement, pour lesquels les solutions optimales sont connues, et nous les avons résolus chacun 100 fois, en partant de solutions initiales tirées au hasard.

Vérifier que l’on peut obtenir systématiquement les solutions optimales de petits problèmes ne suffit pas encore à montrer l’efficacité de la méthode : il faut encore voir que pour des problèmes plus gros on obtient aussi de bons résultats. Cela a déjà été abordé en section 1.1 dans le tableau 1 où il a été mentionné que l’on avait pu obtenir les solutions optimales d’une proportion honorable de gros problèmes (1’000 à 2’000 opérations) tirés aléatoirement. Il convient de remarquer que les proportions indiquées dans ce tableau correspondent aux problèmes pour lesquels nous avons pu prouver que les solutions

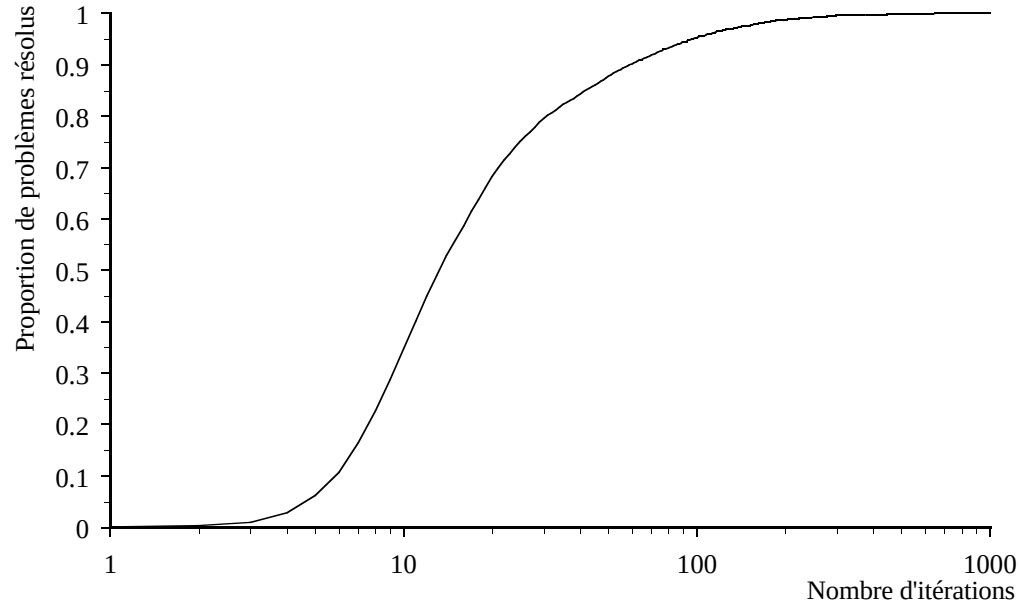

Figure 13 : Proportion de problèmes résolus en fonction du nombre d'itérations.

obtenues étaient optimales et non pas aux problèmes pour lesquels nous avons atteint (éventuellement sans le savoir) la solution optimale.

La comparaison de la méthode que nous proposons avec celles dues à d'autres auteurs est problématique car les codes de leurs programmes ne sont pas disponibles. Remarquons cependant qu'à notre connaissance, l'algorithme génétique de Reeves, [82], est un des plus performants à l'heure actuelle pour le problème de la chaîne de traitement car on trouvera dans cette référence des comparaisons lui étant favorables, notamment sur de gros problèmes, avec les recuits simulés d'Ogbu et Smith [75], [76] et d'Osman et Potts [78]. À défaut de pouvoir tirer des conclusions sur les temps de calcul nécessités par cet algorithme génétique, on peut toutefois remarquer que de façon asymptotique (lorsque le temps de calcul tend vers l'infini, c'est-à-dire quelques heures pour un acteur humain plus ou moins patient) l'avantage est à notre méthode puisque l'algorithme génétique de Reeves n'a pas réussi à améliorer une seule des solutions d'un jeu de 120 problèmes que nous proposons dans [93].

Par rapport à la recherche itérative dirigée de Widmer et Hertz, [98], l'originalité de notre démarche réside principalement dans le choix et l'évaluation du voisinage, puisque ce dernier semble à la fois mieux adapté pour le problème considéré et qu'il peut s'évaluer plus rapidement. Une autre innovation faite par notre méthode est l'interdiction de solutions pendant un nombre d'itérations variable au cours de la recherche, ce qui a permis d'éviter de visiter cycliquement les mêmes solutions comme cela arrive très souvent avec un nombre d'itérations fixe.

3.3.2. Problème de traitement à séquences fixées

Modélisation et voisinage

Pour ce problème, nous avons choisi une modélisation classique, consistant à orienter des arêtes dans un graphe de manière à minimiser la longueur d'un plus long chemin dans ce dernier. Nous restreignons notre recherche itérative dirigée aux solutions admissibles, et la fonction à minimiser est donc la fonction-objectif initiale, qui donne l'étendue de travail entre le début et la fin des opérations. Lorsque nous avons développé notre recherche itérative dirigée, le but était de la comparer à un recuit simulé existant, proposé par van Laarhoven et al. [62] ; nous avons par conséquent choisi le même voisinage que dans cette référence, à savoir l'inversion de deux opérations, appartenant à un plus long chemin, qui doivent s'exécuter successivement sur une machine et qui mettent en jeu deux objets différents. Il est montré dans [62] que ce voisinage est tel que la nouvelle solution est admissible (si la solution courante l'est), qu'il est possible d'atteindre une solution optimale de ce problème en partant de n'importe quelle solution initiale admissible et que permuter deux opérations n'appartenant pas à un plus long chemin (appelé aussi chemin critique plus loin) ne peut améliorer la solution, si cette dernière reste admissible.

Ce type de voisinage est très commode pour un recuit simulé car il est possible de calculer très efficacement la valeur d'un mouvement dégradant la solution (un mouvement non dégradant est toujours accepté) : en reprenant les notations introduites en section 1.2, page 7, déterminer si une opération est critique peut se faire en temps constant si l'on a mémorisé les valeurs d_i de début au plus tôt d'une opération i de durée t_i et la longueur

d'un plus long chemin f_i à partir de la fin du traitement de i car une opération est critique si, et seulement si la quantité $d_i + t_i + f_i$ est égale à la longueur d'un plus long chemin pour la solution courante ; si un mouvement inversant les opérations a et b (= sm_a, l'opération succédant à a sur la même machine) est détériorant, on peut calculer la valeur de la nouvelle solution en temps constant, en remarquant que, lorsque a et b sont permutées, les nouvelles valeurs d'_a, d'_b, f'_a, f'_b de débuts au plus tôt et de longueurs de plus longs chemins partiels sont données par :

$$d'_b = max(d_{pm_a} + t_{pm_a}, d_{po_b} + t_{po_b})$$
$$d'_a = max(d'_b + t_b, d_{po_a} + t_{po_a})$$
$$f'_a = max(f_{sm_b} + t_{sm_b}, f_{so_a} + t_{so_a})$$
$$f'_b = max(f'_a + t_a, f_{so_b} + t_{so_b})$$

et que la nouvelle valeur de la fonction-objectif est donnée par :

$$max(d'_b + t_b + f'_b, d'_a + t_a + f'_a) \quad (6)$$

Pour une recherche itérative dirigée où l'on désire élire le meilleur mouvement non interdit, il serait nécessaire de recalculer intégralement le plus long chemin après inversion. Comme cela est très onéreux, nous proposons d'élire le mouvement qui minimise la valeur donnée par (6), même si celle-ci n'est pas forcément la valeur de la fonction-objectif dans le cas de mouvements non détériorants. L'évaluation du voisinage pourra donc se faire en $O(N)$ où N est le nombre total d'opérations du problème.

Direction de la recherche

Comme solution initiale, nous avons choisi de démarrer avec un atelier inactif et d'y insérer les objets, choisis dans un ordre quelconque, en effectuant le plus tôt possible toutes les opérations qui doivent être faites sur l'objet à insérer, sans modifier les heures

de traitement des objets déjà insérés. Cet algorithme glouton donne en $O(N^2)$ des solutions de qualité très mauvaise.

Un mouvement pouvant être caractérisé par l'opération *a* qui est permutée avec celle qui lui succède sur la même machine, on interdit pendant *t* itérations de permuter *a* avec l'opération qui la précède. Parmi les problèmes générés aléatoirement — toutes les durées des opérations sont tirées de manière uniforme selon la même loi et les séquences des objets ne sont pas en corrélation les unes avec les autres et comportent toutes *m* opérations, une par machine —, nous avons remarqué que les problèmes avec $n \not\gg m$ étaient beaucoup plus difficiles à résoudre que ceux pour lesquels $n \gg m$. Cela influence directement la valeur qu'il faut donner au paramètre *t* pour obtenir les meilleurs résultats. Si $n \not\gg m$, $\frac{n+m}{2}$ est une bonne valeur moyenne de *t* ; si $n \gg m$, il faut plutôt choisir $\frac{nm}{2}$, d'après des résultats expérimentaux que nous avons obtenus. La frontière entre les problèmes difficiles et les problèmes faciles se situe aux environs de $n = 5m$. Par conséquent, nous proposons de choisir comme valeur moyenne de *t* : $\tau = \frac{n+m}{2} \cdot e^{-\frac{n}{5m}} + \frac{nm}{2} \cdot e^{-\frac{5m}{n}}$. Comme nous avons observé qu'un *t* fixe n'empêche pas certains cycles, nous tirerons sa valeur aléatoirement entre $\lfloor 0.8\tau \rfloor$ et $\lceil 1.2\tau \rceil$.

Comme mémoire à long terme, nous proposons de pénaliser les mouvements fréquemment élus et d'ajouter à l'évaluation du mouvement qui permute l'opération *b* et celle qui la précède une pénalisation de $0.5 \cdot \Delta_k^{max} \cdot \sqrt{N} \cdot fréq(b, k)$ où Δ_k^{max} est la différence maximale entre les valeurs de deux solutions successivement visitées par la recherche jusqu'à l'itération *k* et *fréq*(*b*, *k*) la fréquence, à l'itération *k*, à laquelle l'opération *b* a été séquencée plus tôt. Intuitivement, cette pénalisation peut être justifiée : plus la valeur entre deux solutions successives est élevée, cela dépend de la fonction-objectif choisie et de l'exemple de problème, plus on devra, pour obtenir le même résultat, augmenter la pénalité ; ceci « explique » le terme Δ_k^{max}. On observe que plus le voisinage est grand, plus la répartition des fréquences se concentre vers de petites valeurs. Pour que la pénalité ne devienne pas nulle lorsque la taille du problème augmente, il faut la multiplier par une fonction strictement croissante avec la taille du voisinage. La fonction identité s'avérant trop grande en pratique, nous avons choisi la fonction $\sqrt{N}$, qui, avec un

coefficient de 0.5, fournit de bons résultats quels que soient le type et la taille du problème considéré.

Si un mouvement mène à une solution, dont on estime la valeur par (6), potentiellement meilleure que toutes celles visitées jusqu'ici, on supprimera sa pénalité sur sa fréquence d'utilisation et son éventuel statut interdit.

Résultats numériques et comparaisons avec d'autres méthodes.

Nous avons pu, pour ce problème, comparer amplement notre méthode avec d'autres. Tout d'abord, les auteurs d'une méthode de recuit simulé, van Laarhoven et al. [62], donnent un tableau de résultats et des temps de calcul sur un type d'ordinateur dont nous disposions. Ensuite, les auteurs de la méthode généralisée de déplacement du goulot d'étranglement, Appleggate et Cook [4], nous ont fait parvenir leur code. Il a donc été possible de comparer toutes ces méthodes de manière fiable.

En figure 14, nous donnons l'évolution de notre recherche itérative pour 5 problèmes à 10 objets et 10 machines proposés par Lawrence [66]. Nous avons résolu chacun de ces

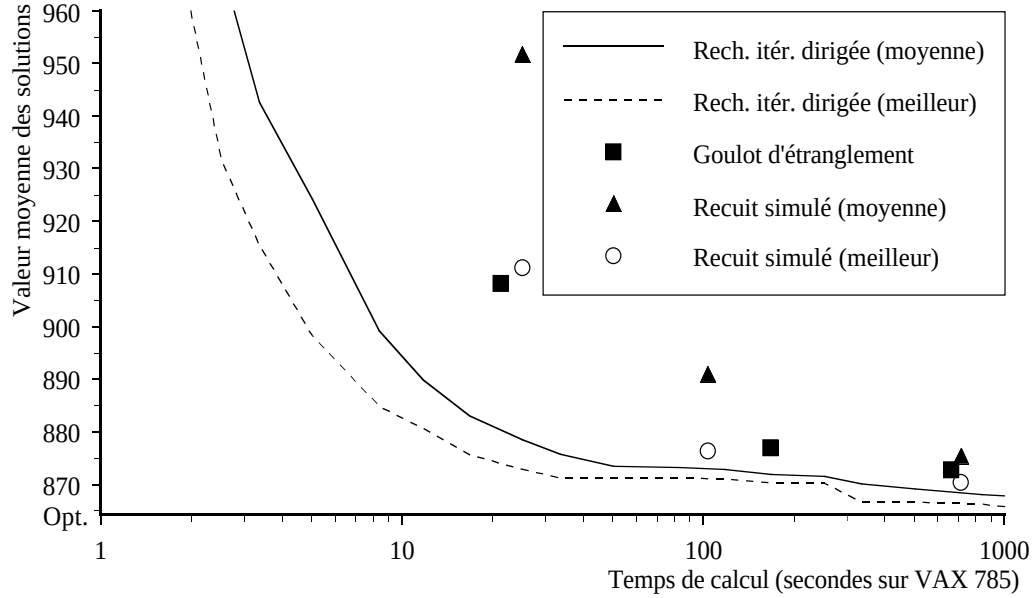

Figure 14 : Comparaison de quelques méthodes sur un type de problème particulier.

problèmes 5 fois, en modifiant la séquence aléatoire des nombres d'itérations pendant lesquelles on interdit des mouvements, ce qui engendre autant de trajectoires différentes. Les valeurs finales du recuit simulé, données par van Laarhoven et al. [62] ont été obtenues de manière similaire et sont données pour trois schémas de recuit différents. Les méthodes de déplacement du goulot d'étranglement sont entièrement déterministes et les résultats ne sont donc donnés que pour une exécution. Comme plusieurs exécutions ont été faites pour les méthodes non déterministes, nous pouvons exhiber deux statistiques : celle donnant la valeur moyenne obtenue sur toutes les exécutions pour tous les problèmes et celle donnant la valeur moyenne obtenue sur la meilleure exécution concernant chaque problème.

Nous remarquons sur la figure 14 que la recherche itérative dirigée est nettement plus rapide que le recuit simulé puisqu'elle fournit des solutions de valeur moyenne donnée en un temps à peu près 10 fois inférieur. L'écart entre les méthodes de déplacement du goulot d'étranglement et la recherche itérative dirigée est moindre mais néanmoins significatif. En ce qui concerne les autres problèmes proposés par Lawrence dans [66], on arrive à des comparaisons similaires, si ce n'est que l'écart entre les méthodes de déplacement du goulot d'étranglement et la recherche itérative dirigée peut être très faible pour certaines tailles de problèmes. Dans le tableau 4, nous comparons ces deux types de méthodes pour

15 objets, 15 machines	20 objets, 15 machines	20 objets, 20 machines	30 objets, 15 machines	30 objets, 20 machines
9.0 (5.2)	10.1 (7.3)	10.0 (6.6)	8.3 (11.9)	13.1 (13.6)
4.5 (1.7)	5.6 (2.6)	6.2 (1.6)	4.8 (2.1)	8.4 (3.5)
3.4 (0.9)	4.4 (1.5)	4.8 (0.9)	4.3 (1.1)	7.3 (1.4)

Tableau 4 : Comparaison des méthodes de déplacement du goulot d'étranglement et de recherche itérative dirigée (entre parenthèses), en% au-dessus de la meilleure solution connue.

diverses tailles de problèmes et nous remarquons que pour certaines tailles, il peut être légèrement avantageux d'utiliser la méthode initiale de déplacement du goulot d'étranglement (première ligne du tableau), comme l'ont proposé Adams et al. [2]. Dans ce tableau, pour un jeu de problèmes que nous proposons dans [93], nous donnons en pour cent au-dessus de la meilleure solution connue, les valeurs moyennes obtenues par trois

méthodes de déplacement du goulot d'étranglement et la recherche itérative dirigée si elles fonctionnent pendant le même temps. Dans ce tableau, nous n'avons pas pu inclure les problèmes les plus gros que nous proposons (à 50 et 100 objets) car le code que nous avons obtenu pour les méthodes de déplacement du goulot d'étranglement n'est pas conçu pour traiter de si gros problèmes et génère des erreurs à l'exécution, vraisemblablement dues à des dépassements de capacité lors de l'élaboration d'arbres de recherche dont ces méthodes ont besoin. En effet, elles résolvent exactement une série de problèmes plus simples, mais néanmoins NP-difficiles.

En figure 15, nous avons représenté le temps de calcul mis par la méthode initiale de déplacement du goulot d'étranglement en fonction du nombre d'objets, pour résoudre des problèmes (générés aléatoirement) où le nombre de machines est fixé à 5 (pour ce type de

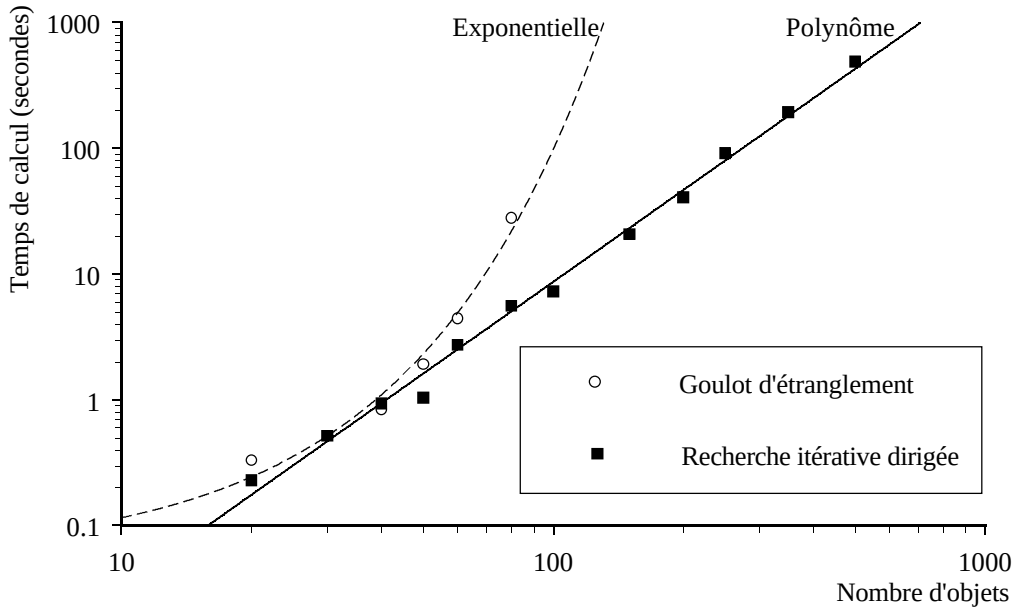

Figure 15 : Temps mis pour résoudre de façon optimale des problèmes à 5 machines.

problèmes, lorsque le nombre d'objets est suffisamment grand, cette méthode semble toujours trouver une solution optimale). Nous avons superposé dans cette figure le temps moyen mis par la recherche itérative dirigée pour trouver une solution optimale. Nous remarquons, dans cette figure où les échelles sur les deux axes sont logarithmiques, que

la méthode de déplacement du goulot d'étranglement semble nécessiter un temps de calcul exponentiel en le nombre d'objets, alors que la recherche itérative dirigée semble prendre un temps qui augmente seulement de manière polynomiale, bien que pour certaines valeurs du nombre d'objets, l'exponentielle pourrait se situer au-dessous de la courbe polynomiale.

Comme on peut le voir en figure 16, l'augmentation du nombre moyen d'itérations nécessaires à la résolution de problèmes générés aléatoirement où le nombre d'objets *n*

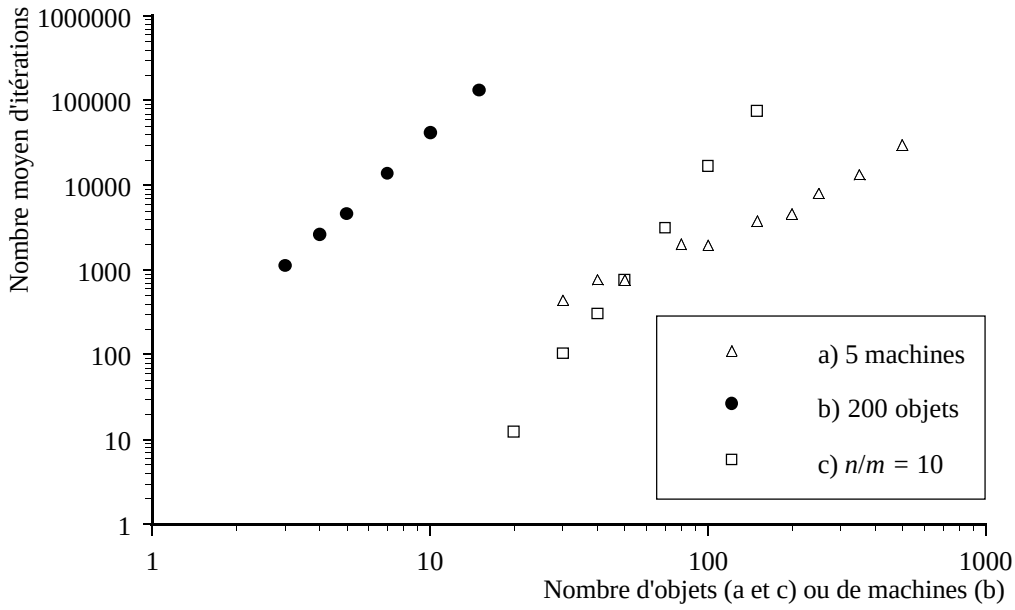

Figure 16 : Nombre moyen d'itérations nécessaires pour résoudre des problèmes aléatoires.

est nettement plus grand que le nombre de machines *m*, suit aussi une fonction polynomiale avec l'augmentation du nombre de machines, ce qui suggère que ce type de problèmes est simple et qu'il doit exister un algorithme polynomial pour le résoudre, comme nous l'avons conjecturé dans la section 1.2. Nous ne pouvons pas prouver que notre algorithme trouve une solution optimale en un temps polynomial dans tous les cas, même s'il semble, expérimentalement et par régression, que notre méthode nécessite un temps de calcul moyen en $O(n^{2.26} \cdot m^{3.88})$.

La modélisation du problème, le choix et l'évaluation du voisinage ont été proposés par d'autres auteurs et notre recherche itérative n'a rien innové dans ces domaines ; en revanche, notre manière d'implanter la mémoire à long terme n'avait jamais été envisagée comme ceci et les résultats asymptotiques de notre méthode sont encourageants, comme on peut le constater dans le tableau 5 où l'on remarque que grâce à elle, nous avons pu améliorer toutes les meilleures solutions connues des problèmes non encore résolus

Problème	Nombre d'objets et de machines	Goulot d'étrangle-ment initial [2]	Goulot d'étrangle-ment généralisée [4]	Recuit simulé [62]	Recherche itérative dirigée
ABZ7	20, 15	730	668	—	665
ABZ8	20, 15	774	687	—	676
ABZ9	20, 15	751	707	—	691
LA21	15, 10	1084	1053	1063	1047
LA 27	20, 10	1294	1269	1269	1240
LA 29	20, 10	1239	1195	1218	1170
LA 38	15, 15	1255	1209	1215	1202

Tableau 5 : Meilleures solutions obtenues par diverses méthodes pour des problèmes classiques.

proposés par Adams et al. dans [2] et par Lawrence dans [66]. Il est aussi à noter que les meilleures solutions publiées pour les problèmes que nous proposons, [93], ont toutes été trouvées par notre méthode itérative dirigée.

3.3.3. Problème d'affectation quadratique

Modélisation et voisinage

De même que pour le problème de la chaîne de traitement, le problème d'affectation quadratique revient à chercher une permutation optimale. Comme toute permutation est une solution admissible de ce problème, la modélisation est triviale. Pour le choix du voisinage, il n'y a pas autant de possibilités raisonnables que pour le problème de la chaîne de traitement : en effet, déplacer l'objet en $i^{\text{ième}}$ position dans la permutation pour le mettre en $j^{\text{ième}}$ ne correspond pas à grand-chose, il en est de même pour la transposition

des objets en $i^{\text{ième}}$ et $i + 1^{\text{e}}$ position dans la permutation. En réalité, si l'on désire se restreindre à des voisinages ne modifiant que les sites attribués à deux objets, il n'est envisageable que de permuter les objets $\pi(i)$ et $\pi(j)$ occupant les sites i et j. Comme l'ont fait remarquer Burkard et Rendl dans [13], on peut évaluer chacun de ces mouvements en $O(n)$ (où n est la taille du problème). Avec des matrices de distances d et de flots f symétriques à diagonales nulles, la valeur d'un mouvement (i, j) sur une solution π est donnée par :

$$\Delta(\pi, i, j) = 2 \cdot \sum_{k \neq i, j} (f_{jk} - f_{ik}) \, (d_{\pi(i)\pi(k)} - d_{\pi(j)\pi(k)}) \tag{7}$$

En mémorisant la valeur de $\Delta(\pi, i, j)$, on peut calculer en $O(1)$, lorsque $i \neq r, s$ et $j \neq r, s$ et en utilisant l'équation (8), qui peut être trouvée dans Frieze et al. [38], la valeur de $\Delta(\mu, i, j)$, où μ est la permutation obtenue en échangeant dans π les éléments r et s, i.e. $\mu(k) = \pi(k), \ (k \neq r, \ k \neq s), \ \ \mu(r) = \pi(s), \ \mu(s) = \pi(r)$.

$$\begin{gathered} \Delta(\mu, i, j) = \\ \Delta(\pi, i, j) + 2 \cdot (f_{ri} - f_{rj} + f_{sj} - f_{si}) \, (d_{\mu(s)\mu(i)} - d_{\mu(s)\mu(j)} + d_{\mu(r)\mu(j)} - d_{\mu(r)\mu(i)}) \end{gathered} \tag{8}$$

Par conséquent, en mémorisant pour tout i et j les valeurs de $\Delta(\pi, i, j)$, on peut calculer en $O(n^2)$ l'ensemble du voisinage : en utilisant (8) on peut évaluer les $O(n^2)$ mouvements ne faisant pas intervenir les indices r et s et en utilisant (7), on peut évaluer les $O(n)$ mouvements qui font précisément intervenir ces indices.

Direction de la recherche

Nous aurions pu utiliser, pour diriger la recherche à court terme, le même type d'interdiction que pour le problème de la chaîne de traitement, à savoir éviter de retourner pendant un certain nombre d'itérations à une valeur donnée de la fonction-objectif. Ce type d'interdiction fonctionne très bien pour des problèmes où les matrices de distances et de flots sont générées aléatoirement, uniformément entre 0 et 99 par exemple, car la fonction-objectif prend un éventail de valeurs très grand. Cependant, on rencontre fréquemment des problèmes où une proportion importante des valeurs de flots entre deux objets sont nulles et par conséquent, la fonction-objectif ne prend qu'un nombre restreint

de valeurs. On peut contourner la difficulté en associant une autre fonction prenant une plage de valeurs très large lorsqu'elle est appliquée à l'ensemble des permutations. On peut donner en exemple la fonction caractéristique : $\sum_{i=1}^{n} i^2 \cdot \pi(i)$ qui prend un nombre potentiel de valeurs différentes proportionnel à $O(n^4)$. Avec une telle fonction, il est possible d'intégrer mémoire à court terme et mémoire à long terme dans la même structure de données, en tirant aléatoirement sur une plage très étendue le nombre t d'itérations pendant lesquelles on interdit de revenir à une valeur donnée de la fonction caractéristique. Pour ne pas bloquer la recherche, il faut alors privilégier les petits t. Si l'on note par $u(0, 1)$ une variable aléatoire uniformément distribuée entre 0 et 1, on pourra prendre comme valeur de t : $\lfloor (b-a) \cdot u^c(0, 1) + a \rfloor$. Quelques essais avec comme paramètres $a = 0.5n$, $b = n^3$ et $c = 16$, ont montré que des résultats excellents pouvaient être obtenus sur des problèmes faiblement structurés comme le sont ceux générés aléatoirement. Cependant, si le problème présente des optima locaux avec de grands bassins d'attraction, une telle méthode éprouve des difficultés à s'en échapper et peut s'avérer inefficace. De plus, il semble difficile de justifier la fonction de répartition de la variable aléatoire t. A-t-on eu de la chance dans la proposition que nous avons faite ou au contraire existe-t-il des fonctions bien meilleures ?

Comme nous ne voyons pas comment apporter des éléments de réponse à ces questions, nous allons développer une méthode plus facile à justifier. En section 3.1.3, nous avons proposé de diriger à court terme la recherche en interdisant pendant t itérations d'effectuer le mouvement inverse de celui que l'on vient de faire. Dans le cas du problème d'affectation quadratique, si l'on applique le mouvement (i, j) à la permutation π, nous définissons par mouvement inverse un mouvement qui place à nouveau l'objet i sur le site $\pi(i)$ et l'objet j sur le site $\pi(j)$. Il est possible de définir de plusieurs manières l'inverse d'un mouvement, mais comme l'a montré Rogger dans [83], celui que nous avons choisi est un des plus efficaces pour empêcher des cycles et le moins sensible à la valeur du paramètre t. Si ce dernier est fixé une fois pour toutes en début de recherche, nous avons remarqué que $t = n$ donne de bons résultats (on peut du reste le voir en figure 9, page 40, où nous

avons précisément étudié l'influence de la valeur de t pour des problèmes d'affectation quadratique).

Nous avons vu qu'un t fixe ne produisait pas une recherche très robuste, car, même pour des t assez grands, des cycles pouvaient apparaître (cf. figure 10, page 41). C'est pourquoi nous proposons de tirer t aléatoirement, uniformément entre $\lfloor 0.9n \rfloor$ et $\lceil 1.1n + 4 \rceil$ (les petits problèmes ont besoin d'une liste proportionnellement plus grande que les grands pour être résolus de manière optimale, d'où le « + 4 » dans la borne supérieure de la valeur de t).

Pour bénéficier des avantages d'un t grand ou petit (échapper à une vallée ou la visiter en détail), nous pensons qu'il faut maintenir pendant un nombre u d'itérations la valeur du t qui vient d'être tirée. Pour être cohérent avec cette notion de bénéfice d'un grand t, il faut maintenir sa valeur pendant au moins un nombre d'itérations égal à la plus grande valeur qu'il peut prendre ; dans la méthode que nous proposons, nous avons choisi de fixer u au double de la plus grande valeur de t (cependant, remarquons que le paramètre u a peu d'influence sur la recherche en pratique ; poser $u = 1$, c'est-à-dire tirer à chaque itération la valeur de t produit une méthode presque aussi efficace ; par contre il faut éviter de donner à u une très grande valeur).

Pour être à même de détruire la structure de la solution associée à un optimum local, donc pour pouvoir s'échapper de la vallée lui étant associée, une mémoire à long terme est nécessaire pour les grands problèmes, comme pour ceux de dimension plus modeste mais très structurés (comme l'est par exemple celui proposé par Elshafei dans [32], de dimension $n = 19$) ; pour cela nous élisons tout mouvement qui n'a pas été effectué pendant v itérations, quelle que soit son évaluation. Pour ne pas avoir à traiter le cas de plusieurs mouvements devant être élus simultanément à cause de cette règle, on fera comme si, avant de débuter la recherche, on avait effectué l'ensemble des $|M|$ mouvements (le voisinage, défini par un ensemble M de mouvements, est statique) durant d'hypothétiques itérations $-|M|, -|M| + 1, \ldots, -1$. Bien entendu, il faut que v soit (bien) plus grand que $|M|$ et nous proposons de poser $v = 5n^2 \sim 10|M|$.

Résultats numériques

Il existe pour ce problème quantité d'exemples publiés dans la littérature. Nous donnerons des résultats pour les problèmes dûs à Elshafei [32], Krarup et Pruzan [61], Nugent et al. [74], Roucairol [84], Steinberg [90], Skorin-Kapov [88] et Wilhelm et Ward [99], ainsi que pour des problèmes que nous avons générés aléatoirement. Chacun de ces auteurs propose non seulement des exemples de problèmes, mais également des méthodes heuristiques ou exactes pour les résoudre. Dans le tableau 6, nous rappelons les valeurs

Problème dû à :	Taille	Méthode : [74]	[32]	[13]	[22]	[88]	[89]	[16]	[94]
[74]	15	1206	1228	1150	1150	1150	1150	1150	1150
[74]	20	2678	2598	2570	2570	2570	2570	2570	2570
[74]	30	6379	6250	6148	6124	6124	6194	6124	6124
[61]	30	—	—	88900	—	88900	92210	88900	88900
[61]	30	—	—	91420	—	91420	93520	91420	91420
[90]	36	—	—	9572	—	9526	9654	9526	9526
[90]	36	—	—	15852	—	15852	16978	15852	15852
[88]	42	—	—	15956	—	15864	15864	15818	15812
[88]	49	—	—	23662	—	23536	23472	23398	23386
[88]	56	—	—	34916	—	34736	34788	34658	34458
[88]	64	—	—	49020	—	48964	48928	48760	48498
[88]	72	—	—	66924	—	66756	66794	66268	66256
[88]	81	—	—	91432	—	91778	91770	91362	91008
[88]	90	—	—	116460	—	116360	116258	116116	115534
[99]	50	—	—		48816				48416
[99]	100	—	—		273400				273044

Tableau 6 : Meilleures valeurs de solutions obtenues par quelques méthodes.

des meilleures solutions obtenues par quelques auteurs sur les problèmes les plus grands publiés dans la littérature. Nous voyons que toutes les meilleures solutions connues ont pu être obtenues par la méthode que nous proposons (dernière colonne du tableau), et il est intéressant de remarquer que les méthodes constructives de Nugent et al. [74] et d'Elshafei [32] produisent des solutions nettement moins bonnes que les autres méthodes de ce tableau, qui sont toutes des recherches itératives. On notera aussi que les adaptations de recuit simulé de Burkard et Rendl [13] et de Connoly [22] sont moins performantes que

les recherches itératives dirigées (toutes les autres méthodes de ce tableau) ; ceci peut être expliqué par le fait qu'en $O(n^2)$ ces méthodes peuvent examiner $O(n^2)$ solutions tandis que les recuits simulés n'en examinent que $O(n)$ durant ce temps.

Notre méthode est extrêmement robuste car elle a fourni la meilleure solution connue de tous les problèmes traités et pour toutes les exécutions partant de solutions initiales tirées aléatoirement, pour autant qu'elle fonctionne suffisamment longtemps. Dans le tableau 7, nous donnons le nombre moyen d'itérations nécessaires à notre recherche itérative pour

Problème : auteur et référence	Taille	Nombre de résolutions	Nombre moyen d'itérations
Nugent et al. [74]	5	100	9.16
[74]	6	100	7.51
[74]	7	100	24.33
[74]	8	100	9.75
[74]	12	100	226.60
[74]	15	100	715.96
[74]	20	100	1170.94
[74]	30	100	19473.6
Elshafei [32]	19	100	14527.17
Krarup et al. [61]	30	100	28103.27
[61]	30	100	19936.09
Steinberg [90]	36	100	45736.98
[90]	36	100	14000.60
[90]	36	100	24498.28
Roucairol [84]	10	100	86.02
[84]	12	100	889.42
[84]	15	100	768.12
[84]	20	30	12206.07

Tableau 7 : Nombre moyen d'itérations nécessaires à la résolution de certains problèmes.

trouver la meilleure solution connue de la littérature (c'est-à-dire l'optimum prouvé pour les problèmes de taille inférieure à 20 et conjecturé pour les plus grands). Nous remarquons dans ce tableau que ce nombre d'itérations n'est pas forcément croissant avec la taille du problème (ceci peut être dû à une mauvaise adaptation des paramètres aussi

bien qu'à un degré de difficulté différent entre problèmes) et que les problèmes entièrement aléatoires de Roucairol [84], semblent plus difficiles à résoudre que d'autres mieux structurés, tels les problèmes de Nugent et al. [74]. Dans la figure 17, nous avons

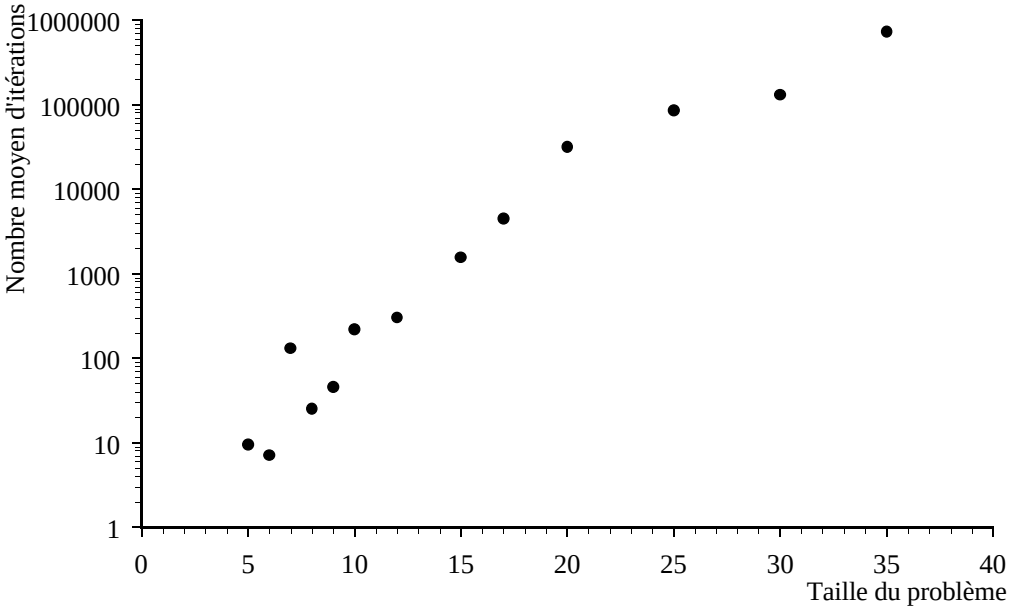

Figure 17 : Nombre moyen d'itérations pour la résolution de problèmes aléatoires.

représenté en fonction de leur taille le nombre moyen d'itérations requises pour résoudre des problèmes aléatoires. Nous remarquons que la croissance du nombre d'itérations est quasi exponentielle avec la taille du problème, ce qui montre que le problème d'affectation quadratique reste difficile, même si l'on arrive à traiter avec une recherche itérative dirigée des problèmes beaucoup plus gros qu'avec une méthode exacte. Il est clair que les résultats donnés dans le tableau 7 et dans la figure 17 ne sont qu'indicatifs de l'effort moyen qui doit être consenti si l'on désire que notre méthode trouve de bonnes solutions, car en pratique on ne connaît généralement pas la valeur d'une solution optimale.

L'originalité de la méthode que nous proposons réside avant tout dans sa simplicité et sa robustesse. Si ce n'est l'incontournable paramètre fixant la condition d'arrêt, obligatoire pour toute recherche itérative, notre méthode ne comporte pas de paramètres ouverts,

puisque nous donnons des valeurs standard à *t*, *u* et *v*, valeurs qui sont telles que pour tous les problèmes traités notre méthode n'a jamais présenté de signes de faiblesse. Il va de soi que ces valeurs standard ne sont pas optimales pour tout problème et que de meilleurs résultats peuvent être obtenus en choisissant d'autres valeurs.

3.3.4. Problème réel de distribution de biens

Modélisation et voisinage

Comme il peut être difficile de trouver une solution admissible pour tout exemple de ce problème, nous avons dû étendre l'ensemble des solutions admissibles dans sa modélisation pour traitement par recherche itérative. Si l'on nomme T_1, T_2, …, T_m les tournées, on ajoutera une tournée T_0, effectuée par un véhicule fictif dont la capacité est infinie, mais qui ne livre qu'une commande avant de retourner au dépôt. Un autre point de vue serait de considérer des solutions où toutes les commandes ne sont pas livrées, mais en attribuant des coûts de non livraison à celles-ci. Avec cette modélisation, trouver une solution admissible est trivial : il suffit de mettre toutes les commandes dans la tournée T_0.

Pour le problème réel de distribution de biens, nous avons choisi un voisinage très simple qui consiste à déplacer une commande i ($i \in \{1, \ldots, n\}$) de la tournée T_j à laquelle elle appartient, dans une autre, T_k, ($j \neq k$, $j \in \{0, \ldots, m\}$). Ainsi, seules deux tournées seront modifiées ; pour déterminer l'ordre dans lequel les commandes doivent être livrées, nous avons utilisé une heuristique d'insertion rapide, mais assez longue à décrire car il faut discuter les divers cas de figure qui peuvent survenir selon le type d'accès aux clients (camion seulement ou train routier) ; comme cette procédure ne présente pas un intérêt particulier, nous ne la décrirons pas ici.

Lorsqu'une tournée comprend des clients qui peuvent être desservis par train routier, elle peut prendre deux formes : soit celle d'un circuit passant une seule fois par chaque client dans le cas d'une livraison par camion seul, soit celle d'un circuit que l'on nommera principal, passant par les clients accessibles par train routier, auquel on adjoindra des

circuits auxiliaires desservant les clients qui ne le sont pas. Comme on ne sait pas a priori si la tournée sera effectuée par un train routier ou par un camion seul, les deux variantes devront être ouvertes à tout moment de la recherche.

Une fois que l'on a défini les commandes attachées aux diverses tournées et l'ordre dans lequel les délivrer, il faut affecter les véhicules aux tournées de manière optimale puisqu'ils ont des coûts différents d'utilisation au kilomètre. Ce problème est un problème d'affectation tridimensionnelle car il faut introduire un coût a_{krs} de livraison de la tournée k avec le camion r auquel on attelle la remorque s (pouvant éventuellement être fictive en cas de livraison avec camion seul). Le coût a_{krs} sera naturellement fixé à une valeur très grande en cas d'incompatibilités (si la tournée k contient des clients ne pouvant être desservis par le camion r ou un train routier, ou encore si le camion r ne peut tirer la remorque s).

Simplification des calculs

Moyennant quelques hypothèses peu contraignantes, il est possible, dans le cas du problème particulier que l'on traite, de réaliser cette affectation tridimensionnelle, problème NP-difficile [39] en général, à l'aide d'une affectation bidimensionnelle (ou affectation linéaire), problème pouvant être résolu en $O(m^3)$ (où m est la taille du problème d'affectation, qui est égale au nombre de tournées). Les hypothèses sont les suivantes :

— Lorsque c'est possible, il est toujours plus avantageux d'effectuer une tournée avec un camion seul plutôt qu'avec un train routier composé avec le même camion.

— Le nombre de remorques de chaque type est illimité.

Ainsi, en posant $b_{kr} = \min_s(a_{krs})$, le problème d'affectation bidimensionnelle sur les b_{kr} sera équivalent au problème d'affectation tridimensionnelle sur les a_{krs}.

Comme il n'y a pas un nombre illimité de remorques dans notre problème, il est possible qu'une affectation optimale fasse usage d'un nombre de remorques plus grand que celui qu'on a effectivement. Pour simplifier, nous éliminons de l'ensemble des solutions

admissibles les ensembles de tournées présentant cette caractéristique, même si une affectation admissible existe.

Calculer la valeur d'un mouvement requiert un volume de calcul important : il faut tout d'abord trouver les nouveaux ordres (à la fois pour camion seul ou train routier) dans lesquels sont desservis les clients des deux tournées modifiées, puis calculer les coefficients du problème d'affectation, ce qui demande aussi un effort important puisqu'il faut vérifier que toutes les contraintes (fenêtre de temps, capacité des véhicules, restrictions d'accès) soient satisfaites, et enfin procéder au calcul encore plus onéreux que constitue l'affectation proprement dite.

Le problème dont nous nous occupons englobe une vingtaine de véhicules et une cinquantaine de clients qui passent chacun une ou deux commandes, ce qui signifie qu'en tout il faut traiter entre 70 et 90 biens. Calculer le voisinage complet, même avec les simplifications exposées ci-dessus requiert un temps de calcul prohibitif (quelques minutes sur station de travail individuelle performante) qu'il convient de réduire. L'originalité de la méthode que nous proposons consiste précisément à simplifier les calculs et à accélérer l'évaluation du voisinage tout en ne perdant qu'un minimum d'informations essentielles sur celui-ci.

Un premier examen de la méthode nous a révélé que plus de 90% du temps de calcul était passé à construire des affectations optimales, bien que nous ayons utilisé un des algorithmes les plus performants pour les trouver, l'algorithme dit « hongrois » (cf. Papadimitriou et Steiglitz [79]). Cependant, vu le peu de choses qu'un mouvement change, on observe que bien souvent l'affectation est identique d'une solution à la suivante ; comme l'algorithme hongrois construit une affectation de a à z, il est plus judicieux de recourir à un algorithme vérifiant si une affectation donnée est optimale et la modifiant si elle ne l'est pas. Un tel algorithme, basé sur des méthodes de transbordement, peut être trouvé dans Bradley et al. [8] et bien qu'il soit d'un ordre de complexité supérieur, $O(m^4)$, il accélère d'un facteur d'environ 5 l'examen du voisinage.

Cependant, l'obtention d'affectations optimales demande encore plus de 70% du temps de calcul ; comme on ne peut espérer trouver une procédure beaucoup plus rapide, il convient de réduire drastiquement le nombre d'affectations calculées. Si π est la permutation optimale de la solution courante, c'est-à-dire si $\pi(k)$ donne le véhicule qu'il faut attribuer à la tournée T_k, nous avons choisi de ne pas remettre en cause cette affectation dans l'évaluation des mouvements éligibles. Donc, le calcul de la valeur d'un mouvement (prendre la commande i de la tournée T_j pour l'inclure dans la tournée T_k) se fait simplement en évaluant les nouveaux coefficients $b_{j(j)}$ et $b_{k(k)}$. L'affectation optimale ne se fait donc plus pour tous les mouvements éligibles mais seulement pour le mouvement élu, dont l'évaluation en sera notablement accélérée, au détriment d'une certaine perte de qualité dans l'élection du mouvement.

Pour rendre plus efficace l'examen du voisinage, l'étape suivante consiste à remarquer que seules deux tournées sont modifiées d'une itération à l'autre. Par conséquent, en mémorisant pour tout mouvement les modifications qu'il apporte aux deux tournées concernées, il suffit de recalculer les modifications des mouvements mettant en jeu une ou deux de ces tournées. Une telle technique permet d'accélérer encore une fois notablement l'examen du voisinage, au détriment uniquement de la place mémoire, au demeurant très raisonnable, nécessaire à stocker les modifications de tournées.

Une autre astuce que nous avons utilisée pour accélérer l'élection d'un mouvement est de n'examiner qu'environ un quart du voisinage ; pour être plus précis, à l'itération k, on essaie de ne changer de tournée que les commandes $\left((k-1) \cdot \left\lfloor \frac{n}{4} \right\rfloor \bmod n \right) + 1$ à $\left(\left(k \cdot \left\lfloor \frac{n}{4} \right\rfloor - 1 \right) \bmod n \right) + 1$. De cette manière, on réalise un examen partiel mais cyclique du voisinage, ce qui permettra d'élire plus rapidement un mouvement, au détriment de sa qualité puisque tous les mouvements ne sont pas pris en considération à chaque itération. Cependant, à un niveau global, cette restriction ne fait pas trop baisser la qualité des solutions produites et nous avons même observé, pour certains problèmes, que les solutions trouvées étaient meilleures qu'avec un examen complet du voisinage. Cet apparent paradoxe s'explique par le fait qu'un examen partiel peut engendrer une certaine diversité dans les solutions visitées, précisément parce que des mouvements qui

ont été élus ne l'auraient jamais été avec un examen complet du voisinage. Remarquons aussi que ce n'est souvent pas un mouvement très bon dans son évaluation qui améliore sensiblement la meilleure solution trouvée, mais au contraire une suite de mouvements, pas forcément bons, voire très mauvais dans leur évaluation, qui changent la structure des solutions visitées, offrant ainsi la possibilité d'aboutir à des solutions de qualité supérieure.

La dernière technique qui nous a permis d'accélérer la recherche est de réduire la taille du problème traité. En effet, le bon sens nous dicte souvent que certaines structures de solutions ont peu de chances d'être bonnes ; par exemple, il semble aberrant a priori de placer sur une même tournée deux clients éloignés du dépôt et à l'opposé par rapport à ce dernier ; par contre il pourrait sembler raisonnable d'imposer que deux commandes livrables en des lieux très rapprochés appartiennent à la même tournée. Sans aller si loin dans l'étude de chaque exemple de problème, nous avons adopté la politique suivante de réduction de la taille du problème : comme chaque client peut passer deux commandes, nous avons agrégé toutes les commandes devant être livrées au même client en une seule insécable, pour autant que celle-ci ne dépasse pas la capacité du plus petit véhicule pouvant se rendre chez ce client.

Le gain en temps de calcul obtenu par cette technique de réduction de la taille du problème est double car, d'une part le voisinage est plus petit, et d'autre part, l'ensemble des solutions admissibles est fortement réduit. Ce gain se fait au détriment de la valeur de la solution optimale, mais nous n'avons observé que dans de très rares cas une diminution de la qualité des meilleures solutions trouvées par une recherche travaillant sur un problème avec commandes agrégées.

L'ensemble de ces astuces pour réduire les temps de calcul nous a permis d'accélérer la recherche d'un facteur de plusieurs centaines car nous arrivons à effectuer quelques dizaines d'itérations en une seconde, alors que la méthode initiale nécessitait quelques minutes par itération.

Direction de la recherche

Nous avons envisagé plusieurs manières de diriger la recherche. Celle qui semble la plus appropriée est la suivante : Soit i, une commande du client c_i qui faisait partie de la tournée T_j et qui a été déplacée dans la tournée T_k ; on interdira pendant t itérations de déplacer sur la tournée T_j n'importe quelle commande issue du client c_i. Comme dans les adaptations précédentes, le paramètre t n'est pas fixé, mais choisi aléatoirement, uniformément entre deux bornes.

Les accélérations successives de l'évaluation du voisinage font que l'on pourrait passer à côté de bonnes solutions ; lorsqu'on se trouve dans une région prometteuse, nous avons décidé d'intensifier la recherche. En effet, les bonnes solutions sont caractérisées par un chargement important des véhicules, et faire l'hypothèse que l'affectation de ces derniers aux tournées ne change pas d'une itération à l'autre (dans l'évaluation des mouvements éligibles) devient irréaliste ; de même, l'examen partiel du voisinage ne se justifie plus. Soit s_0, s_1, … les solutions visitées par la recherche itérative décrite ci-dessus (sans intensification). Soit $\hat{s}_k = \underset{p = 1, \ldots, k}{opt} (s_p)$ la meilleure solution rencontrée par la recherche jusqu'à l'itération k. Pour tout $\hat{s}_k$, nous effectuons une courte recherche itérative dirigée semblable à celle décrite ci-dessus (condition d'arrêt : avoir effectué quelques itérations sans améliorer la meilleure solution trouvée) mais qui considérera le voisinage dans son intégralité et qui recalculera une affectation optimale pour chaque mouvement éligible. Cette courte recherche aura donc pour but de trouver l'optimum local associé à la solution $\hat{s}_k$.

Commentaires sur la méthode

Vu la spécificité du problème, il est difficile de présenter des résultats numériques parlants, puisqu'il faut une base de comparaison valable ; or ce problème n'apparaît dans la littérature qu'au travers de nos travaux. Nous n'avons d'autres moyens d'évaluer la méthode proposée que de comparer les solutions qu'elle produit avec celles qui ont été effectivement adoptées par l'entreprise. Comme on ne peut raisonnablement penser que cette dernière dilapide son argent intentionnellement, on est en droit de supposer que les

solutions qu'elle adopte ne sont pas trop mauvaises, en tout cas pas évidentes à améliorer à la main, et en effet, elles font usage en moyenne de presqu'un véhicule en moins que celles produites par notre recherche itérative dirigée, mais en violant souvent, bien que légèrement, certaines contraintes, ce que nous avons formellement banni de l'ensemble des solutions admissibles.

À part sur ce critère du nombre de véhicules engagés, qui n'entre pas dans la fonction-objectif de notre méthode car elle ne prend en compte que les frais occasionnés par l'usage des véhicules (coûts globaux par kilomètre parcouru), cette dernière produit de bien meilleures solutions si elle fonctionne suffisamment longtemps : en moyenne, les solutions obtenues ont un coût inférieur de 16% aux solutions adoptées par l'entreprise et les véhicules effectuent un parcours global moins long de 10%, ce qui représente tout de même quelques centaines de francs ou quelques centaines de kilomètres épargnés chaque jour.

Un des buts du présent travail était de faire ressortir un des avantages décisifs des recherches itératives, à savoir leurs capacités de traitement de problèmes pratiques comportant de multiples contraintes ; un autre était de montrer qu'une implantation naïve, bien que parfaitement cohérente, pouvait se révéler très décevante au niveau de ses performances. Nous avons pu illustrer quelques astuces permettant d'accélérer énormément des recherches itératives, et c'est là qu'il faut chercher le côté original de notre démarche.

3.3.5. Problème académique de distribution de biens

Modélisation et voisinage

Nous avons à nouveau choisi une modélisation très simple du problème, restreinte aux solutions admissibles, auxquelles on ajoute des solutions où toutes les commandes ne sont pas livrées par l'introduction d'une tournée fictive T_0 de capacité infinie et dont le mode de livraison consiste en une succession d'allers et retours : dépôt-client-dépôt. D'autres

auteurs, Gendreau et al. [42], ont adopté une modélisation moins immédiate où des violations de contraintes sont autorisées mais pénalisées.

Comme nous avons opté pour une évaluation rapide du voisinage, nous avons pu en utiliser un plus étendu que pour le problème précédent : non seulement nous considérons le déplacement d'une commande d'une tournée dans une autre, mais aussi l'échange de deux commandes appartenant à des tournées différentes. L'évaluation des mouvements se fait par simple heuristique d'insertion : une commande est placée dans une tournée là où elle occasionne le détour le moins important ; l'ordre dans lequel les biens sont distribués dans une tournée perdant une commande n'est pas modifié, on saute directement de la commande précédant la commande enlevée à celle qui lui succède.

L'usage de cette heuristique simple implique que la qualité des tournées, c'est-à-dire l'ordre dans lequel on visite les clients, se détériore au fur et à mesure que l'on avance dans la recherche. Pour remédier à cela, nous proposons de réoptimiser les tournées périodiquement, ou lors de la visite de solutions particulièrement bonnes. Nous avons utilisé le programme de Volgenant et Jonker [96] pour réaliser ces optimisations exactes de tournées.

Pour accélérer l'examen du voisinage, nous avons à nouveau utilisé la technique de mémorisation des valeurs de mouvements, étant donné que seules deux tournées sont modifiées d'une itération à l'autre. On considère le nombre maximal m de véhicules engagés dans le problème comme étant donné ; si ce nombre n'est pas connu, il est en général aisé à déterminer.

Direction de la recherche

Pour diriger la recherche, nous avons utilisé à la fois le mécanisme d'interdiction de mouvements, réalisant une mémoire à court terme, et celui de pénalisation de mouvements fréquemment exécutés, implantant ainsi une mémoire à long terme. Si i est la commande qui appartient à la tournée T_j et qui est déplacée dans la tournée T_k, nous avons choisi d'interdire pendant t itérations de remettre i dans T_j. Le paramètre t est à

nouveau choisi aléatoirement, uniformément entre $\lfloor 0.4n \rfloor$ et $\lceil 0.6n \rceil$, où n est le nombre de commandes.

De manière analogue au problème de traitement à séquences fixées, nous avons pénalisé le mouvement qui déplace la commande i dans la tournée T_j de $u \cdot \Delta_k^{max} \cdot \sqrt{n \cdot m} \cdot fréq(i, T_j, k)$, où $fréq(i, T_j, k)$ est la fréquence, à l'itération k, à laquelle on a utilisé ce mouvement et Δ_k^{max} est l'évaluation maximale d'un mouvement durant les k premières itérations. Le paramètre u est lui aussi choisi aléatoirement, uniformément entre 0.1 et 0.5. Ce mécanisme de mémoire à long terme joue un rôle très important car sans y recourir, la recherche se comporte comme suit : lorsqu'un optimum local est atteint, le mouvement autorisé dégradant le moins la solution mettra en jeu des commandes proches du dépôt. Avec un paramètre t fixé à une valeur petite, la recherche aura tendance à ne déplacer que des clients proches du dépôt, rendant impossible un changement important dans la structure de la solution, intimement liée aux parcours effectués en périphérie. Inversement, si t est grand, en arrivant dans un optimum local, elle commencera par effectuer un certain nombre de mouvements à faible coût (positif ou négatif), qu'il n'est pas possible de « défaire » lorsqu'est effectuée une suite de mouvements de coût plus important et modifiant la structure de la solution. Ainsi, dans un cas, on ne visite que des solutions présentant toutes la même structure générale alors que dans l'autre, on visite des solutions variées mais jamais très bonnes. Conjugué à une valeur de t assez petite, le rôle de la mémoire à long terme revient à pénaliser les mouvements de clients situés près du dépôt, favorisant ainsi les déplacements de commandes en périphérie, qui modifient la structure des solutions ; comme t est relativement petit, il est toujours possible de bien visiter une vallée associée à une solution présentant une certaine structure, tout en pouvant s'en échapper après un certain temps grâce à la mémoire à long terme.

Comparaison avec d'autres méthodes

La méthode la plus proche de celle que nous venons de décrire est celle due à Osman [77]. Elle diffère en ceci qu'elle ne fait pas usage de mémoire à long terme et qu'à la place de

réoptimiser exactement et périodiquement les tournées, elle effectue sur les deux tournées modifiées une optimisation à chaque itération par échanges successifs de deux arêtes, selon la méthode heuristique de Lin et Kernighan [67]. Une autre différence entre cette méthode et la nôtre est qu'elle part d'une solution admissible, obtenue par l'heuristique de Clark et Wright [21], et qu'elle ne visite que des solutions admissibles (pas de commande non livrée).

La seconde recherche itérative efficace que nous devons comparer à la nôtre est celle de Gendreau et al. [42]. Cette méthode présente la particularité d'utiliser un voisinage très simple (déplacement d'une commande d'une tournée sur une autre seulement) mais sur un domaine de solutions étendu, autorisant, tout en les pénalisant, des solutions violant des contraintes de capacité des véhicules ou de longueur maximale des tournées.

L'évaluation de la longueur des tournées modifiées se fait au moyen de méthodes heuristiques spécialement développées pour cette application et décrites par les mêmes auteurs dans [41]. Afin de réduire la taille du voisinage, l'évaluation des mouvements n'est pas réalisée pour toute commande et toute tournée, mais seul est évalué le déplacement d'un sous-ensemble des commandes, tiré aléatoirement, dans les tournées les plus proches des commandes concernées.

Sur notre conseil, les auteurs de cette méthode ont introduit une mémoire à long terme fonctionnant de manière tout à fait similaire à notre pénalisation sur la fréquence d'utilisation des mouvements.

Nous remarquons que cette méthode est assez évoluée et que si elle fonctionne un peu plus lentement que la nôtre sur de petits problèmes (50 – 70 commandes, 5 – 10 véhicules), elle est en revanche plus efficace pour de gros problèmes. Cependant, notre méthode a été développée dans l'esprit de traiter de petits sous-problèmes de distribution, issus de la décomposition d'un problème de taille plus grande. Ces sous-problèmes présentent la particularité d'englober des commandes proches les unes des autres, rendant par là pertinente l'utilisation d'un voisinage complet qui consiste à essayer (à l'intérieur d'un sous-problème) de déplacer n'importe quelle commande dans n'importe quelle tournée.

Nous décrirons plus en détail dans le chapitre suivant comment décomposer les problèmes de distribution de biens, et nous y donnerons les résultats numériques qui montreront qu'en général, sur les problèmes étudiés, la méthode que nous proposons est plus efficace que les deux autres auxquelles elle est comparée ci-dessus.

L'originalité de notre méthode, si l'on excepte les techniques de décomposition qui seront abordées dans le chapitre suivant, se manifeste dans l'usage parcimonieux, mais nécessaire, que nous faisons d'un coûteux algorithme exact d'optimisation des tournées de véhicules. L'implantation de la mémoire à long terme ainsi que l'utilisation de paramètres aléatoirement modifiés en cours de recherche est aussi à signaler.

3.4. SYNTHÈSE DES RÉSULTATS OBTENUS

Dans ce chapitre, nous nous sommes tout d'abord penché sur le rôle de certains paramètres de direction des recherches itératives. L'idée primordiale retenue pour les diriger à court terme est d'interdire d'effectuer pendant t itérations le mouvement ne faisant qu'annuler celui que l'on vient de faire. Nous avons étudié l'influence de la valeur du paramètre t et montré que cette technique de direction est insuffisante pour garantir à une recherche itérative de ne pas visiter cycliquement le même sous-ensemble de solutions, quelle que soit la valeur de t, bien que cette politique très simple semble pourtant efficace en pratique.

Nos expériences numériques ont réussi à montrer que le choix d'un t fixé pour toute la durée de la recherche ne mène pas à des méthodes robustes et que le simple fait de choisir aléatoirement le nombre d'itérations pendant lesquelles interdire certains mouvements permet dans bien des cas d'éviter des cycles.

Lorsque l'ensemble des mouvements potentiellement applicables à toute solution admissible ne dépend pas de la solution, nous avons identifié une manière de diriger la recherche à long terme, en pénalisant simplement les mouvements d'un facteur proportionnel à leur fréquence d'utilisation. Une autre technique que nous proposons pour

diversifier la recherche est d'effectuer, quelle que soit son évaluation, un mouvement qui n'a pas été élu durant un grand nombre d'itérations.

Finalement, au travers d'applications sur des problèmes concrets, nous avons pu illustrer quelques techniques de structuration des informations gérées par les recherches itératives, techniques qui ont permis de multiplier le nombre d'itérations effectuées par unité de temps d'un facteur parfois énorme. Les solutions excellentes obtenues sur des problèmes classiques de la littérature sont dues tant aux politiques de direction qu'à l'implantation soignée de nos méthodes, ce qui nous permet d'effectuer un très grand nombre d'itérations en des temps raisonnables.

4. RECHERCHES ITÉRATIVES DIRIGÉES PARALLÈLES

L'obtention de bonnes solutions avec une recherche itérative dirigée se paie cher en temps de calcul ; comme il est fréquent d'avoir à réaliser plusieurs dizaines de milliers d'itérations, tout gain de temps est le bienvenu. En supposant que la recherche est déjà programmée de manière optimale sur un calculateur doté d'un unique processeur (il existe actuellement des utilitaires très efficaces pour optimiser les codes de programmes, on peut citer en particulier les « profileurs » qui permettent d'analyser avec précision le temps passé dans chaque instruction d'un programme), on est donc obligé de faire appel à un ordinateur possédant plusieurs processeurs pour accélérer la recherche. Il n'y a qu'une alternative pour augmenter le débit du nombre d'itérations effectuées par unité de temps : soit accélérer les calculs nécessaires pour effectuer une itération, soit réaliser plusieurs mouvements simultanément.

La première voie suppose une parallélisation du code utilisé pour effectuer une itération, parallélisation qui se fait souvent en décomposant les boucles que le code contient, afin d'évaluer tant la fonction-objectif que la valeur d'un mouvement ou le choix du meilleur mouvement. La seconde manière revient à décomposer ou à dupliquer le problème ; s'il est possible d'accomplir plusieurs mouvements simultanément sur le même problème, c'est que ceux-ci sont indépendants et que le problème est susceptible d'être décomposé. La duplication de ce dernier peut sembler farfelue à premier abord, mais nous en avons déjà parlé de manière cachée lorsqu'on intensifiait la recherche : pour le problème réel de distribution de biens, nous considérions une recherche accélérée à laquelle on prélevait

les solutions jugées prometteuses, pour trouver l'optimum local associé à ces solutions par un processus indépendant et qui était aussi une recherche itérative dirigée.

Dans le cas précis qui vient d'être abordé, les deux recherches itératives ne sont pas identiques, mais rien n'empêche de choisir des processus effectuant un travail similaire, se distinguant par exemple uniquement par un germe de générateur aléatoire différent ; c'est ce que proposent de faire Aarts et al. dans [1] pour paralléliser un recuit simulé : un premier processus est lancé ; après un certain temps, on prend une solution qu'il a produite comme solution initiale d'un second processus qui démarre avec une température et un germe différents, et ainsi de suite. Malheureusement, si les processus d'optimisation sont des recherches itératives dirigées, nous ne sommes pas parvenu à identifier la solution qu'il faut choisir pour lancer le processus suivant et les politiques essayées n'ont jamais réussi à surpasser de façon convaincante la méthode que nous présentons en dernière partie de ce chapitre et qui consiste à faire démarrer tous ces processus en même temps avec des solutions ou des germes initiaux différents.

4.1. PARALLÉLISATION DE BAS NIVEAU

Dans cette partie, nous supposerons que l'évaluation de la fonction-objectif ou de la valeur d'un mouvement est un travail conséquent qui peut être parallélisé et nous illustrerons ce type de parallélisation sur deux problèmes : celui du traitement à séquences fixées et celui de distribution de biens.

4.1.1. Problème de traitement à séquences fixées

Dans l'adaptation de recherche itérative que nous avons vue pour ce problème, l'effort de calcul le plus important est d'évaluer les heures de début au plus tôt d_i des opérations ($i = 1, \ldots, n$). Nous rappelons ici l'algorithme utilisé pour calculer les d_i (en reprenant les notations du paragraphe 1.2.1, page 7) :

Algorithme pour calculer les d_i :

— Introduire toutes les premières opérations à effectuer sur chaque objet dans un ensemble Q d'opérations pour lesquelles les heures de début au plus tôt sont calculables.

— Répéter tant que $Q \neq \emptyset$:

Marquer $i \in Q$.

$Q := Q \setminus \{i\}$.

$d_i = max(d_{pm_i} + t_{pm_i}, d_{po_i} + t_{po_i})$.

Si pm_{so_i} est marqué (ou n'existe pas) poser $Q := Q \cup \{so_i\}$.

Si po_{sm_i} est marqué (ou n'existe pas) poser $Q := Q \cup \{sm_i\}$.

Il est clair que l'on peut calculer simultanément tous les débuts au plus tôt des opérations se trouvant à un moment donné dans l'ensemble Q ; notons qu'il peut y avoir au plus m opérations dans Q (m : nombre de machines) car il ne peut y en avoir plus d'une par machine. Une parallélisation de cet algorithme pourra mener à une accélération d'au plus m (l'accélération étant définie comme le rapport entre le temps d'exécution d'un algorithme séquentiel et celui d'un algorithme parallèle). Pour réaliser cette parallélisation, nous proposons de créer un processus par machine, qui exécutera l'algorithme suivant :

Processus *j*:

— Pour toute première opération i d'un objet devant s'exécuter sur la machine j, faire :

Envoyer aux processus m_{so_i} la valeur provisoire t_i et marquer i

— Répéter tant que tous les d_i ne sont pas calculés :

Attendre une valeur provisoire de début au plus tôt d'une opération i telle que $m_i = j$.

Mémoriser cette valeur.

Pour toute opération i pour laquelle la valeur provisoire est connue et d_i calculable (i.e. d_{pm_i} déjà calculé), marquer i, calculer d_i, envoyer au processus m_{so_i} (s'il existe) la valeur provisoire $d_i + t_i$.

Nous avons implanté cet algorithme sur un réseau de processeurs appelés « Transputer » ; pour ce faire, nous avons connecté *m* de ces processeurs sous la forme d'une configuration en anneau (un Transputer ne disposant que de quatre connections avec d'autres processeurs, il n'est pas possible d'avoir des connections directes entre tous), et par conséquent, nous avons été obligé d'implanter aussi un contrôleur de communications, les messages qu'un processus reçoit ne lui étant pas forcément adressés. En figure 18, nous

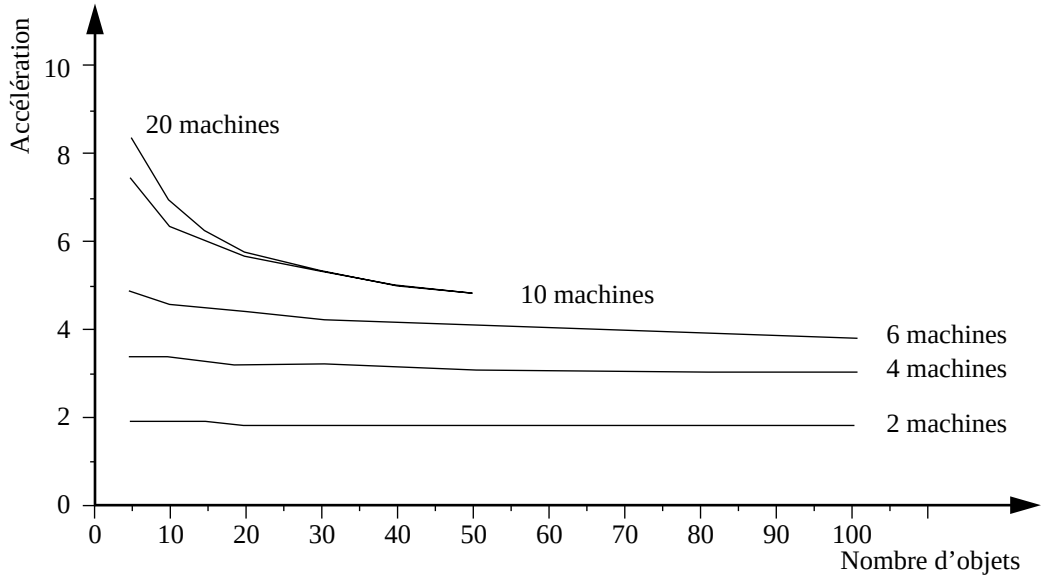

Figure 18 : Accélération de la recherche itérative pour divers nombres de processeurs.

représentons, en fonction du nombre d'objets, l'accélération qu'il est possible d'obtenir pour quelques valeurs de *m*. Nous remarquons que jusqu'à 6 machines, une telle parallélisation est efficace puisqu'il est possible d'atteindre des efficacités supérieures à 70% (l'efficacité étant le rapport entre l'accélération et le nombre de processeurs). Cependant, si le nombre de machines et le nombre d'objets augmentent, les performances diminuent car le réseau de communications se sature rapidement. Remarquons toutefois que les calculs devant être effectués sont extrêmement simples (une addition et une comparaison avant l'envoi d'informations). Si le problème traité avait été plus compliqué (comme cela n'aurait certainement pas manqué d'être le cas s'il s'était agi d'un problème réel avec changements d'outils, machines parallèles, etc.), les calculs à effectuer par

chaque processus auraient pu être plus compliqués et les problèmes de communication relégués au second plan, rendant par là pertinente notre approche de parallélisation.

4.1.2. Problème de distribution de biens

Comme dans notre méthode deux tournées sont modifiées à chaque itération et que trouver le nouvel ordre dans lequel il faut livrer les biens revient à un problème de voyageur de commerce, il n'est pas difficile d'imaginer que le calcul parallèle de l'évaluation d'un mouvement pourra notablement accélérer la recherche itérative. Nous allons voir ici que même la très simple procédure d'insertion que nous avons utilisée peut se paralléliser efficacement avec quelques processeurs.

Lorsqu'on déplace la commande i dans la tournée T_k, l'algorithme d'insertion consiste simplement à trouver la valeur : $\min_{j = 0 \ldots |T_k|} (d_{ji} + d_{i\,j+1} - d_{j\,j+1})$, si l'on suppose que la tournée T_k contient les commandes 1 à $|T_k|$ et que 0 et $|T_k| + 1$ représentent les deux le dépôt.

En connectant 3 ou 7 Transputers selon une topologie d'arbre binaire, nous avons représenté en figure 19 les accélérations qu'il est possible d'obtenir en fonction de $|T_k|$

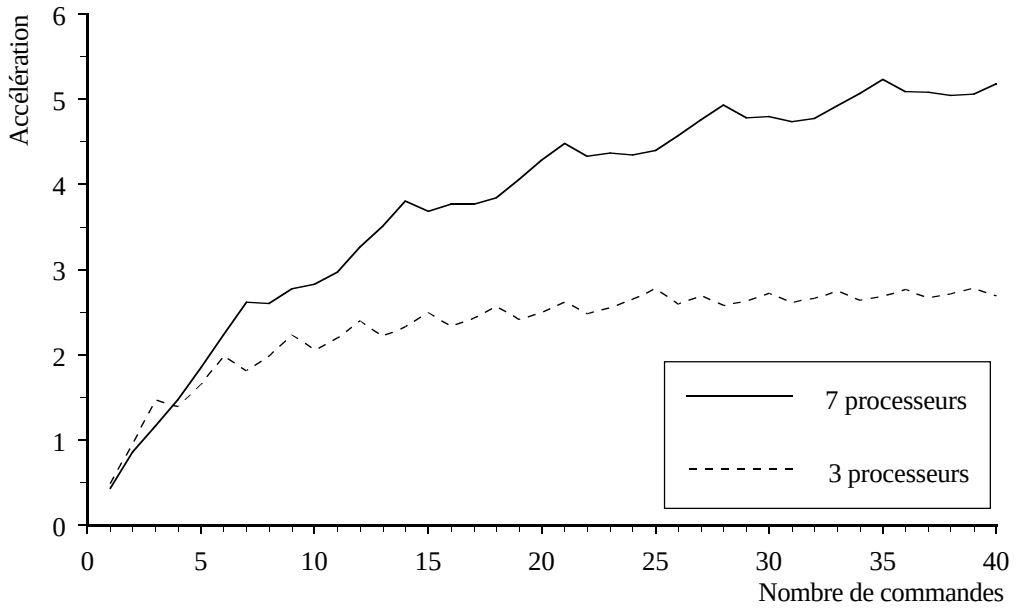

Figure 19 : Accélération de la procédure d'insertion.

pour déterminer l'endroit où insérer une commande sur la tournée T_k. Nous voyons dans cette figure que si les tournées contiennent plus de 10 à 20 commandes, la parallélisation peut devenir intéressante ; et elle le sera d'autant plus qu'une méthode plus onéreuse est utilisée, à condition que le travail puisse être réparti entre divers processeurs. À nouveau, nous n'avons envisagé que le cas le plus simple ; en examinant le problème réel de distribution de biens on pourra se convaincre que pour une application pratique, les calculs peuvent sérieusement se compliquer, et par là rendre souhaitable une accélération de l'évaluation des mouvements.

4.2. PARALLÉLISATION DE L'EXAMEN DU VOISINAGE

Si le calcul de la valeur d'un mouvement est un processus coûteux qui ne peut pas se paralléliser, il est toujours possible de procéder concurremment à l'élection du meilleur mouvement du voisinage. Cependant, cette élection n'est pas toujours triviale à réaliser en parallèle, et nous le montrerons dans le paragraphe suivant à propos du problème d'affectation quadratique.

4.2.1. Problème d'affectation quadratique

Pour ce problème, l'évaluation d'un mouvement peut être très rapide, en $O(1)$ si l'on peut utiliser la formule (8), page 64, ou plus onéreuse, en $O(n)$, si l'on est obligé d'utiliser la formule (7). De manière analogue à ce qui a été fait dans la section précédente, il serait évidemment possible d'utiliser un algorithme parallèle pour évaluer la formule (7), et cet algorithme aurait au mieux une complexité en $O(\log n)$. Sans aller si bas dans la programmation parallèle, nous avons choisi de décomposer l'ensemble des mouvements, de taille $O(n^2)$ en $O(n)$ sous-ensembles de taille $O(n)$. Il est important de répartir équitablement le travail entre processeurs et par conséquent, il faut que chacun de ces sous-ensembles ne contienne que $O(1)$ mouvements demandant un calcul par la formule (7). Pour réaliser ceci, il est évident qu'une décomposition dynamique de l'ensemble des mouvements est toujours possible. Cependant, pour limiter les échanges d'informations entre processeurs, on peut décomposer l'ensemble des mouvements,

pouvant être représenté par l'ensemble des couples non ordonnés (i, j) avec $1 \leq i < j \leq n$, en $n - 1$ (respectivement n lorsque n est impair) sous-ensembles comportant chacun $n/2$ (respectivement $(n - 1)/2$) mouvements, chacun de ces sous-ensembles ne comportant jamais deux indices identiques. Lorsque n est pair, le $k^{\text{ième}}$ de ces sous-ensembles, $k = 0, \ldots, n - 2$ sera composé des mouvements :

$$\{(n,\ 1 + k),\quad (((n - v + k - 1) \text{ modulo } (n - 1)) + 1,\ ((v + k) \text{ modulo } (n - 1)) + 1)\},$$
$$v = 1, \ldots, n/2 - 1$$

Il est facile de constater que la réunion de ces sous-ensembles donne le voisinage complet et que chacun ne contient qu'une fois chaque indice de 1 à n. Lorsque n est impair, le $k^{\text{ième}}$ de ces sous-ensembles, $k = 0, \ldots, n - 1$ sera composé des mouvements :

$$\{(((n - v + k) \text{ modulo } n) + 1,\ ((v + k) \text{ modulo } n) + 1)\},\ v = 1, \ldots, (n - 1)/2$$

Ainsi, chacun de ces sous-ensembles n'aura qu'au maximum deux mouvements à calculer en $O(n)$ avec la formule (7). Il est donc possible, avec $O(n)$ processeurs, d'évaluer en $O(n)$ l'ensemble du voisinage. En utilisant $n/10$ Transputers, nous avons réussi à obtenir des efficacités tout à fait honnêtes, comme on peut le voir en figure 20 où l'on a représenté, en fonction de la taille du problème, l'efficacité de notre parallélisation.

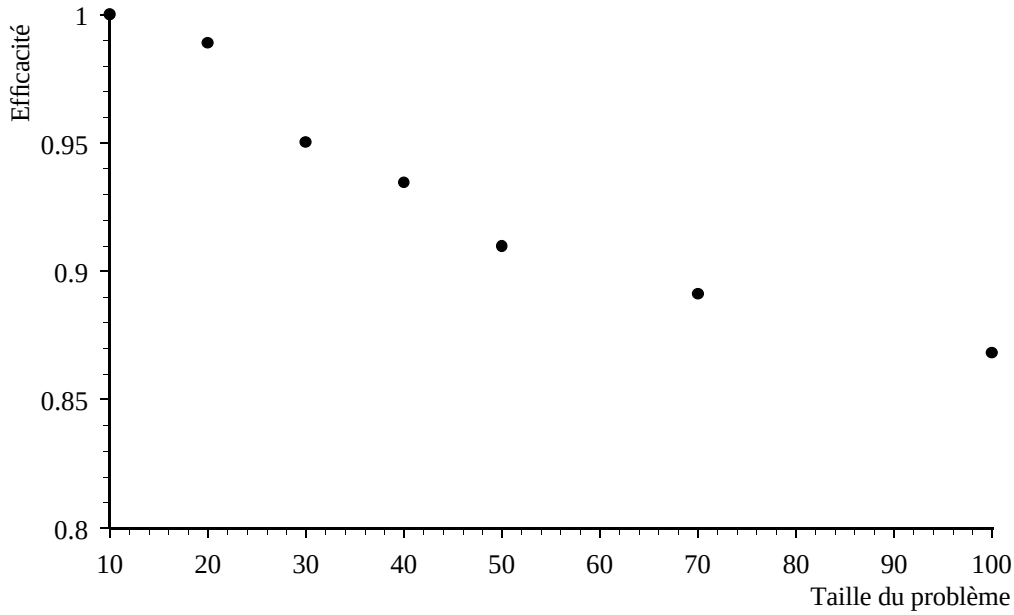

Figure 20 : Efficacité de la parallélisation de l'examen du voisinage avec n/10 processeurs.

En utilisant une décomposition plus fine, on arriverait à évaluer le voisinage en $O(\log n)$ avec $O(n^2/\log n)$ processeurs ; Chakrapani et Skorin-Kapov ont une implantation sur « Connection Machine », présentée dans [16], qui réalise précisément cette évaluation en $O(\log n)$, mais en utilisant $O(n^2)$ processeurs.

4.3. DÉCOMPOSITION DU PROBLÈME

Lorsque la taille du problème que l'on cherche à résoudre augmente, il peut devenir très onéreux d'évaluer le voisinage ou même de trouver une solution admissible. Dans un premier paragraphe, nous présenterons une méthode de décomposition de très grands problèmes de traitement à séquences fixées, basée sur une recherche itérative dirigée, qui permet de trouver de bonnes solutions tout en étant plus rapide qu'une simple heuristique gloutonne.

Dans un second paragraphe, nous proposerons des méthodes de décomposition qui nous ont permis de trouver toutes les meilleures solutions connues des problèmes de distribution de biens posés par Christofides et al. dans [18].

4.3.1. Problème de traitement à séquences fixées

Dans ce paragraphe, nous traiterons le cas de problèmes où le nombre d'objets n est beaucoup plus grand que le nombre de machines m. Nous avons vu dans le paragraphe 3.3.2 que ces problèmes pouvaient être résolus de manière très satisfaisante par recherche itérative dirigée. Cependant, si n est très grand (plus de 1000 objets), il devient même très onéreux de trouver une solution admissible de ce problème puisque l'algorithme glouton est en $O(n^2m)$. Une manière de le décomposer est de créer des lots de $g(m)$ objets. Nous avons vu en effet, dans le cas de problèmes générés aléatoirement, que les problèmes ayant plus de $n = 5m$ objets devenaient faciles à résoudre. ($g(m) = 5m$ n'est certainement pas une fonction universelle ; cette dernière doit dépendre aussi de la façon de générer les problèmes.)

Par conséquent, s'il est possible d'obtenir en $O(h(m))$ une très bonne séquence de traitement des opérations des $g(m)$ objets d'un lot, on est en droit d'espérer qu'en $O(h(m)n/g(m))$ on puisse trouver une bonne séquence de traitement des opérations de tous les objets du problème, simplement en plaçant les $n/g(m)$ lots les uns après les autres, indépendamment.

Naturellement, on n'obtiendra pas forcément une solution optimale, mais, en prenant $g(m) = 5m$ et pour des problèmes où les temps de travail des opérations sur les machines sont tirés aléatoirement entre 80 et 100, nous avons pu trouver des solutions bien meilleures que celles fournies par un algorithme glouton tout en ayant besoin d'un temps de calcul moindre. En figure 21, nous avons représenté la qualité des solutions obtenues,

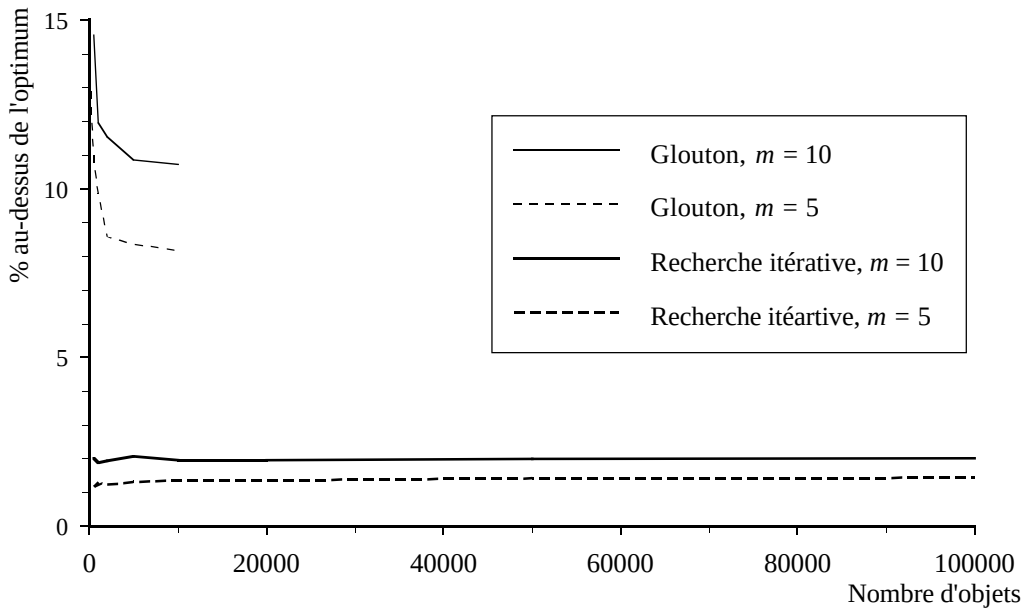

Figure 21 : Qualité des solutions obtenues par deux heuristiques sur de très grands problèmes.

en pour cent au-dessus d'une borne inférieure à la valeur d'une solution optimale et en fonction du nombre d'objets, pour $m = 5$ et $m = 10$. Nous voyons dans cette figure que l'algorithme glouton est beaucoup plus mauvais que la méthode basée sur la décomposition du problème et que cette dernière fournit des solutions de qualité constante, quel que soit le nombre d'objets considérés. Comme le temps de calcul

devenait trop important pour lui, nous n'avons pas traité les problèmes de plus de 10'000 objets avec l'algorithme glouton.

Lorsqu'on parle de recherches itératives dirigées, on les associe souvent à des méthodes coûteuses en temps de calcul et il est original d'avoir mis en évidence qu'elles pouvaient aussi mener à des méthodes de faible complexité, plus performantes à tout point de vue que de simples algorithmes gloutons.

4.3.2. Problème de distribution de biens

Mieux que tout autre problème abordé dans ce travail, celui de distribution de biens se prête à être décomposé parce que les commandes à livrer le sont en des lieux pouvant souvent être localisés sur une carte de géographie et qu'il est par là naturel de les regrouper par secteurs. Notons cependant que ce regroupement, si naturel apparaisse-t-il à première vue, ne correspond à aucune réalité en général, puisqu'il est tout à fait possible de créer des problèmes où les tournées des véhicules sont totalement entrelacées dans une solution optimale ; il suffit pour cela de considérer un ensemble de commandes, proches les unes des autres en regard de leur distance par rapport au dépôt et dont les volumes correspondent à un problème de partition en sous-ensembles de poids donnés, d'un ensemble dont on connaît le poids de chaque élément. Toutefois, les problèmes auxquels nous sommes confronté n'ont rien à voir avec ces hypothétiques ravitaillements de satellites artificiels, mais présentent au contraire la caractéristique d'avoir un certain nombre de commandes proches du dépôt, ce qui laisse une certaine flexibilité dans le choix de bonnes solutions, puisqu'il est toujours possible d'ajouter à moindre coût un véhicule supplémentaire desservant des commandes proches du dépôt et permettant ainsi de libérer un volume dans les autres véhicules, volume qui augmentera la marge de manœuvre d'une recherche itérative.

Nous supposerons donc par la suite que les problèmes de grande taille peuvent être légitimement décomposés, tout en étant conscient qu'il n'y a plus aucune garantie de pouvoir aboutir à une solution optimale. Simchi-Levi et Bramel ont montré théoriquement dans [87] qu'il était effectivement possible sous certaines conditions de décomposer les

problèmes comprenant un grand nombre de véhicules, tout en conservant des solutions dont les valeurs asymptotiques sont celles des solutions optimales.

Taille des sous-problèmes à créer

Les méthodes les plus efficaces disponibles actuellement permettent de trouver routinièrement les solutions probablement optimales de problèmes comprenant au plus 50 à 100 commandes réparties sur 5 à 10 véhicules ; la difficulté d'un problème de taille donnée peut être très variable selon la répartition spatiale et volumique des commandes, les contraintes sur la longueur maximale des tournées, … Ainsi, des problèmes à 100 commandes et plus peuvent être résolus en quelques secondes alors qu'un exemple à 75 commandes n'a pas pu être résolu de manière optimale par les méthodes de Gendreau et al. [42] et Osman [77], malgré un effort de calcul beaucoup plus important. Dans ce contexte, on comprend bien les avantages d'une décomposition des problèmes plus gros : d'une part, une recherche itérative travaillant globalement sur un problème de grande taille sera difficile à diriger car il faut à la fois pouvoir changer la structure générale des solutions et obtenir simultanément pour toutes les régions du problème des tournées logiquement organisées ; d'autre part, travailler sur de petits problèmes permet de recourir à un voisinage de faible taille ce qui diminue notablement le temps mis pour effectuer une itération. La limitation du voisinage proposée par Gendreau et al. [42] (limiter les essais de déplacements des commandes dans les quelques tournées les plus proches) est certainement un pas dans la bonne direction, mais insuffisant à nos yeux : en effet, s'il est logique de ne tester l'insertion d'une commande située en périphérie que dans deux ou trois des tournées (aussi périphériques) les plus proches, une commande située près du dépôt peut tout aussi logiquement être desservie par une tournée momentanément périphérique. Une autre faiblesse des méthodes de Gendreau et al. et Osman est de ne pas pouvoir assurer qu'un ensemble restreint de tournées soit logiquement organisé, alors que la même recherche itérative pourrait très facilement trouver des améliorations si elle était appliquée à un sous-ensemble de quelques tournées de la meilleure solution qu'elle a trouvée en travaillant sur le problème complet.

Décomposition de problèmes euclidiens réguliers

L'idée à la base de notre méthode de décomposition de gros problèmes est de considérer une solution admissible (avec éventuellement des commandes non livrées) et de grouper quelques tournées pour les optimiser séparément des autres. Afin de pouvoir utiliser plusieurs processus pour réaliser ces optimisations simultanément, nous décomposerons une solution en plusieurs ensembles de tournées indépendants. Naturellement, un de ces sous-ensembles ne devra contenir que des tournées proches les unes des autres. Dans le cas de problèmes euclidiens, pour définir la proximité d'une tournée par rapport à une autre, nous avons utilisé la notion de centre de gravité : à la place de l'ensemble des commandes d'une tournée, on considérera le centre de gravité des lieux où elles doivent être livrées. Pour créer des sous-ensembles qui ne s'intersectent pas deux à deux et dont la réunion comporte l'intégralité des commandes du problème initial, nous avons adopté l'algorithme de décomposition en $s \cdot c$ régions polaires suivant (où s est le nombre de secteurs et c le nombre de partitions des secteurs) :

Algorithme de décomposition :

- Tirer $\boldsymbol{g}$, un vecteur aléatoire de longueur non nulle.
- Calculer les coordonnées polaires (θ_i, ρ_i) du centre de gravité de chaque tournée i ($i = 1, \ldots, m$),
  θ_i étant l'angle entre les vecteurs (dépôt $\rightarrow$ centre de gravité de la tournée i) et $\boldsymbol{g}$,
  ρ_i étant l'éloignement du centre de gravité par rapport au dépôt.
- Trier les tournées par θ_i croissants ; on supposera donc que $\theta_i \leq \theta_{i+1}$ ($i = 1, \ldots, m - 1$).
- Attribuer au secteur j les tournées $\left\lfloor \frac{m}{s}(j-1) + 1.5 \right\rfloor$ à $\left\lfloor \frac{m}{s} j + 0.5 \right\rfloor$ $(1 \leq j \leq s)$.
- Renuméroter les tournées attribuées à chaque secteur par rayon ρ_i croissant ; pour le secteur j qui contient m_j tournées, nous aurons dorénavant $\rho_i \leq \rho_{i+1}$ ($i = 1, \ldots, m_j - 1$).
- Attribuer à la région k du secteur j les tournées $\left\lfloor \frac{m_j}{c}(k-1) + 1.5 \right\rfloor$ à $\left\lfloor \frac{m_j}{c} k + 0.5 \right\rfloor$ $(1 \leq k \leq c)$.
- Attribuer une commande non livrée au sous-problème qui contient la tournée dont le centre de gravité est le plus proche du lieu de livraison de la commande.

Nous avons supposé dans cet algorithme que le nombre *m* de véhicules disponibles était précisément égal au nombre de tournées formées. Si ce n'est pas le cas, c'est-à-dire s'il existe des véhicules inutilisés, on pourra les répartir aléatoirement entre les sous-problèmes créés par l'algorithme. Ce dernier produit une partition aléatoire du problème, dépendant du vecteur $\boldsymbol{g}$ tiré au début. Ainsi, une fois que l'on estime avoir bien résolu chaque sous-problème, il est possible de l'utiliser à nouveau, en partant de la meilleure solution trouvée par la recherche jusqu'à ce point, pour obtenir une nouvelle partition. Dans le cas où une ébauche de solution n'existe pas (pas de tournées déjà formées), on peut utiliser un algorithme tout à fait similaire pour grouper les commandes et former ainsi les sous-problèmes sur lesquels on va travailler. On retrouve ainsi la méthode de décomposition de Marchetti Spaccamela et al. [68]. L'avantage de ce type de décomposition par rapport à la partition de Karp [59] en rectangles est qu'il engendre une plus grande diversité de découpes.

Résolution des sous-problèmes

Ce qu'il reste à définir, c'est l'effort qu'il faut consacrer à la résolution des sous-problèmes : si peu d'itérations sont effectuées avant une nouvelle décomposition, la méthode trouvera rapidement des solutions relativement bonnes mais aura de la peine à les améliorer lors des décompositions subséquentes ; inversement, si un effort important est consenti pour la résolution des sous-problèmes, il faudra patienter un certain temps avant d'obtenir une bonne solution mais il sera alors certainement possible de l'améliorer par la suite. Nous avons illustré ce phénomène en figure 22 où nous comparons (en pour cent au-dessus de la valeur de la meilleure solution connue) en fonction du temps de calcul global, les solutions que l'on obtient en effectuant 50 ou 500 itérations avant de redécomposer en quatre parties un problème dû à Christofides et al. [18] et comprenant 199 commandes. Nous voyons que dans un cas on obtient en quelques dizaines de secondes des solutions admissibles mais pas très bonnes et qu'il est difficile d'améliorer puisqu'en 10 minutes les valeurs de solutions restent à plus de 4% au-dessus de la meilleure valeur connue, alors que dans l'autre cas, il faut attendre une centaine de secondes pour obtenir une solution admissible à environ 5% au-dessus de la meilleure

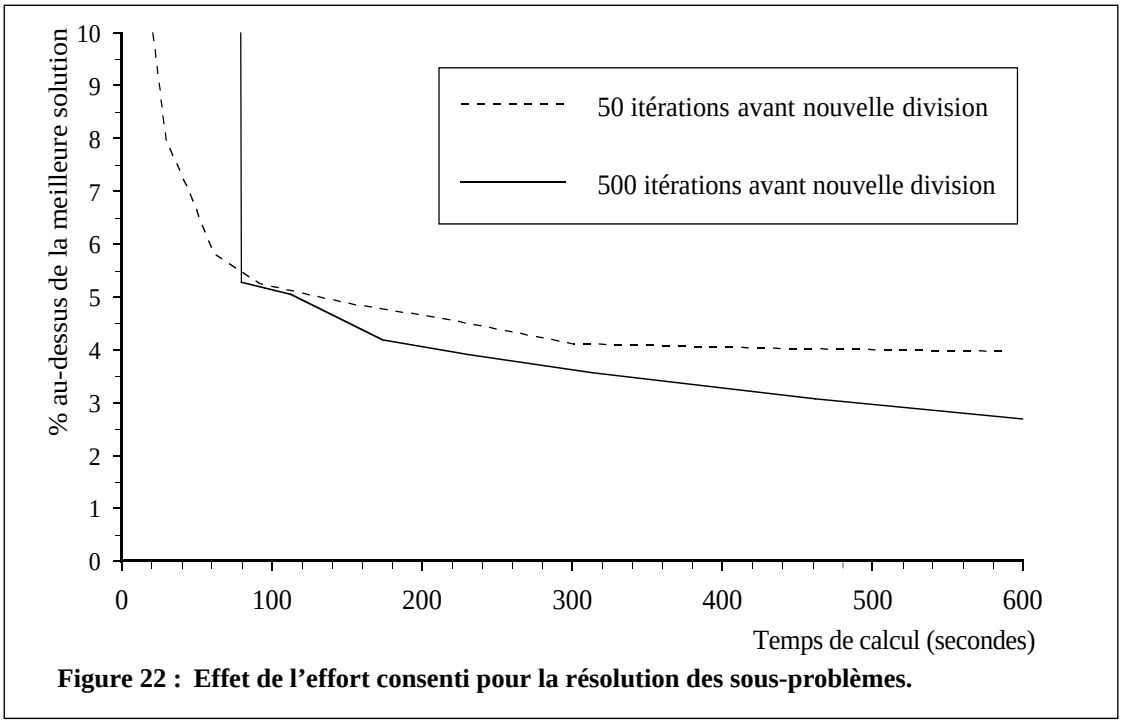

Figure 22 : Effet de l'effort consenti pour la résolution des sous-problèmes.

solution connue mais qu'après 10 minutes de calcul il est possible de descendre à moins de 3%.

Pour bénéficier à la fois de l'avantage d'un petit effort entre deux partitions (obtenir rapidement une solution honorable) et d'une recherche plus fouillée permettant de réviser en profondeur l'organisation des tournées à l'intérieur d'un sous-problème, nous avons opté pour une augmentation progressive du nombre d'itérations entre deux découpes. Ainsi, après la $d^{\text{ième}}$ découpe, nous effectuons $\left\lfloor 2d \cdot \frac{n}{sc} \right\rfloor$ itérations ; par exemple, après la découpe initiale du problème en s secteurs décomposés eux-mêmes en c régions concentriques, le nombre d'itérations effectuées correspond environ au double du nombre de commandes attribuées à chaque sous-problème.

Résultats numériques et comparaisons avec d'autres méthodes

En figure 23, nous comparons sur le même problème notre méthode à celle de Gendreau et al. [42], dont le programme nous a été transmis par Hertz [54], en dessinant l'évolution moyenne de la valeur des solutions produites en fonction du temps de calcul. Nous remarquons que pour des temps très courts (20 à 60 secondes), la méthode de Gendreau

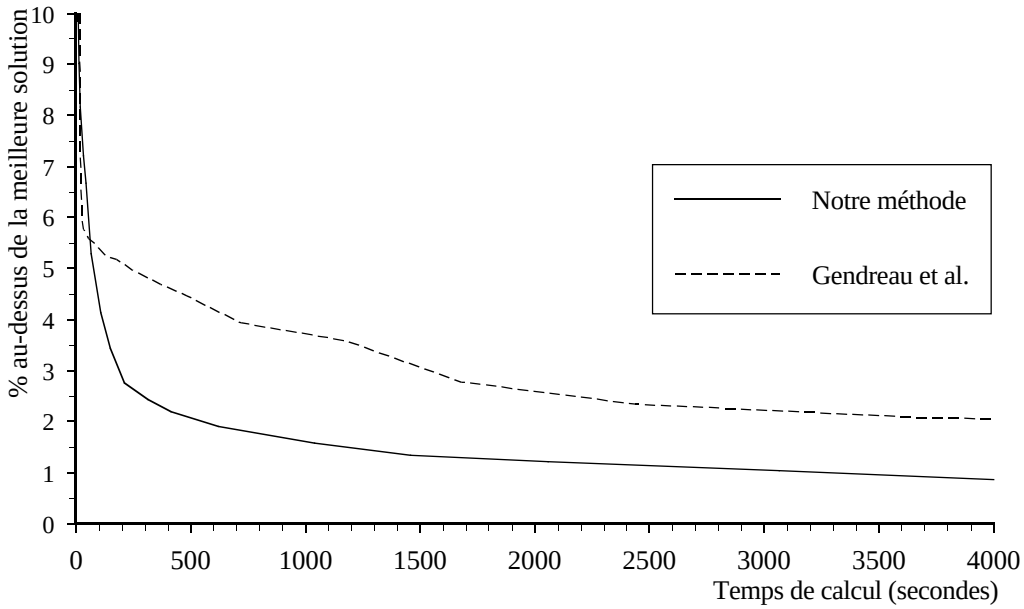

Figure 23 : Comparaison de deux méthodes sur un problème particulier.

et al. est meilleure, mais qu'elle a ensuite énormément de peine à trouver une solution à moins de 3% au-dessus de la meilleure solution connue, ce que notre méthode trouve très rapidement.

Pour obtenir ce graphique, nous avons exécuté 12 fois notre algorithme. Comme les partitions sont aléatoires, il s'en est suivi autant de solutions finales différentes (l'optimum de ce problème n'est certainement pas encore trouvé). Pour l'anecdote et pour illustrer le fait que trouver une bonne partition initiale des problèmes de distribution de biens n'est pas trivial, mentionnons que l'exécution ayant produit la meilleure solution après 2000 itérations en tout a produit la solution la moins bonne à la fin et que celle dont la solution était la moins bonne après 2000 itérations a fourni la meilleure solution en définitive.

Dans le tableau 8, nous comparons de manière plus exhaustive qu'en figure 23 notre méthode avec celle de Gendreau et al., en indiquant les temps nécessaires aux deux méthodes pour trouver des solutions en moyenne à 5%, 2% et 1% au-dessus de la meilleure solution connue des 14 problèmes proposés par Christofides et al. dans [18]. Dans ce tableau, un temps indiqué en italique signifie que l'autre méthode est nettement

			Partition en secteurs [95]			Implantation de Gendreau et al. [42]		
Nombre de commandes	Longueur limitée	Nombre de secteurs	5%	2%	1%	5%	2%	1%
50	—	1	*7*	*23*	*49*	6	53	> 90
75	—	2	*3*	*12*	*53*	19	> 110	—
100	—	2	12	68	580	*< 5*	*9*	*53*
150	—	3	86	*450*	3800	*18*	> 1800	—
199	—	4	*75*	*510*	*3000*	240	4600	—
50	oui	1	*2*	*5*	*17*	12	21	> 34
75	oui	2	*5*	*20*	*51*	33	> 99	—
100	oui	2	*11*	*120*	*1100*	190	310	> 1400
150	oui	3	*64*	*400*	*1100*	550	> 2800	—
199	oui	4	*100*	*890*	—	1300	> 2300	—
120	—	2	4600	—	—	—	—	—
100	—	2	81	170	340	*14*	*32*	*72*
120	oui	2	*70*	*300*	3900	> 2100	—	—
100	oui	2	82	330	1500	79	280	*520*

Tableau 8 : Temps de calcul (en secondes sur station personnelle) nécessaire à deux méthodes pour trouver des solutions en moyenne à quelques pour cent de la meilleure solution connue.

plus lente. Nous remarquons que notre méthode est en général plus rapide que celle de Gendreau et al., particulièrement lorsqu'il y a une restriction sur la longueur des tournées. Nous avons également indiqué dans ce tableau en combien de sous-problèmes nous avons divisé les problèmes initiaux ; ainsi, ceux à 50 commandes n'ont pas été divisés et nous constatons que la recherche itérative dirigée que nous avons développée (sans partition) est plus rapide pour de petits problèmes que celle de Gendreau et al. Enfin, nous voyons que pour obtenir ces résultats, les sous-problèmes devaient comporter entre 4 et 6 tournées et 35 à 60 commandes. (Le tableau 9 donne le nombre de véhicules utilisés par les meilleurs plans connus de livraison.)

Un problème qui n'a pas pu être résolu de façon acceptable par ces méthodes est celui comportant 120 commandes. Cet exemple de problème a la particularité de présenter un dépôt très excentré par rapport aux lieux de livraison des commandes et ces derniers sont répartis en groupes. Le manque d'efficacité de notre méthode de décomposition peut s'expliquer par les caractéristiques spéciales de cet exemple de problème : comme le

dépôt est très excentré, notre algorithme de partition peut créer deux sous-problèmes dans des secteurs d'étendue angulaire très variable ; ensuite, comme les commandes forment des groupes qui doivent être livrés par le même véhicule dans les bonnes solutions, une partition peut diviser un ou plusieurs de ces groupes et il est par la suite très difficile de fusionner les différentes tournées livrant les commandes d'un de ces groupes. Nous verrons plus loin comment décomposer les problèmes où les commandes sont irrégulièrement réparties.

Dans le tableau 9, nous donnons les meilleures valeurs de solutions des problèmes de Christofides et al. obtenues par les trois meilleures méthodes actuellement disponibles

Nombre de commandes	Nombre de véhicules	Gendreau et al.	Osman	Partition en secteurs
50	5	*524.61*	*524.61*	*524.61*
75	10	835.32	838.62	*835.26*
100	8	*826.14*	829.18	*826.14*
150	12	1031.07	1044.35	*1028.42*
199	17	1311.35	1334.16	*1298.79*
50	6	*555.43*	*555.43*	*555.43*
75	11	*909.68*	*909.68*	*909.68*
100	9	*865.94*	866.75	*865.94*
150	14	*1162.55*	1164.12	*1162.55*
199	18	1404.75	1417.85	*1397.94*
120	7	*1042.11*	*1042.11*	*1042.11*
100	10	*819.56*	819.59	*819.56*
120	11	1545.93	1545.98	*1541.14*
100	11	*866.37*	*866.37*	*866.37*

Tableau 9 : Meilleures valeurs de solutions trouvées par quelques méthodes.

pour résoudre le problème de distribution de biens, à savoir les recherches itératives dirigées de Gendreau et al. [42], Osman [77] et celle que nous venons de décrire. Nous voyons que la méthode d'Osman semble la moins bonne et que la nôtre a permis de trouver toutes les meilleures solutions connues de ces problèmes.

Dans le tableau 10, nous nous sommes intéressé à la résolution des problèmes générés sur des grilles de $(2q \cdot k)^2$ points correspondant aux lieux de livraison de commandes de

$n = (2q \cdot k)^2$	m	k	q	s	c	Nombre d'itérations	Temps de calcul (secondes)	% au-dessus de l'optimum
64	16	2	2	2	1	150	1.7	2.4
64	16	2	2	2	1	2'000	17.9	0.9
64	16	2	2	4	1	150	0.7	2.8
64	16	2	2	4	1	2'000	6.2	1.7
256	64	4	2	4	1	500	7.9	1.6
256	64	4	2	4	1	3'000	94.8	0.9
256	64	4	2	2	2	500	8.5	1.8
256	64	4	2	2	2	3'000	96.7	1.2
1024	256	8	2	8	4	3'000	123.7	0.8
144	36	2	3	2	1	1'000	23.7	1.9
144	36	2	3	2	1	5'000	107.5	0.9
324	54	3	3	6	1	1'000	27.9	2.0
324	54	3	3	4	1	10'000	289.9	1.0

Tableau 10 : Résultats numériques pour des problèmes générés sur des grilles.

volume unitaire (volume des véhicules : $2q$). Nous donnons dans ce tableau la qualité en pour cent au-dessus de la valeur qui a été conjecturée comme optimale dans le paragraphe 1.4.2, page 16, et le temps mis par notre recherche itérative dirigée parallèle pour certaines tailles de grille (données par k et q) et différentes décompositions (données par le nombre s de secteurs et le nombre c de décompositions des secteurs). Nous pouvons émettre plusieurs remarques à l'examen de ce tableau : ces problèmes ont pu être résolus de manière très satisfaisante (un pour cent ou moins au-dessus de l'optimum conjecturé) en des temps de calcul fort courts ; plus le problème est décomposé finement, plus les itérations sont effectuées rapidement mais au détriment parfois de la qualité de la solution produite ; une décomposition des secteurs ne semble utile que pour les très gros problèmes ; finalement, la complexité du problème semble croître assez fortement avec le nombre moyen de commandes que contiennent les tournées.

Décomposition de problèmes irréguliers et/ou non euclidiens

La méthode de décomposition en régions polaires fonctionne mal si les commandes ne sont pas réparties régulièrement autour du dépôt et si ce dernier n'est pas au centre géographique du problème ; de plus, elle ne peut pas s'appliquer aux problèmes non

euclidiens puisqu'elle est basée entièrement sur les coordonnées des lieux où doivent être livrées les commandes. Examinons tout d'abord le cas des problèmes non euclidiens. Pour les décomposer, nous suggérons d'élaguer l'arborescence constituée de l'ensemble des plus courts chemins entre le dépôt et chaque client ; cet élagage se fait en enlevant des arcs à l'arborescence de manière à ce que les diverses composantes connexes contiennent un ensemble de commandes dont le volume total est aussi proche que possible d'un multiple de la capacité d'un véhicule. Plus formellement, l'algorithme se formule comme suit :

Partition de problèmes non euclidiens :

Données:

n clients, 1 dépôt.

d_{ij} : distancesentre les clients ($i, j = 0, \ldots, n$).

V : volume des véhicules.

v_i : volume demandé par le client i.

c : nombre maximal de véhicules par sous-problème créé.

— Construction de l'arborescence :

Construire l'ensemble des plus courts chemins du dépôt à chaque client. Cet ensemble est une arborescence α ayant comme racine le dépôt. Soit w_i le volume total des commandes des clients d'une sous-arborescence de α dont i est la racine. (Pour une feuille on a $w_i = v_i$; pour le dépôt, on a $w_0 = \sum_{k=1}^{n} v_k$.) Soit D_i la profondeur du client i dans l'arborescence (c'est-à-dire le nombre d'arcs du dépôt à i).

— Décomposition de l'arborescence :

Tant que $w_0 \neq 0$ répéter :

Soit j une commande telle que :

— j n'est pas déjà affectée à un sous-problème.

— $w_j \leq c \cdot V$

— $w_k > cQ$ où k est le client (s'il existe) connecté à j avec $D_k = D_j - 1$

— $D_j \geq D_{j'}$ pour tout j' respectant les trois premières conditions

ci-dessus.

— $w_j \geq w_{j''}$ pour tout j'' respectant les quatre conditions ci-dessus.

Affecter à un même sous-problème les commandes contenues dans la sous-arborescence ayant j comme racine et qui n'ont pas été affectées à un autre sous-problème.

Poser $w_i := w_i - w_j$ pour tout i appartenant au plus court chemin du dépôt à j.

Cet algorithme résout en fait le problème qui consiste à supprimer un nombre minimum d'arcs d'une arborescence de manière à ce que chaque composante connexe résultante ait un volume total de commandes plus petit ou égal à cV ; pour cela, il faut détruire les arcs qui sont le plus loin possible du dépôt (en termes de nombre d'arcs) de sorte que les sous-problèmes ainsi créés contiennent une quantité de biens aussi proche que possible de cV.

Décomposition de problèmes euclidiens irréguliers

Dans le cas de problèmes euclidiens, l'arborescence des plus courts chemins consiste en une étoile : chaque client est connecté directement au dépôt ; par conséquent, l'algorithme que nous venons de proposer, appliqué à cette arborescence, ne décomposera pas le problème logiquement. Pour remédier à cela, nous allons créer une arborescence qui engendre une décomposition pertinente du problème. L'utilisation d'un simple arbre de poids minimum (le terme poids étant synonyme de distance, voir figure 24 où l'on a représenté celui construit sur un ensemble de 385 points correspondants aux administrations municipales du canton de Vaud) n'est pas satisfaisante non plus car cet arbre ne présente pas une réelle structure et il est indépendant de l'emplacement du dépôt. Nous proposons donc de modifier les poids associés aux arcs et de favoriser ceux qui sont dirigés en direction du dépôt. Ainsi, le poids p_{ij} d'un arc ne sera plus égal à d_{ij}, mais obtenu par la formule :

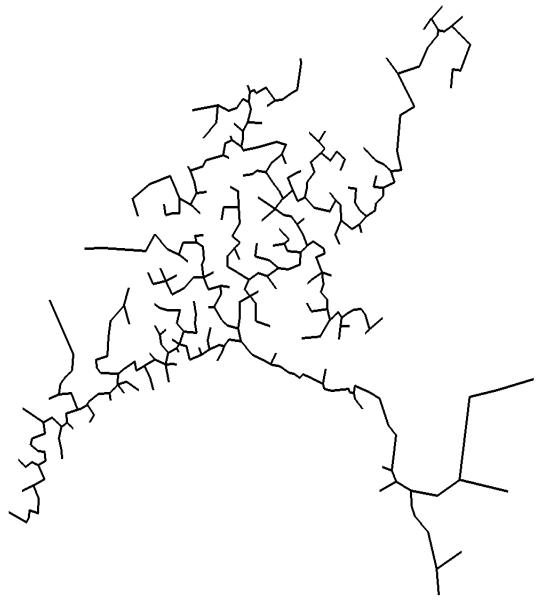

Figure 24 : Arbre de poids minimum sur les administrations municipales du canton de Vaud.

$$p_{ij} = \begin{cases} \|\boldsymbol{d}_{ij}\|\left(1 + \mu\left(1 + \dfrac{\boldsymbol{d}_{ij} \cdot \boldsymbol{d}_{i0}}{\|\boldsymbol{d}_{ij}\| \cdot \|\boldsymbol{d}_{i0}\|}\right)\right) & \text{si } \|\boldsymbol{d}_{ij}\| \cdot \|\boldsymbol{d}_{i0}\| \neq 0 \\ \|\boldsymbol{d}_{ij}\| & \text{sinon} \end{cases} \quad (9)$$

Dans cette formule, $\boldsymbol{d}_{ij} = \begin{pmatrix} x_j - x_i \\ y_j - y_i \end{pmatrix}$ où (x_i, y_i) et (x_j, y_j) sont les coordonnées rectangulaires des clients *i* et *j*, $\boldsymbol{d}_{ij} \cdot \boldsymbol{d}_{i0}$ est le produit scalaire entre les vecteurs $\boldsymbol{d}_{ij}$ et $\boldsymbol{d}_{i0}$ et $\|\boldsymbol{d}_{ij}\|$ est la longueur du vecteur $\boldsymbol{d}_{ij}$, ou en d'autres termes d_{ij}.

Le paramètre μ permet de varier l'importance du privilège des arcs selon leur direction. Ainsi, un μ nul n'induit aucun privilège et l'arborescence construite est identique à l'arbre de poids minimum ; un μ très grand favorise fortement les arcs pointant vers le dépôt. Le rôle de ce paramètre est clairement illustré en figure 25 où nous avons représenté l'arborescence obtenue avec $\mu = 2$ sur le même ensemble de points que pour la figure 24 ; le point noir situe le dépôt. Il est intéressant de noter que visuellement l'arborescence construite avec un μ proche de l'unité ressemble fortement à l'arborescence des plus courts chemins, obtenue en considérant les voies de communication réelles.

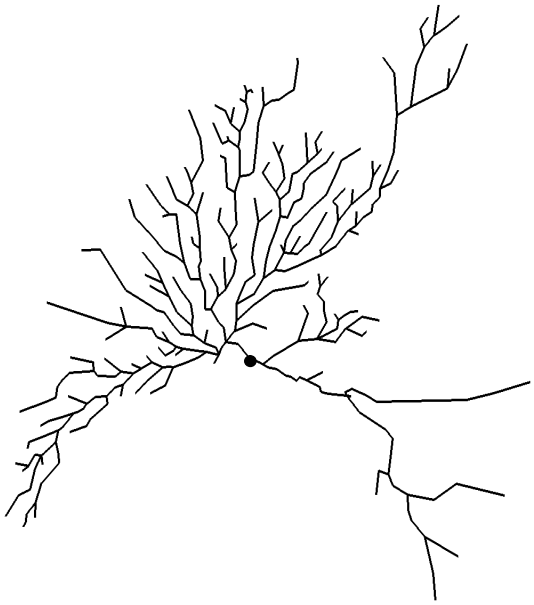

Figure 25 : Arborescence sur les mêmes points que pour la figure 24, obtenue en posant $\mu = 2$.

Par rapport à la décomposition en régions polaires, cette méthode présente l'énorme avantage de tenir compte de la structure du problème ; en variant les paramètres μ et c, il est possible d'obtenir des partitions variées. Cependant, cette méthode ne s'applique qu'à la décomposition initiale du problème, lorsqu'aucune tournée n'est encore formée. Comme une redécomposition du problème en régions polaires peut aussi être vue comme la transmission par un processus, de tournées, de véhicules non utilisés et de commandes non livrées, à un autre processus chargé de s'occuper d'un sous-problème lui étant adjacent, on peut dans la même optique tenir pour adjacents deux sous-problèmes obtenus par décomposition d'une arborescence s'ils étaient reliés par un arc avant la partition. Dans le cas euclidien, il est possible de créer plusieurs liens pour la transmission d'informations entre processus en considérant plusieurs arborescences obtenues avec des μ variés.

Dans le tableau 11, nous comparons (pour divers temps de calcul et en pour cent au-dessus de la meilleure solution connue) les deux méthodes de partition sur deux problèmes euclidiens irréguliers : le premier est celui dû à Christofides et al. [18] à 120

commandes et sans limite en ce qui concerne la longueur des tournées, et le second a été généré en considérant des commandes issues des administrations communales du canton de Vaud et de volume proportionnel au nombre d'habitants de chaque commune, le dépôt étant placé à l'École Polytechnique Fédérale de Lausanne. Nous remarquons que la

Nombre de commandes	Partition en régions polaires	Décomposition d'une arborescence	Temps de calcul
120	40.4	8.7	3.2
	21.5	1.8	19.7
	19.3	1.3	28
385	26.9	5.8	76
	7.9	2.7	520

Tableau 11 : Comparaison de deux méthodes de partition.

méthode de partition de l'arborescence permet de trouver beaucoup plus rapidement de bonnes solutions aux problèmes irréguliers que la méthode de décomposition en régions polaires, qui peut, en l'occurrence, être très mauvaise. Pour obtenir ces résultats, nous avons opéré avec une unique partition statique du problème dans le cas de la décomposition de l'arborescence, alors que nous avons appliqué au besoin de multiples décompositions dans le cas de la partition en régions polaires.

4.4. RECHERCHES INDÉPENDANTES

Une dernière technique de parallélisation, qui peut être appliquée à toute recherche itérative à comportement aléatoire, est d'effectuer simultanément plusieurs recherches indépendantes les unes des autres. Dans le cadre de recherches itératives dirigées, nous n'avons en effet pas pu élaborer des stratégies de séparation ou de fusion de trajectoires qui soient plus efficaces que de simplement laisser évoluer indépendamment plusieurs recherches et de retenir finalement la meilleure de toutes les solutions produites.

Il est de plus possible de caractériser les bonnes recherches itératives susceptibles de se paralléliser ainsi. Soit A un algorithme à comportement aléatoire qui a une probabilité $1 - q(t)$ de trouver une solution « intéressante » lorsqu'il fonctionne pendant un temps t.

A pourrait être par exemple une recherche itérative qui démarre avec une solution initiale aléatoire et/ou qui utilise des paramètres aléatoires ; une solution intéressante pourrait être une solution de valeur donnée. S'il existe $a > 1$, $t' > 0$ et $p \geq 2$ entier tels que :

$$q(t) < a^{-t} \quad (t' \leq t \leq pt') \text{ et } q(t) \geq a^{-t} \quad (pt' \leq t) \tag{10}$$

alors, il y a une plus grande probabilité d'obtenir une solution intéressante si *A* est exécuté p fois pendant un temps $t \in [t', pt']$ que si *A* n'est exécuté qu'une fois mais pendant un temps pt ; ceci se montre très simplement car on a $q^p(t) < (a^{-t})^p = a^{-pt} \quad (t' \leq t \leq pt')$ et $q(pt) \geq a^{-pt} \quad (t' \leq t)$ ce qui entraîne que $q^p(t) < q(pt) \quad (t' \leq t \leq pt')$.

Les conditions de (10) peuvent sembler artificielles, mais voici deux cas de figure où des recherches indépendantes sont recommandées :

1) $\lim_{t \to \infty} q(t) = b > 0$ et on désire obtenir une solution intéressante avec probabilité supérieure à $1 - b$.
2) On dispose de plusieurs processeurs faiblement interconnectés (par exemple des stations de travail personnelles connectées entre elles par un réseau de communication standard conçu pour le transfert de fichiers, la messagerie électronique, … mais pas pour le transfert rapide d'une grande quantité de messages courts) et il n'est pas possible de réaliser une parallélisation de bas niveau ou une décomposition du problème ; par contre, la fonction $q(t)$ n'est pas très éloignée d'une courbe exponentielle.

La première de ces conditions est très souvent remplie : en effet, on ne peut par exemple pas assurer qu'une recherche itérative dirigée ne visite pas de manière cyclique un ensemble restreint de solutions ne contenant pas de solution intéressante ; en général, on aura intérêt à fractionner une très longue recherche itérative en plusieurs recherches plus courtes et indépendantes. Il est cependant difficile de caractériser formellement les cas où ce type de parallélisation est efficace. Ainsi, pour la seconde condition, on est contraint de remplacer la fonction $q(t)$ par une fonction de répartition empirique, obtenue en observant le temps mis par un tel algorithme probabiliste pour parvenir à une solution intéressante en l'appliquant un grand nombre de fois à un problème, à l'image de ce que nous avons fait pour obtenir la figure 13, page 55, qui donne la probabilité empirique d'obtenir une solution optimale de problèmes de chaîne de traitement en fonction du temps de calcul de

notre recherche itérative dirigée. Si dans cette figure nous avions ajusté une courbe du type $1 - a^{-t}$, nous nous serions rendu compte que la seconde propriété était satisfaite dans ce cas. On observe en pratique que les recherches itératives dirigées satisfont très souvent les conditions de la seconde propriété. Dans les figures 26 et 27, nous avons représenté en même temps qu'une courbe exponentielle la proportion empirique de problèmes non

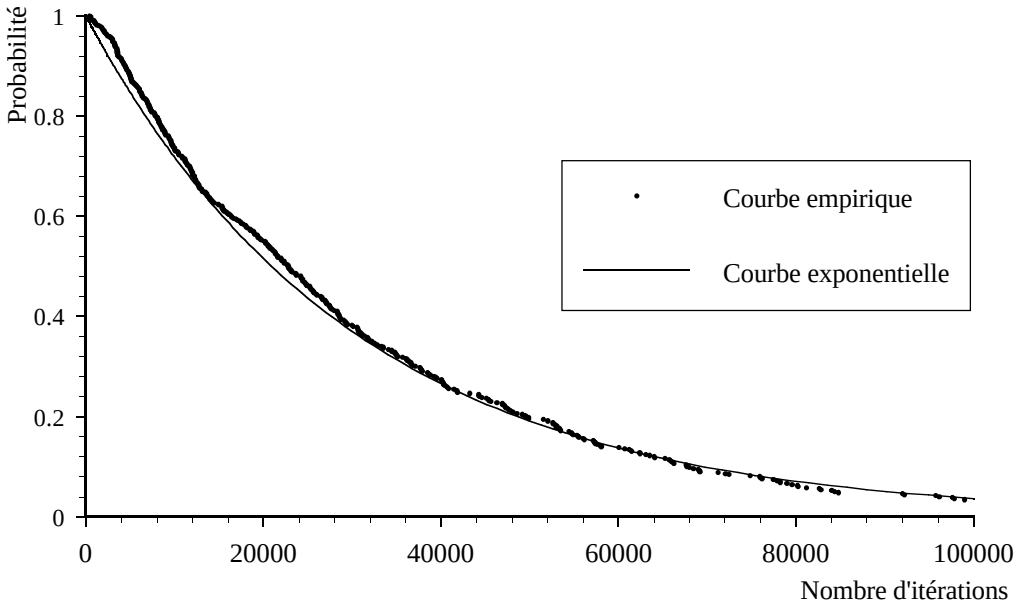

Figure 26 : Fonction empirique *q*(*t*) pour un problème de traitement à séquences fixées.

résolus de manière optimale par nos recherches itératives dirigées adaptées respectivement à des problèmes de traitement à séquences fixées et d'affectation quadratique. Pour obtenir ces courbes, nous avons résolu un grand nombre de fois le même problème, en modifiant la valeur des germes aléatoires utilisés pour générer les valeurs successives des divers paramètres et, pour le problème d'affectation quadratique, en partant de solutions initiales générées aléatoirement.

Bien que les deux problèmes considérés n'aient rien à voir l'un avec l'autre, et que les méthodes itératives dirigées utilisées pour les résoudre soient bien différentes, les deux courbes empiriques (de même que celle de la figure 13) de la fonction $q(t)$ sont tout à fait similaires et comparables à une exponentielle ; nous ne savons pas pourquoi il en est ainsi.

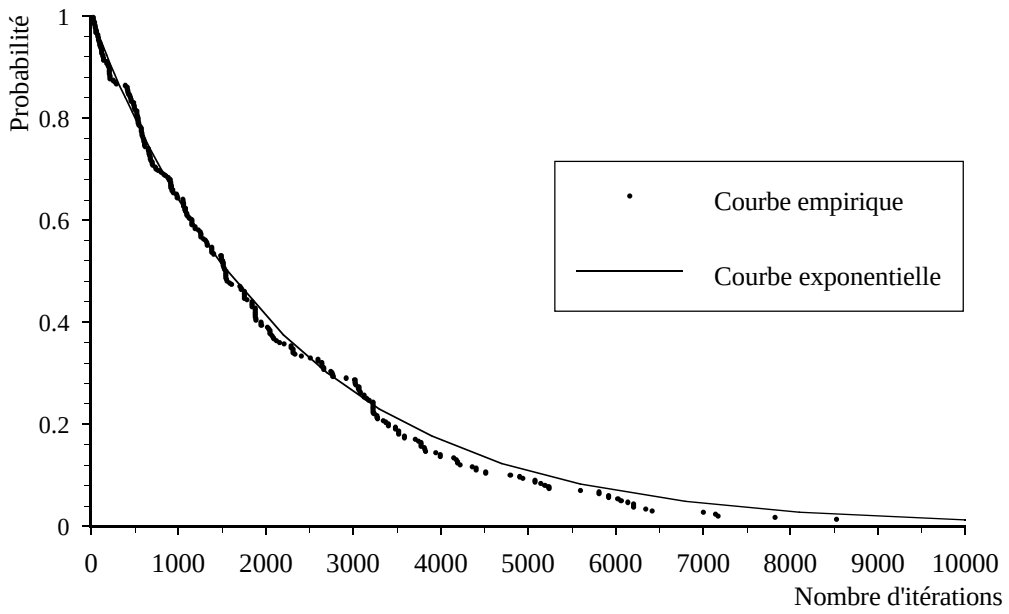

Figure 27 : Fonction empirique *q*(*t*) pour un problème d'affectation quadratique.

Pour ces trois algorithmes, une parallélisation par réalisation de trajectoires indépendantes sera efficace et facile à réaliser.

Synthèse

Nous avons présenté ce chapitre en le décomposant par type de parallélisation ; il aurait été possible de le découper par problème, comme pour le précédent. En guise de synthèse sur ce qui vient d'être dit, nous présenterons ici la manière d'utiliser tous ces types de parallélisation simultanément sur le problème de distribution de biens.

Nous avons vu qu'il pouvait être très difficile de trouver une bonne partition initiale du problème en sous-problèmes et par conséquent une multiplication des trajectoires (au sens de la section 4.4) s'impose si l'on désire obtenir de très bonnes solutions ; pour ceci, il faut compter 10 à 20 recherches indépendantes, selon la taille du problème.

Si l'on suppose que l'on est en face d'un problème avec 400 commandes qui doivent être réparties sur 30 à 40 camions, on doit le décomposer en 8 à 10 sous-problèmes. À ce stade, nous avons déjà créé une bonne centaine de processus qui sont relativement faciles à synchroniser et il est donc possible d'obtenir une efficacité de parallélisation honorable.

Avec des sous-problèmes comportant 50 commandes et 5 camions, le nombre de mouvements dont il faut recalculer la valeur à chaque itération d'une recherche itérative dirigée peut être évalué à une soixantaine ; une élection concurrente du meilleur mouvement peut donc englober quelques processeurs. Si chaque tournée comporte 10 à 15 commandes, on pourra à nouveau paralléliser efficacement la procédure permettant d'évaluer d'un mouvement avec quelques processeurs.

Globalement, il est donc possible de décomposer une recherche itérative multiple pour un problème de distribution de biens en un nombre de processus aussi grand que la taille du problème. Nous n'avons malheureusement pas pu tester une parallélisation à pareille échelle, ne disposant pas d'une machine adéquate ; cependant, il n'est pas utopique de penser que lorsque la technique nous fournira des calculateurs suffisamment puissants, ce qui devrait être le cas dans un avenir très proche, il sera possible de trouver en des temps très courts d'excellentes solutions à ce type de problèmes en utilisant les méthodes élaborées dans ce travail.

Dans ce chapitre, nous avons montré que la parallélisation d'un processus pourtant aussi séquentiel qu'une recherche itérative dirigée pouvait être efficacement réalisée. La première technique présentée, qui consiste à paralléliser l'évaluation des mouvements, aboutit à des efficacités très dépendantes du problème. La deuxième, préconisant l'évaluation concurrente des solutions éligibles, peut s'appliquer à plus de problèmes et permet d'obtenir souvent de bonnes accélérations.

Les techniques de décomposition de gros problèmes permettent de diriger efficacement les recherches itératives si celles-ci ont des difficultés à trouver de bonnes structures locales partout à la fois dans une solution ; ces techniques nécessitent de transmettre au processus de l'information sur les gros problèmes. En plus d'être facilement implantables sur calculateurs distribués, nous avons montré qu'elles peuvent notablement accélérer une recherche qui doit s'exécuter sur un ordinateur séquentiel.

Bien qu'elle soit utilisée de longue date, il est original d'avoir fait ressortir que la réalisation de plusieurs recherches itératives totalement indépendantes peut mener (en

tout cas pour tous les problèmes abordés dans ce travail) à des accélérations presque idéales lorsque ces recherches sont effectuées en parallèle sur des processeurs différents.

CONCLUSIONS

Nous terminerons cette étude sur les méthodes de recherches locales dirigées et leur parallélisation par une synthèse et des considérations générales sur ce qui a été présenté. Tout d'abord, nous devons insister sur le fait que seule a été étudiée une infime partie de la mine d'idées que constituent les deux articles fondamentaux de Glover, ainsi que plusieurs autres, écrits parfois avec la collaboration d'autres auteurs et qui n'ont pas toujours été cités dans ce travail, parce qu'en dehors du propos que nous voulions tenir.

Il nous semble en effet beaucoup plus logique de partir du problème pour arriver, comme nous l'avons fait, à une méthode de résolution, car il n'est pas possible d'intégrer en un seul coup toutes les politiques décrites dans la littérature, souvent en contradiction les unes avec les autres. Le but de notre approche est donc d'abord de définir une modélisation du problème ainsi que la notion de voisinage ou de mouvement qui permettent à une recherche locale de se « sentir à l'aise ». Pour qu'il en soit ainsi, il ne doit pas être trivial d'imaginer des situations où la recherche itérative se trouve piégée dans un ensemble de solutions et où il semble très difficile d'édicter des règles simples permettant à cette dernière de s'en échapper avec une grande probabilité. Ainsi, nos adaptations aux problèmes de traitement à séquences fixées et de distribution de biens peuvent être critiquées, car dans le premier cas, si l'on imagine une solution dont plusieurs opérations critiques sont réalisées successivement par une machine, trouver un ordre optimal de ces opérations critiques par simple inversion peut demander un nombre important d'itérations ; dans le second cas, réaliser un mouvement parfaitement pertinent qui consiste à échanger deux morceaux de deux tournées différentes peut aussi demander

un travail important si l'on ne considère que le déplacement d'une commande ou l'échange de deux commandes à la fois. Nous ne doutons pas que les modélisations et les voisinages adoptés dans nos méthodes puissent être nettement surpassés, en tout cas pour certains exemples de problèmes, même si les résultats obtenus semblent plus qu'encourageants.

Pour toutes les applications traitées, notre démarche a débuté invariablement par l'élaboration d'un moteur de recherche simple mais soigneusement implanté : modélisation triviale ou presque, accompagnée d'un voisinage de taille réduite ; en effet, nous avons toujours pu trouver dans toutes nos adaptations des moyens d'évaluer beaucoup plus rapidement que par une méthode naïve l'ensemble des solutions éligibles et s'il fallait dire pourquoi nos méthodes sont plus efficaces que certains recuits simulés, nous n'hésiterions pas à avancer l'argument que le nombre de solutions examinées par unité de temps par nos implantations est beaucoup plus élevé, d'un facteur pouvant être proportionnel à la taille du problème.

Comme dans toutes nos adaptations nous examinons le voisinage de façon systématique, sans tirer aléatoirement un sous-ensemble de solutions éligibles effectivement évaluées, nous avons dû apporter une attention particulière à la direction de la recherche. Là encore, nous préconisons des mécanismes simples ; les deux principaux étant une mémoire à court terme réalisée en interdisant pendant un nombre aléatoire d'itérations d'effectuer l'inverse d'un mouvement, et une mémoire à long terme basée sur la pénalisation des mouvements fréquemment élus ; ces mécanismes doivent aussi être implantés efficacement et nous avons toujours pu le faire de sorte qu'en temps constant on puisse dire si un mouvement est interdit et quelle pénalité lui est attribuée. Nous avons été particulièrement soucieux de donner automatiquement des valeurs aux paramètres associés aux mécanismes de direction de la recherche, afin que tous les exemples de problèmes que nous avons étudiés aient pu être résolus de manière acceptable. Nous sommes parfaitement conscient que les valeurs automatiques ne sont pas optimales et qu'elles pourraient même être très mauvaises pour des exemples de problèmes différents de ceux traités ici ; néanmoins, les règles qui ont été adoptées devraient pouvoir aider le

concepteur d'une recherche itérative dirigée dans l'élaboration d'une méthode adaptée au problème concret qu'il désire résoudre.

Même si nous n'avons abordé cet aspect que timidement au travers du problème de distribution de biens par le biais des méthodes de décomposition, l'élément fondamental qui permet à une recherche itérative dirigée de produire rapidement de bonnes solutions est de lui inculquer des bribes de notre savoir ; et nous sommes intimement convaincu que c'est là que réside un potentiel immense et encore très largement méconnu pour l'élaboration de méthodes efficaces. En effet, si nous reprenons l'exemple du problème de distribution de biens, le commun des mortels n'arrivera pas à trouver d'excellentes solutions à des problèmes non structurés comportant 50 commandes et 5 véhicules alors qu'une simple recherche itérative dirigée pourra le faire très rapidement, parce qu'un calculateur arrive à gérer beaucoup plus vite qu'un humain la masse d'informations que constitue un petit problème ; mais la même recherche itérative est incapable de traiter un gros problème, submergée par le nombre énorme d'informations qu'elle ne peut trier, la connaissance géographique et structurelle du problème, pourtant très intuitive chez l'homme, lui faisant défaut. Par contre, correctement dirigée en lui apprenant comment découper le problème, cette recherche itérative pourra traiter efficacement des problèmes de toutes les tailles.

Si nous devions nous prononcer sur un avantage d'un recuit simulé par rapport à une recherche itérative dirigée, nous indiquerions qu'un recuit est capable, dans la phase apparemment désordonnée qu'il effectue à haute température, d'identifier de lui-même une structure du problème sans qu'il soit nécessaire de lui fournir des indications supplémentaires, ce qui suggère que la mise au point d'un recuit simulé ne nécessite pas autant de connaissance sur le problème que celle d'une recherche itérative dirigée.

Finalement, mentionnons que la dernière ressource que nous proposons pour augmenter la probabilité de trouver de bonnes solutions une fois que la recherche itérative dirigée est entièrement définie, à savoir de réaliser plusieurs recherches en partant de solutions initiales différentes, ou, pour le problème d'élaboration de tournées, de diverses décompositions, peut être vue comme la traduction de notre ignorance sur le problème et

sa structure ; en effet, en reprenant le problème de distribution de biens, si l'on savait comment faire une décomposition optimale, une solution excellente pourrait être trouvée en un temps ridiculement court. La politique aléatoire des décompositions du problème ou des solutions initiales est critiquable et une voie plus systématique mériterait d'être explorée : on peut en effet imaginer d'élaborer un arbre de recherche partiel dont chacune des feuilles correspondrait à un problème dont certaines caractéristiques seraient fixées par les alternatives définies par les noeuds, à l'image de ce qui se fait pour les méthodes d'évaluation et séparation. Dans l'optique d'identifier de bonnes structures, notons que l'approche génétique nous semble également prometteuse.

Nous espérons que ce travail a permis de mettre en évidence certains avantages et faiblesses des recherches itératives dirigées et qu'il constitue un jalon dans une épopée qui débute à peine, si l'on en juge par la litanie de résultats expérimentaux encourageants obtenus, sans pour autant avoir réussi à poser la première pierre d'un outil mathématique qui analyserait et prédirait le comportement des recherches itératives dirigées. En effet, le seul outil actuellement disponible est l'analyse probabiliste et nous pensons que les théorèmes de convergence du recuit simulé sont à la fois insuffisants en pratique et inapplicables pour un processus déterministe.

Nous espérons également que le lecteur, bien que frustré de cet outil, trouvera néanmoins des principes de base qui lui seront utiles pour mettre au point une recherche itérative dirigée.

BIBLIOGRAPHIE

[1] E. Aarts, F. de Boyt, E. Habers, P. van Laarhoven, Parallel Implementations of the Statistical Cooling Algorithm, Integration, the VLSI Journal 4, 1986, 209–238.

[2] J. Adams, E. Balas, D. Zawack, The shifting bottle-neck procedure for job shop scheduling, Management Science 34, 1988, 391−401.

[3] Y. Agarwal, K. Mathur, H. Salkin, A set partitioning based exact algorithm for the vehicle routing problem, Networks 19, 1989, 731−749.

[4] D. Applegate, W. Cook, A computational Study of the Job-Shop Scheduling Problem, Operations Research Society of America Journal on Computing 3, 1991, 149−156.

[5] K. R. Baker, Introduction to Sequencing and Scheduling, Wiley, New York, 1974.

[6] R. Battiti, G.Tecchiolli, The reactive tabu search, préimpression, département de mathématiques de l'université de Trento, octobre 1992.

[7] L. D. Bodin, B. L. Golden, A. A. Assad, M. O. Ball, Routing and Scheduling of Vehicles and Crews. The State of the Art, Computers & Operations Research 10, 1983, 69−211.

[8] G. H. Bradley, G. G. Brown, G. W. Graves, Design and implementation of large scale primal transshipment algorithms, Management Science 24, 1977, 1–34.

[9] P. Brucker, B. Jurisch, B. Sievers, A Fast Branch & Bound Algorithm for the Job-Shop Scheduling Problem, cahier 136 des Osnabrücker Schriften zur Mathematik, Fachbereich Mathematik/Informatik, Universität Osnabrück, 1991.

[10] R. E. Burkard, Quadratic assignment problems, European Journal of Operational Research 15, 1984, 283−289.

[11] R. E. Burkard, U. Finke, Probabilistic asymptotic properties of some combinatorial optimization problems, Discrete Applied Mathematics 12, 1985, 21−29.

[12] R. E. Burkard and J. Offermann, Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme, Z. Operations Res. 21, 1977, B121−B132.

[13] R. E. Burkard, F. Rendl, A thermodynamically motivated simulation procedure for combinatorial optimization problems, European Journal of Operational Research 17, 1984, 169−174.

[14] J. Carlier, Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité, Méthodologie et architecture des systèmes informatiques, Institut de programmation, Université P. et M. Curie, Paris, 1984

[15] J. Carlier, E. Pinson, An algorithm for solving the job-shop problem, Management Science 35, 1989, 164-176.

[16] J. Chakrapani, J. Skorin-Kapov, Connectionist approaches to the quadratic assignment problem, dans : F. Glover, M. Laguna, É. Taillard, D. de Werra, Annals of Operations Research 41, 1993, 327–341.

[17] N. Christofides, Vehicle Routing, dans : The Traveling Salesman Problem : A Guided Tour of Combinatorial Optimization (E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, éditeurs), Wiley, Chichester, 1985, 431−448.

[18] N. Christofides, A. Mingozzi, P. Toth, The vehicle routing problem, dans : Combinatorial Optimization, (N. Christofides, A. Mingozzi, P. Toth, C. Sandi, éditeurs), Wiley, Chichester, 1979, 315−338.

[19] N. Christofides, A. Mingozzi, P. Toth, Exact algorithms for the vehicle routing problem, based on spanning tree shortest path relaxation, Mathematical programming 20, 1981, 255−282.

[20] N. Christofides, A. Mingozzi, P. Toth, Space state relaxation procedures for the computation of bounds to routing problems, Networks 11, 1981, 145−164.

[21] G. Clarke, J. W. Wright, Scheduling of Vehicles from a Central Depot to a Number of Delivery Points, Operations Research 12, 1964, 658−581.

[22] D. T. Connolly, An improved annealing scheme for the QAP, European Journal of Operational Research 46, 1990, 93−100.

[23] D. Costa, A Tabu Search Algorithm for Computing an Operational Time Table, à paraître dans European Journal of Operational Research, 1993.

[24] V. Černy, Thermodynamical Approach to the Traveling Salesman Problem : An Efficient Simulation Algorithm, Journal of Opt. Theory Appl. 45, 1985, 41–51.

[25] D. G. Dannenbring, An evaluation of flow shop sequencing heuristics, Management Science 23, 1977, 1174−1182.

[26] L. Davis (éditeur), Genetic Algorithms and Simulated Annealing, Pitman, Londres, 1987.

[27] L. Davis (éditeur), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

[28] M. Desrochers, T. W. Verhoog, A Matching Based Saving Algorithm for the Vehicle Routing Problem, Cahier du GERAD G−89−04, École des Hautes Études Commerciales de Montréal, 1989.

[29] J. W. Dickey, J. W. Hopkins, Campus building arrangement using TOPAZ, Transportation Research 6, 1972, 59−68.

[30] M. Dror, L. Levy, A Vehicle Routing Improvement Algorithm Comparison of a Greedy and a Matching Implementation for Inventory Routing, Computers & Operations Research 13, 1986, 33−45.

[31] G. Dueck, T. Scheuer, Threshold accepting : A general purpose optimization algorithm appearing superior to simulated annealing, IBM Scientific Center Heidelberg, rapport technique 88.10.011, 1988.

[32] A. E. Elshafei, Hospital layout as a quadratic assignment problem, Operations Research Quaterly 28, 1977, 167−179.

[33] U. Faigle, W. Kern, Some Convergence Results for Probabilistic Tabu Search, Operations Research Society of America Journal on Computing 4, 1992, 32–37.

[34] U. Faigle, R. Schrader, On the convergence of stationary distributions in simulated annealing algorithms, Information Processing Letters 27, 1988, 189–194.

[35] G. Finke, R. E. Burkard and F. Rendl, Quadratic assignment problems, Annals of Discrete Mathematics 31, 1987, 61−82.

[36] H. Fisher, G. L. Thompson, Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules, dans : J.F. Muth, G.L. Thompson éditeurs, Industrial Scheduling, Prentice Hall, Englewood Cliff, N.J., 1963, 225−251.

[37] M. L. Fisher, R. Jaikumar, A Generalized Assignment Heuristic for Vehicle Routing, Networks 11, 1981, 109−124.

[38] A. M. Frieze, J. Yadegar, S. El-Horbaty, D. Parkinson, Algorithms for assignment problems on an array processor, Parallel Computing 11, 1989, 151−162.

[39] M. R. Garey, D. S. Johnson, Computers and intractability : a guide to the theory of NP-completeness, W.H. Freeman & Co, New York, 1979.

[40] T. Gaskel, Bases for Vehicle Fleet Scheduling, Operational Research Quaterly 18, 1967, 281−295.

[41] M. Gendreau, A. Hertz, G. Laporte, New Insertion and Post-Optimization Procedures for the Traveling Salesman Problem, Operations Research 40, 1992, 1086–1094.

[42] M. Gendreau, A. Hertz, G. Laporte, A tabu search heuristic for the vehicle routing problem, rapport CRT-777, Centre de recherche sur les transports, Université de Montréal, 1991.

[43] B. Gillett, L. Miller, A Heuristic Algorithm for the Vehicle Dispatch Problem, Operations Research 22, 1974, 340−349.

[44] F. Glover, Future Paths for Integer Programming and Links to Artificial Intelligence, Computers and Operations Research 13, 1986, 533–549.

[45] F. Glover, Tabu Search – part I, Operations Research Society of America Journal on Computing 1, 1989, 190−206.

[46] F. Glover, Tabu Search – part II, Operations Research Society of America Journal on Computing 2, 1990, 4−32.

[47] F. Glover, É. Taillard, D. de Werra, A user's guide to tabu search, dans : F. Glover, M. Laguna, É. Taillard, D. de Werra, Annals of Operations Research 41, 1993, 3–28.

[48] B. L. Golden, A. A. Assad, Vehicle Routing : Methods and Studies, North-Holland, Amsterdam, 1988.

[49] M. Haimovich, A. H. G. Rinnooy Kan, Bounds and Heuristics for Capacitated Routing Problems, Mathematics of Operations Research 10, 1985, 527−542.

[50] B. Hajek, A Tutorial Survey of Theory and Applications of Simulated Annealing, actes de la « 24th Conference on Decision and Control, Ft. Landerdale » 1985, 755–760.

[51] P. Hansen, The steepest Ascent Mildest Descent Heuristic for Combinatorial Programming, présenté au « Congress on Numerical Methods in Combinatorial Optimization », Capri, 1986.

[52] A. Hertz, Tabu Search for Large Scale Timetabling Problems, European Journal of Operationnal Research 54, 1991, 39–47.

[53] A. Hertz, Finding a Feasible Course Schedule Using Tabu Search, Discrete Applied Mathematics 36, 1992, 255–270.

[54] A. Hertz, communication privée, (20. 2. 1992).

[55] A. Hertz, É. Taillard, D. de Werra, Tabu search — a chapter in the book : Local search in combinatorial optimization, edited by J. K. Lenstra, rapport

ORWP 92/18, Département de mathématiques, École Polytechnique Fédérale de Lausanne, 1992.

[56] A. Hertz, D. de Werra, The tabu search metaheuristic : how we used it, Annals of Mathematics and Artificial Intelligence 1, 1990, 111–121.

[57] J. H. Holland, Adaptations in Natural and Artificial Systems, presses de l'Université de Michigan, Ann Arbor, 1976.

[58] S. M. Johnson, Optimal Two- and Three-Stage Production Schedule with Setup Times Included, Naval Research Logistics Quaterly 1, 1954, 61−68.

[59] R. M. Karp, Probabilistic analysis of partitioning algorithms for the travelling salesman problem in the plane, Mathematics of Operations Research 2, 1977, 209−244.

[60] S. Kirkpatrick, C. D. Gelatt Jr, M. P. Vecchi, Optimization by Simulated Annealing, Science 220, 1983, 671–680.

[61] J. Krarup, P. M. Pruzan, Computer-aided layout design, Mathematical Programming Study 9, 1978, 75–94.

[62] P. J. M. van Laarhoven, E. H. L. Aarts, J. K. Lenstra, Job shop scheduling by simulated annealing, Operations Research 40, 1992, 113–125.

[63] G. Laporte, Y. Nobert, Exact Algorithms for the Vehicle Routing Problem, dans : Surveys in Combinatorial Optimization, (S. Martello, G. Laporte, M. Minoux, C. Ribeiro, éditeurs), North-Holland, Amsterdam, 1984, 147−184.

[64] G. Laporte, Y. Nobert, M. Desrochers, Optimal routing under capacity and distance restrictions, Operations Research 33, 1985, 1050−1073.

[65] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, Sequencing and Scheduling : Algorithms and complexity, Rapport BS−R89xx, Centrum voor Wiskunde en Informatica, Amsterdam, Pays-Bas, 1989.

[66] S. Lawrence, Ressource Constrained Project Scheduling : An Experimental Investigation of Heuristic Scheduling Techniques (supplément), Graduate School of Industrial Administration, Carnegie Mellon University, 1984.

[67] S. Lin, B. W. Kernighan, An effective heuristic algorithm for the travelling salesman problem, Operation Research 21, 1973, 2245–2269.

[68] A. Marchetti Spaccamela, A. H. G. Rinnoy Kan, L. Stougie, Hierarchical vehicle routing problems, Networks, 1984, 571−586.

[69] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of State Calculations by fast Computing Machines, Journal of Chem. Physics 21, 1953, 1087–1092.

[70] R. H. Mole, S. R. Jameson, A Sequential Route-Building Algorithm Employing a Generalised Savings Criterion, Operational Research Quaterly 27, 1976, 503−511.

[71] M. Navaz, E. E. Enscore Jr., I. Ham, A Heuristic Algorithm for the m−Machine, n−Job Flow-Shop Sequencing Problem, Omega 11, 1983, 91−95.

[72] M. D. Nelson, K. E. Nygard, J. H. Griffin, W. E. Shreve, Implementation Techniques for the Vehicle Routing Problem, Computers & Operations Research 12, 1985, 273−283.

[73] C. E. Noon, J. Mittenthal, R. Pillai, A TSSP + 1 Decomposition Approach for the Capacity-Constrained Vehicle Routing Problem, papier en préparation, Management Science Program, The University of Tennesse, Knoxville, TN, 1991.

[74] Ch. E. Nugent, Th. E. Vollmann, J. Ruml, An experimental comparison of techniques for the assignment of facilities to locations, Operations Research 16, 1968, 150−173.

[75] F. A. Ogbu, D. K. Smith, Simulated annealing for the permutation flowshop problem, OMEGA International Journal of Management Science 19, 1990, 64−67

[76] F. A. Ogbu, D. K. Smith, The application of the simulated annealing algorithm to the solution of the $n|m|C_{max}$ flowshop problem, Computers and Operations Research 17, 1990, 243−253.

[77] I. H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, dans : F. Glover, M. Laguna, É. Taillard, D. de Werra, Annals of Operations Research 41, 1993, 421–451.

[78] I. H. Osman, C. N. Potts, Simulated annealing for the permutation flow-shop scheduling, OMEGA International Journal of Management Science 17, 1989, 551−557.

[79] C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization Algorithms and Complexity, Prentice Hall, Englewood Cliffs, 1982.

[80] H. Passens, The Savings Algorithm for the Vehicle Routing Problem, European Journal of Operational Research 34, 1988, 336−344.

[81] V. M. Pureza, P. M. França, Vehicle Routing Problems via Tabu Search Metaheuristic. publication CRT-747, Centre de recherche sur les transports, Montréal, 1991.

[82] C. R. Reeves, A genetic algorithm for flowshop sequencing, à paraître dans Computers and Operations Research.

[83] A. Rogger, Applications des techniques parallèles de recherche tabou au problème de l'affectation quadratique, rapport de projet de diplôme, Département de mathématiques, École Polytechnique Fédérale de Lausanne, 1989.

[84] C. Roucairol, A parallel branch and bound algorithm for the quadratic assignment problem, Discrete Applied Mathematics 18, 1987, 211–225.

[85] F. Semet, É. Taillard, Solving real-life vehicle routing problems efficiently using taboo search, Annals of Operations Research 41, 1993, 469–488.

[86] S. Sahni, T. Gonzalez, P-Complete approximation problems, Journal of the Association for Computing Machinery 23, 1976, 555–565.

[87] D. Simchi-Levi, J. Bramel, On the Optimal Solution Value of the Capacitated Vehicle Routing Problem with Unsplit Demands, Département d'ingénierie industrielle et de recherche opérationnelle, Université de Columbia, New-York, 1990 (révision 1991).

[88] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, Operations Research Society of America Journal on Computing 2, 1990, 33–45.

[89] J. Skorin-Kapov, Extention of a tabu search adaptation to the quadratic assignment problem, Annals of Operations Research 41, 1993, 327–341.

[90] L. Steinberg, The backboard wiring problem : a placement algorithm, Society for Industrial and Applied Mathematics review 3, 1961, 37–50.

[91] É. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, European Journal of Operational Research 47, 1990, 65–74.

[92] É. Taillard, Parallel taboo search techniques for the job shop scheduling problem, rapport ORWP 89/11, Département de mathématiques, École Polytechnique Fédérale de Lausanne, 1989, à paraître dans Operations Research Society of America Journal on Computing.

[93] É. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64, 1993, 278–285.

[94] É. Taillard, Robust taboo search for the quadratic assignment problem, Parallel Computing 17, 1991, 443–455.

[95] É. Taillard, Parallel iterative search methods for vehicle routing problems, Networks 23, 1993, 661–673.

[96] A. Volgenant, R. Jonker, The symmetric traveling salesman problem and edge exchanges in minimal 1-tree, European Journal of Operational Research 12, 1983, 394–403.

[97] F. Werner, A. Winkler, Insertion Techniques for the Heuristic Solution of the Job Shop Problem, Préimpression Math 26/91, Institut für Mathematik, Fakultät für Mathematik, Technische Universität « Otto von Guericke » Magdeburg, 1991.

[98] M. Widmer, A. Hertz, A new heuristic method for the flow shop sequencing problem, European Journal of Operational Research 41, 1989, 186−193.

[99] M. R. Wilhelm, T. L. Ward, Solving quadratic assignment problems by simulated annealing, Institute of electrical and electronical engineers Transactions 19, 1987, 107−119.

[100] J. A. G. Willard, Vehicle Routing Using *r*-Optimal Tabu-Search, thèse de maîtrise, The Management School, Imperial College, London, 1989.

[101] D. L. Woodruff, E. Zemel, Hashing Vectors for Tabu Search, dans : F. Glover, M. Laguna, É. Taillard, D. de Werra, Annals of Operations Research 41, 1993, 123–137.

[102] A. Wren, Computers in Transport Planning and Operation, Ian Allan, Londres, 1971.

[103] A. Wren, A. Holliday, Computer Scheduling of Vehicles from one or more Depots to a Number of Delivery Points, Operational Research Quaterly 23, 1972, 333−344.

[104] P. Yellow, A Computational Modification the Savings Method of Vehicle Scheduling, Operational Research Quaterly 21, 1970, 281−283.