Titre: Title:	Affectation de cellules à des commutateurs dans les réseaux de communications personnelles
Auteur: Author:	Fabien Houeto
Date:	1999
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Houeto, F. (1999). Affectation de cellules à des commutateurs dans les réseaux de communications personnelles [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8782/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8782/
Directeurs de recherche: Advisors:	Samuel Pierre
Programme: Program:	Génie électrique

UNIVERSITÉ DE MONTRÉAL

AFFECTATION DE CELLULES À DES COMMUTATEURS DANS LES RÉSEAUX DE COMMUNICATIONS PERSONNELLES

FABIEN HOUETO

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

AOÛT 1999

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

AFFECTATION DE CELLULES À DES COMMUTATEURS DANS LES RÉSEAUX DE COMMUNICATIONS PERSONNELLES

présenté par : HOUETO Fabien

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen composé de :

M. CONAN Jean, Ph.D., président

M. PIERRE Samuel, Ph.D., membre et directeur de recherche

M. PESANT Gilles, Ph.D., membre

REMERCIEMENTS

À M. Samuel Pierre dont le soutien et les conseils m'ont guidé tout au long de ma recherche.

À Marionne, ma sœur aînée qui a montré la voie.

À mes parents, frère et sœurs pour leur soutien constant et indéfectible.

À M. Ronald Beaubrun et à tous mes collègues du Laboratoire de Recherche en Réseautique et Informatique Mobile (LARIM) pour leur collaboration et surtout pour l'ambiance de travail chaleureuse.

À mon pays dont la contribution financière m'a permis de mener toutes mes études universitaires.

À ma grande famille et à tous ceux qui m'ont soutenu et aidé, en particulier à Linda et Indira pour leurs relectures.

RÉSUMÉ

Durant les dernières décennies, le domaine des communications a connu beaucoup de changements. L'introduction du concept de mobilité à travers de nouveaux développements comme les cellulaires a amené de nouveaux problèmes qui ne se posaient pas avec les télécommunications fixes traditionnelles (réseaux de téléphone, d'ordinateurs, ...). Le problème d'affectation des cellules à des commutateurs dans des réseaux de communications personnelles en est un. Étant donné un ensemble de cellules et de commutateurs (dont les emplacements sont connus), on veut affecter les cellules aux commutateurs de façon à minimiser une fonction de coût qui intègre une composante de coût de liaison et une autre composante de coût de relève. En outre, l'affectation doit tenir compte de la contrainte de capacité des commutateurs qui ne peuvent accepter qu'un nombre limité d'appels. On introduit des variantes en permettant aux cellules d'être affectées à un ou plusieurs commutateurs. Le nombre d'affectations possibles est énorme et risque d'entraîner une explosion combinatoire.

Ce mémoire a pour objectif principal la mise au point d'une méthode heuristique de type *recherche taboue* (tabu search) pour résoudre le problème d'affectation énoncé précédemment. Pour y parvenir, nous avons défini une formulation mathématique du problème qui explicite la fonction objectif à minimiser, les contraintes à satisfaire et les variables dont dépendent la fonction et les contraintes. Puis, nous avons défini les paramètres de base de la recherche taboue dans le contexte spécifique de notre problème, et les avons raffinés dans le but d'obtenir de meilleurs résultats.

Lors de la mise en œuvre de la méthode, comme contributions de ce mémoire, nous avons d'abord établi un parallèle entre le problème d'affectation et celui bien connu en optimisation combinatoire de localisation d'entrepôts. Ensuite, pour implémenter notre adaptation, nous avons défini une structure de gains et des procédures de mise à jour pour choisir efficacement à chaque itération la meilleure solution dans le voisinage courant. Enfin, à défaut de connaître l'optimum global, nous avons déterminé deux de ses bornes inférieures afin de juger de la qualité des solutions.

Les résultats obtenus par notre méthode se comparent avantageusement à ceux fournis par d'autres heuristiques appliquées au même problème, en particulier pour des problèmes de grande et moyenne taille.

ABSTRACT

During the last decades, the introduction of the concept of mobility through new developments such as the cellular brought changes to the communications' fields. From these changes arose new difficulties which were unknown with traditional fixed telecommunications (computers, phone networks, ...). The assignment of cells to switches in personal communication networks is one of these new problems. Given a set of cells and switches (whose locations are fixed and known), the problem is to assign the cells to switches in order to minimise the costs of trunking (or cabling) and handoffs. Moreover, the assignment must take into account the restrained capacity of the switches which can accept only a limited number of calls. Variants of the problem are introduced by allowing the cells to be assigned to one or many switches. The enormous number of possible assignments makes the problem NP-hard.

This thesis has as principal objective the development of a tabu search method to solve the assignment problem. For that purpose, we defined an integer programming formulation of the problem, the basic parameters of tabu search in the specific context of our problem, and refined them in order to obtain better results. For the implementation of the method, we established a parallel between our problem and the one of warehouse location. We also defined a structure of gains and the appropriate update procedures so that the best solution in the current neighbourhood could be chosen efficiently. Finally we determined two lower bounds of the problem in order to judge the quality of our solutions. The results obtained by our method are compared advantageously with those

provided by other heuristics applied to the same problem, in particular for big and average size instantiations.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

Sigle ou abréviation	Signification
SCP	Système de communications personnelles
RCP	Réseau de communications personnelles
BB	Branch and bound
RT	Recherche taboue
LT	Liste taboue
Comm.	Commutateur
Cell.	Cellule
Heur.	Heuristique

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION

Durant les dernières décennies, la révolution des communications a changé le monde et conduit à une véritable démocratisation de l'information. Les communications sont devenues une nécessité vitale pour les entreprises et les personnes. Malgré tous les changements dont elles ont déjà été la cause, les communications modernes nous réservent encore bien des bouleversements. En effet, l'avènement du téléphone cellulaire a introduit dans le domaine l'important concept de mobilité des utilisateurs. L'introduction de ce concept ne va pas sans un certain nombre de problèmes qui ne se posaient pas avec les télécommunications fixes traditionnelles (réseaux de téléphone, d'ordinateurs, ...). Ce mémoire traite un de ces problèmes, soit celui de l'affectation des cellules à des commutateurs dans des réseaux de communications personnelles. Dans ce chapitre d'introduction, nous préciserons d'abord quelques concepts de base et les éléments de la problématique, par la suite nous résumerons nos objectifs de recherche, les résultats attendus et nos principales contributions, avant d'esquisser les grandes lignes du mémoire.

1.1 Définitions et concepts de base

Un *réseau de communications* est un ensemble d'équipements de communications répartis géographiquement mais reliés entre eux pour permettre à des usagers distants

d'échanger de l'information. Les systèmes de télécommunications qui intègrent la transmission de voix, de messages numériques, de textes, de courrier vocal et divers autres services sur un même support, éventuellement sans fil, dans un seul contrat et sur une seule facture, sont appelés des *systèmes de communications personnelles (SCP).*

Dans les SCP, les usagers ont un identifiant leur permettant de se faire reconnaître par le réseau et ce, peu importe l'endroit où ils se trouvent. Par conséquent, un usager peut se connecter au réseau et profiter des services offerts de n'importe quel endroit où l'accès au réseau est possible. Les SCP sont offerts sur des *réseaux de communications personnelles (RCP)* utilisant une organisation géographique du territoire semblable à celle des communications mobiles.

Dans un RCP, la zone de couverture est souvent découpée géographiquement en *cellules* (ou *nœuds*) hexagonales, comme l'illustre la figure 1.1. Ces cellules sont organisées hiérarchiquement afin de réduire les coûts de liaison (ou de câblage) . Ainsi, toutes les cellules ne communiquent pas directement entre elles. On choisit un certain nombre de cellules privilégiées, où l'on installe des *commutateurs* qui communiquent entre eux et servent de relais pour des communications entre paires de cellules.

Pour diverses raisons et surtout à cause de la mobilité, les commutateurs servant de relais à un usager donné peuvent changer si ce dernier change de cellule. L'opération qui consiste à noter le changement de cellule d'un utilisateur et à effectuer les mises à jour nécessaires constitue une *relève (handoff).* Quand la relève s'effectue entre deux cellules reliées à un même commutateur, on parle de *relève simple* car les mises à jour à effectuer sont peu nombreuses. Par contre, quand la relève se déroule entre deux cellules

reliées à des commutateurs différents, on parle alors de *relève complexe* puisque les mises à jour consomment plus de ressources.

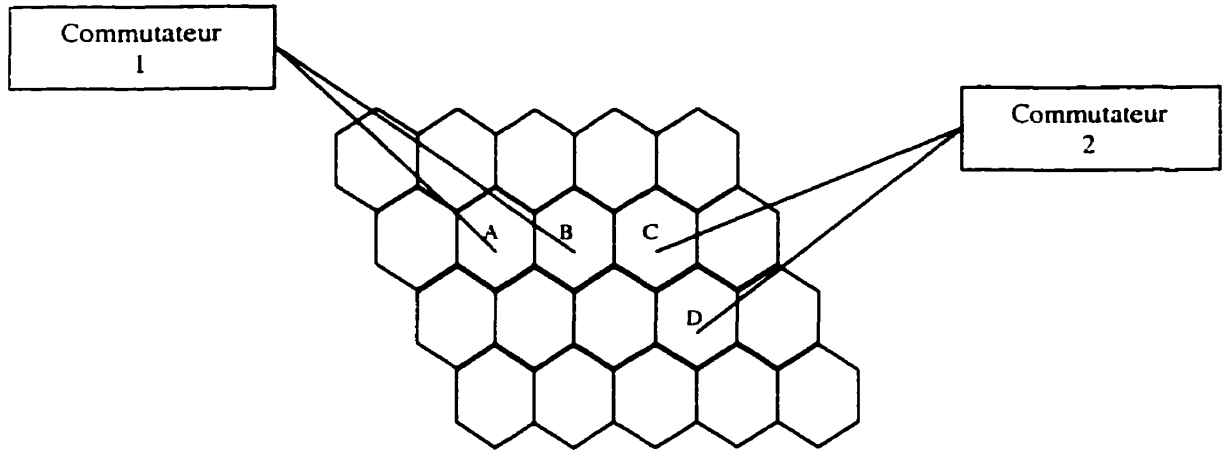

Figure 1.1 Découpage géographique dans un réseau mobile et relève (handoff) entre commutateurs

1.2 Éléments de la problématique

Dans les RCP, chaque cellule est desservie par une antenne utilisée pour communiquer avec les usagers sur des fréquences pré-affectées. Les communications des usagers d'une cellule sont ainsi relayées par le commutateur qui dessert celle-ci. Pour un usager qui se déplace, le signal envoyé est pris en charge par les cellules les plus proches. Souvent, on définit un seuil de filtrage au-delà duquel le signal de l'usager est suffisamment important pour être pris en compte par la cellule, ce qui peut entraîner quelques complications.

En se référant à la figure 1.1 par exemple, les cellules A et B relèvent du commutateur 1, tandis que les cellules C et D relèvent du commutateur 2. Considérons

un usager se trouvant dans la cellule B, le signal que reçoivent les cellules B et C peut être supérieur au seuil de filtrage compte tenu de la proximité de ces deux cellules. Il faut donc un réseau de signalisation (qui n'est pas dessiné sur la figure 1.1) pour savoir quelle cellule reçoit le plus fort signal et déterminer, de ce fait, le commutateur qui gérera la communication. Intuitivement, si l'usager se trouve dans la cellule B, la communication sera acheminée par le commutateur 1 auquel est reliée ladite cellule.

Si l'usager se déplace de la cellule B vers la cellule A, on assiste à une relève simple. La base de données du réseau qui garde en mémoire le commutateur gérant chaque usager n'a pas besoin d'être mise à jour. De plus, seul le commutateur 1 est concerné par cette mise à jour et aucune autre entité du réseau n'intervient. Par contre, si l'usager se déplace de la cellule B vers la cellule C, on a une relève complexe. En effet, les commutateurs 1 et 2 doivent échanger de l'information sur l'usager et la base de données du réseau doit être mise à jour. De plus, si le commutateur 1 est responsable de la facturation, la relève (handoff) ne peut pas simplement remplacer le commutateur 1 par le commutateur 2. En fait, la communication continue à être relayée à travers le commutateur 1 (à cause de la facturation) même après la relève. On aura donc une connexion de l'usager au commutateur 2, puis au commutateur 1, et enfin au réseau. Le coût d'une relève complexe est donc beaucoup plus élevé que celui d'une relève simple.

Si la fréquence des relèves entre les cellules A et B de la figure 1 est très élevée tandis que la fréquence des relèves entre les cellules B et C est faible, il apparaît alors raisonnable et moins cher de connecter les cellules A et B au même commutateur (si cela est possible) en vue de réduire les coûts de relève. Ainsi apparaît le problème

d'affectation de cellules à des commutateurs qui peut être résumé comme suit: étant donné un ensemble de cellules et de commutateurs (dont les positions sont connues), le problème consiste donc à affecter les cellules aux commutateurs de façon à minimiser une fonction de coût qui intègre une composante de coût de liaison et une autre composante de coût de relève. En outre, l'affectation doit tenir compte de la contrainte de capacité des commutateurs qui ne peuvent accepter qu'un nombre limité d'appels.

Ce problème d'affectation a été étudié en particulier par Merchant et Sengupta (1994, 1995). Leur algorithme part d'une solution initiale qu'il essaye d'améliorer par une série de mouvements de type glouton, tout en évitant de s'enfermer dans un minimum local. Les mouvements proposés pour échapper à un minimum local n'explorent qu'une partie très limitée des possibilités. De plus, ils dépendent de la solution initiale, et celle adoptée dans leur algorithme ne garantit pas forcément une bonne solution finale.

1.3 Objectifs de recherche et résultats attendus

Ce mémoire a pour objectif principal la mise au point d'une méthode heuristique de type *recherche taboue* (tabu search) pour résoudre le problème d'affectation énoncé précédemment. Pour y parvenir, nous commencerons par une formulation mathématique du problème qui explicite la fonction objectif à minimiser, les contraintes à satisfaire et les variables dont dépendent fonction et contraintes. Puis, nous définirons les paramètres de base de la recherche taboue dans le contexte spécifique de notre problème, et essayerons par la suite de les raffiner dans le but d'obtenir de meilleurs résultats.

Nous espérons que les résultats ainsi obtenus pourront se comparer avantageusement à la méthode proposée par Merchant et Sengupta (1995) et qui est spécialement adaptée au problème d'affectation. De plus, notre méthode devrait aussi pouvoir tenir la comparaison avec d'autres méta-heuristiques bien connues telles que le *recuit simulé* et les *algorithmes génétiques* appliquées au même problème.

1.4 Principales contributions

Les principales contributions de ce mémoire sont au nombre de quatre et consistent en :

- l'établissement d'une équivalence entre notre problème d'affectation et celui bien connu en optimisation combinatoire de localisation d'entrepôts ;
- la définition de mouvements, de critères d'aspiration, de mécanismes d'intensification et de diversification à la fois efficaces et adaptés au problème ;
- la définition d'une structure de gains et de ses procédures de mise à jour pour une implémentation efficace des mouvements ;
- la définition de deux bornes inférieures de la solution optimale du problème.

1.5 Plan du mémoire

Ce mémoire comprend six chapitres. Suivant ce premier chapitre d'introduction, le chapitre 2 présente une formulation mathématique, de type programmation en nombres entiers, du problème d'affectation de cellules et fait une revue sélective des méthodes existantes de résolution. Le chapitre 3 décrit la méthode générale de recherche taboue

qui sert de base à l'approche que nous proposons ici pour résoudre le problème d'affectation. Le chapitre 4 présente les détails d'adaptation et d'implémentation de la recherche taboue dans le cadre du problème d'affectation de cellules dans les réseaux mobiles. Au chapitre 5, nous présentons une analyse détaillée des résultats fournis par l'implémentation du chapitre 4 et faisons quelques comparaisons avec d'autres méthodes. Enfin, nous concluons au chapitre 6 en résumant les principaux résultats obtenus, les limitations de notre méthode et les extensions possibles aux travaux déjà entrepris.

CHAPITRE 2

AFFECTATION DE CELLULES À DES COMMUTATEURS DANS LES RCP

Le problème d'affectation de cellules, comme son nom l'indique, consiste à déterminer un patron d'affectation de cellules à des commutateurs dans le but de minimiser une fonction de coût quadratique, tout en respectant un certain nombre de contraintes notamment sur la capacité des commutateurs. Dans ce chapitre, nous faisons une revue sélective des travaux recensés dans la littérature qui traitent d'un ou de plusieurs aspects de ce problème. Après avoir défini les concepts de base, nous présenterons une formulation mathématique de ce problème, suivie d'une analyse mettant en évidence les points de ressemblance avec d'autres problèmes déjà connus de recherche opérationnelle. Ceci nous permettra d'évaluer les méthodes appliquées ou applicables au problème d'affectation ainsi que leurs limitations.

2.1 Définitions et concepts de base

Dans les RCP, les cellules, les commutateurs, ainsi que les liens entre cellules et commutateurs définissent la *topologie* du réseau pouvant être représentée par un graphe G=(N,A). *N* désigne l'ensemble des nœuds qui sont tantôt des commutateurs, tantôt des antennes desservant les cellules, et *A* désigne l'ensemble des arcs ou des liaisons. Ces

dernières possèdent des *attributs* tels la *capacité,* le *flot,* la *longueur*, etc. Les liaisons de la topologie peuvent être représentées, entre autres, par une *matrice d'incidence*, *une matrice d'adjacence,* ou un *vecteur caractéristique.*

On parle aussi parfois de patron. Un *patron* est un ensemble de données qui définit complètement l'état d'un attribut du réseau. Ainsi, un *patron d'appels* définit le flot d'appels entre deux cellules quelconques du réseau; un *patron d'affectation* définit complètement le commutateur auquel est affectée chacune des cellules du réseau.

Dans un patron d'affectation, on peut avoir une domiciliation simple ou double des cellules. On parle de *domiciliation simple* des cellules lorsqu'une cellule ne peut être reliée qu'à un et un seul commutateur. Par contre, on parlera de *domiciliation double* quand une cellule peut être connectée à au plus deux commutateurs. Si le patron des appels et des relèves ne reste pas le même pendant toute la journée, on peut avoir deux patrons : un pour le matin et l'autre pour l'après-midi, par exemple. Dans ce cas, les cellules ont une domiciliation double. Les deux commutateurs auxquels une cellule est reliée sont actifs alternativement et jouent le rôle de commutateur de base à des périodes précises de la journée.

Dans un *problème d'affectation*, on dispose de deux ensembles disjoints de même cardinalité, et on essaie d'établir une correspondance biunivoque entre les éléments de ces ensembles. Un *problème de transport* est un peu plus général. En effet, on retrouve toujours deux ensembles disjoints : des usines de production d'une part et des entrepôts d'autre part. Mais, on essaie plutôt de stocker la production des usines dans des entrepôts, tout en respectant les contraintes relatives à la capacité de ces

derniers. En effet, la capacité de stockage totale doit être supérieure à la production totale. En général, on ajoute des centres de production fictifs pour équilibrer production et capacité de stockage.

2.2 Formulation du problème

Le problème d'affectation de cellules à des commutateurs consiste essentiellement à minimiser une fonction de coût composée du coût de liaison et du coût des relèves sous des contraintes de capacité de commutateurs et selon une domiciliation simple ou double des cellules. Nous formulerons d'abord le problème général pour une domiciliation simple, tel que présenté par Merchant et Sengupta (1994, 1995). Ensuite, nous apporterons quelques simplifications et aborderons le problème avec domiciliation double.

2.2.1 Problème général pour une domiciliation simple

Soient n cellules à affecter à m commutateurs. La localisation des cellules et commutateurs est fixe et connue. Soient H_{ij} le coût par unité de temps d'une relève simple entre les cellules i et j impliquant un seul commutateur, et H'_{ij} le coût par unité de temps d'une relève complexe entre les cellules i et j $(i, j = 1, \ldots, n \text{ avec } i \neq j)$ impliquant deux commutateurs. H_{ij} et H'_{ij} sont proportionnels à la fréquence des relèves entre les cellules i et j. Soit c_{ik} le coût d'amortissement de la liaison entre la cellule i et le commutateur k $(i = 1, \ldots, n; k=1, \ldots, m)$ et soit λ_i le nombre d'appels par unité de

temps destinés à la cellule i. La capacité d'un commutateur en nombre d'appels par unité de temps est notée M_k.

Soit

$$x_{ik} = \begin{cases} 1 \text{ si la cellule } i \text{ est reliée au commutateur } k, \\ 0 \text{ sinon.} \end{cases}$$

L'affectation des cellules aux commutateurs est sujette à un certain nombre de contraintes. En effet, chaque cellule doit être assignée à un et un seul commutateur, ce qui se traduit par la relation suivante :

$$\sum_{k=1}^{m} x_{ik} = 1 \quad \text{pour} \quad i = 1, \ldots, n. \tag{2.1}$$

Soient z_{ijk} et y_{ij} définis par :

$$z_{ijk} = x_{ik} x_{jk} \quad \text{pour } i, j = 1, \ldots, n \text{ et } k = 1, \ldots, m, \text{ avec } i \neq j.$$

$$y_{ij} = \sum_{k=1}^{m} z_{ijk} \quad \text{pour } i, j = 1, \ldots, n, \text{ et } i \neq j.$$

z_{ijk} est égal à 1 si les cellules i et j, avec $i \neq j$, sont toutes deux connectées au même commutateur k, sinon il est nul.

y_{ij} prend la valeur 1 si les cellules i et j sont toutes deux connectées au même commutateur, et la valeur 0 si les cellules i et j sont connectées à des commutateurs différents.

Le coût par unité de temps f s'exprime comme suit:

$$f = \sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik} x_{ik} + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} H'_{ij} (1 - y_{ij}) + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} H_{ij} y_{ij} \tag{2.2}$$

Le premier terme est le coût de liaison, le deuxième terme prend en compte le coût des relèves complexes et le troisième terme celui des relèves simples. On note que la fonction de coût est quadratique en x_{ik}, car y_{ij} est une fonction quadratique des x_{ik}.

Si λ_i est le nombre d'appels par unité de temps destinés à la cellule i, la capacité limitée des commutateurs impose la contrainte suivante:

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leq M_k \quad \text{pour } k = 1, \ldots, m \tag{2.3}$$

selon laquelle la charge totale de toutes les cellules assignées à un commutateur ne dépasse pas la capacité de ce commutateur.

Enfin, pour compléter les contraintes du problème, on a :

$$x_{ik} = 0 \text{ ou } 1 \text{ pour } i=1, \ldots, n \text{ et } k = 1, \ldots, m. \tag{2.4}$$

$$z_{ijk} = x_{ij}x_{ik} \text{ pour } i, j = 1, \ldots, n \text{ et } k = 1, \ldots, m. \tag{2.5}$$

$$y_{ij} = \sum_{k=1}^{m} z_{ijk} \quad \text{pour } i, j = 1, \ldots, n \tag{2.6}$$

En rajoutant au besoin des cellules fictives avec des volumes d'appels fictifs non nuls, on peut ramener la contrainte (2.3) à une égalité. Les contraintes (2.1), (2.3) et (2.4) sont alors des contraintes de problème de transport. En effet, chaque cellule i peut être assimilée à une usine qui produit un volume d'appels λ_i. Les commutateurs sont alors des entrepôts de capacité M_k où on peut stocker la production des cellules. Le problème revient donc à minimiser (2.2) sous les contraintes (2.1), et (2.3) à (2.6). Ainsi formulé, le problème ne peut être résolu avec les méthodes standards de programmation

linéaire à cause de la non-linéarité de la contrainte (2.5). Merchant et Sengupta (1994, 1995) l'ont remplacée par l'ensemble équivalent de contraintes suivant :

$$z_{ijk} \leq x_{ik} \tag{2.7}$$

$$z_{ijk} \leq x_{jk} \tag{2.8}$$

$$z_{ijk} \geq x_{ik} + x_{jk} - 1 \tag{2.9}$$

$$z_{ijk} \geq 0 \tag{2.10}$$

Le problème peut alors se reformuler comme suit :

Minimiser (2.2) sous les contraintes (2.1), (2.3), (2.4) et (2.6) à (2.10).

On peut encore procéder à une simplification du problème en posant :

$$h_{ij} = H'_{ij} - H_{ij} .$$

h_{ij} est le coût réduit par unité de temps d'une relève complexe entre les cellules i et j. La relation (2.2) se récrit alors :

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik} x_{ik} + \sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} h_{ij}(1 - y_{ij}) + \underbrace{\sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} H_{ij}}_{constante}$$

Le problème d'affectation prend alors la forme suivante :

Minimiser

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik} x_{ik} + \sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} h_{ij}(1 - y_{ij}) \tag{2.11}$$

sujet à :

$$x_{ik} = 0 \text{ ou } 1 \text{ pour } i = 1, \ldots, n \text{ et } k = 1, \ldots, m \tag{2.4}$$

$$\sum_{k=1}^{m} x_{ik} = 1 \text{ pour } i = 1, \ldots, n \tag{2.1}$$

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leq M_k \quad \text{pour} \quad k = 1, \ \ldots, \ m \tag{2.3}$$

$$z_{ijk} \leq x_{ik} \tag{2.7}$$

$$z_{ijk} \leq x_{jk} \tag{2.8}$$

$$z_{ijk} \leq x_{ik} + x_{jk} \tag{2.9}$$

$$z_{ijk} \geq 0 \tag{2.10}$$

Ainsi transformé, le problème d'affectation peut se résoudre par les méthodes usuelles de programmation en nombres entiers. De plus, nous avons montré que la fonction de coût est quadratique et que les contraintes peuvent se ramener à celles d'un problème de transport.

2.2.2 Problème général pour une domiciliation double

Pour une domiciliation double, le patron des appels et des relèves ne reste pas le même pendant toute la journée. On a par exemple deux patrons pour la journée : un pour la matinée et l'autre pour l'après-midi. Les deux problèmes reliés à chaque partie de la journée ne sont pas découplés et ne peuvent donc être résolus séparément. Il faut optimiser non pas par rapport à chacun des patrons, mais par rapport aux deux simultanément, de telle sorte que le coût total (surtout celui de liaison) soit minimum.

Nous proposons une formulation introduite par Merchant et Sengupta (1995). Gardons les mêmes définitions de λ_i et h_{ij}, sauf qu'elles s'appliquent seulement pour un patron donné, en l'occurrence celui de la matinée. Soient λ'_i et h'_{ij} respectivement le

nombre d'appels par unité de temps destinés à la cellule i et le coût réduit par unité de temps d'une relève complexe entre les cellules i et j dans le patron 2 de l'après-midi. Dans ce cas, c_{ik} peut être interprété comme le coût des mises à jour nécessaires pour assigner la cellule i au commutateur k. M_k est toujours défini comme la capacité maximale du commutateur k. Les valeurs de M_k et c_{ik} restent les mêmes dans les deux patrons.

Selon le principe de la double domiciliation, une cellule peut être connectée à un ou deux commutateurs mais, dans chaque patron, seulement un commutateur à la fois est actif. On doit alors trouver deux patrons d'affectation pour chacune des parties de la journée, en indiquant s'il est plus économique ou non (avec les variations entre les deux patrons) d'avoir une cellule connectée à deux commutateurs avec un basculement d'un commutateur à un autre, ou d'avoir la cellule connectée à un seul commutateur. La différence entre les deux formulations est qu'il faut éviter que le coût de liaison c_{ik} ne soit compté doublement dans le cas où une cellule reste connectée au même commutateur dans les deux patrons d'affectation.

Pour le patron 1, définissons comme précédemment les variables x_{ik}, z_{ijk}, y_{ij}. Elles doivent, avec les variables λ_i et h_{ij}, satisfaire les contraintes (2.1), (2.3), (2.4) et (2.6) à (2.10). Pour le patron 2, définissons les variables équivalentes x'_{ik}, z'_{ijk}, y'_{ij} qui, avec les variables λ'_i et h'_{ij}, satisfont les mêmes contraintes (2.1), (2.3), (2.4) et (2.6) à (2.10). Le coût de liaison entre la cellule i et le commutateur k est compté si la cellule i est affectée au commutateur k dans au moins un des deux patrons d'affectation, c'est-à-

dire si $x_{ik} = 1$ ou $x'_{ik} = 1$. Soit w_{ik} la nouvelle variable qui indique si l'on doit ou non compter le coût de liaison entre i et k. On a :

$$w_{ik} = 1 \text{ si } x_{ik} = 1 \text{ ou } x'_{ik} = 1.$$

w_{ik} est donc défini par :

$$w_{ik} = x_{ik} \vee x'_{ik} \text{ pour } i=1, \ldots, n \text{ et } k=1, \ldots, m \qquad (2.12)$$

avec « $\vee$ » désignant l'opérateur logique « ou ».

La fonction objectif est alors :

$$f = \sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik} w_{ik} + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} h_{ij} (1 - y_{ij}) + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} h'_{ij} (1 - y'_{ij}) \qquad (2.13)$$

sous les contraintes (2.1), (2.3), (2.4) et (2.6) à (2.10), pour les variables x_{ik}, z_{ijk}, y_{ij}, λ_i et h_{ij} d'une part, et x'_{ik}, z'_{ijk}, y'_{ij}, λ'_i et h'_{ij} d'autre part. À tout cela, s'ajoute la contrainte supplémentaire :

$$w_{ik} = x_{ik} \vee x'_{ik} \text{ pour } i=1, \ldots, n \text{ et } j=1, \ldots, m \qquad (2.12)$$

Le problème est totalement défini mais n'est pas linéaire à cause de la contrainte (2.12) que l'on peut alors remplacer par l'ensemble équivalent de contraintes suivant :

$$w_{ik} \geq x_{ik} \qquad (2.14)$$

$$w_{ik} \geq x'_{ik} \qquad (2.15)$$

$$w_{ijk} \leq x_{ik} + x'_{ik} \qquad (2.16)$$

$$w_{ik} \leq 1 \qquad (2.17)$$

D'où la formulation finale suivante :

Minimiser (2.13) sous les contraintes (2.1), (2.3), (2.4), (2.6) à (2.10) et (2.14) à (2.17) pour les variables x_{ik}, w_{ik}, z_{ijk}, y_{ij}, λ_i et h_{ij} d'une part, et les variables x'_{ik}, z'_{ijk}, y'_{ij}, λ'_i et h'_{ij} d'autre part.

2.3 Fondements du problème d'affectation

Le problème d'affectation peut se ramener à un certain nombre de types de problèmes bien connus en recherche opérationnelle, tels le partitionnement de graphes ou le problème de localisation d'entrepôts ou de concentrateurs. Notre analyse portera d'abord sur le problème à domiciliation simple.

2.3.1 Problème de localisation d'entrepôts

La formulation du problème de localisation d'entrepôts qui correspond le mieux au problème d'affectation de cellules est connue sous le nom de problème de localisation de p concentrateurs fixes (p-fixed hubs location) et a été introduit par Skorin-Kapov et al. (1994) et Sohn et Park (1998).

Soient H un ensemble de positions connues et fixes de concentrateurs avec $|H| = m$ et c_{ik} le coût fixe pour établir une liaison entre le nœud i et le concentrateur k. L'ensemble des concentrateurs H et l'ensemble des cellules (ou nœuds) N sont disjoints. Le problème s'énonce alors comme suit :

Minimiser

$$f_L = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{m} \sum_{p=1}^{m} v_{ij} A_{ijkp} x_{ijkp} + \sum_{i=1, i \notin H}^{n} \sum_{k=1}^{m} c_{ik} z_{ik} \quad (2.18)$$

sujet à :

$$z_{kk} = 1 \text{ pour } k = 1, \ldots, m \quad (2.19)$$

$$\sum_{k=1}^{m} z_{ik} = 1 \quad \text{pour} \quad i = 1, \ldots, n \quad (2.20)$$

$$\sum_{p=1}^{m} x_{ijkp} = z_{ik} \quad \text{pour} \quad i = 1, \ldots, n-1 \text{ et } j = i+1, \ldots, n, \; k = 1, \ldots, m \quad (2.21)$$

$$\sum_{k=1}^{m} x_{ijkp} = z_{jp} \quad \text{pour} \quad i = 1, \ldots, n-1 \text{ et } j = i+1, \ldots, n, \; p = 1, \ldots, m \quad (2.22)$$

$$z_{ik} = 0 \text{ ou } 1 \text{ pour } i = 1, \ldots, n \text{ et } k = 1, \ldots, m \quad (2.23)$$

$$x_{ijkp} \geq 0, \text{ pour } i = 1, \ldots, n-1, \; j = i+1, \ldots, n, \; k = 1, \ldots, m \text{ et } p = 1, \ldots, m \quad (2.24)$$

où :

- $v_{ij} = W_{ij} + W_{ji}$ où W_{ij} désigne le flot du nœud i au nœud j;
- $A_{ijkp} = a_{ik} + \alpha a_{kp} + a_{pj}$ où a_{ij} est le coût par unité de flot entre les nœuds i et j;
- $\alpha \leq 1$ est un facteur sur le coût par unité de flot entre concentrateurs (on suppose que $c_{ii}=0$ pour $i = 1, \ldots, n$).
- x_{ijkp} est la fraction du flot entre les nœuds i et j acheminée à travers les concentrateurs se trouvant aux nœuds k et p (i est relié à k et j à p).
- $z_{ik} = 1$ si le nœud i est affecté au commutateur k et 0 sinon.

Le fait que les nœuds de H soient des concentrateurs se traduit par la contrainte (2.19). La contrainte (2.20) traduit le fait que chaque nœud est affecté à un seul concentrateur. Les contraintes (2.21) et (2.22) traduisent le fait que tout le flot d'un nœud est acheminé.

Le problème de localisation de p concentrateurs fixes, tel que présenté ici, a été relativement peu étudié. Dans la littérature, il est en général décomposé en deux sous-problèmes: (1) localiser la position des concentrateurs, (2) affecter les nœuds aux concentrateurs (Abdinnour-Helm, 1998; Klincewicz, 1991; Skorin-Kapov et al., 1994; Sohn et Park, 1998). Seule la deuxième partie du problème nous intéresse dans ce mémoire.

Dans le problème d'affectation, les commutateurs équivalent aux concentrateurs et les cellules aux nœuds. On a $H \cap N=\varnothing$, où H est l'ensemble des commutateurs et N l'ensemble des cellules. De plus, $W_{ii} = 0$ pour $i=1, \ldots, n$, c'est-à-dire qu'il n'y a pas de flot d'une cellule i vers la même cellule. Supposons que $A_{ijkp}=A_{jipk}$ pour $i,j=1,\ldots,n$ et $k,p=1, \ldots, m$ (c'est-à-dire que le coût de la relève entre les cellules i et j est le même que le coût de la relève entre les cellules j et i). De ce fait, (2.18) se réécrit comme suit :

$$f_L = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{m} \sum_{p=1}^{m} W_{ij} A_{ijkp} x_{ijkp} + \sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik} z_{ik} \qquad (2.18')$$

Dans le cas du problème d'affectation, le coût unitaire du flot entre les nœuds i et j, A_{ijkp}, sera le coût par unité de temps de la relève entre i et j. Posons alors :

$$A_{ijkp} = \begin{cases} B'_{ij} = a_{ik} + \alpha a_{kp} + a_{pj} & \text{pour un transfert complexe entre les cellules } i \text{ et } j \\ B_{ij} = a_{ik} + a_{pj} & \text{pour un transfert simple entre les cellules } i \text{ et } j. \end{cases}$$

Pour le problème d'affectation, la relève entre les cellules i et j est totalement acheminée par les commutateurs k et p auxquels elles sont reliées, c'est-à-dire qu'on n'a pas de flot fractionnaire. De ce fait, $x_{ijkp} = x_{ik}x_{jp}$, $z_{ik} = x_{ik}$ et on peut poser :

$$\sum_{k=1}^{m}\sum_{p=1}^{m} A_{ijkp}.x_{ik}.x_{jp} = A_{ijk'p'}.x_{ik'}.x_{jp'}$$

$$= \begin{cases} B'_{ij}\, x_{ik'}.x_{jp'} = b'_{ij} & \text{si } k' \neq p' \text{ et } i \text{ relié à } k',\ j \text{ relié à } p' \\ B_{ij}.x_{ik'}.x_{jp'} = b_{ij} & \text{si } k' = p',\ i \text{ et } j \text{ reliés au même commutateur } k' \end{cases}$$

En posant $H_{ij} = W_{ij}b_{ij}$ et $H'_{ij} = W_{ij}b'_{ij}$, (2.18') devient :

$$f_L = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{ij}\, y_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} H'_{ij}\,(1 - y_{ij}) + \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}\, x_{ik} \qquad (2.18'')$$

où y_{ij} est défini comme dans (6).

(2.18'') se ramène donc à (2.2) et au niveau des contraintes, on a :

- la contrainte (2.19) peut être éliminée car elle est déjà traduite par le fait que toutes les cellules de H sont des concentrateurs;
- la contrainte (2.20) équivaut à la contrainte (2.1) et traduit le fait que chaque cellule doit être affectée à un et seul concentrateur;
- les contraintes (2.21) et (2.22) entraînent que $z_{ik} = x_{ik}$ et en fait se ramènent à la contrainte (2.20);
- les contraintes (2.23) et (2.24) sont évidentes et correspondent au fait que le patron d'affectation est composé seulement de 0 (pour l'absence de liaisons) et de 1 (pour la présence de liaisons).

La relation (2.3) est une contrainte supplémentaire sur la capacité des commutateurs. Le problème d'affectation de cellules peut donc se ramener à un problème de localisation de p concentrateurs fixes à capacité limitée. Notons toutefois que, dans le problème de localisation de p concentrateurs, le flot entre deux cellules peut être partiellement acheminé par des concentrateurs différents, ce qui donne des valeurs de x_{ijkp} non entières et un problème de programmation mixte, alors que le problème d'affectation de cellules ne fait intervenir que des valeurs entières.

2.3.2 Problème de partitionnement de graphes

Le partitionnement de graphes est un problème très courant dans le domaine des technologies de l'information. Il survient par exemple en VLSI dans la disposition des composants, et en calcul parallèle où on doit répartir des calculs entre un nombre donné de processeurs (Kerninghan, 1970; Fidducia et Mattheyses, 1982; Sanchis, 1989). Dans cette section, nous exposons le problème de partitionnement en mettant en évidence ses liens avec le problème d'affectation.

Soit G un graphe avec des coûts associés à chacun de ses arcs. Le problème de partitionnement consiste à diviser l'ensemble N des nœuds du graphe en des sous-ensembles de cardinalité inférieure à un nombre maximal donné, de manière à minimiser le coût total des coupes, c'est-à-dire la somme des coûts des arcs ayant leurs extrémités dans des sous-ensembles différents. La figure 2.1 en est un exemple.

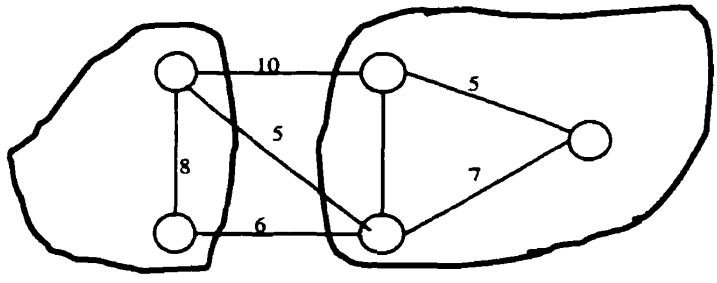

Figure 2.1 Partitionnement d'un graphe avec une coupe de coût 21

Pour représenter le problème d'affectation sous forme d'un graphe, on assimile chaque cellule à un nœud du graphe. Les antennes des cellules sont situées à leur centre et un arc (i,j) a comme coût la somme des coûts $h_{ij}+h_{ji}$ *de* transfert entre les cellules i et j. Les coûts de liaison sont proportionnels aux distances, comme illustré à la figure 2.2.

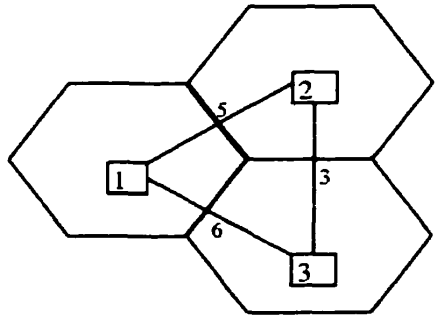

Figure 2.2 Un problème d'affectation (le commutateur est dans la cellule 2)

Le problème est ramené à un problème de partitionnement par les transformations suivantes exposées par Merchant et Sengupta (1994):

- Chaque cellule avec un volume d'appels λ_i est représenté par un *nœud primaire* et $\lfloor K\lambda_i \rfloor - 1$ *nœuds secondaires.* Le facteur multiplicatif K est un entier qui permet de

transformer les volumes d'appels non entiers en nombres entiers. Un facteur K trop petit entraîne une perte de précision dans les données du problème. En théorie donc, K peut prendre des valeurs assez grandes, mais en pratique, une valeur de K comprise entre 1 et 10 est souvent suffisante. Le nœud primaire et les nœuds secondaires sont reliés par des arcs de coût M, où M est un entier très grand. Ceci assure que les nœuds secondaires et primaire correspondant à une même cellule restent à tout moment dans le même sous-ensemble. Une cellule ayant un volume d'appels plus grand sera divisée en un nombre plus grand de nœuds pour refléter son volume d'appels.

- Pour chaque paire (i,j) de cellules, l'ancien arc (i,j) est reporté entre le nœud primaire représentant i et le nœud primaire de j avec les mêmes coûts.
- Un commutateur k est représenté par un nœud et demeure associé de manière permanente à un sous-ensemble de taille maximale $\lfloor KM_k \rfloor + 1$, où K est le facteur multiplicatif entier et M_k la capacité du commutateur k. Le fait de lier le commutateur de manière permanente au sous-ensemble fait que l'on ne peut pas déplacer le commutateur de son sous-ensemble.
- Enfin, le coût de liaison est représenté par l'ajout d'un arc entre le nœud primaire de chaque cellule i et le nœud représentant le commutateur k. À cet arc, on associe un coût c_a défini par :

$$c_a = \frac{1}{m-1}\left(\sum_{j=1}^{m} c_{ij}\right) - c_{ik} .$$

Ainsi, si la cellule i est associée au commutateur k', le coût de liaison c_L pris en compte par la coupe sera :

$$c_L = \sum_{\substack{k=1 \\ k \neq k'}}^{m} [\frac{1}{m-1}(\sum_{j=1}^{m} c_{ij}) - c_{ik}] = (m-1)\frac{1}{m-1}(\sum_{j=1}^{m} c_{ij}) - \sum_{\substack{k=1 \\ k \neq k'}}^{m} c_{ik} = \sum_{j=1}^{m} c_{ij} - \sum_{\substack{k=1 \\ k \neq k'}}^{m} c_{ik} = c_{ik'}$$

qui est bien le coût de liaison entre la cellule i et le commutateur k'. On a donc une équivalence entre les deux problèmes. En conséquence, les méthodes de résolution du problème de partitionnement peuvent donc s'appliquer au problème d'affectation.

Le problème de partitionnement de graphes et celui de localisation d'entrepôts sont connus comme étant des problèmes NP-difficiles (Garey et Johnson, 1979; Sanchis, 1989; Skorin-Kapov et al., 1994). Le problème d'affectation de cellules pouvant être ramené à un de ses problèmes et vice-versa, il est lui aussi NP-difficile. De ce fait, on ne connaît pas d'algorithme pour le résoudre en temps polynomial. Les méthodes exactes de programmation en nombres entiers ont une complexité exponentielle et explosent avec la taille du problème. On doit donc se contenter d'heuristiques qui fournissent en temps raisonnable des résultats éventuellement sous optimaux, mais assez proches de l'optimum.

Comme nous l'avons déjà mentionné, le problème d'affectation intègre une fonction de coût quadratique et des contraintes qui peuvent se réduire à celles d'un problème de transport : les commutateurs sont des usines qui ont une demande égale à leur capacité et les cellules sont des fournisseurs qui ont une offre égale à leur charge. Chaque cellule doit être affectée à un commutateur et la capacité des commutateurs doit être respectée (en ajoutant des cellules fictives s'il le faut). Le problème d'affectation de

cellules pourrait donc être appelé problème de transport quadratique, par analogie au problème d'affectation quadratique (QAP) qu'on appelle ainsi parce qu'il possède une fonction de coût quadratique et des contraintes d'affectation.

2.4 Méthodes classiques de résolution

Le problème d'affectation étant un problème NP-difficile, les algorithmes exacts souvent basés sur une énumération de toutes les possibilités ne peuvent être appliqués qu'à des problèmes de petite taille. En ramenant le problème à un problème de partitionnement de graphe, Kerninghan et Lin (1970) ont montré que le nombre de combinaisons possibles croît très rapidement. Pour un problème de localisation de m concentrateurs avec n cellules par exemple, on doit examiner à première vue m^n combinaisons.

2.4.1 Approches globales et voraces

Des méthodes globales ont été développées pour essayer de limiter l'espace des combinaisons à explorer. Ainsi, la technique de limitation d'exploration (BB pour Branch and Bound) permet de limiter le domaine de recherche grâce à une fonction de coût associée à chaque possibilité ou solution du domaine de recherche (Horowitz et Sahni, 1990). On part donc d'un point ou solution quelconque considéré comme la racine, sur laquelle on effectue une seule transformation à la fois. La transformation à effectuer est celle qui, (théoriquement) de toutes les transformations possibles, minimise le coût des efforts additionnels pour atteindre l'optimum. Malheureusement, calculer le

coût des efforts additionnels pour atteindre l'optimum revient presqu'à connaître cet optimum, ce qui est aussi difficile que le problème initial. On propose donc souvent une fonction de coût qui est une approximation de la fonction de coût réelle.

Des algorithmes comme ADD et DROP (Kuehn et Hamburger, 1963) prennent comme estimation du coût additionnel requis pour atteindre l'optimum, l'opposé du gain fait par rapport à la solution initiale. Ceci se justifie par le fait que plus on a déjà gagné par rapport à la solution de départ, moins il reste d'effort à faire pour parvenir à une solution. Ces algorithmes sont dits voraces car ils améliorent la solution au maximum à chaque étape. Toutefois, ils dépendent fortement de la manière dont les concentrateurs sont ajoutés et ne peuvent échapper au piège d'un minimum local.

2.4.2 Localisation de p concentrateurs fixes à capacité limitée

Dans la littérature, le problème de p concentrateurs fixes a été relativement peu étudié. Sohn et Park (1998) sont parmi les rares à aborder le problème de localisation de p concentrateurs avec des coûts fixes pour la création des liaisons. Cependant, le problème ressemble assez à celui d'affectation simple étudié antérieurement par les mêmes auteurs. Le problème d'affectation simple a été formulé par O'Kelly (1987) et consiste à déterminer la localisation des concentrateurs et l'affectation des cellules sous la seule contrainte que chaque cellule doit être affectée à un seul concentrateur.

En général, les algorithmes proposés jusqu'ici se concentrent surtout sur la partie localisation et font l'affectation d'abord selon la distance (les cellules sont affectées au commutateur le plus proche), ou selon le taux de transfert entre deux cellules (les

cellules qui échangent beaucoup d'informations sont affectées au même concentrateur), ou encore selon un critère mixte. Ensuite, cette première affectation est améliorée par différentes techniques. O'Kelly (1987) a donc proposé deux méthodes : la première affecte une cellule au concentrateur le plus proche, tandis que la deuxième considère les deux concentrateurs les plus proches. Bien qu'explorant toutes les possibilités de localisation des concentrateurs, la partie affectation (des cellules aux concentrateurs) de l'algorithme est très pauvre et n'est efficace que pour un nombre très restreint de concentrateurs (l'optimum est atteint à coup sûr quand on a un ou deux concentrateurs).

Klincewicz (1991) a exposé une heuristique basée sur un critère multiple prenant en compte aussi bien la distance que le taux de transfert entre cellules, avec des poids normalisés respectifs w_1 et w_2. Cette heuristique procède comme suit :

Étape 1 : Répéter pour chaque cellule i,

a) Calculer, pour chaque concentrateur k, l'inverse de la distance de i à k et normaliser les distances inverses de telle sorte que le maximum soit 1. Soit D_{ik} la distance inverse normalisée de i à k.

b) Calculer, pour chaque concentrateur k, le trafic total échangé par i avec les cellules déjà affectées à k. Appeler cette mesure T_{ik} et la normaliser de telle sorte que le maximum soit 1.

c) Calculer, pour chaque concentrateur k, $G_{ik}=w_1 D_{ik} + w_2 T_{ik}$.

d) Affecter i au concentrateur ayant le plus grand G_{ik} et ayant une capacité suffisante pour prendre i.

Étape 2 : Pour chaque cellule i, faire les calculs suivants : pour chaque concentrateur k, calculer le gain S_{ik} de la fonction objectif résultant de la réaffectation de i à k au lieu de son affectation actuelle.

Étape 3 : Choisir le S_{ik} maximum pour tous les i et tous les k qui respectent les contraintes de capacité.

Étape 4 : Si $S_{ik} > 0$, réaffecter i à k et retourner à l'étape 2, sinon arrêter.

Un désavantage des heuristiques proposées réside dans le fait qu'elles essayent à la fois de localiser les concentrateurs et de faire l'affectation. Cela donne souvent lieu à des algorithmes d'affectation qui ne prennent en compte ni les contraintes de capacité du problème d'affectation, ni le fait que les positions des commutateurs sont fixées et connues. Par exemple, l'algorithme de Klincewicz (1991) exposé précédemment peut être adapté au problème d'affectation en utilisant la fonction objectif à la place de G_{ik}. Cependant, cet algorithme est myope et ne peut échapper à un minimum local.

Sohn et Park (1998), après avoir énoncé le problème de localisation de p concentrateurs fixes avec coût fixe de liaison, ont utilisé une résolution de la relaxation lagrangienne du problème pour obtenir une borne inférieure de la fonction objectif. Cette borne est utilisée pour évaluer les différents schémas de localisations des concentrateurs afin d'en choisir rapidement un.

2.4.3 Partitionnement de graphes

L'une des premières heuristiques de partitionnement de graphes a été introduite par Kernighan et Lin (1970). Cette heuristique permet de partitionner un graphe $G=(N, A)$ avec c sommets (c pair) en 2 blocs de $c/2$ sommets chacun. On part d'une partition quelconque (A,B) que l'on améliore itérativement en choisissant une cellule dans chaque bloc, puis en les permutant selon les étapes suivantes :

Étape 1 : Affecter à p la valeur 1. A_p devient l'ensemble A et B_p l'ensemble B.

Étape 2 : Enlever les marques de tous les sommets et initialiser i à 1.

Étape 3 : Pour tout élément (ou sommet) n de N non marqué, calculer le gain (éventuellement négatif) sur la fonction objectif résultant du déplacement de l'élément n de son bloc (par exemple A) vers l'autre bloc (par exemple B).

Étape 4 : Choisir le couple (a_i, b_i) avec $a_i \in A_p$ et $b_i \in B_p$, a_i et b_i non marqués, tel que la permutation de a_i et b_i donne le plus grand gain g_i sur la fonction objectif (ce gain peut être négatif).

Étape 5 : On permute a_i et b_i et on obtient ainsi A_{p+1} et B_{p+1} . On marque a_i et b_i et on incrémente i et p.

Étape 6 : S'il reste des sommets non marqués, retourner à l'étape 3.

Étape 7 : Choisir k tel que le gain $G = \sum_{i=1}^{k} g_i$ soit maximal.

Étape 8 : Si $G > 0$ alors A devient $A-\{a_1, \ldots, a_k\} \cup \{ b_1, \ldots, b_k\}$ et B devient $B-\{b_1, \ldots, b_k\} \cup \{ a_1, \ldots, a_k\}$ et on retourne à l'étape 1, sinon on arrête.

Des formules sont fournies pour calculer les gains, faire les mises à jour et choisir les permutations de façon optimale. L'algorithme peut être adapté pour partitionner en sous-ensembles de tailles inégale (bornée supérieurement ou inférieurement). Pour faire un partitionnement multiple, on utilise le même algorithme en décomposant l'optimisation du partitionnement multiple en plusieurs optimisations entre sous-paires. On peut, par exemple, faire un partitionnement en m blocs, en faisant un partitionnement entre le premier bloc et les m-1 autres, et ensuite appliquer le même schéma m-2 fois aux m-1 blocs restants. Cependant, cette optimalité par paires ne garantit par l'optimalité globale des m blocs entre eux.

Fiduccia et Mattheyses (1982) ont amélioré l'algorithme de Kernighan et Lin (1970) en proposant le déplacement d'une cellule à la fois, ce qui introduit une plus grande flexibilité dans la taille des blocs et permet de choisir plus facilement les permutations à effectuer. Ils ont introduit la notion de gain d'une cellule C noté $gain(C)$ et défini comme l'amélioration sur la fonction objectif obtenue en déplaçant la cellule C d'un bloc à un autre; $gain(C)$ est compris entre p et $-p$, où p est le degré du graphe, c'est-à-dire le degré du sommet ayant le plus grand degré. Dans un tableau de taille $2p+1$ indexé par les valeurs possibles du gain $[-p,p]$, on peut maintenir des pointeurs vers les cellules ayant comme gain la valeur de l'index. Les cellules sont ainsi constamment gardées en ordre de gains croissants et on peut facilement choisir les cellules donnant le plus fort gain. Bien entendu, des formules et méthodes sont fournies pour mettre à jour le tableau des gains de façon simple.

Krishnamurthy (1987) a ajouté des améliorations telles le $i^{ème}$ niveau de gain qui ont permis à Sanchis (1989) de pouvoir adapter l'algorithme au partitionnement multiple, mais de façon uniforme, en considérant à chaque itération tous les mouvements possibles d'une cellule de son bloc vers n'importe lequel des autres blocs.

Merchant et Sengupta (1995) ont introduit une méthode plus efficace spécialement adaptée au problème d'affectation de cellules. Elle consiste à trouver une solution initiale de la manière suivante :

Étape 1 : Classer les cellules par ordre décroissant de volume d'appels. On commence avec une seule affectation vide.

Étape 2 : Pour chaque cellule prise dans l'ordre imposé ci-dessus, étendre les affectations précédemment retenues en leur ajoutant toutes les affectations de la $k^{ième}$ cellule aux différents commutateurs.

Étape 3 : Éliminer les affectations qui violent la capacité des commutateurs. S'il ne reste pas d'affectations, l'algorithme échoue. S'il reste b ou moins que b affectations, les retenir toutes. Sinon retenir les b meilleures affectations selon la fonction objectif (b est un paramètre, fixé au départ, qui ne doit pas être trop grand pour éviter tout risque d'explosion combinatoire, mais assez grand pour garantir une meilleure solution initiale).

Étape 4 : S'il reste des cellules non encore affectées, retourner à l'étape 2, sinon arrêter et retourner la meilleure des affectations trouvées.

Pour améliorer la solution initiale trouvée, on définit un mouvement faisable comme un mouvement qui n'implique pas de cellules marquées et qui conduit à une solution faisable ne violant pas la contrainte de capacité des commutateurs. Ensuite, on complète les étapes suivantes :

Étape 1 : Effacer les marques de toutes les cellules.

Étape 2 : De tous les mouvements faisables, choisir celui qui affecte une cellule i à un commutateur k et qui réduit la fonction objectif de la plus grande valeur. Si aucun des mouvements faisables ne réduit la fonction objectif, prendre le mouvement faisable qui augmente le moins la fonction objectif.

Étape 3 : Affecter la cellule i au commutateur k, marquer la cellule i et noter le schéma d'affectation courant.

Étape 4 : S'il reste des cellules non marquées, retourner à l'étape 2.

Étape 5 : De tous les schémas d'affectation générés, choisir celui ayant la plus petite valeur de fonction objectif.

Étape 6 : Si la valeur de la fonction objectif obtenue à l'étape 5 est inférieure à celle du schéma d'affectation courant, alors celui-ci devient le schéma choisi à l'étape 5 et on retourne à l'étape 1 en ignorant alors tous les schémas d'affectation précédents, sinon on arrête.

Merchant et Sengupta (1995) ont aussi proposé un algorithme pour résoudre le problème d'affectation avec domiciliation double. En voici les principales étapes :

Étape 1 : Former deux problèmes d'affectation avec domiciliation simple correspondant chacun à un patron d'une partie de la journée.

Étape 2 : Résoudre les deux problèmes avec l'algorithme précédent. Soit Q le problème dont la solution A a la plus petite valeur de fonction objectif et Q' l'autre solution.

Étape 3 : Soit Q_1 le problème d'affectation avec domiciliation simple identique au problème Q', à la différence que si la solution A affecte la cellule i au commutateur k, le coût de liaison c_{ik} est nul dans Q_1. Résoudre Q_1 et soit A' sa solution.

Étape 4 : Similairement, soit Q_2 le problème d'affectation avec domiciliation simple identique au problème Q à la différence que si la solution A' affecte la cellule i au commutateur k, alors le coût de liaison c_{ik} est nul dans Q_2. Résoudre Q_2 et réinitialiser A comme solution de Q_2.

Étape 5 : Répéter les étapes 3 et 4 jusqu'à ce que les affectations ne donnent plus d'amélioration de la fonction objectif. Les affectations dans A et A' forment la solution du problème d'affectation avec domiciliation double.

Les méthodes exposées précédemment, bien qu'assez efficaces et parvenant à une solution en des temps assez courts, intègrent des mécanismes souvent complexes pour échapper au piège du minimum local (Kerninghan et Lin, 1970; Merchant et Sengupta, 1994, 1995). Ces méthodes comptent surtout sur une bonne solution initiale

pour atteindre l'optimum. Les approches exactes telles que la programmation linéaire profitent de la convexité de l'espace de recherche pour développer des algorithmes optimaux qui sont très performants. Malheureusement, en programmation entière, on ne bénéficie pas de cette convexité. Dans ces circonstances, les méthodes d'exploration conduisent à des optima qui souvent sont locaux. Dans le chapitre suivant, nous exposons la méthode de recherche taboue qui est aussi une méta-heuristique générale qui tire profit de l'histoire de l'exploration du domaine de recherche pour converger vers une « bonne » solution.

CHAPITRE 3

LA MÉTHODE DE RECHERCHE TABOUE

La méthode de recherche taboue (RT) est une technique adaptative introduite assez récemment en optimisation combinatoire pour résoudre des problèmes difficiles. Elle est considérée comme une méta-heuristique, en ce sens qu'elle peut s'appliquer à différentes instances de problèmes et y fournir de bonnes solutions. Dans ce chapitre, nous présentons d'abord les fondements de la méthode en exposant brièvement son historique et ses concepts de base. Ensuite, nous la caractérisons en décrivant ses différents mécanismes de fonctionnement et les paramètres qui permettent de les adapter à un problème spécifique. Enfin, nous explorons et illustrons les raffinements possibles inhérents à cette méthode.

3.1 Fondements de la méthode de recherche taboue

Le concept d'algorithme de RT remonte aux années 70 et provient des procédures combinatoires appliquées aux problèmes non linéaires. La méthode a été introduite dans sa forme actuelle par Glover (1989) et est maintenant devenue une approche d'optimisation qui s'étend à plusieurs domaines. Les résultats obtenus par l'application de RT à des problèmes comme le design de circuits intégrés, la coloration de graphes, le problème du commis voyageur, ont démontré la capacité de RT à fournir des solutions de bonne qualité au prix d'un effort de calcul relativement modeste

comparé à des méthodes alternatives. Après avoir introduit les notions de mouvement et de voisinage nécessaires à la compréhension de RT, nous exposerons la méthode de RT proprement dite en la comparant avec l'algorithme plus simple de descente générale.

3.1.1 Mouvements et voisinage

De façon générale, un problème d'optimisation combinatoire peut se représenter sous la forme :

Minimiser $f(x)$,

sujet à: $x \in X \subseteq R^n$.

La fonction objectif $f(x)$ peut être linéaire ou non et la condition $x \in X$ contraint souvent x à prendre des valeurs discrètes, souvent entières.

Trouver directement la solution x^* qui minimise $f(x)$ sous la condition $x \in X$ n'est souvent pas possible. La plupart des méthodes et procédures proposées partiront d'une solution initiale x_0, et lui appliquerons de petites perturbations pour aboutir à de meilleures solutions (Pierre, 1998a, 1998b). Ces perturbations qui permettent de se déplacer d'une solution à une autre définissent des *mouvements* (Pierre, 1997). Un mouvement m est donc défini comme une application :

$$m : \quad X(m) \rightarrow X$$

$$s \rightarrow m(s)= s'=s \oplus m \text{ (par abus de notation)}$$

où $X(m) \subseteq X$ est l'ensemble des solutions auxquelles peut être appliqué le mouvement m. En général, les mouvements sont inversibles et on définit *l'inverse* m^{-1} d'un mouvement m par :

$$m^{-1}: \quad X=X(m^{-1}) \quad \rightarrow \quad X(m)$$

$$s'=m(s) \quad \rightarrow \quad m^{-1}(s')=s' \oplus m^{-1} = s \oplus m \oplus m^{-1}=s$$

Soient M l'ensemble des mouvements applicables aux éléments de X, et $M(s)$ l'ensemble des mouvements applicables à la solution s. On a :

$$M(s)= \{m \in M : m(s)=(s \oplus p) \in X\},$$

et on définit le *voisinage* d'une solution s par :

$$N(s) =\{s' \in X : \exists\, m \in M(s)\,,\ s' = m(s)\}.$$

3.1.2. Algorithme de descente simple

L'algorithme général de descente part d'une solution initiale qu'il essaye d'améliorer de manière itérative. Pour cela, il génère à chaque étape un sous-ensemble V du voisinage $N(s)$ de la solution courante s. Ensuite, il choisit parmi cet ensemble V la meilleure solution, c'est-à-dire celle qui minimise la fonction objectif sur V. Cette dernière solution devient la solution courante et l'algorithme continue jusqu'au moment où aucun élément de V ne permet d'avoir une meilleure fonction objectif. L'algorithme général de la méthode de descente est présenté à la figure 3.1.

Initialisation :	Choisir une solution initiale s dans X.
	Continuer := vrai
Tant que	Continuer **faire**
	Générer un sous-ensemble V de solutions dans N(s)
	Trouver la meilleure solution s' dans V (f(s')=min f(s) avec s $\in$ V)
	Si f(s') $\geq$ f(s) **alors** Continuer := faux
	Sinon s := s'
	Fin si
Fin tant que	

Figure 3.1 Algorithme général de la méthode de descente

À l'examen de cet algorithme, on constate que :

- Le choix du sous-ensemble V de $N(s)$ n'est pas précisé. $V(s)$ peut donc être choisi égal à $N(s)$ (c'est le cas si $|N(s)|$ est assez petit) ou à l'autre extrême, V peut être un singleton dont le seul élément est généré aléatoirement (c'est le cas dans certaines versions de la méta-heuristique de recuit simulé). Le choix de V aura donc une influence sur les résultats obtenus.
- La recherche de la meilleure solution dans V constitue un sous-problème (local) d'optimisation. Elle peut donc s'avérer assez compliquée et nécessiter l'utilisation d'autres heuristiques. La méthode de descente est donc elle aussi une méta-heuristique.

- L'optimum final est un minimum qui n'est pas garanti global et qui souvent est local.

RT essaie surtout de pallier ce dernier défaut, en améliorant la méthode de descente.

3.1.3 Algorithme général de recherche taboue

La méthode de recherche taboue est une amélioration de l'algorithme général de descente. Elle essaie principalement d'éviter le piège des minima locaux. Pour cela, il est nécessaire d'accepter de temps en temps des solutions qui n'améliorent pas la fonction objectif, en espérant ainsi parvenir plus tard à de meilleures solutions. La condition d'arrêt de l'algorithme décrit à la figure 3.1 doit donc être modifiée en conséquence. Cependant, le fait de vouloir accepter des solutions non forcément meilleures introduit un risque de cycle, c'est-à-dire un retour vers des solutions déjà explorées. D'où l'idée de conserver une *liste taboue T* (tabu list) des solutions déjà visitées. Ainsi, lors de la génération de l'ensemble V des solutions voisines candidates, on enlève toutes les solutions appartenant à la liste taboue.

Si la conservation d'une liste taboue permet de conjurer le risque de cycle, elle s'avère peu pratique par ailleurs. En effet, le stockage de toutes les solutions déjà visitées peut nécessiter beaucoup de mémoire. D'autre part, il peut s'avérer utile de revenir à une solution déjà visitée pour continuer la recherche dans une autre direction. Un compromis est obtenu en gardant dans la liste taboue seulement les k dernières solutions. Ainsi, quand une nouvelle solution devient taboue, elle remplace la plus ancienne dans la liste,

ce qui permet d'éviter des cycles de longueur inférieure ou égale à k. Toutefois, des cycles de longueur supérieure à k peuvent toujours survenir.

L'algorithme s'arrête quand aucune amélioration n'est intervenue depuis un nombre *kmax* d'itérations, ou si toutes les solutions voisines candidates sont taboues, c'est-à-dire V- $T = \varnothing$. La figure 3.2 expose l'algorithme général de la méthode de RT. Notons que :

- L'utilisation de la liste taboue T et d'une liste de solutions candidates V introduit une contrainte dans l'exploration de l'espace de recherche et, de ce fait, les solutions obtenues dépendront de la composition de T et de V.
- La méthode de RT ne fait pas référence à des conditions d'optimalité (même locales), sauf implicitement quand un optimum local est meilleur que la meilleure solution s^* obtenue jusque là.
- La méthode choisit à chaque itération la meilleure solution parmi les solutions candidates V, même si cette solution ne constitue pas une amélioration par rapport à s^*, la meilleure solution actuellement connue.

Initialisation : Choisir une solution initiale s dans X.
s* := s (s* est la meilleure solution obtenue à jusqu'ici)
nbiter := 0 (nbiter est le compteur des itérations)
bestiter := 0 (bestiter est le numéro de l'itération à laquelle on a obtenu la dernière amélioration, i.e le dernier s*)
T := ∅ (T est la liste taboue)
Continuer := vrai
Tant que Continuer **faire**
Si (nbiter – bestiter > kmax) **ou** (V-T=∅)
alors Continuer := faux
Sinon nbiter := nbiter + 1
Générer V ⊆ N(s)
Trouver la meilleure solution s' dans V (f(s')=min f(s) avec s ∈ V - T)
s := s'
Mettre à jour T
Si f(s') < f(s*) **alors** s* := s'
bestiter:= nbiter
Fin si
Fin si
Fin tant que

Figure 3.2 Algorithme général de la méthode de recherche taboue

3.2 Caractéristiques de la méthode de recherche taboue

La méthode de RT étant itérative, elle procède par améliorations successives à partir d'une solution initiale. Elle essaie d'explorer « intelligemment » l'espace des solutions possibles (ou espace de recherche). Dans cette section, nous exposerons les

éléments essentiels du fonctionnement de RT, notamment sa manière de parcourir l'espace de recherche, sa structure de mémoire à court terme, ses critères d'aspiration et de fin.

3.2.1 Exploration de l'espace de recherche

Pour minimiser la fonction objectif, la méthode de recherche taboue part d'une solution initiale et essaie d'atteindre un optimum global par l'application de mouvements permettant de passer d'une solution s_i à une solution s_{i+1} choisie dans le voisinage $N(s_i)$ de s_i. À chaque itération i, le voisinage où les solutions sont sélectionnées est redéfini comme le voisinage de la solution courante s_i. À cela, s'ajoute le fait que les conditions taboues changent et les solutions admissibles ne sont plus les mêmes.

L'exploration de l'espace de recherche peut être représentée par un graphe $G=(X,A)$ où X désigne l'ensemble des solutions et A l'ensemble des arcs $(x,m(x))$, $m(x)$ étant la solution obtenue en appliquant le mouvement m à x. Un mouvement donné équivaudra donc à l'ensemble des arcs $\{(x,m(x)),\ x \in X\}$. Le graphe G est symétrique car, pour chaque arc $(x,\ m(x))$, il existe un arc $(m(x),\ x)$ obtenu en appliquant le mouvement inverse m^{-1} à $m(x)$. La méthode de recherche taboue part d'une solution initiale x_0, nœud du graphe G, et cherchera dans G un chemin $x_0, x_1, \ldots, x_k$, où $x_i = m(x_{i-1})$ avec $i=1,\ldots,k$. Chaque solution intermédiaire du chemin est obtenue en appliquant un mouvement à la précédente. Les arcs $(x_i,\ x_{i+1})$ du chemin sont choisis en résolvant le problème d'optimisation :

$$f(x_{i+1}) = \min f(x_i) \text{ avec } x_i \in V - T.$$

La méthode de RT se distingue donc de celle de descente générale par l'utilisation d'une structure de mémoire à court terme qui permet d'accepter des solutions moins bonnes pour sortir des optima locaux, tout en évitant les cycles.

3.2.2 Mémoire à court terme

La liste taboue de RT est une structure de mémoire à court terme qui garde les $k=|T|$ dernières solutions visitées (ou mouvements appliqués) en vue d'éviter les cycles. Elle est implémentée sous forme d'une liste FIFO. La dernière solution visitée (ou le dernier mouvement) remplace la première, c'est-à-dire la plus ancienne de la liste. La LT permet aussi, si un cycle survient malgré toutes les précautions, d'amener avec une plus grande probabilité, l'exploration à se faire dans une direction différente. La LT intervient donc surtout lors de la sélection de la meilleure solution admissible dans V. La figure 3.3 résume le fonctionnement de la mémoire à court terme dans le processus global de RT. Dans cette figure, on remarque l'introduction des notions de mémoire à moyen et long terme dont on reparlera plus loin. On introduit aussi la notion de critère d'aspiration.

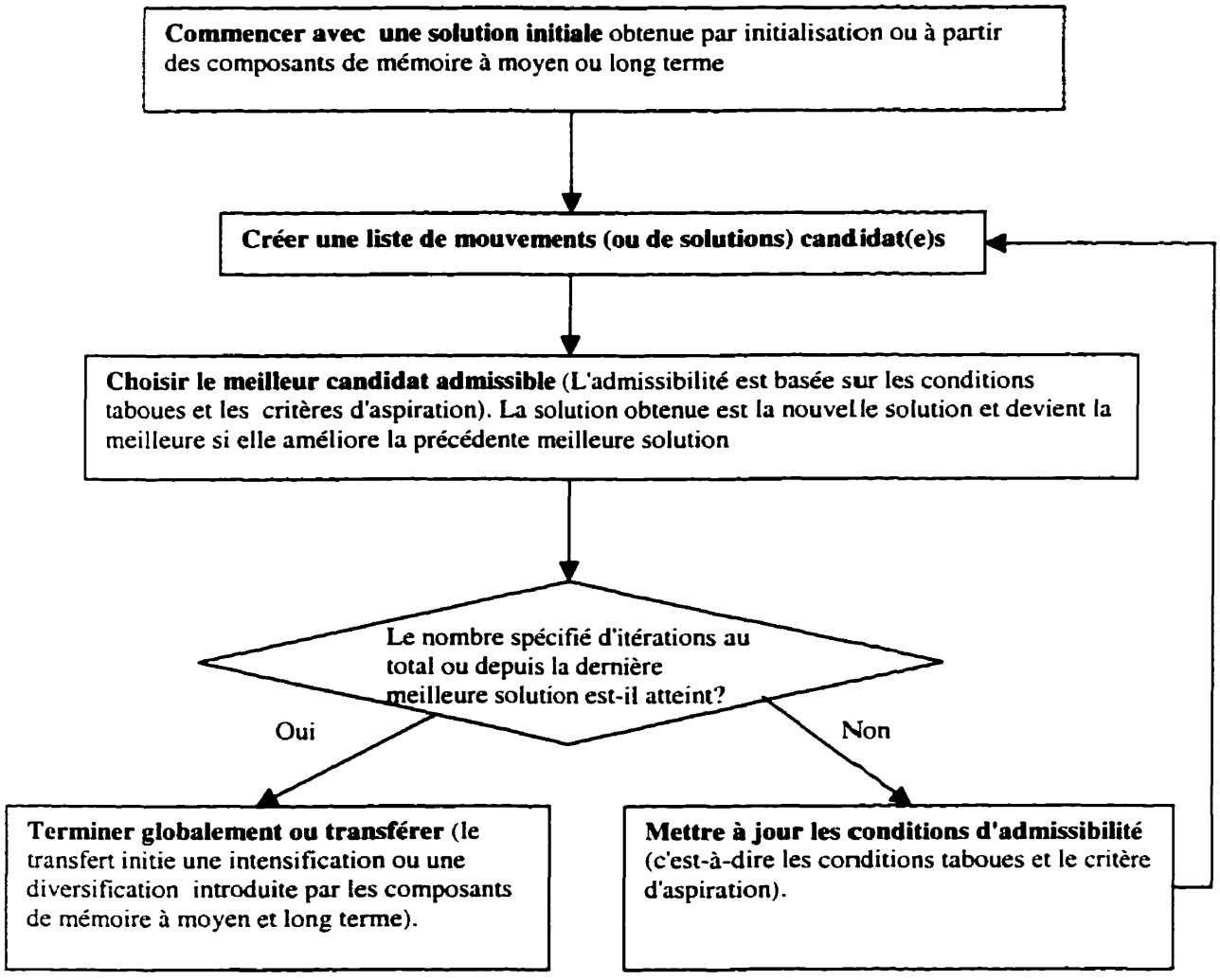

Figure 3.3 Composante de mémoire à court terme de la méthode de recherche taboue

3.2.3 Critère d'aspiration

La méthode de RT inclut des structures lui évitant de boucler en revenant à des solutions déjà visitées. Cependant, il peut être nécessaire de revenir à une solution déjà explorée pour continuer la recherche dans une autre direction. Pour ce faire, on modifie la mémoire à court terme par l'introduction d'un critère d'aspiration qui permet d'annuler

temporairement le statut tabou d'un mouvement (ou d'une solution), afin de le rendre admissible.

On définit un *seuil ou critère d'aspiration* $A(s,m)$ et on associe à chaque mouvement m applicable à une solution s, un *niveau d'aspiration* $a(s,m)$. Plus généralement, le critère d'aspiration $A(s,m)$ est défini comme un ensemble de valeurs préférentielles prises par le niveau d'aspiration $a(s,m)$. Une solution vérifie le critère d'aspiration si son niveau d'aspiration est une des valeurs préférentielles, c'est-à-dire $a_i(s,m) \in A_i(s,m)$, $i=1,\ldots,I$, où $a(s,m)$ est une fonction vectorielle de composantes $a_i(s,m)$ et $A(s,m)$ un ensemble de points de l'espace de dimension I. Le statut tabou d'un mouvement m est annulé si ledit mouvement vérifie une ou un nombre spécifié des conditions ci-dessus.

Par exemple, le niveau d'aspiration d'une solution peut être son coût, et le seuil ou critère d'aspiration $A(s,m)$ sera l'ensemble des valeurs inférieures au coût de la meilleure solution actuellement connue. Ainsi, un mouvement tabou m sera accepté si son coût est inférieur à celui de la meilleure solution courante.

La sélection de la meilleure solution admissible dans V est donc modifiée. La figure 3.4 décrit en détail les différents mécanismes mis en œuvre.

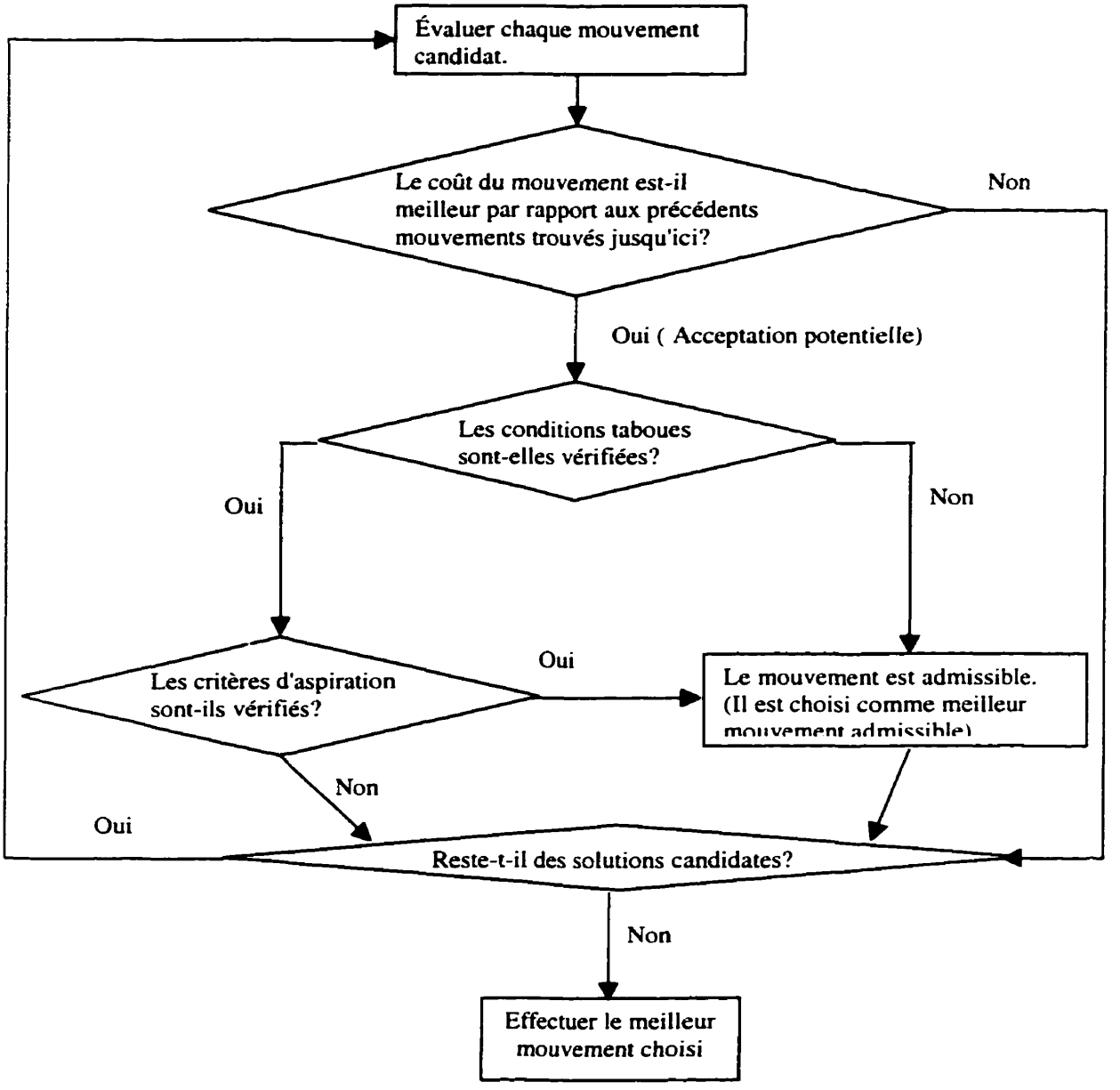

Figure 3.4 Procédure de sélection du meilleur candidat admissible

En résumé, RT trouve la meilleure solution du voisinage *V* en évaluant chaque solution de *V*. Ensuite, elle vérifie son statut tabou. Si le mouvement générant la solution est tabou, RT examine son critère d'aspiration. En fin de compte, la meilleure solution

non taboue ou la meilleure solution taboue qui vérifie les critères d'aspiration est sélectionnée. Si l'évaluation du niveau d'aspiration des mouvements n'est pas trop coûteuse, il est préférable de tester d'abord si un mouvement aboutit à une solution ayant un « bon » niveau d'aspiration, avant de vérifier son statut tabou.

3.2.4 Critères de fin

Puisqu'elle n'a pas un test pour déterminer l'optimum global, la méthode de RT, lorsqu'elle arrive à un optimum, continue sa recherche en considérant le meilleur résultat actuel comme un optimum local et en cherchant de meilleurs optima. Ainsi, en l'absence d'un test d'optimum global, on doit définir d'autres critères d'arrêt. La méthode s'arrête lorsque l'une au moins des quatre situations suivantes se présente :

- aucun mouvement n'est possible à partir de la solution courante, c'est-à-dire tous les mouvements du voisinage sont tabous et ne vérifient pas le critère d'aspiration donc ne sont pas admissibles;
- le nombre total d'itérations est supérieur au nombre permis;
- le nombre d'itérations depuis la dernière meilleure solution est supérieur à un nombre spécifié;
- un certain temps s'est écoulé depuis le début de l'algorithme.

Si on connaît une «bonne» borne inférieure f^* de la fonction objectif $f(x)$, on peut aussi arrêter quand la meilleure solution actuelle a un coût suffisamment proche de f^*.

Les critères de fin peuvent être aussi utilisés dans l'implémentation de stratégies de recherche plus élaborées, notamment les stratégies de diversification ou

d'intensification. Quand un optimum local est atteint, on peut, au lieu de terminer la recherche, faire un transfert vers une autre région de recherche (diversification) ou raffiner la recherche (intensification) dans la région courante. La figure 3.3 illustre ces différentes possibilités. Nous reviendrons plus tard sur les stratégies de diversification et d'intensification.

3.3 Adaptation de la méthode de recherche taboue

La méta-heuristique de RT peut s'appliquer à divers types de problèmes. Pour être efficace, elle nécessite donc une adaptation de certains de ces paramètres au type du problème. Les définitions de ces adaptations peuvent être plus ou moins intuitives. Nous présenterons les paramètres généraux d'adaptation de RT, plus particulièrement la liste de solutions candidates et la liste taboue.

3.3.1 Liste de solutions candidates

Pour chaque application, la méthode de RT possède un certain nombre d'attributs qui doivent être adaptés au contexte. Globalement, on définit des mouvements adaptés au problème et à partir de ces mouvements, on déduit la notion de voisinage d'une solution. Ceci étant fait, on se penche sur la liste des solutions candidates V qui est un sous-ensemble du voisinage $N(s)$ défini précédemment. Ensuite, on définit les attributs de la liste taboue, plus précisément sa taille et sa représentation. La définition des critères d'aspiration et d'arrêt dans le contexte du problème complète les éléments de RT.

L'une des difficultés de la méthode de RT est de déterminer la liste des solutions candidates, c'est-à-dire le sous-ensemble $V \subseteq N(s)$. Prendre $V = N(s)$ donne de bons résultats (Glover et al., 1993) mais demande beaucoup de calcul, car on doit examiner tout le voisinage. On choisit donc en général comme ensemble V, un petit sous-ensemble du voisinage $N(s)$ tout en ayant soin de considérer quand même les bonnes solutions. Le choix de V peut se faire selon deux stratégies : les meilleurs candidats et les mouvements préférés.

Cas 1 : ***Meilleurs candidats***

V est représenté sous forme de mouvements candidats et contient les mouvements qui conduisent à une solution de meilleur coût. À une itération donnée, on examine d'abord les mouvements dans V (qui proviennent des itérations précédentes). Ensuite, on considère une partie du voisinage courant $N(s)$ et on remplace dans V les solutions devenues obsolètes ou peu intéressantes par des solutions issues du voisinage courant. Périodiquement, après un certain nombre d'itérations ou quand la qualité globale de V se détériore, on inspecte une plus grande portion de $N(s)$ pour reconstruire la liste V. Cette approche est motivée par le fait qu'un bon mouvement, s'il n'est pas appliqué à l'itération courante, demeure encore un bon mouvement, pour un certain nombre d'itérations.

Cas 2 : *Mouvements préférés*

Dans certaines applications, on isole certains attributs communs aux mouvements menant à une bonne solution. On limite ensuite V aux solutions obtenues par des mouvements ayant les attributs préférentiels détectés.

3.3.2 Liste taboue

Selon le problème et l'importance accordée au caractère tabou, la LT peut être représentée sous diverses formes. Celles-ci donnent lieu à autant de cas que nous allons expliciter ci-après.

Cas 1 : *Représentation implicite de la LT*

Si la taille du voisinage est assez petite, on peut associer à chaque mouvement des éléments qui définissent son statut tabou. Par exemple, on peut lier aux mouvements le numéro d'itération (c'est-à-dire le moment) où ils ne sont plus tabous. Ainsi, le statut tabou d'un mouvement se vérifie en temps constant, par examen de son numéro d'itération. De plus, l'espace mémoire nécessaire ne dépend pas de la taille de la LT mais de celle du voisinage. Cependant, dans la plupart des cas, le voisinage est assez grand et la LT est représentée par un ensemble de solutions interdites ou de mouvements interdits.

Cas 2 : *Représentation de la LT sous forme d'une liste de solutions taboues*

On conserve dans la LT les $k=|T|$ dernières solutions générées. À chaque fois qu'on génère un voisinage $V \subseteq N(s)$, les solutions candidates ne doivent pas appartenir à la LT. Ainsi, on est sûr qu'on n'aura pas de cycle de longueur inférieure à k. D'un autre côté, on ne peut pas revenir, avant k itérations, à une solution déjà visitée afin de continuer l'exploration dans une autre direction. Pour pallier cet inconvénient, on peut adopter des listes taboues de tailles variables portant sur différents attributs des solutions.

En pratique, il est difficile de représenter la LT sous forme de liste de solutions car cela exige une grande capacité de stockage. De plus, comme on teste assez souvent si une solution est dans la LT, ceci peut prendre beaucoup de temps. On représente donc la LT sous forme de mouvements interdits.

Cas 3 : *Représentation de la LT sous forme d'une liste de mouvements tabous*

Dans ce cas, ce sont des mouvements qui sont enregistrés dans la liste taboue. À chaque itération où on effectue un mouvement m, on met dans la liste taboue l'inverse m^{-1} de m. On espère ainsi interdire tous les mouvements qui retournent à des solutions déjà visitées.

Cette représentation a aussi ses inconvénients. Ainsi, le fait d'interdire des mouvements à la place des solutions peut s'avérer trop rigide (en interdisant un mouvement, on proscrit toute une classe de solutions au lieu d'une seule : exemple 3.1)

ou trop flexible (on peut boucler par un chemin autre que celui interdit par la LT : exemple 3.2)

Exemple 3.1

Soit un ensemble X de solutions faisables S qui correspondent aux paires de l'ensemble $W= \{a,b,c,d,...\}$. Un mouvement m se définit ici comme le remplacement de i dans S par un j dans W-S, et on note $m= (i \leftarrow j)$. La liste taboue est représentée comme un ensemble de mouvements interdits et sa taille est fixée à 3. On part de la solution $S=\{a,b\}$ et on cherche à aller à la solution $S=\{a,e\}$.

Tableau 3.1 Interdiction d'un mouvement susceptible d'amener à des solutions non encore visitées

S Solution courante	m Mouvement	LT Liste taboue
a b	c ← b	b ← c
a c	d ← a	b ← c; a ← d
c d	e ← c	b ← c; a ← d; c ← e
d e		

On observe que la LT interdit le mouvement $a \leftarrow d$ de telle sorte qu'on ne peut pas atteindre la solution $\{a,e\}$ non encore visitée.

Exemple 3.2

Les solutions S sont des sous-ensembles de W de cardinalité 3. La LT a une taille supérieure ou égale à 4. On part de $S=\{a,b,c\}$. On observe qu'après trois itérations on retourne à la solution initiale, alors que la LT a une taille supérieure à 3.

Tableau 3.2 Retour à des solutions déjà visitées

S Solution courante	m Mouvement	LT Liste taboue
a b c	d ← c	c ← d
a b d	c ← b	c ← d; b ← c
a c d	b ← d	c ← d; b ← c; d ← b
a b c		

On constate qu'il n'est pas toujours facile d'adapter la liste taboue au problème courant. Même une bonne adaptation peut se révéler inefficace dans certaines conditions. C'est la raison pour laquelle la liste taboue est utilisée conjointement avec des mécanismes d'aspiration et parfois de mémoire à moyen et long terme que nous aborderons dans la prochaine section.

En général, il est préférable d'adopter une représentation de la LT qui ne limite pas trop le choix des mouvements disponibles et qui n'introduit pas trop de rigidité. Cependant, le mieux est toujours un compromis entre la flexibilité et la rigidité des contraintes introduites par la LT. Dans certains cas où on a des attributs sujets à des

contraintes différentes, on peut utiliser plusieurs listes taboues. La taille de ces listes ne devrait être ni trop élevée, ni trop petite. Selon Glover (1989), elle devrait se situer autour de 7, indépendamment de la taille et de la structure du problème.

3.4 Raffinements possibles

Dans plusieurs applications, l'implémentation de RT avec la structure de mémoire à court terme suffit pour générer de meilleurs résultats comparés à d'autres méthodes. Toutefois, on peut ajouter des composants de mémoire à moyen et long terme (Skorin-Kapov, 1989) qui intensifient et diversifient la recherche. La forme modulaire de RT permet d'implanter d'abord la composante à court terme, et d'ajouter par la suite les autres composantes, si cela s'avère nécessaire.

3.4.1 Mémoire à moyen terme

La mémoire à moyen terme procède par comparaison des meilleures solutions générées par RT pendant une période donnée. Les attributs qui sont communs à toutes les solutions ou à une grand partie de celles-ci (comme par exemple, la valeur prise par certaines variables) sont considérés comme des prémices d'une bonne solution. Pendant la période d'intensification de la recherche, RT cherche des solutions ayant les attributs désirés en pénalisant ou en restreignant les mouvements disponibles.

Le cas de problèmes où la taille des données est grande se prête bien à l'application d'un tel mécanisme. Au début, on cherche les solutions en exploitant toutes les données disponibles. Après un certain nombre d'itérations, on déduit un nouvel

ensemble de données constitué uniquement des attributs présents dans les premières solutions générées. On suppose que les attributs qui ne sont pas utilisés dans les premières solutions ne le seront pas non plus dans d'autres solutions. La taille du problème est donc réduite et on peut examiner plus de possibilités de solutions, ce qui augmente les chances d'en trouver une « bonne ».

3.4.2 Mémoire à long terme

À l'inverse de la mémoire à moyen terme, celle à long terme cherche à diversifier la recherche. Pour cela, elle utilise des principes intrinsèquement opposés à ceux de la mémoire à moyen terme. Au lieu de concentrer la recherche dans des régions qui contiennent de bonnes solutions trouvées précédemment, la mémoire à long terme guide l'exploration vers des régions qui contrastent avec celles explorées jusque là. Elle essaie donc de tirer profit de l'histoire de la recherche pour diversifier l'exploration. En cela, elle diffère des méthodes qui diversifient leur recherche en générant aléatoirement plusieurs solutions initiales sans tenir compte des explorations déjà effectuées. Cependant, certaines versions de RT allient l'aspect aléatoire et l'utilisation des statistiques de recherche (Glover, 1989).

La mémoire à long terme crée un critère d'évaluation qui pénalise les attributs qui sont fréquents dans les solutions déjà trouvées. Ce critère est ensuite utilisé pour générer de nouvelles solutions initiales. La mémoire à long terme peut être vue comme une liste taboue sanctionnant les solutions les plus fréquentes, alors que la mémoire à court terme, elle, sanctionne les solutions les plus récentes.

3.5 Exemples d'application de la méthode de recherche taboue

La méthode de RT est très utilisée en optimisation combinatoire. Elle a été appliquée avec succès à plusieurs problèmes difficiles tels : le problème d'affectation quadratique (Skorin-Kapov, 1989), le problème de coloration de graphes (Hertz et de Werra, 1987), le problème de conception topologique de réseau (Lee, 1989; Pierre et Elgibaoui, 1997). Nous présentons deux exemples d'application de RT. Le premier concerne le problème d'arbre de recouvrement minimum avec contraintes et le deuxième, le problème de partitionnement de graphes.

3.5.1 Problème d'arbre de recouvrement minimum avec contraintes

Le problème d'arbre de recouvrement minimum (Glover, 1990a) consiste à trouver dans un graphe pondéré $G=(N,A)$, l'arbre de recouvrement minimum sous des contraintes de type :

(i) certains arcs ne peuvent apparaître ensemble dans l'arbre;

(ii) la présence de certains arcs dans l'arbre de recouvrement est reliée à celle d'autres arcs.

La complexité du problème d'arbre de recouvrement sans aucune contrainte est polynomiale. L'introduction des contraintes (i) et (ii) complique le problème et le rend difficile. Pour appliquer la méthode de RT à ce problème, on définit des mouvements m caractérisés par :

m : *ajout d'un arc et retrait d'un autre afin de transformer l'arbre courant en un nouvel arbre.*

Le mouvement *m* a pour attributs l'arc à retirer x_i et l'arc à ajouter x_j. x_i et x_j sont reliés par le fait que x_i est toujours dans le cycle résultant de l'ajout de x_j. La liste des solutions candidates *V* est égale au voisinage *N(s)*. La liste taboue *T* a une longueur *k* égale à 2 et est constituée en affectant un statut tabou aux deux derniers arcs ajoutés. Un mouvement sera tabou s'il implique le retrait d'un arc tabou. La fonction objectif est la somme des coûts de tous les arcs présents dans l'arbre. On définit le critère d'aspiration par l'ensemble des valeurs inférieures au coût de la meilleure solution courante. Si une solution *s*, résultant d'un mouvement tabou, a un coût meilleur que celui de la meilleure solution déjà obtenue, on ne tient pas compte du caractère tabou de *s*.

Cet exemple simple est cité pour illustrer rapidement les divers détails de l'implémentation de RT. L'application de la méthode de RT à ce problème mène à l'optimum global.

3.5.2 Partitionnement de graphe

Le problème de partitionnement de graphes consiste à diviser l'ensemble *N* des nœuds d'un graphe en des sous-ensembles de cardinalité inférieure à un nombre maximal donné, de manière à minimiser le coût total des coupes, c'est-à-dire la somme des coûts des arêtes ayant leurs extrémités dans des sous-ensembles différents. Ce problème avait été déjà défini au chapitre 2, illustré à la figure 2.1 et comparé avec le problème d'affectation de cellules. Nous exposons dans cette section, une application de

RT à un cas particulier du problème de partitionnement : la bipartition uniforme de graphes. Dans ce cas, on cherche à diviser l'ensemble des nœuds N en deux sous-ensembles dont les cardinalités diffèrent d'au plus un élément (Kernighan et Li, 1970).

Plus formellement, soit (i,j) un arc e de G, w_e ou $w(i,j)$, le coût (ou poids) de cet arc, et $s=(S,N\setminus S)$ une bipartition uniforme. Soit $\delta(S) = \{(i,j) \in N : i \in S, j \in N\setminus S\}$ l'ensemble des arcs de la coupe définie par la partition s. Si $|N|=n$, on cherche à :

Minimiser $z(s) = \sum_{e \in \delta(S)} w_e$

sujet à :

$|S| = n/2$

On peut supposer, sans perte de généralité, que n est pair et $n/2$ entier.

Dell'Amico et Trubian (1998) utilisent la méthode de RT pour résoudre le problème de bipartition uniforme. Les mouvements m sont définis par :

m(a,b) : *permutation de la paire de nœuds (a, b) avec* $a \in A=S$ *et* $b \in B=N\setminus S$

Soient $E_a = \sum_{j \in B} w(a,j)$ le *coût externe du nœud* $a \in A$ et $I_a = \sum_{j \in A} w(a,j)$ son *coût interne*. On définit de façon similaire les coûts interne et externe du nœud $b \in B$. Pour un nœud j, la différence $D_j=E_j - I_j$ est le gain sur la fonction objectif $z(s)$ si on déplace j de son sous-ensemble (par exemple A) vers l'autre sous-ensemble (par exemple B). Après l'application du mouvement $m(a,b)$, a se retrouve dans $B=N\setminus S$, b dans $A=S$ et la fonction objectif $z(s')$ de la nouvelle solution s' est diminuée de $D_a + D_b - 2w(a,b)$.

Le voisinage $N(s)$ d'une solution $s=(A,B)$ est défini par tous les mouvements qui permutent une paire de nœuds dont l'un provient de l'ensemble A et l'autre de l'ensemble

B. Pour déterminer la liste *V* de solutions candidates, on calcule pour chaque nœud *n* de *N*, le gain D_n : en fait, les D_n sont calculés au début et il existe des techniques très efficaces pour les mettre à jour (Kernighan et Li, 1970). Les ensembles *A* et *B* sont ordonnés par valeur non décroissante des gains D_n. Les mouvements candidats sont choisis en parcourant les deux ensembles *A* et *B* selon les gains décroissants. La logique qui sous-tend cette approche est que le gain sur la fonction objectif croît avec les gains de chacun des nœuds permutés. On a donc des mouvements préférés qui sont ceux qui permutent deux nœuds ayant des gains élevés.

L'algorithme démarre avec une solution faisable et applique des mouvements qui conservent la faisabilité. La composante de mémoire à court terme est constituée de deux listes taboues : *was_in_A* et *was_in_B*. Un mouvement est caractérisé par les noms et les ensembles d'origine des deux nœuds permutés. Pour un mouvement *m(a,b)*, on enregistre le nœud *a* dans *was_in_A* et le nœud *b* dans *was_in_B*, et pendant un certain nombre d'itérations subséquentes, tous les mouvements qui proposent le retrait du nœud *a* de *B* (ou le retrait du noeud *b* de *A*) seront tabous. La longueur *k* des listes taboues est variable. Initialement, *k* est fixé à une valeur *tabu_tenure* égale à 10. Ensuite, si on a Δip=5 itérations consécutives qui diminuent la valeur de la fonction objectif, on diminue *k* et on le fixe au *max(k-1, 0.5*tabu_tenure)*. Si au contraire, on a Δwp=3 itérations consécutives qui ne diminuent pas la valeur de la fonction objective, on augmente *k* en le fixant au *min(k+1, 1.5*tabu_tenure)*.

Le statut tabou est supprimé par deux critères d'aspiration : le premier consiste simplement à accepter un mouvement tabou s'il améliore la meilleure solution

actuellement connue: le deuxième critère prend en compte les cas où, après la permutation de deux nœuds, disons *(a,b)*, et l'exécution d'au moins une autre permutation n'impliquant ni *a* ni *b*, une bonne solution peut être atteinte en permutant *a* ou *b*, mais non *a* et *b*.

Les éléments d'intensification et de diversification sont mis en œuvre par la variation de la longueur *k* de la liste taboue. Une réduction de *k* correspond à une intensification locale, et une augmentation à une diversification. À cela, s'ajoute une structure de mémoire à long terme. Dans cette structure appelée *Second*, on garde quelques-unes des bonnes solutions examinées, mais qui n'ont pas été choisies car elles n'étaient pas les meilleures solutions admissibles. *Second* est implanté avec une liste ordonnée de longueur *L*. Soit s_2 la meilleure seconde solution trouvée dans le voisinage courant. s_2 est ajouté à *Second* si ce dernier n'est pas plein, ou bien s_2 remplace s_w, la pire solution dans *Second*, si s_2 est mieux que s_w. La solution s_2 est ajoutée à *Second* avec une copie, dans leur état courant, des deux listes taboues. Notons que si lors de l'exploration des solutions candidates dans V, on trouve directement la meilleure solution, on ne continue pas à chercher une seconde meilleure solution pour l'ajouter à *Second*. On passe plutôt directement à l'étape suivante. Une deuxième procédure d'intensification est mise en œuvre en utilisant la meilleure solution dans *Second* (avec les listes taboues qui lui ont été associées) comme nouveau point de départ, dans l'un des cas suivants :

- on ne trouve aucune solution admissible dans le voisinage courant;
- pendant une série de MC=7 itérations, la valeur de la fonction objectif ne s'améliore pas;

- pendant une série de *MB*=200 itérations la meilleure solution trouvée ne s'améliore pas.

La longueur *L* de *Second* est fixée entre 5 et le degré moyen des nœuds. Cela permet de ramener la recherche dans des zones prometteuses non encore totalement explorées. Le fait d'ajouter au plus une solution à *Second* dans chaque voisinage joue un rôle de diversification de la recherche.

Enfin, l'algorithme se termine après un temps *max_time* prédéfini. Il fournit un plus grand nombre de meilleures solutions en des temps plus courts que d'autres méthodes utilisées pour le partitionnement de graphes (Dell'Amico et Trubian, 1998).

Globalement donc, la méthode de RT s'est révélée assez souple et efficace pour résoudre une grande variété de problèmes. Dans le chapitre suivant, nous entreprendrons de l'adapter à la résolution du problème spécifique d'affectation de cellules à des commutateurs dans les RCP

CHAPITRE 4

AFFECTATION DE CELLULES PAR LA MÉTHODE DE RECHERCHE TABOUE

Le problème d'affectation de cellules est un problème difficile qui ne peut pas se résoudre de manière exacte en des temps de calcul acceptables. Le compromis que nous proposons consiste à utiliser la méta-heuristique de recherche taboue (RT) pour obtenir des solutions acceptables en des temps de calcul raisonnables. Ce chapitre expose une adaptation de la méthode de RT à ce problème d'affectation. La démarche consiste globalement à modifier itérativement une solution initiale en espérant aboutir à une solution finale respectant les contraintes du problème. Après avoir esquissé les grandes lignes de notre adaptation, nous exposons les détails d'implémentation et de mise en œuvre des trois composantes de mémoire à court, moyen et long termes.

4.1 Adaptation de la méthode de RT

La méthode de RT utilise des mouvements pour passer d'une solution à une autre à l'intérieur d'un espace de recherche prédéfini. Si le sous-ensemble de solutions à explorer est trop grand, le coût induit par l'algorithme peut devenir prohibitif. À l'opposé, un sous-ensemble trop petit limite la qualité des solutions obtenues. Dans notre adaptation, l'espace de recherche choisi est libre des contraintes de capacité sur les commutateurs, mais respecte la contrainte d'affectation unique des cellules aux

commutateurs. La faisabilité de la solution finale n'est donc pas garantie, mais le fait de pouvoir examiner un plus grand nombre de possibilités augmente les chances d'aboutir à de bonnes solutions. RT associe à chaque solution deux valeurs numériques : la première est le coût intrinsèque (ou tout simplement coût) de la solution, calculé à partir de la fonction objectif; la deuxième est une évaluation de la solution prenant en compte le coût et une sanction pour le respect des contraintes de capacité. Cette sanction comprend une partie fixe appelée *PENALITEFIXE* et une partie variable par unité de violation désignée par *PENALITE*. À chaque étape, RT choisit la solution ayant la meilleure évaluation.

Contrairement à la méthode de descente, la méthode de RT, quand elle arrive à un optimum local, choisit la solution voisine qui dégrade le moins la fonction objectif. Pour éviter les cycles autour de cet optimum, une liste taboue garde les k dernières solutions et interdit momentanément un retour vers ces solutions. Les solutions sont libérées après k itérations ou lorsqu'elles satisfont un critère d'aspiration. Une composante de mémoire à moyen terme garde les m dernières meilleures solutions et essaie, plus tard, d'intensifier la recherche dans leurs voisinages. En se basant sur les statistiques de recherche, la composante de mémoire à long terme détermine un nouveau point de départ contrastant le plus possible avec les solutions visitées jusqu'ici.

Notre adaptation de la méthode de RT construit une solution initiale à partir des données du problème. La composante de mémoire à court terme essaie par la suite d'améliorer itérativement cette solution tout en évitant les cycles. Une composante de mémoire à moyen terme intensifie la recherche dans des voisinages précis, tandis que la

composante de mémoire à long terme permet une diversification de l'exploration du domaine.

4.2 Mémoire à court terme

La mémoire à court terme passe itérativement d'une solution à une autre par l'application de mouvements, tout en interdisant un retour vers les *k* dernières solutions visitées. Elle débute donc avec une solution initiale.

4.2.1 Génération de la solution initiale

Le domaine d'exploration défini plus haut étant libre des contraintes de capacité sur les commutateurs, la solution initiale doit seulement respecter les contraintes d'affectation unique des cellules. Le Tableau 4.1 illustre les coûts de liaison pour un réseau de 14 cellules et 3 commutateurs.

Tableau 4.1 Coûts de liaison pour un réseau de 14 cellules et 3 commutateurs

Cellules	Comm. 0	Comm. 1	Comm. 2
0	0	2	2
1	1	1.73	1.73
2	1	1	2.65
3	1	1.73	3
4	1	2.65	2.65
5	1	3	1.73
6	1	2.65	1
7	1.73	1	2.65
8	1.73	1	3.61
9	2	0	3.46
10	2	2	2
11	1.73	2.65	1
12	2	3.46	0
13	1.73	3.61	1

La solution initiale est générée selon l'algorithme de la Figure 4.1. En effet, selon cet algorithme, on affecte tout simplement chaque cellule au commutateur le plus proche en terme de distance qui les sépare. Si une cellule se trouve à égale distance de deux commutateurs, elle est affectée au premier commutateur dans l'ordre spécifié par le problème.

Initialisation : Obtenir la matrice de coût de liaison.
Pour chaque cellule *i* faire :
Meilleur := +∞;
Meilleur_comm := 0; (*Meilleur_comm* est le commutateur le plus près de *i*)
Pour chaque commutateur *k* faire :
Si c_{ik} < *Meilleur*
alors *Meilleur* := c_{ik};
Meilleur_comm := *k*;
Fin si
Fin pour
Affecter la cellule *i* au commutateur *Meilleur_comm* ;
Fin pour

Figure 4.1 Algorithme de génération de la solution initiale

La solution initiale obtenue à partir du Tableau 4.1 est celle de la Figure 4.2. Son coût est de 132 unités.

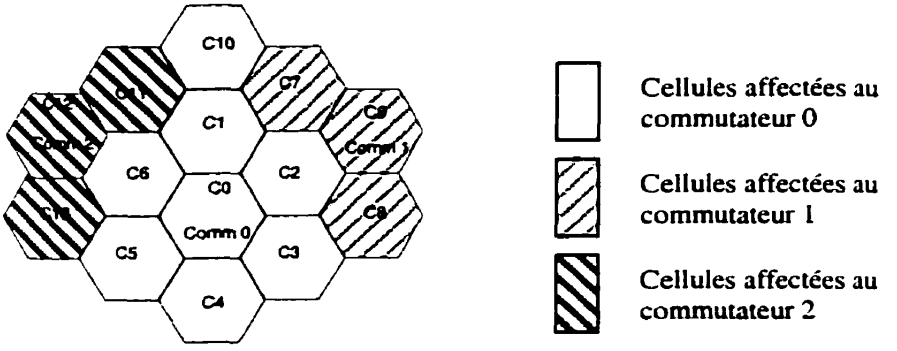

Figure 4.2 Solution initiale

Le Tableau 4.2 indique les volumes d'appels ou le trafic des cellules, ainsi que les capacités des commutateurs. La première ligne donne le numéro de la cellule ou du commutateur, et les deux autres lignes les volumes d'appels des cellules ou les capacitées des commutateurs. Ainsi, le commutateur 1 a une capacité de 8 unités, et la cellule 10 génère un volume d'appels d'une unité.

Tableau 4.2 Volumes d'appels et capacités

N°	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Cell.	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Comm.	7	8	7											

La solution de la Figure 4.2 viole les contraintes de capacité notamment au niveau du commutateur 0 où le volume total des appels affectés est de 8 unités, alors que sa capacité est de 7 unités. De ce fait, son évaluation diffère de son coût . Pour une valeur de 100 unités de la constante *PENALITEFIXE* et une valeur de 50 unités de *PENALITE*, l'évaluation s'élève à 282 unités.

4.2.2 Mouvements et gain

Le but de la composante de mémoire à court terme est d'améliorer la solution courante, soit en diminuant son coût, soit en diminuant les pénalités encourues. Pour y parvenir, elle utilise des mouvements *m(a,b)* définis comme suit :

m(a,b) *: réaffectation de la cellule au commutateur b.*

Ce type de mouvement offre une grande souplesse. En effet, il ne comporte pas de contraintes de capacité. De plus, il constitue un mouvement de base permettant d'effectuer toutes sortes de mouvements plus complexes, comme par exemple la permutation de deux cellules. Les solutions explorées sont alors plus nombreuses, puisqu'on n'est pas limité aux seules solutions faisables.

Le voisinage *N(S)* d'une solution *S* est défini par toutes les solutions accessibles à partir de *S* par l'application d'un mouvement *m(a,b)* à *S*. Pour évaluer rapidement les solutions du voisinage *N(S)*, on définit le gain $G_S(a,b)$ associé au mouvement *m(a,b)* et à la solution *S* par :

$$G_S(a,b) = \begin{cases} \sum_{i=1, i\neq a}^{n} (h_{ai} + h_{ia}) x_{ib_0} - \sum_{i=1, i\neq a}^{n} (h_{ai} + h_{ia}) x_{ib} + c_{ab} - c_{ab_0} & \text{si } b \neq b_0 \\ M & \text{sinon} \end{cases} \qquad (4.1)$$

où :

- h_{ij} désigne le coût de relève entre les cellules i et j ;
- b_0 le commutateur de la cellule a dans la solution S, c'est-à-dire avant l'application du mouvement *m(a,b)* ;
- x_{ik} prend la valeur 1 si la cellule i est affectée au commutateur k, 0 sinon ;

- c_{ik} est le coût de liaison de la cellule i au commutateur k ;
- M est un nombre arbitrairement grand.

À chaque itération, la méthode de RT sélectionne, parmi tous les mouvements possibles, celui ayant le gain minimum. Si $b \neq b_0$, $G_S(a,b)$ représente le gain sur le coût $f(S)$ de la solution S lorsqu'on effectue le mouvement $m(a,b)$. Pour $b = b_0$, le mouvement $m(a,b)$ ne modifie pas la solution et le gain $G_S(a,b)$ devrait donc être nul. Mais, dans ce dernier cas, on affecte au gain une valeur arbitrairement grande. Ainsi, lorsqu'on arrive à un optimum local et qu'aucun des mouvements disponibles n'améliore le coût de la solution, le mouvement ayant le gain minimum ne sera pas le mouvement $m(a,b_0)$ qui cycle sur la même solution. Le coût de la nouvelle solution S' est tout simplement obtenue à partir de la formule :

$$f(S') = f(S) + G_S(a,b) \tag{4.2}$$

Au début, on génère dans un tableau $n \times m$, où n désigne le nombre de cellules et m le nombre de commutateurs, tous les gains $G_S(i,k)$. Après chaque mouvement $m(a,b)$, on met à jour le tableau des gains. Cette mise à jour se fait rapidement, car seules les colonnes correspondant aux commutateurs b et b' et la ligne de la cellule a changent. Pour une cellule p et un commutateur q, notons C_p le commutateur de p dans la nouvelle solution S'. Les nouveaux gains $G_{S'}(a,b)$ sont obtenus à partir des anciens par les formules suivantes :

$$G_{S'}(p,q) = G_S(p,q) \qquad \text{si } C_p \neq b',\ C_p \neq b,\ q \neq b',\ q \neq b \tag{4.3}$$

$$G_{S'}(p,q) = M \qquad \text{si } C_p = q \tag{4.4}$$

$$G_{S'}(p,q) = G_S(p,q) - 2(h_{ap} + h_{pa}) \quad \text{si } C_p = b',\ q = b \tag{4.5}$$

$$G_{S'}(p,q) = G_S(p,q) - (h_{ap} + h_{pa}) \quad \text{si } C_p = b',\ q \neq b',\ q \neq b \tag{4.6}$$

$$G_{S'}(p,q) = G_S(p,q) + 2(h_{ap} + h_{pa}) \quad \text{si } C_p = b,\ p \neq a,\ q = b' \tag{4.7}$$

$$G_{S'}(p,q) = G_S(p,q) + (h_{ap} + h_{pa}) \quad \text{si } C_p = b,\ q \neq b',\ q \neq b \tag{4.8}$$

$$G_{S'}(a,b') = -G_S(a,b) \tag{4.9}$$

$$G_{S'}(a,q) = G_S(a,q) - G_S(a,b) \quad \text{si } q \neq b',\ q \neq b \tag{4.10}$$

4.2.3 Liste taboue et aspiration

À chaque itération, RT définit un voisinage de la solution courante composée des solutions engendrées par tous les mouvements $m(a,b)$. Elle choisit ensuite le mouvement qui améliore le plus le coût de la solution, c'est-à-dire celui ayant le gain minimum. Si aucun mouvement n'améliore le coût actuel, la solution engendrant une dégradation minimum du coût est sélectionnée. Pour chaque mouvement $m(a,b)$ effectué, la méthode garde dans une liste taboue de taille *TAILLELT*, le mouvement inverse $m(a,b')$ où b' désigne le commutateur de la cellule a avant l'application du mouvement $m(a,b)$. On arrête s'il s'est écoulé *kmax* itérations depuis la dernière meilleure solution ou s'il n'y a plus de mouvements disponibles.

A priori, on a plus de chance d'aboutir à une meilleure solution en augmentant le délai *kmax*. Cependant, on constate que, si on prend des valeurs de plus en plus grandes de *kmax*, la recherche s'éloigne des solutions faisables avec peu de chance d'y revenir. On utilise alors un mécanisme de « rappel » pour ramener l'exploration vers les

solutions faisables. Pour y parvenir, après un nombre *DRESPECT* de solutions consécutives non faisables, on pénalise les gains des mouvements menant à des solutions non faisables. On ajoute progressivement une pénalité, composée d'une partie fixe *PRIMECAPAFIXE* et d'une partie variable *PRIMECAPA*, au gain des mouvements qui ajoutent une cellule à un commutateur ayant déjà une capacité résiduelle négative. Le caractère progressif de la pénalité est mis en œuvre en introduisant un multiplicateur *SEVERITE* sur la partie variable. À chaque solution non faisable, le multiplicateur *SEVERITE* est incrémenté jusqu'à un maximum de *MAXSEV*. À la première solution faisable, le dispositif de « rappel » est désactivé. En pénalisant progressivement les mouvements, on évite une transition brutale qui pourrait ignorer de bonnes solutions. Le mouvement *m(a,b)* défini plus haut s'applique donc dans deux contextes : un premier qui ignore totalement les contraintes de capacité et un second qui en tient compte d'une certaine manière.

La liste taboue définie plus haut permet à RT d'éviter les cycles. Toutefois, il est parfois avantageux de revenir à une solution taboue pour continuer la recherche dans une autre direction. Le critère tabou d'un mouvement sera donc annulé si ce dernier conduit à une solution dont l'évaluation est inférieure à celle de la meilleure solution actuellement connue.

En résumé, la mémoire à court terme génère une solution initiale, calcule le tableau des gains qui lui permet de choisir dans le voisinage de la solution courante, celle donnant le meilleur coût. Si la recherche s'éloigne trop des zones de solutions faisables, un dispositif de « rappel » est mis en jeu pour ramener l'exploration vers de

bonnes régions. Enfin, si un mouvement tabou a un gain assez faible, la mémoire à court terme vérifie son niveau d'aspiration. Le mouvement est choisi s'il respecte les critères d'aspiration. Pour avoir de meilleurs résultats, on ajoute à la composante de mémoire à court terme, une composante à moyen terme.

4.3 Mémoire à moyen terme

La composante de mémoire à moyen terme de RT a pour but d'intensifier la recherche localement dans des régions prometteuses. Cela permet de revenir à des solutions qu'on avait peut-être omises. Il s'agit donc d'abord de définir les régions d'intensification, et ensuite de choisir les types de mouvements à appliquer.

4.3.1 Régions d'intensification

Définir des régions prometteuses pour y intensifier la recherche n'est pas chose aisée. Dans notre adaptation de RT, nous avons choisi de conserver dans une liste *FIFO* les *ITAILLEBEST* dernières meilleures solutions. Les solutions sont gardées avec les valeurs associées des gains pour permettre de restaurer plus tard le contexte de la recherche. Chaque fois qu'on trouve une solution s^* qui améliore la meilleure solution trouvée jusqu'à présent, s^* remplace dans la liste *FIFO* la plus ancienne meilleure solution. Cette approche est justifiée par le fait que, dans le problème d'affectation, nous supposons que les bonnes solutions ne sont pas topologiquement très éloignées les unes des autres. Ceci nous amène du coup à définir les mouvements d'intensification.

4.3.2 Mouvements, liste taboue, aspiration et critères d'arrêt

L'idée fondamentale de l'intensification est de varier les mouvements pour aller vers des solutions ignorées par les mouvements de la mémoire à court terme. Dans notre cas, deux mouvements d'intensification ont été définis :

- $i_1(a,c)$: *permutation des cellules a et c selon les plus faibles gains ;*
- $i_2(a,b)$: *déplacement de la cellule a vers le commutateur b en vue de rétablir les contraintes de capacité.*

Cas 1 : *Mouvement $i_1(a,c)$*

Le but de ce mouvement est d'améliorer l'évaluation des solutions en diminuant le coût associé. Toutefois, pour ne pas reprendre le mouvement *m(a,b)* déjà défini dans la composante de mémoire à court terme, nous avons choisi de parcourir tout le tableau des gains et de choisir les deux cellules dont la permutation semble engendrer le plus grand gain. En fait, les deux cellules sont sélectionnées sans tenir compte du fait qu'après le déplacement de la première cellule *a*, le tableau de gain change et la cellule *b* n'est plus forcément la meilleure à déplacer. Cependant, comme ce mouvement s'applique généralement à des solutions faisables, le but est de ne pas trop perturber la topologie de la solution, tout en essayant quand même de l'améliorer. L'estimation du gain ainsi obtenue se calcule rapidement même si elle ne tient pas compte des contraintes de capacité, elle s'avère suffisante pour guider la recherche. En pratique, dans un souci d'efficacité et sans perte de généralité, nous avons implémenté la permutation avec les mécanismes déjà définis pour les mouvements *m(a,b)*. La

permutation est décomposée en deux sous-déplacements consécutifs de cellules. De ce fait, deux mouvements inverses seront enregistrés dans la liste taboue *LT1* de taille *2*ITAILLELT* associée aux mouvements i_1. Une permutation est taboue si l'un au moins de ses sous-déplacements l'est. Le facteur multiplicatif 2 est mis pour pouvoir conserver à chaque fois les deux sous-déplacements tabous. Le critère d'aspiration est défini de la même manière que dans le cas de la composante de mémoire à court terme ; il consiste à accepter une permutation taboue si elle mène à une solution meilleure que celles déjà obtenues.

Le mouvement $i_1(a,c)$ s'applique donc généralement à des solutions déjà faisables. Il ne tient pas compte des contraintes de capacité et s'appuie exclusivement sur une estimation des gains pour le choix de la permutation. Pour des solutions non faisables, le mouvement $i_2(a,b)$ sera plus souvent appliqué.

Cas 2 : *Mouvement* $i_2(a,b)$

Comme nous l'avons déjà dit, le but visé par ce type de mouvement est de restaurer les contraintes de capacité et, de ce fait, diminuer les pénalités et l'évaluation de la solution. Le mouvement $i_2(a,b)$ consiste à :

- déterminer le commutateur c' ayant la capacité résiduelle minimale,
- trouver la cellule a affectée à c' qui génère le volume d'appel minimal,
- l'affecter au commutateur b qui a une capacité résiduelle suffisante et qui permet d'obtenir le gain minimal.

Ce mouvement se fonde donc à la fois sur les contraintes de capacité et les gains. Il est activé dès qu'on a *IDRESPECT* solutions non faisables consécutives et le demeure tant que les solutions trouvées ne sont pas faisables. Il possède une liste taboue séparée *LT2* de même taille que la liste *LT1* et ne met en œuvre aucun critère d'aspiration.

La composante de mémoire à moyen terme, tout comme celle de mémoire à court terme, s'arrête si la meilleure solution n'a pas été améliorée pendant les *ikmax* dernières itérations ou s'il n'y a plus de mouvements disponibles. On peut alors enchaîner avec des mécanismes de diversification.

4.4 Mémoire à long terme

Pour diversifier sa recherche, RT utilise une structure de mémoire à long terme pour amener la recherche dans des régions jusqu'ici peu explorées. Ceci se fait souvent en générant de nouvelles solutions initiales. Dans ce cas-ci, un tableau $n \times m$ (où n est le nombre de cellules et m le nombre de commutateurs) compte, pour chaque arc (a,b), le nombre de fois où ce dernier apparaît dans les solutions visitées. Une nouvelle solution initiale est générée en choisissant pour chaque cellule a, l'arc (a,b) le moins visité. Les solutions visitées lors de la phase d'intensification ne sont pas prises en compte car elles résultent de mouvements différents de ceux appliqués dans les composantes de mémoire à court et long termes. À partir de la nouvelle solution initiale, on démarre une nouvelle recherche en appliquant les mécanismes de mémoire à court et à moyen termes.

4.5 Implémentation

Dans cette section, nous présentons les détails de l'implémentation de RT pour résoudre le problème d'affectation de cellules. Nous précisons d'abord les formats de fichiers utilisés, ensuite nous décrivons les algorithmes des principales parties de l'adaptation proposée, ainsi que les classes utilisées.

4.5.1 Formats de fichiers

Pour résoudre le problème d'affectation, le programme doit d'abord acquérir les spécifications du problème. Celles-ci sont fournies dans deux types de fichiers :

- Le premier appelé « fichier de données » fournit les données concernant le nombre de cellules n et le nombre de commutateurs m sur sa première ligne. Ensuite vient une matrice $n \times m$, où chaque ligne i donne les coûts de liaison de la cellule i aux m commutateurs. La dernière matrice $n \times n$ du fichier présente sur chaque ligne i les coûts de relève de la cellule i par rapport aux n autres cellules.
- Le deuxième appelé « fichier de capacité » donne sur sa première ligne les capacités des n cellules, et celles des m commutateurs sur sa deuxième ligne.

Les résultats obtenus par le programme sont écrits dans un fichier « result.dat » qui contient l'évaluation de la solution, ainsi que le patron d'affectation sous forme d'une matrice $n \times m$.

4.5.2 Implémentation des principaux algorithmes

La méthode de RT comprend globalement trois parties : la mémoire à court terme, la mémoire à moyen terme et celle à long terme. La Figure 4.3 illustre l'organigramme du noyau de la composante de mémoire à court terme. Ce noyau sera réutilisé dans le mécanisme de diversification illustré à la Figure 4.6. La composante de mémoire à court terme est elle-même illustrée à la Figure 4.4. Enfin, la Figure 4.5 présente l'organigramme du mécanisme d'intensification.

4.5.3 Détails d'implémentation

Les expériences ont été réalisées sur une machine « *SUN SPARCStation 5 Model 110* » en environnement UNIX. La Figure 4.7 donne le diagramme UML des principales classes utilisées dans notre adaptation. La plupart des classes possèdent une méthode *init* et une méthode *detruit* qui se charge de la gestion de la mémoire.

La classe données

Elle regroupe les principales données statiques du problème. Elle comprend : le nombre de cellules, *nbcell*, le nombre de commutateurs, *nbcomm*, le tableau des coûts de liaison, *tab_cable*, le tableau des coûts de relève, *tab_releve*, le tableau des volumes d'appel des cellules, *capa_cell*, le tableau des capacités des commutateurs, *capa_comm*.

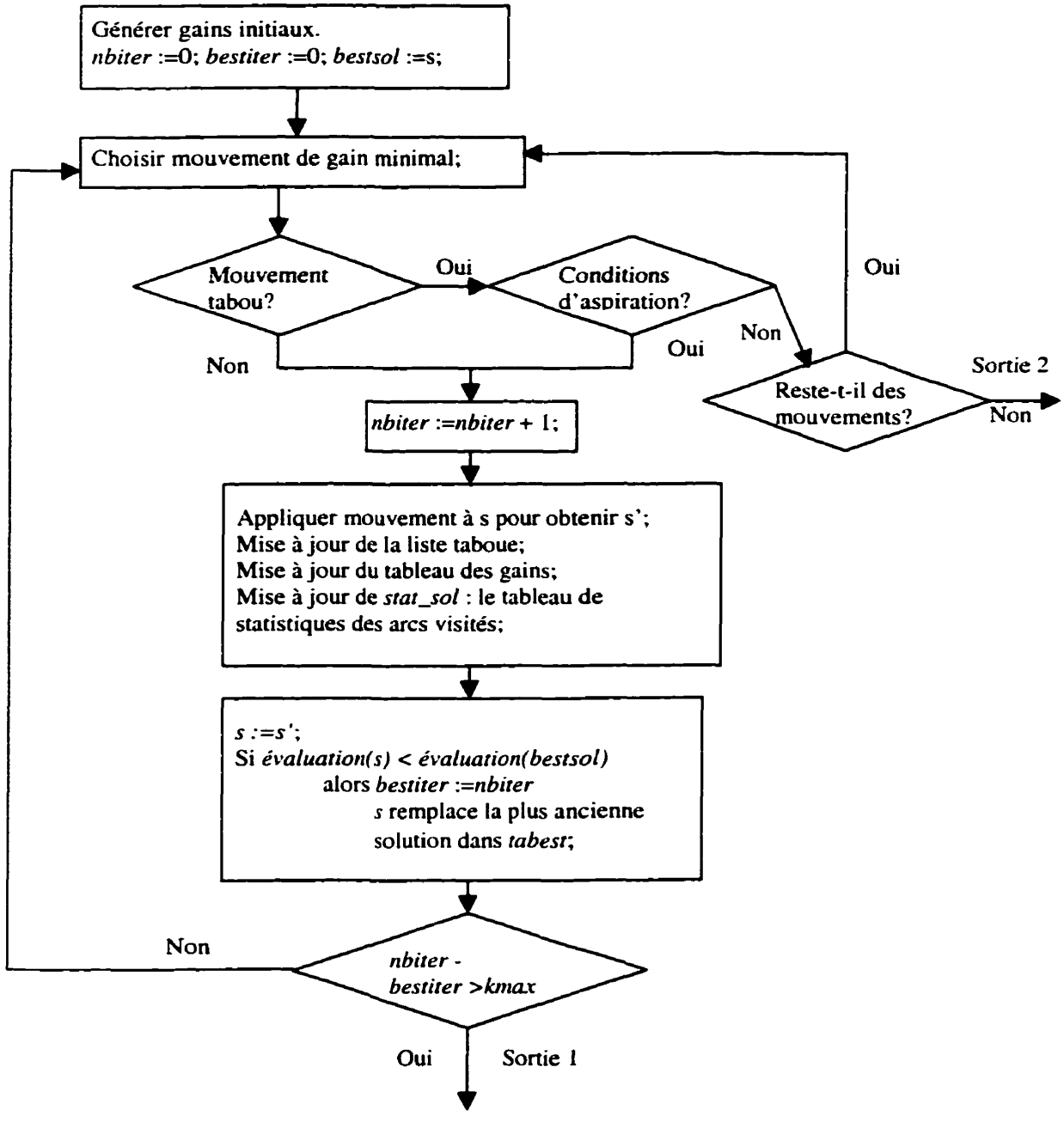

Figure 4.3 Noyau de la composante de mémoire à court terme

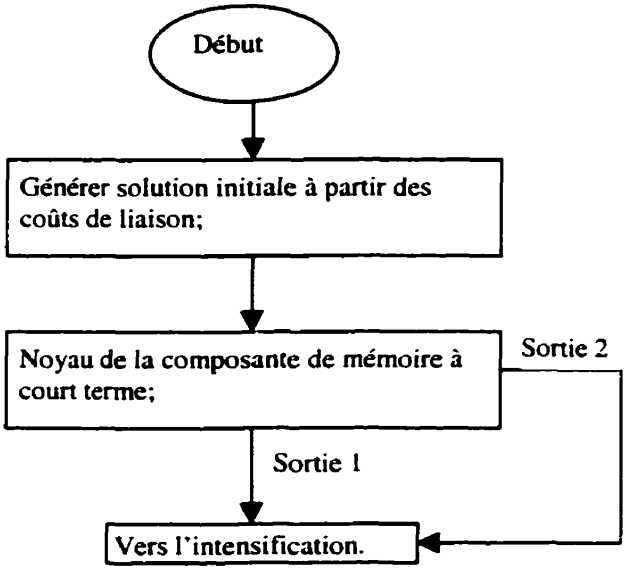

Figure 4.4 Composante de mémoire à court terme

On y trouve aussi un tableau dynamique, *capa_resi* qui donne les capacités résiduelles des commutateurs. Les principales méthodes sont :

- *lire()* qui lit les données du fichier de relève et les transfère dans les structures adéquates de la classe ;
- *lirecapa()* qui lit les données du fichier de capacité et les garde dans les champs adéquats de la classe ;
- *evaluation()* et *neweval()* qui calculent respectivement l'évaluation et la nouvelle évaluation de la solution après modifications.

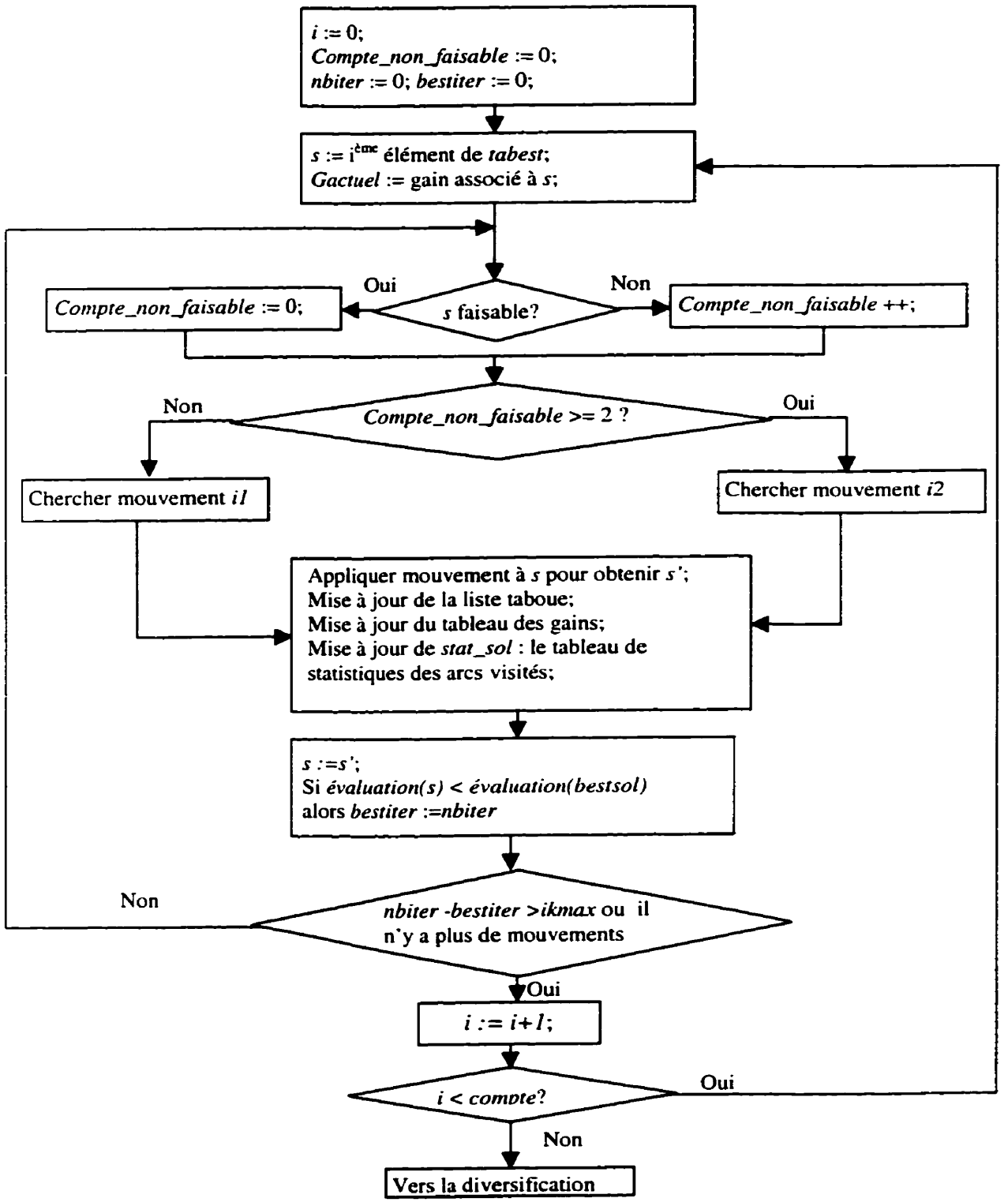

Figure 4.5 Mécanisme d'intensification

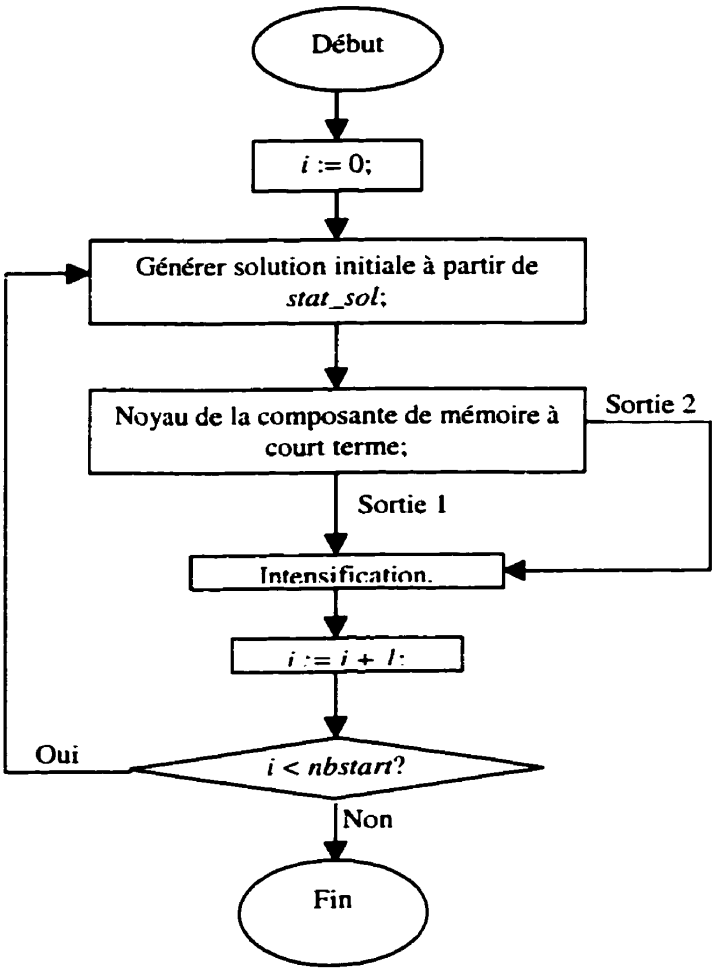

Figure 4.6 Mécanisme de diversification

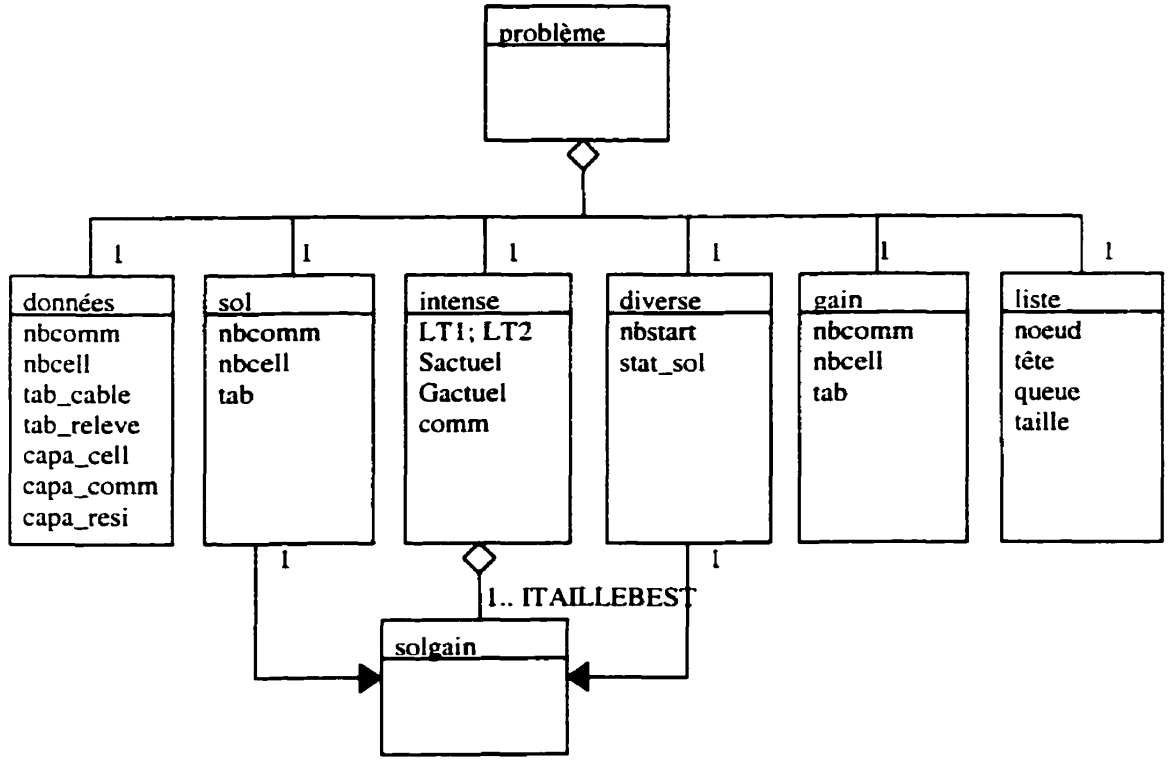

Figure 4.7 Diagramme des principales classes

La classe sol

Elle représente la solution et comprend principalement un tableau $n \times m$ nommé *tab* qui donne le patron d'affectation de la solution. Les principales fonctions sont :

- *initiale()* qui calcule la solution initiale à partir des coûts de liaison ;
- *objectif()* qui calcule la valeur de la fonction objectif pour la solution courante.

La classe intense

Elle représente le mécanisme d'intensification. Elle est composée de :

- deux listes taboues, *LT1* et *LT2*, pour les mouvements de type i_1 et i_2 ;
- deux pointeurs, *Sactuel* et *Gactuel*, qui pointent respectivement sur la solution courante ayant servi de départ à l'intensification et le tableau des gains associés à *Sactuel* ;
- un tableau, *comm*, qui donne pour chaque cellule le numéro du commutateur auquel elle est affectée ;
- un tableau dynamique *tabest* qui garde les *ITAILLEBEST* dernières meilleures solutions avec les gains associés.

Les principales méthodes sont :

- *ajoute()* qui ajoute une meilleure solution à *tabest* ;
- *mvt_inten()* qui cherche les mouvements de type i_1 ;
- *mvt_inten2()* qui cherche les mouvements de type i_2 ;
- *intensifie()* qui applique le processus d'intensification.

La classe diverse

Elle représente le mécanisme de diversification. Elle est composée d'un entier *nbstart* qui indique le nombre de redémarrages, et d'une table $n \times m$ qui indique les statistiques de visite de chaque arc. Elle a une seule fonction principale, *compte()*, qui met à jour, pour chaque solution, les statistiques des arcs visités.

La classe gain

Elle représente le tableau des gains et comprend principalement un tableau $n \times m$ nommé *tab* qui donne les gains potentiels de tous les mouvements *m(a,b)*. Les principales fonctions sont :

- *generer()* qui génère le tableau initial des gains ;
- *maj()* qui effectue la mise à jour des gains après chaque mouvement.

La classe liste

C'est une liste circulaire servant à implémenter les listes taboues. Chaque élément de la liste est un *nœud* qui contient un mouvement et un pointeur sur le nœud suivant. La liste est accessible par un pointeur *tete* qui en indique la tête. Un pointeur *queue* marque la fin de la liste. La variable *taille* donne la taille maximale de la liste. Les principales méthodes sont :

- *insere()* qui ajoute un élément dans la liste ;
- *appartient()* qui vérifie si un élément est dans la liste.

La classe solgain

Elle hérite des classes *sol* et *gain* et sert juste à garder les *ITAILLEBEST* dernières meilleures solutions avec leurs gains, pour leur appliquer le mécanisme d'intensification.

La classe problem

C'est la classe la plus globale. Elle regroupe toutes les autres classes et permet ainsi de définir le problème, les solutions, les gains, etc. Ses principales méthodes sont :

- *choix_mvt()* qui choisit le mouvement *m(a,b)* à appliquer dans la composante de mémoire à court terme ;
- *CMtaboue()* qui applique les composantes de mémoire à court et moyen terme de RT ;
- *rtaboue()* qui applique la méthode de RT avec les différents raffinements possibles.

4.6 Mise en œuvre

Pour améliorer la solution initiale de la Figure 4.2, nous lui avons appliqué notre adaptation de RT. Pour cela, nous avons défini, entre chaque couple *(i,j)* de cellules, un coût horaire de relève h_{ij} comme présenté au Tableau 4.3.

Dans un premier temps, nous avons appliqué seulement la composante de mémoire à court terme de RT. Au bout de 13 itérations, qui correspondent à 13 perturbations de la solution initiale, l'algorithme trouve la solution de la Figure 4.8 qui ne pourra plus être améliorée. Nous avons fixé à 20 le nombre maximal *kmax* d'itérations entre deux améliorations successives de la meilleure solution. Dans ce cas, l'algorithme s'arrête après avoir atteint ce nombre maximal.

Tableau 4.3 Coût de relève entre cellules

	Cellules													
Cellules	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	3	8	3	18	15	7	0	0	0	0	0	0	0
1	3	0	1	0	0	0	4	2	0	0	4	1	0	0
2	8	1	0	6	0	0	0	5	10	1	0	0	0	0
3	3	0	6	0	6	0	0	0	7	0	0	0	0	0
4	18	0	0	6	0	13	0	0	0	0	0	0	0	0
5	15	0	0	0	13	0	6	0	0	0	0	0	0	11
6	7	4	0	0	0	6	0	0	0	0	0	9	2	4
7	0	2	5	0	0	0	0	0	0	11	3	0	0	0
8	0	0	10	7	0	0	0	0	0	4	0	0	0	0
9	0	0	1	0	0	0	0	11	4	0	0	0	0	0
10	0	4	0	0	0	0	0	3	0	0	0	5	0	0
11	0	1	0	0	0	0	9	0	0	0	5	0	7	0
12	0	0	0	0	0	0	2	0	0	0	0	7	0	6
13	0	0	0	0	0	11	4	0	0	0	0	0	6	0

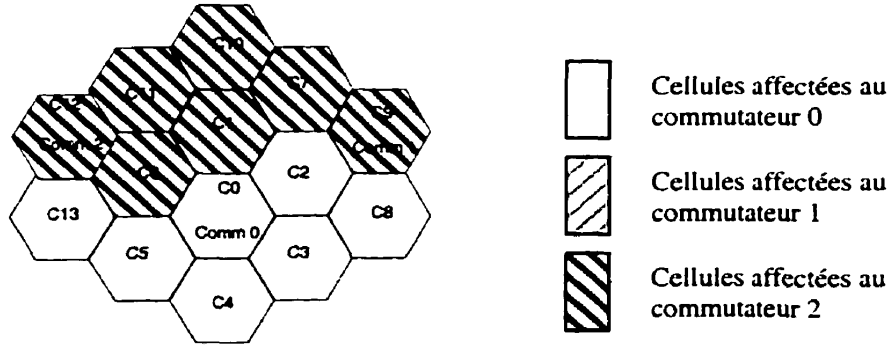

Figure 4.8 Solution finale obtenue par la composante de mémoire à court terme

La solution obtenue respecte les contraintes de capacité. Son coût et son évaluation ont la même valeur de 93.3 unités. On peut noter qu'aucune cellule n'est affectée au commutateur 1.

Une application simultanée des composantes de mémoire à court et moyen termes ne donne aucune amélioration sur la solution finale. Par contre, l'application de

la diversification, avec un nombre de redémarrages *nbstart* égal à 3, nous donne la solution finale de la Figure 4.9. Elle a un coût et une évaluation de 85.1 unités, soit une amélioration de 8.8 %.

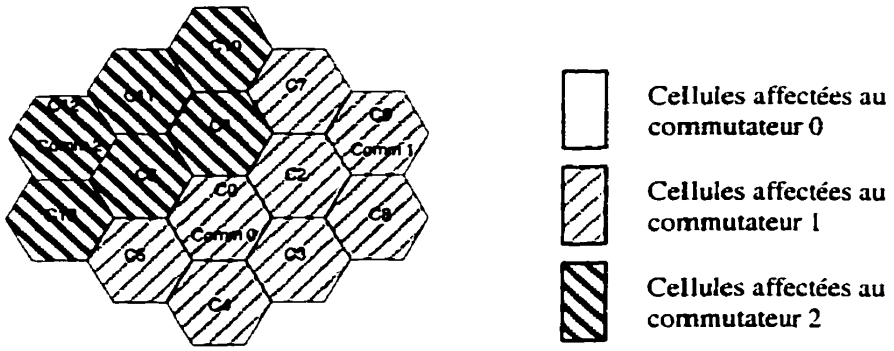

Figure 4.9 Solution finale obtenue avec le mécanisme de diversification

Cette fois-ci, c'est le commutateur 0 qui ne reçoit aucune affectation. Cette solution faisable a été obtenue à l'itération 24 après le premier redémarrage. La solution initiale pour ce premier redémarrage est présentée à la Figure 4.10.

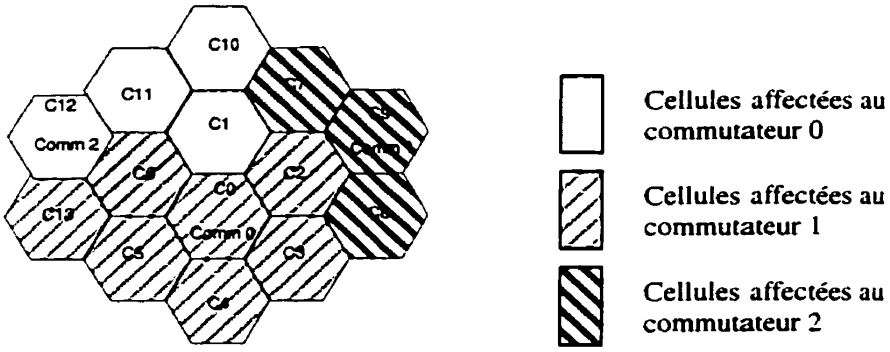

Figure 4.10 Solution initiale pour le premier redémarrage

On remarque que, sur les 14 cellules, seules les cellules 1 et 10 sont affectées au même commutateur que dans la solution initiale de la Figure 4.2.

Enfin, avec une activation simultanée des mécanismes d'intensification et de diversification, on aboutit à la solution de la Figure 4.11 qui a un coût et une évaluation de 77.8 unités, soit une amélioration de 16.6 % par rapport à la solution obtenue seulement avec la composante de mémoire à court terme.

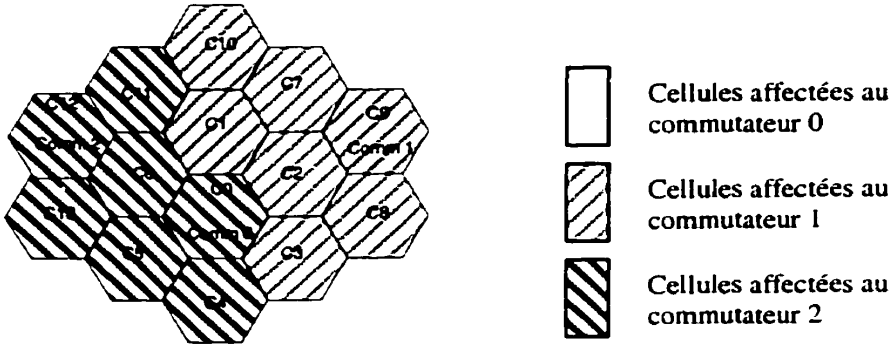

Figure 4.11 Solution finale obtenue avec l'activation de tous les mécanismes

Même si l'intensification seule ne donne pas d'amélioration, elle se révèle assez efficace lorsqu'elle est combinée avec la diversification. En effet, chaque redémarrage de la diversification lui fournit de nouvelles zones prometteuses. La recherche a donc plus de chance de tomber sur une meilleure solution.

Ces résultats préliminaires présagent d'un bon comportement de la méthode de RT. Dans le chapitre suivant, nous soumettrons notre adaptation à une série de tests pour déterminer l'influence de certains paramètres; puis nous comparerons ses performances avec celles d'autres heuristiques appliquées au même problème.

CHAPITRE 5

ANALYSE DES RÉSULTATS

La performance de l'adaptation de la méthode de RT repose sur un bon choix de ses paramètres. Pour évaluer notre implémentation, nous l'avons soumise à une série de tests pour déterminer son efficacité et sa sensibilité par rapport à différents paramètres. Les résultats obtenus sont ensuite comparés à une borne inférieure, estimation de l'optimum global. Enfin, nous avons comparé notre adaptation à d'autres heuristiques appliquées au même problème d'affectation.

5.1 Génération des tests

Pour dégager un comportement global valable, nous avons exécuté notre programme sur un grand nombre de cas tests. Les tests ont été générés par un programme *Matlab®* en supposant que les cellules sont disposées sur une grille hexagonale de longueur et de largeur à peu près égales. Les antennes sont situées au centre des cellules et distribuées uniformément sur la grille. Toutefois, lorsque deux ou plusieurs antennes sont trop proches, la disposition des antennes est rejetée et on la tire de nouveau. Le coût de liaison d'une cellule à un commutateur est proportionnel à la distance les séparant. Nous avons pris un coefficient de proportionnalité égal à l'unité. Le taux d'appel γ_i d'une cellule i suit une loi gamma de moyenne et de variance égales à l'unité. Les temps de service (ou temps de séjour) des appels à l'intérieur des cellules

sont distribués selon une loi exponentielle de paramètre 1. Si une cellule j possède k voisins, l'intervalle [0,1] est divisé en $k+1$ sous-intervalles en choisissant k nombres aléatoires distribués uniformément entre 0 et 1. À la fin de la période de service dans la cellule j, l'appel peut être, soit transféré au $i^{ème}$ voisin ($i=1, ..., k$) avec une probabilité de relève r_{ij} égale à la longueur du $i^{ème}$ intervalle, soit clos avec une probabilité égale à la longueur du $k+1^{ème}$ intervalle. Pour trouver des volumes d'appel et des taux de relève cohérents, les cellules sont considérées comme des files d'attente *M/M/1* formant un réseau de Jackson (Kleinrock, 1975). Les taux d'arrivées α_i dans les cellules à l'équilibre sont obtenus en résolvant le système :

$$\alpha_i - \sum_{j=1}^{n} \alpha_j r_{ji} = \gamma_i \qquad \text{avec} \quad i = 1, ..., n$$

Les α_i sont normalisés en les divisant par le double du plus grand taux d'arrivées α_i, ce qui garantit le respect de la condition de stabilité $\alpha_i \leq \mu_i = 1$ pour $i=1, ..., n$. On choisit comme volume d'appel d'une cellule j, la longueur moyenne de sa file d'attente. Le taux de relève h_{ij} est défini par :

$$h_{ij} = \lambda_i . r_{ij}$$

Tous les m commutateurs ont la même capacité M calculée comme suit :

$$M = \frac{1}{m}(1 + \frac{K}{100}) \sum_{i=1}^{n} \lambda_i$$

où K est choisi uniformément entre 10 et 50, ce qui assure un surplus global de 10 à 50% de la capacité des commutateurs par rapport au volume d'appel des cellules.

5.2 Plan d'expériences

La méthode de RT est constituée essentiellement de trois composantes : les mémoires à court, moyen et long termes. Notre plan d'expériences vise à tester les paramètres mis en jeu par chaque composante. Compte tenu du grand nombre de combinaisons possibles, nous faisons varier un seul paramètre à la fois. Pour chaque paramètre, nous choisissons trois valeurs discrètes possibles. L'algorithme est exécuté pour chacune des valeurs du paramètre sur un ensemble de 300 cas tests avec un nombre de cellules variant de 15 à 200 et un nombre de commutateurs allant de 2 à 7. Le Tableau 5.1 donne les détails des cas utilisés.

Tableau 5.1 Cas tests utilisés pour l'exécution du plan d'expériences

Nombre de cellules	Nombre de commutateurs	Nombre de cas tests
15	2	50
30	3	50
50	4	50
100	5	50
150	6	50
200	7	50

5.2.1 Mémoire à court terme

Dans notre adaptation, la mémoire à court terme met en jeu essentiellement deux processus : une liste taboue pour diminuer les risques de cycle et un mécanisme de « rappel » qui ramène l'exploration vers les solutions faisables. Les tests sont effectués en désactivant toutes les autres composantes de mémoire à moyen et long termes.

Effet de la taille de la LT

Selon Glover (1989), la taille de la LT devrait se situer autour de 7, indépendamment de la taille et de la structure du problème. Nous avons cependant voulu étudier l'impact de différents choix de taille pour la LT sur la qualité des solutions obtenues. À cette fin, nous avons exécuté notre programme avec des listes taboues de taille respective 5, 7 et 9. Ensuite, nous avons fait une moyenne pour tous les cas-tests ayant un même nombre de cellules et de commutateurs. La Figure 5.1 représente la moyenne de l'évaluation des solutions obtenues en fonction du nombre de cellules et de commutateurs, et cela pour chacune des valeurs choisies pour la taille de la LT.

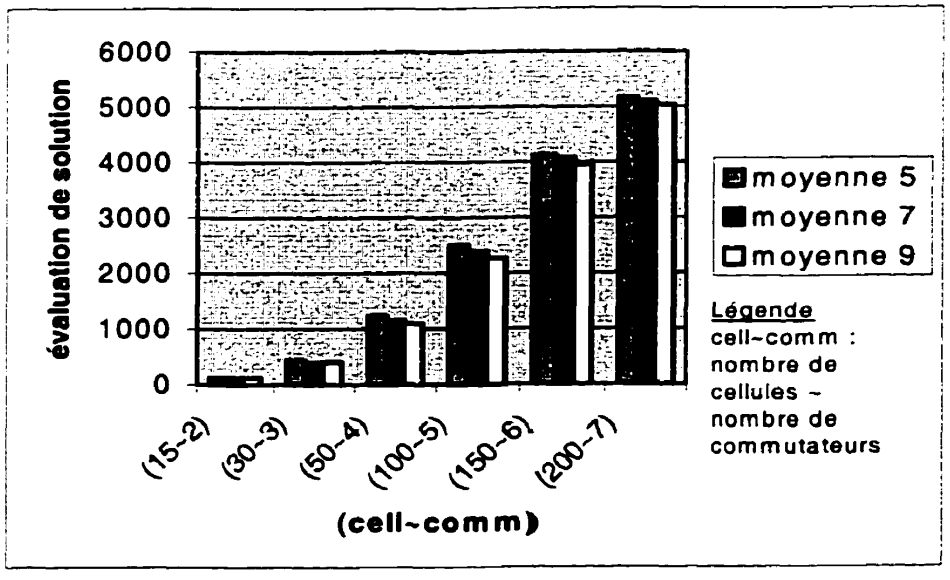

Figure 5.1 Effet de la taille de la LT sur les solutions obtenues

On note qu'une liste taboue de taille 9 donne en moyenne de meilleurs résultats, c'est-à-dire des solutions de coûts moindres. En effet, le mécanisme de « rappel » a tendance à

ramener l'exploration vers des zones de solutions faisables. Par suite, si la liste taboue est courte, la recherche a plus de chance de quitter les zones de solutions non faisables en retournant à une solution déjà visitée. De ce fait, on explore beaucoup moins de solutions et on a moins de chance d'aboutir à de bonnes.

Effet du délai de déclenchement du mécanisme de « rappel »

Lors de l'exploration de la zone de recherche, la composante de mémoire à court terme utilise un mécanisme de « rappel » pour ramener la recherche vers des régions de solutions faisables. Le mécanisme de « rappel » est activé après un certain délai mesuré en nombre de solutions consécutives non faisables. Ce délai ne doit pas être a priori, ni trop grand sinon la recherche s'éloigne beaucoup trop des zones prometteuses, ni trop petit, sinon la recherche est constamment ramenée vers des zones de solutions faisables et peut ainsi omettre de bonnes solutions. Nous avons étudié les cas où le dispositif de « rappel » se déclenche après respectivement 2, 5 et 8 solutions consécutives non faisables. La Figure 5.2 donne la moyenne de l'évaluation des solutions obtenues en fonction du nombre de cellules et de commutateurs, et cela pour chacune des valeurs choisies pour le délai de déclenchement du dispositif de « rappel ».

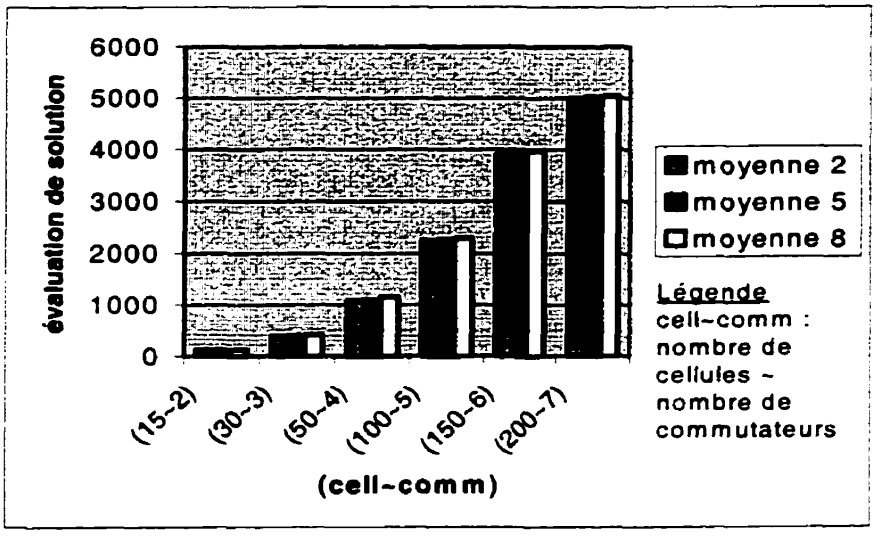

Figure 5.2 Effet du délai de déclenchement du dispositif de rappel sur les solutions obtenues

Un délai de déclenchement de deux solutions consécutives non faisables donne en moyenne des solutions meilleures. En effet, comme nous l'avions déjà dit, le mécanisme de « rappel » est appliqué progressivement. Il doit donc se déclencher le plus tôt possible pour avoir le temps d'agir (c'est-à-dire, attirer la recherche vers des régions de solutions faisables) sans cependant avoir à le faire soudainement.

Effet de la variation de l'intensité maximale du mécanisme de retour

L'intensité du mécanisme de « rappel » est modulée progressivement à l'aide d'un multiplicateur *SEVERITE* qui est incrémenté à chaque solution non faisable jusqu'à une valeur maximale *MAXSEV*. La Figure 5.3 représente la moyenne de l'évaluation des solutions obtenues en fonction du nombre de cellules et de commutateurs, et cela pour des valeurs de *MAXSEV* de 10, 15 et 18 respectivement.

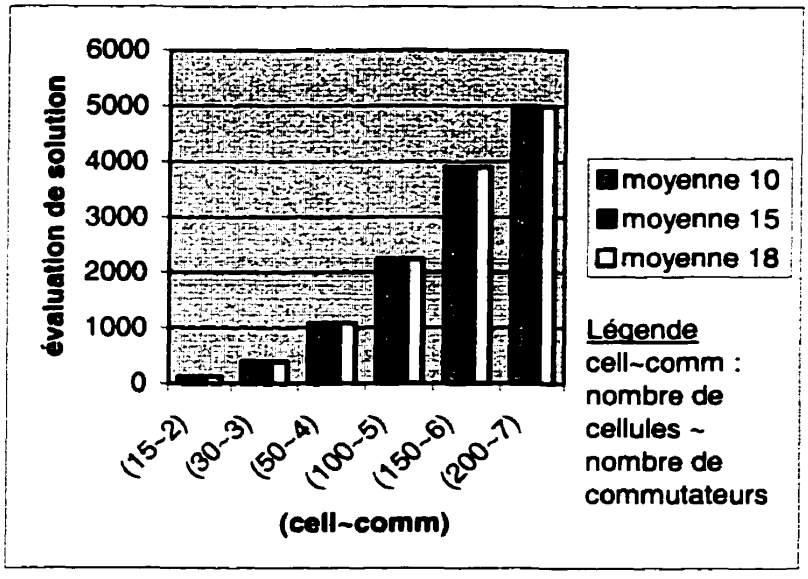

Figure 5.3 Effet de la variation de l'intensité maximale du dispositif de rappel sur les solutions obtenues

On ne note pas de différence sensible entre les résultats pour les trois valeurs de *MAXSEV* choisie. Cela signifierait donc qu'en général, le mécanisme de « rappel » n'est pas appliqué jusqu'à la plus petite des valeurs choisies pour l'intensité maximale, car on ne trouve pas plus de 10 solutions consécutives non faisables lors des exécutions de l'algorithme. Nous avons donc pris une valeur de 15 pour notre adaptation, car elle constitue la moyenne parmi les trois valeurs utilisées pour notre test et le temps d'exécution ne s'en trouve pas particulièrement affecté. De plus, on pense intuitivement assurer ainsi l'efficacité de notre adaptation dans des cas où on aurait plus de 10 solutions consécutives non faisables.

5.2.2 Mémoire à moyen terme

La composante de mémoire à moyen terme raffine l'exploration dans des régions prometteuses. Dans notre adaptation, elle retient les *ITAILLEBEST* dernières meilleures solutions et intensifie la recherche dans leurs voisinages. Pour cela, elle utilise deux types de mouvement : le premier permute deux cellules, le second consiste à réaffecter des cellules pour rétablir les contraintes de capacité. Le premier type de mouvement est celui appliqué par défaut ; le second mouvement, quant à lui, est mis en œuvre après un nombre *IDRESPECT* de solutions consécutives non faisables. Dans cette section, nous étudions l'influence du délai de déclenchement des mouvements du second type en modifiant la variable *IDRESPECT* et l'effet de la variable *ITAILLEBEST* qui est la taille de la région d'intensification. Comme le mécanisme de diversification fait appel à l'intensification à chaque redémarrage, les tests sont effectués en activant tous les mécanismes de mémoire à court, moyen et long termes.

Effet du délai de déclenchement des mouvements du second type

Tout comme dans le cas de la composante de mémoire à court terme, le mécanisme d'intensification utilise des mouvements de type différent pour rétablir les contraintes de faisabilité. Le délai *IDRESPECT* de mise en œuvre de ce type de mouvement ne doit pas être a priori, ni trop grand sinon la recherche sort des zones prometteuses, ni trop petit sinon la recherche est trop contrainte et peut manquer de bonnes solutions. Nous avons choisi des valeurs de *IDRESPECT* de 2, 5 et 8 respectivement.

La Figure 5.4 représente la moyenne de l'évaluation des solutions obtenues en fonction du nombre de cellules et de commutateurs, et cela pour chacune des valeurs de *IDRESPECT* choisies. De meilleures solutions sont obtenues avec un délai de déclenchement faible, soit une valeur de 2 pour la variable *IDRESPECT*. Comme dans le cas de la mémoire à court terme, un déclenchement rapide du mécanisme de choix des mouvements de type 2 permet à l'algorithme de commencer très tôt la recherche des solutions faisables. Pendant la phase d'intensification, le gain fait en ramenant l'exploration dans des zones faisables est donc suffisant pour justifier qu'on applique très tôt les mouvements de type 2 qui sont beaucoup plus contraignants que les permutations.

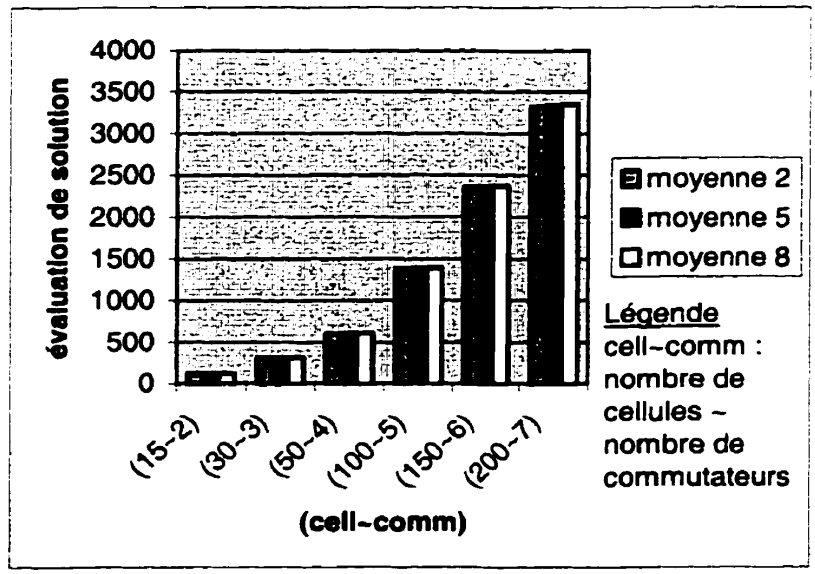

Figure 5.4 Effet du délai de déclenchement des mouvements de type 2 sur les solutions obtenues

Effet de la taille de la région d'intensification

Dans notre adaptation, la taille de la région d'intensification est déterminée par le nombre *ITAILLEBEST* de dernières meilleures solutions retenues. En effet, la recherche est intensifiée dans les voisinages de ces solutions. Nous avons étudié l'impact de la taille de la région d'intensification sur les solutions obtenues. Les valeurs de *ITAILLEBEST* choisies sont de 3, 5 et 7. Il est évident qu'une région d'intensification plus grande donne de meilleures solutions, mais au prix d'un temps de calcul beaucoup plus long. Pour pouvoir comparer les résultats obtenus pour différentes valeurs de *ITAILLEBEST*, nous avons pris une valeur de référence 3. Ensuite, nous avons calculé les améliorations des solutions ainsi que l'augmentation du temps d'exécution par rapport aux valeurs de référence. La Figure 5.5 donne le rapport des améliorations aux augmentations de temps d'exécution en fonction du nombre de cellules et de commutateurs, et cela pour les différentes valeurs de *ITAILLEBEST*. Notons qu'un rapport élevé suppose une bonne amélioration en un temps relativement court, donc une bonne valeur de la variable *ITAILLEBEST*.

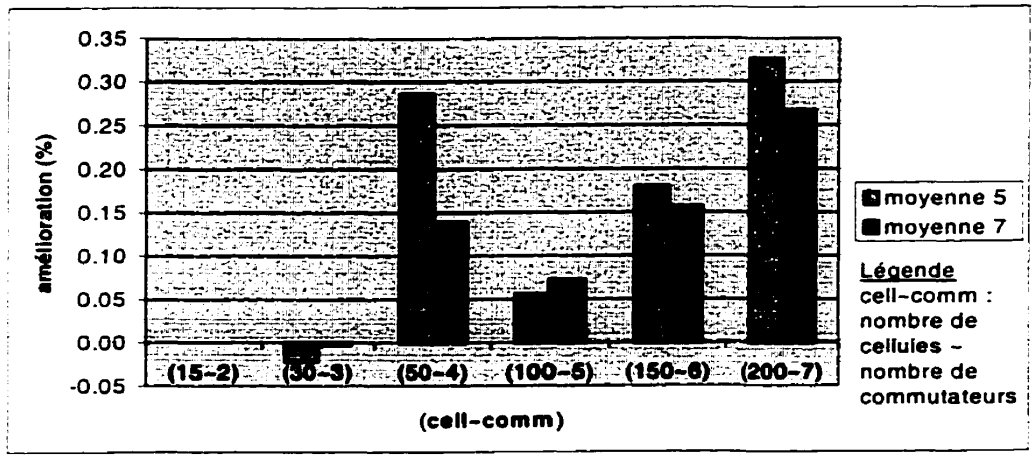

Figure 5.5 Effet de la taille de la région d'intensification sur les solutions obtenues

On constate que pour des problèmes de petite taille (jusqu'à 30 cellules et 3 commutateurs), une zone d'intensification de taille 3 convient parfaitement. Par contre, pour des problèmes de taille plus élevée, une valeur de 5 de la variable *ITAILLEBEST* donne de meilleurs résultats. En effet, pour un problème de petite taille, on imagine que l'intensification améliore assez vite la solution de la composante de mémoire à court terme, puisque les possibilités sont moindres. Pour un problème de taille plus élevée, on a un plus grand nombre de possibilités et les améliorations apportées par l'intensification se justifient bien par rapport au temps investi, jusqu'à une taille de la zone d'intensification de l'ordre de 5 solutions.

5.2.3 Mémoire à long terme

Pour la diversification, le paramètre le plus important est le nombre *nbstart* de redémarrages. Les tests sont effectués sans activation du mécanisme d'intensification.

Effet du nombre de redémarrages

Nous avons étudié l'influence du nombre *nbstart* de redémarrages sur la qualité des solutions obtenues. Les valeurs choisies pour la variable *nbstart* sont de 1, 3, et 5 respectivement. Le cas d'un seul redémarrage sert de référence. Comme dans le cas de la taille de la zone d'intensification, l'influence du temps d'exécution est prise en compte en considérant le rapport des améliorations des solutions aux augmentations de temps d'exécution. La Figure 5.6 illustre les résultats obtenus. Même si l'écart entre les différentes valeurs choisies pour *nbstart* n'est pas très grand, on note tout de même qu'avec trois redémarrages, on obtient en général des solutions meilleures.

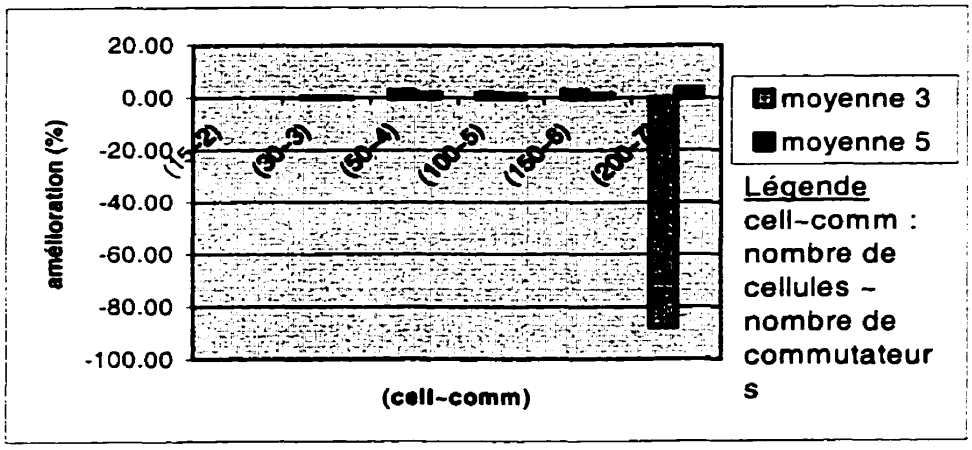

Figure 5.6 Effet du nombre de redémarrage sur les solutions obtenues

5.3 Comportement général de la méthode

Pour étudier le comportement général et l'efficacité des mécanismes mis en œuvre dans notre adaptation de RT, nous avons généré six séries de tests en faisant varier séparément, puis simultanément le nombre de cellules et le nombre de

commutateurs. Le Tableau 5.2 donne le détail des séries de tests exécutés. La composition de chaque série est présentée à l'Annexe A.

Tableau 5.2 Détails des séries de tests exécutés

N° de la série de tests	Nombre de cellules	Nombre de commutateurs	Nombre total de cas de tests
1	Variable (15-200)	2	700
2	Variable (15-200)	3	700
3	Variable (15-200)	4	700
4	50	Variable (2-7)	300
5	150	Variable (2-7)	300
6	Variable (15-200)	Variable (2-7)	300

Pour chacune des séries, nous avons exécuté notre programme respectivement avec la composante de mémoire à court terme, puis avec les combinaisons de mémoire à court et moyen termes, et celles de mémoire à court et long termes, et enfin avec toutes les composantes de RT. Les figures 5.7 à 5.12 et 5.13 à 5.18 décrivent d'une part l'amélioration apportée par l'intensification et la diversification par rapport à la composante de mémoire à court terme uniquement, et d'autre part le pourcentage de solutions faisables obtenues pour chaque mécanisme.

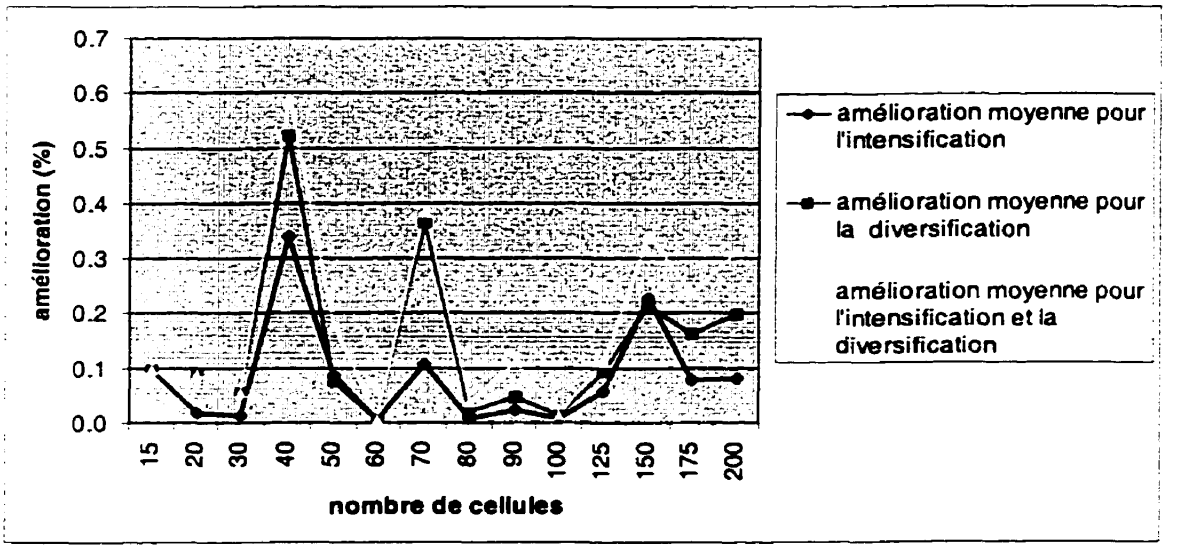

Figure 5.7 Contribution en pourcentage de chaque mécanisme à l'amélioration des solutions obtenues (série n°1)

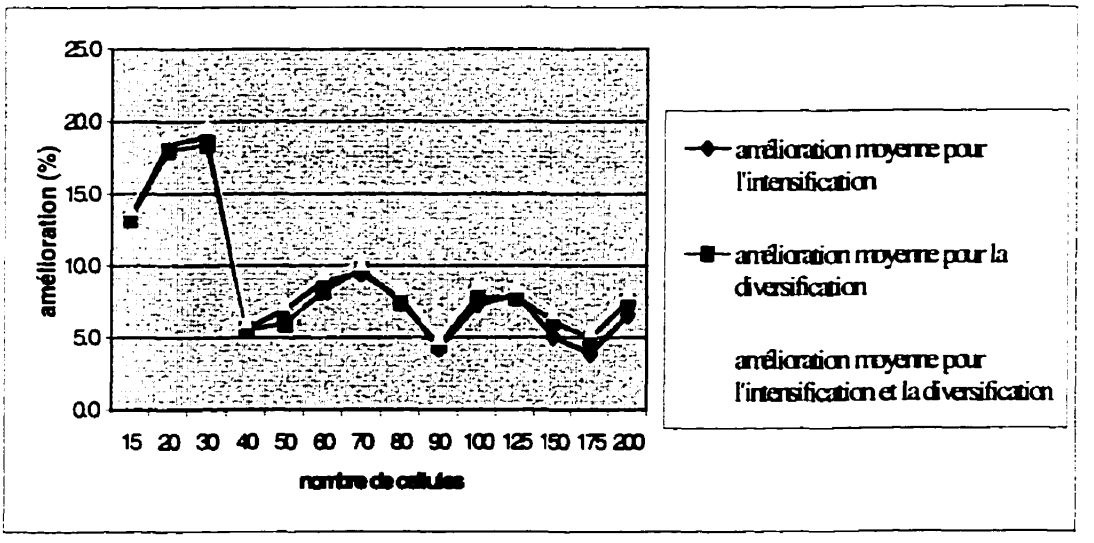

Figure 5.8 Contribution en pourcentage de chaque mécanisme à l'amélioration des solutions obtenues (série n°2)

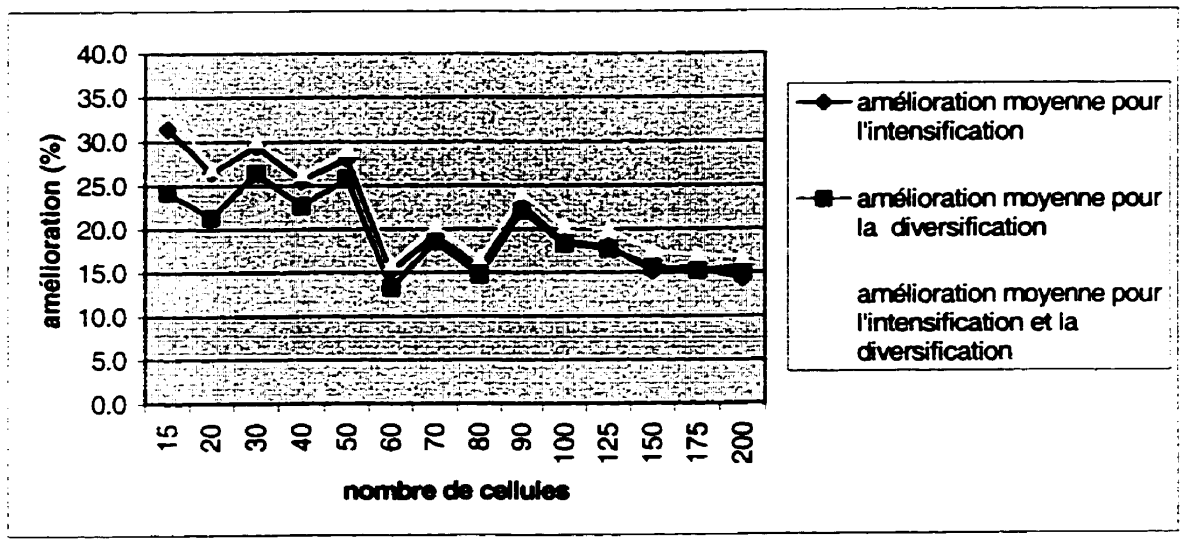

Figure 5.9 Contribution en pourcentage de chaque mécanisme à l'amélioration des solutions obtenues (série n°3)

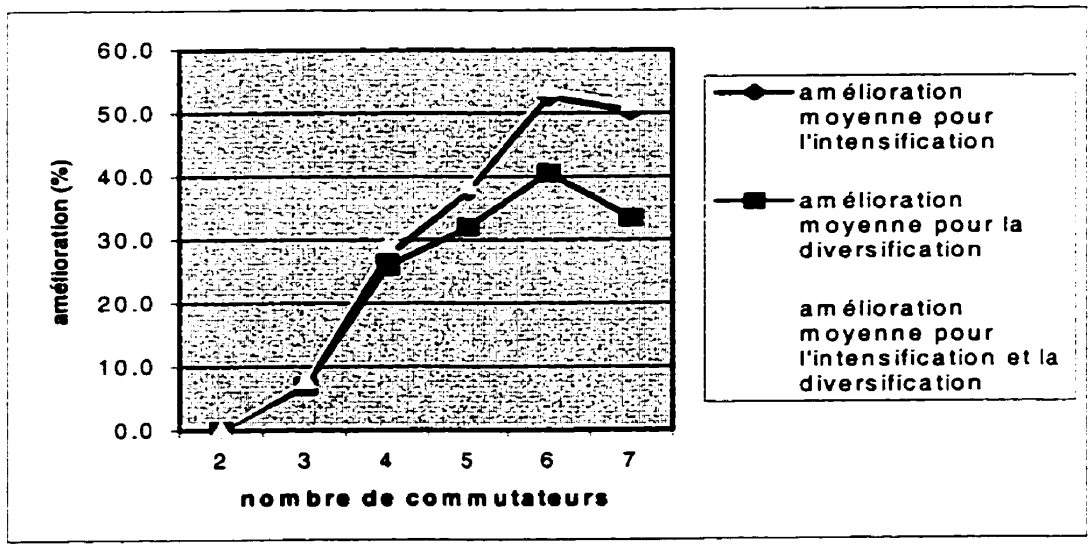

Figure 5.10 Contribution en pourcentage de chaque mécanisme à l'amélioration des solutions obtenues (série n°4)

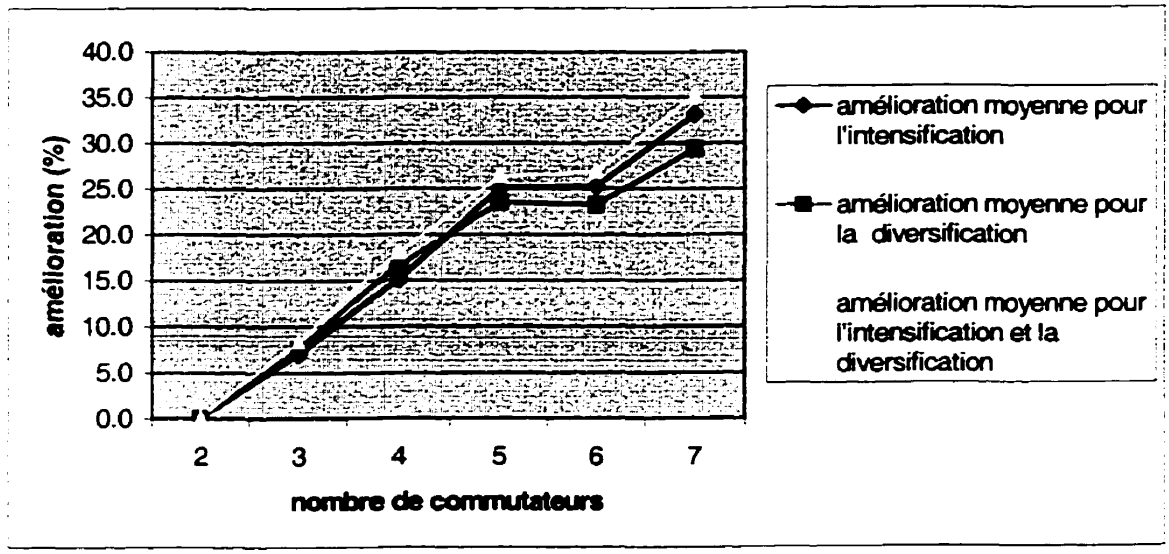

Figure 5.11 Contribution en pourcentage de chaque mécanisme à l'amélioration des solutions obtenues (série n°5)

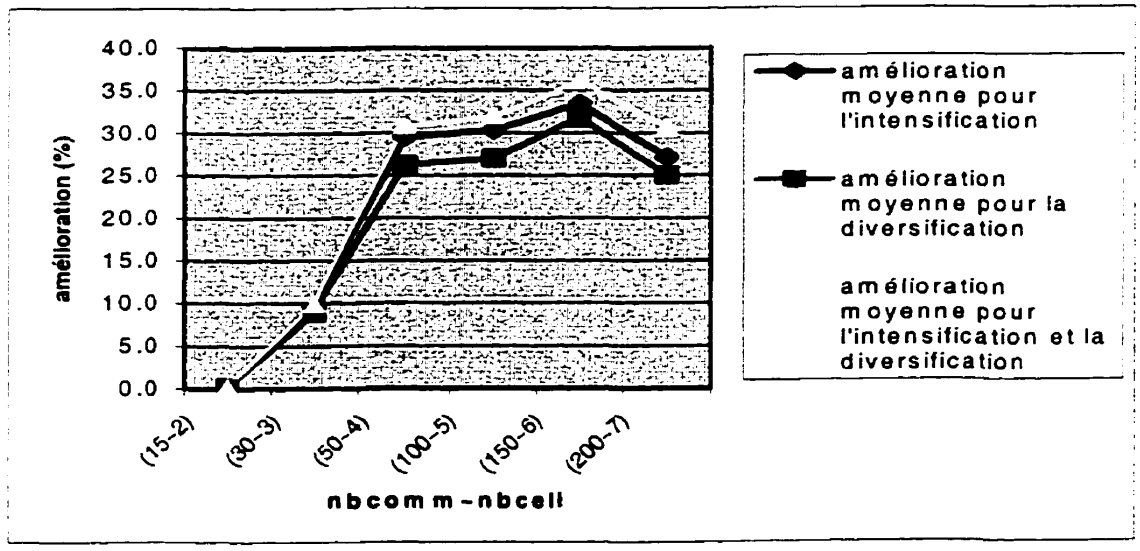

Figure 5.12 Contribution en pourcentage de chaque mécanisme à l'amélioration des solutions obtenues (série n°6)

On constate que la contribution des mécanismes d'intensification et de diversification à l'amélioration des solutions croît avec la taille des données du problème. En particulier, l'intensification et/ou la diversification n'améliorent les

solutions obtenues de façon sensible que pour un nombre de commutateurs supérieur à deux. Ceci s'explique par le fait que, pour deux commutateurs, la composante de mémoire à court terme, lors des mouvements de réaffectation, examine la plupart des possibilités par rapport aux deux commutateurs. L'intensification et la diversification amènent donc rarement la recherche vers des zones non encore explorées. Par contre, pour plus de deux cellules, on a plus de choix qu'une simple réaffectation. L'intensification permet alors de raffiner les choix faits par la composante de mémoire à court terme. De même, la diversification amène la recherche dans des zones comportant des alternatives non encore visitées. Enfin, les mécanismes d'intensification et de diversification sont plus efficaces pour un plus grand nombre de commutateurs, ou à nombre égal de commutateurs, pour un petit nombre de cellules.

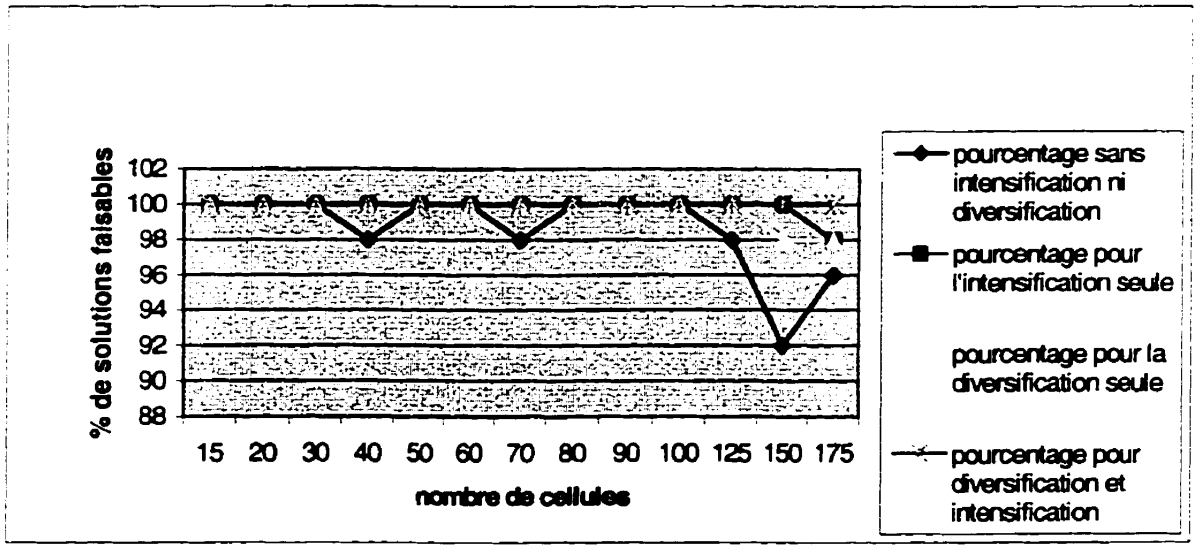

Figure 5.13 Pourcentage de solutions faisables pour chaque mécanisme (série n°1)

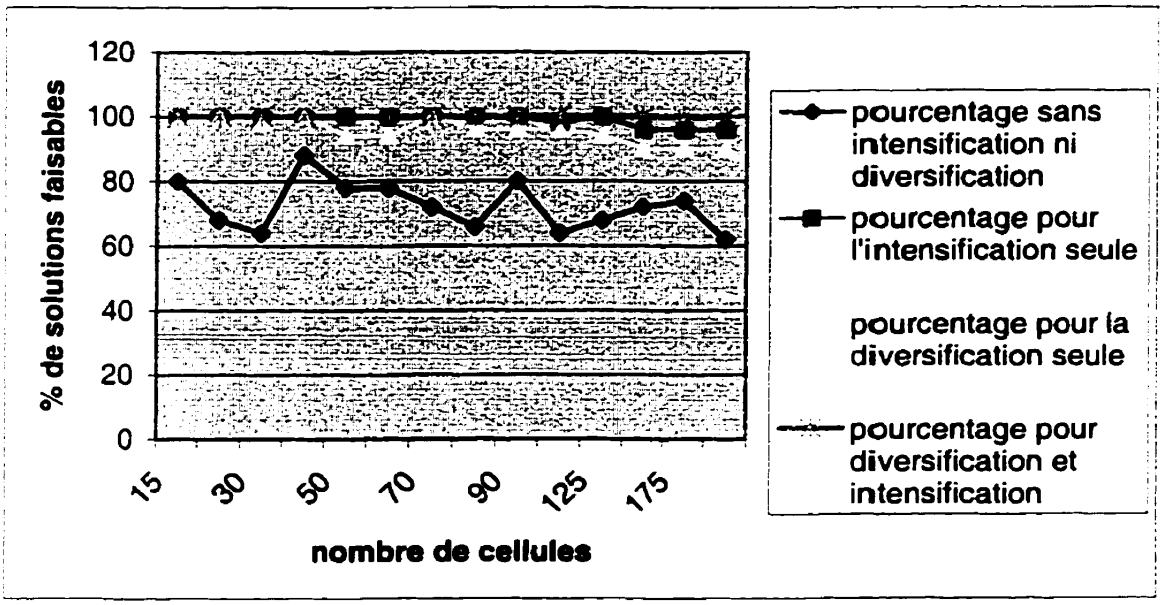

Figure 5.14 Pourcentage de solutions faisables pour chaque mécanisme (série n°2)

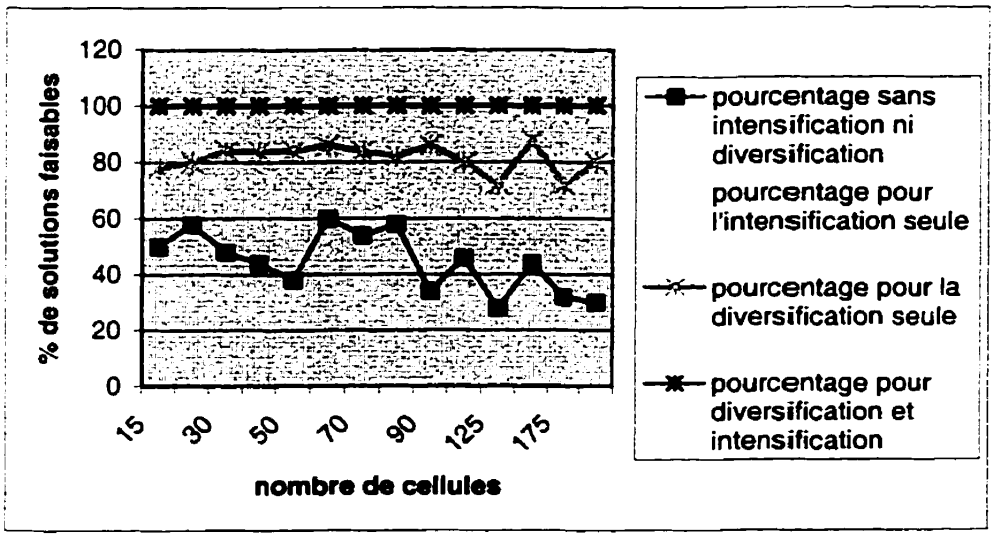

Figure 5.15 Pourcentage de solutions faisables pour chaque mécanisme (série n°3)

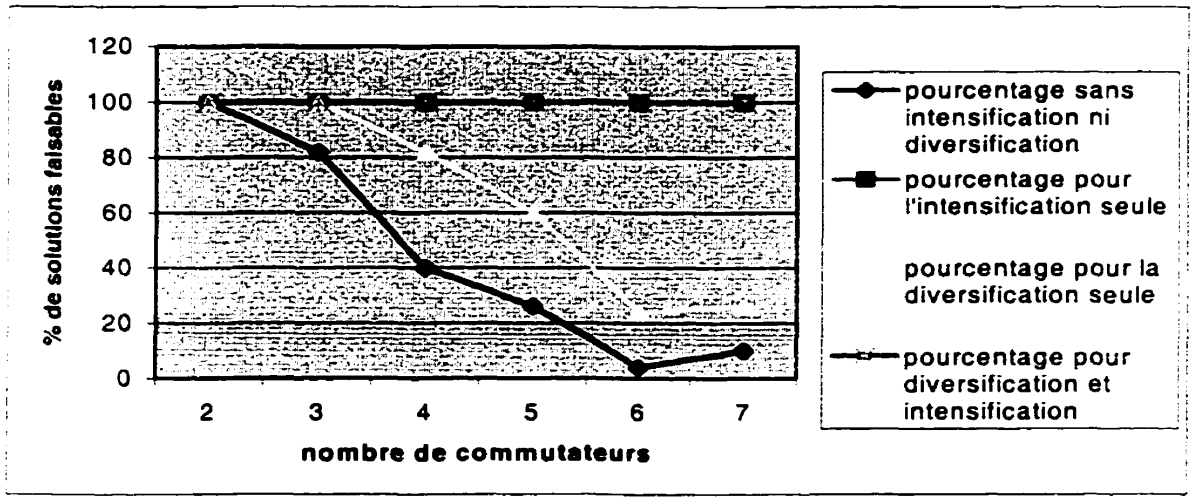

Figure 5.16 Pourcentage de solutions faisables pour chaque mécanisme (série n°4)

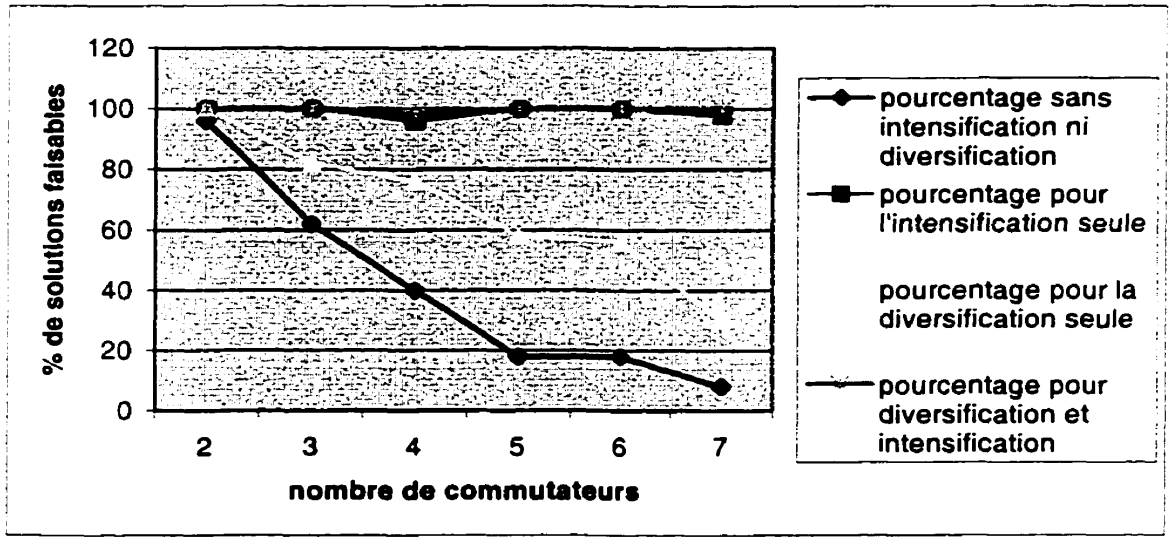

Figure 5.17 Pourcentage de solutions faisables pour chaque mécanisme (série n°5)

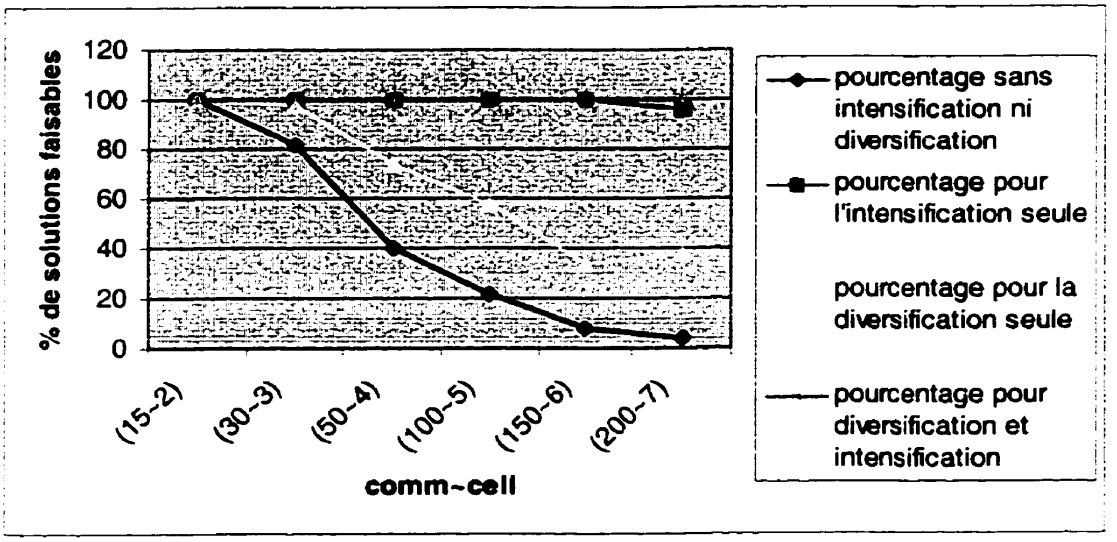

Figure 5.18 Pourcentage de solutions faisables pour chaque mécanisme (série n°6)

Le pourcentage de solutions faisables obtenues par la composante de mémoire à court terme décroît en règle générale avec le nombre de commutateurs. En particulier, pour un nombre élevé de commutateurs, la composante de mémoire à court terme génère difficilement une solution faisable. Pour un nombre de commutateurs fixe et un nombre variable de cellules, le pourcentage de solutions faisables reste à peu près du même ordre, quels que soient les mécanismes mis en jeu. Avec l'activation de l'intensification et/ou de la diversification, on obtient plus de solutions faisables. Toutefois, la composante de mémoire à moyen terme se révèle plus efficace que celle de mémoire à long terme pour générer des solutions faisables. Ceci s'explique par le fait que l'intensification applique des mouvements ayant pour but explicite le rétablissement de la faisabilité, alors que la diversification essaie d'améliorer globalement la solution sans chercher spécifiquement à rétablir la faisabilité. De toute manière, une action combinée

de l'intensification et de la diversification engendre presque toujours des solutions faisables.

Globalement, notre adaptation de la méthode de RT donne d'assez bons résultats. De plus, une exécution du programme requiert un temps raisonnable qui ne croît pas trop vite avec la taille du problème. L'Annexe B donne à titre illustratif les temps d'exécution sur une « *SUN SPARCStation 5 Model 110* ».

5.4 Comparaison avec une estimation de l'optimum global

La solution obtenue par la méta-heuristique de RT est souvent un optimum local. À défaut de connaître l'optimum global, nous définissons une borne inférieure pour évaluer la qualité de nos solutions.

5.4.1 Définition d'une borne inférieure

Soient un ensemble de n cellules à affecter à m commutateurs et x_{ik} une variable de décision telle que :

$$x_{ik} = \begin{cases} 1 \text{ si la cellule } i \text{ est reliée au commutateur } k, \\ 0 \text{ sinon.} \end{cases}$$

Soient λ_i le volume d'appels horaires destiné à la cellule i, M_k la capacité horaire du commutateur k, c_{ik} le coût de liaison de la cellule i au commutateur k, et h_{ij} le coût de relève horaire de la cellule i à la cellule j. Le problème d'affectation consiste essentiellement à :

Minimiser

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} h_{ij}(1-\sum_{k=1}^{m} x_{ik}x_{jk})$$

sujet aux contraintes suivantes déjà formulées au chapitre 2 :

$$x_{ik} = 0 \text{ ou } 1 \text{ pour } i = 1, \ldots, n \text{ et } k = 1, \ldots, m \qquad (2.4)$$

$$\sum_{k=1}^{m} x_{ik} = 1 \quad \text{pour} \quad i = 1, \ldots, n \qquad (2.1)$$

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leq M_k \quad \text{pour} \quad k = 1, \ldots, m \qquad (2.3)$$

En associant les multiplicateurs μ_k et β_i aux contraintes (2.1) et (2.3), le lagrangien s'écrit :

$$L(x,\mu,\beta) = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} h_{ij}(1-\sum_{k=1}^{m} x_{ik}x_{jk}) + \sum_{k=1}^{m}\mu_k\left[(\sum_{i=1}^{n}\lambda_i x_{ik}) - M_k\right] + \sum_{i=1}^{n}\beta_i\left[(\sum_{k=1}^{m} x_{ik}) - 1\right]$$

En utilisant la contrainte (2.1), on a :

$$L(x,\mu,\beta) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m}\left[\frac{c_{ik}}{n} + h_{ij}(1-x_{jk})\right]x_{ik} + \sum_{i=1}^{n}\sum_{k=1}^{m}(\mu_k\lambda_i + \beta_i)x_{ik} - \sum_{k=1}^{m}\mu_k M_k - \sum_{i=1}^{n}\beta_i$$

avec $\mu_k \geq 0$, $k=1, \ldots, m$ et $\beta_i \in \boldsymbol{R}$, $i=1, \ldots, n$.

Le dual du problème ci-dessus s'exprime comme suit :

$$(D) \qquad \sup_{\beta \text{ et } \mu} (\inf_{x} L(x,\mu,\beta))$$

où β est un réel quelconque, μ un réel positif ou nul et x un nombre binaire.

Reécrivons le lagrangien sous la forme suivante:

$$L(x,\mu,\beta) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m}\left[\frac{c_{ik}}{n} + h_{ij}(1-x_{jk}) + \frac{\mu_k\lambda_i + \beta_i}{n}\right]x_{ik} - \sum_{k=1}^{m}\mu_k M_k - \sum_{i=1}^{n}\beta_i$$

Le sous-problème $\inf_x L(x,\mu,\beta)$ a comme solutions :

$$(i) \qquad x_{ik} = 0, \text{ si } Q_{ik} = c_{ik} + \beta_i + \mu_k\lambda_i > 0 \tag{5.1}$$

$$(ii) \qquad x_{ik} = 1, \text{ si } Q_{ik} + \sum_{j=1, j\neq i}^{n} h_{ij}(1-x_{jk}) < 0 \tag{5.2}$$

$$\text{(iii)} \qquad x_{ik} = 0, \text{ si } Q_{ik} < 0 \text{ et } Q_{ik} + \sum_{j=1, j\neq i}^{n} h_{ij}(1-x_{jk}) > 0 \tag{5.3}$$

Prenons le cas (5.1), le plus simple pour tous les i et k. On a donc $Q_{ik} > 0$, $\forall$ i et k. Avec la solution $x_{ik} = 0$ $\forall$ i et k, le dual (D) devient donc :

$$\text{(D)} \quad \max_{\beta\in\mathbf{R},\,\mu\geq 0}\left(-\sum_{i=1}^{n}\beta_i - \sum_{k=1}^{m}\mu_k M_k\right)$$

sujet à :

$$c_{ik} + \beta_i + \mu_k\lambda_i > 0, \quad i=1, \ldots, n \text{ et } k=1, \ldots, m \tag{5.4}$$

Il s'ensuit que:

$$\beta_i > -\mu_k\lambda_i - c_{ik}, \quad i=1, \ldots, n \text{ et } k=1, \ldots, m$$

$$\beta_i \geq \max_k(-\mu_k\lambda_i - c_{ik}), \quad i=1, \ldots, n$$

et par suite :

$$-\sum_{i=1}^{n}\beta_i - \sum_{k=1}^{m}\mu_k M_k \leq \sum_{i=1}^{n}\left(\min_k(\mu_k\lambda_i + c_{ik})\right) - \sum_{k=1}^{m}\mu_k M_k$$

Le problème dual se ramène à :

$$\text{(D)} \quad \max_{\mu\geq 0}\left(\sum_{i=1}^{n}\left[\min_k(\mu_k\lambda_i + c_{ik})\right] - \sum_{k=1}^{m}\mu_k M_k\right)$$

Ce qui équivaut à :

$$\max_{\mu \geq 0} \left[\sum_{i=1}^{n} \left(\min_{k}(c_{ik}) \right) + \sum_{i=1}^{n} \left(\lambda_i (\min_{k} \mu_k) \right) - \sum_{k=1}^{m} \mu_k M_k \right]$$

Posons $\mu_{min} = \min_{k} \mu_k$, alors :

$$\mu_{min} \leq \mu_k$$

et par suite :

$$\mu_{\min} \sum_{i=1}^{n} \lambda_i \leq \mu_{\min} \sum_{k=1}^{m} M_k \text{ car } \sum_{i=1}^{n} \lambda_i \leq \sum_{k=1}^{m} M_k \text{ et } \mu \geq 0$$

D'où :

$$\sum_{i=1}^{n} \lambda_i \left(\min_{k} \mu_k \right) - \sum_{k=1}^{m} \mu_k M_k \leq 0$$

et $\max_{\mu \geq 0} \left[\sum_{i=1}^{n} \left(\min_{k}(c_{ik}) \right) + \sum_{i=1}^{n} \left(\lambda_i (\min_{k} \mu_k) \right) - \sum_{k=1}^{m} \mu_k M_k \right]$ est atteint pour $\mu_k = 0, \ \forall k$.

Ainsi, une borne inférieure pour le problème primal est :

$$LB1 = \sum_{i=1}^{n} \min_{k}(c_{ik})$$

Cette borne inférieure ne tient pas compte des coûts de relève. En fait, on suppose que la contrainte (2.3) de capacité est relaxée et que toutes les cellules peuvent être affectées à un seul commutateur. On a donc une borne inférieure valable quelles que soient les valeurs des M_k et λ_i.

Pour les cas (5.2) et (5.3) ou pour une combinaison quelconque de (5.1), (5.2) et (5.3), l'analyse peut se faire d'une manière similaire. Toutefois, les bornes inférieures obtenues sont très compliquées et dépendent de l'affectation à faire (ce qui revient à résoudre d'abord le problème primal pour trouver une borne inférieure!).

En ne considérant pas le cas trivial où toutes les cellules sont affectées à un même commutateur, nous pouvons supposer sans grande perte de généralité qu'un seul commutateur ne peut prendre en charge toutes les cellules. On a donc au moins une bipartition de l'ensemble des cellules. Dans le cas d'une bipartition (p,q) d'un ensemble de n cellules, le nombre total de coûts de relève à prendre en compte est $t = 2pq$. Le nombre minimal de relèves à considérer pour une bipartition est obtenue en résolvant le problème :

$$\min 2pq$$

$$\text{sujet à : } p + q = n,\ p \geq 1,\ q \geq 1.$$

Le problème ci-dessus a pour solution les bipartitions $(1,n-1)$ et $(n-1,1)$. Ces solutions sont celles qui, de toutes les partitions possibles, engendrent le moins de relèves (car une multipartition est obtenue en divisant plusieurs fois encore une bipartition). Donc, une borne inférieure pour le nombre de relèves est $2(n-1)$. Notons que si la relève $i \rightarrow j$ est prise en compte, il en est forcément de même avec la relève $j \rightarrow i$. Soit alors h_T la partie triangulaire supérieure de la matrice $(H+H^T)$ où H désigne la matrice des coûts de relève et H^T sa transposée. Si on veut prendre en compte le coût des relèves dans la borne inférieure, on doit considérer au moins $n-1$ relèves de h_T. De ce fait, une nouvelle borne inférieure est :

$$LB2 = \sum_{i=1}^{n} \min_{k}(c_{ik}) + \sum_{p=1}^{n-1} \sum_{q=p+1}^{n} \mathbf{1}_{N\_}\{h_T(p,q)\}.h_T(p,q)$$

où $N\_$ désigne l'ensemble des $n-1$ premiers minima de la matrice triangulaire h_T, $\mathbf{1}_{N\_}$ désigne la fonction indicatrice de l'ensemble $N\_$. avec $\mathbf{1}_{N\_}[x] = 1$ si $x \in N\_$ et 0 sinon.

Enfin $h_T(p,q)$ désigne l'élément à la ligne p et à la colonne q de la matrice triangulaire h_T. Pour finir, notons que $LB1 \leq LB2$ et donc $LB2$ est a priori une meilleure borne inférieure que $LB1$.

5.4.2 Comparaison avec la borne inférieure

Comme nous l'avons déjà dit, $LB2$ est a priori une meilleure borne inférieure que $LB1$, mais pour un nombre de cellules dépassant la dizaine, la matrice H contient « beaucoup » d'éléments nuls, car chaque cellule a au plus 6 voisins sur la grille. De ce fait, les $n-1$ plus petits éléments de h_T sont en général tous nuls et les bornes $LB1$ et $LB2$ sont alors égales. C'est le cas pour les tests que nous avons utilisés. Les tableaux 5.3 à 5.8 donnent, pour chacune des séries de tests définies au Tableau 5.2, la distance moyenne par rapport à la borne inférieure des solutions obtenues pour un nombre donné de cellules et de commutateurs.

Tableau 5.3 Distance par rapport à la borne inférieure (série n°1)

#cellules	15	20	30	40	50	60	70	80	90	100	125	150	175	200
Distance moyenne (%)	10	7	4	4	3	2	2	2	2	2	1	2	1	2

Tableau 5.4 Distance par rapport à la borne inférieure (série n°2)

#cellules	15	20	30	40	50	60	70	80	90	100	125	150	175	200
Distance moyenne (%)	25	16	12	7	6	5	5	4	4	4	4	3	3	3

Tableau 5.5 Distance par rapport à la borne inférieure (série n°3)

#cellules	15	20	30	40	50	60	70	80	90	100	125	150	175	200
Distance moyenne (%)	37	24	16	12	10	8	7	6	8	6	6	5	5	4

Tableau 5.6 Distance par rapport à la borne inférieure (série n°4)

#commutateurs	2	3	4	5	6	7
Distance moyenne (%)	3	6	11	15	20	21

Tableau 5.7 Distance par rapport à la borne inférieure (série n°5)

#commutateurs	2	3	4	5	6	7
Distance moyenne (%)	1	3	5	6	8	9

Tableau 5.8 Distance par rapport à la borne inférieure (série n°6)

(cell~comm)	(15~2)	(30~3)	(50~4)	(100~5)	(150~6)	(200~7)
Distance moyenne (%)	11	9	10	9	9	8

La méthode de RT donne des résultats assez « proches » de la borne inférieure, donc de l'optimum. Rappelons que notre borne inférieure ne tient pas compte des coûts de relève, ce qui fait qu'aucune de nos solutions ne peut être égale à la borne inférieure. On note que les solutions obtenues sont plus proches de la borne inférieure pour des problèmes de grande taille, en particulier pour un plus grand nombre de cellules. Ceci s'explique par le fait que, pour les problèmes de taille élevée, les coûts de liaison sont primordiaux. En effet, une cellule a au plus six voisins et, de ce fait, un accroissement du nombre de cellules entraîne des coûts de relève pratiquement nuls. Ces derniers contribuent donc de moins en moins à l'évaluation de la solution.

5.5 Comparaison avec d'autres heuristiques

Du point de vue théorique, notre adaptation de RT génère d'assez bons résultats. Pour confirmer notre analyse, nous la comparons dans cette section avec d'autres heuristiques appliquées au même problème d'affectation de cellules.

5.5.1 Heuristique de Merchant et Sengupta

Cette heuristique (*H1*) proposée par Merchant et Sengupta (1995) est conçue spécialement pour le problème d'affectation de cellules. Elle a été déjà présentée au chapitre 2 de ce mémoire. Les données de comparaison fournies par ces chercheurs sont peu nombreuses et pas très pertinentes dans notre cas. Toutefois, nous avons comparé les temps d'exécution et le pourcentage de solutions faisables pour quelques cas. Nos tests ont porté sur des cas ayant un nombre variable de cellules et quatre commutateurs. Ces cas ont été extraits de la série n°3 de tests. Pour avoir les mêmes conditions que ces chercheurs, nous avons exécuté notre programme sur une « *SUN SPARCStation 2 (4/75)* ». Le Tableau 5.9 présente une comparaison des moyennes obtenues.

Les deux heuristiques trouvent toujours une solution faisable. Pour un grand nombre de cellules, l'algorithme de RT est un peu plus rapide que l'heuristique de Merchant et Sengupta. Par contre, pour des problèmes de taille moins élevée, notre adaptation est un peu plus lente. Toutefois, ces résultats nous renseignent seulement sur la faisabilité des solutions obtenues, sans pour autant révéler si ce sont de bonnes solutions (avec une faible évaluation) ou non. Merchant et Sengupta (1995) donnent un

exemple détaillé d'application de leur méthode. Sur les données de l'exemple, notre adaptation, tout comme la méthode de Merchant et Sengupta, aboutit à l'optimum.

Tableau 5.9 Comparaison de notre adaptation de RT avec l'heuristique de Merchant et Sengupta

cellules	# problèmes		% solutions faisables		Temps CPU (sec)	
	H1	RT	H1	RT	H1	RT
15	74	50	100	100	0.05	0.08
20	74	50	100	100	0.09	0.13
30	74	50	100	100	0.16	0.24
40	29	50	100	100	0.30	0.37
50	9	50	100	100	0.43	0.53
60	9	50	100	100	0.64	0.65
70	9	50	100	100	0.88	0.82
80	9	50	100	100	1.22	1.02
90	9	50	100	100	1.53	1.29
100	9	50	100	100	1.96	1.53

5.5.2 Heuristique de Beaubrun, Pierre et Conan

Beaubrun et al. (1999) présentent aussi une heuristique (*H2*) adaptée spécifiquement au problème d'affectation. L'idée principale de leur algorithme est de trouver une fonction de décision permettant de déterminer à chaque étape une « bonne » affectation de la cellule courante. Pour cela, leur méthode s'inspire de la programmation dynamique et du principe d'optimalité de Bellman. Les cellules sont d'abord triées selon certains critères constitués essentiellement des coûts induits de relève et de liaison. Ensuite, une fonction de décision permet à chaque étape d'affecter les cellules dans l'ordre obtenu par le tri. Les tableaux 5.10 et 5.11 présentent une comparaison entre

l'heuristique *H2* et notre adaptation de RT sur les moyennes des résultats obtenus à partir des séries de tests n°2 et 6.

Tableau 5.10 Comparaison de notre adaptation de RT avec l'heuristique de Beaubrun, Pierre et Conan (série n°2)

# cellules	# commutateurs	% amélioration moyenne RT par rapport H2
15	3	34
20	3	34
30	3	34
40	3	38
50	3	38
60	3	39
70	3	38
80	3	40
90	3	41
100	3	40
125	3	39
150	3	40
175	3	40
200	3	40

Tableau 5.11 Comparaison de notre adaptation de RT avec l'heuristique de Beaubrun, Pierre et Conan (série n°6)

# cellules	# commutateurs	% amélioration moyenne RT par rapport H2
15	2	31
30	3	36
50	4	42
100	5	53
150	6	56
200	7	59

La méthode de RT se révèle de manière constante meilleure à l'heuristique *H2*. Les résultats trouvés par notre adaptation sont en général de 30 à 50% meilleurs à ceux

de l'heuristique *H2*. En particulier pour des problèmes de taille élevée, notre adaptation fournit des solutions de coût remarquablement moindre. Ceci s'explique par le fait que l'heuristique *H2* affecte les cellules au fur et à mesure en tenant compte seulement des cellules déjà affectées auparavant. Si la taille du problème croît, les premières décisions prises ont beaucoup plus de chances de ne pas être bonnes, puisqu'elles ne tiennent pas compte des affectations faites à leur suite.

5.5.3 Recuit simulé

La méthode du recuit simulé (SA pour simulated annealing) est une approche algorithmique probabiliste pour la résolution de problèmes d'optimisation. Le nom de la méthode dérive d'une analogie entre la résolution de problèmes d'optimisation et la simulation du recuit des solides proposée par Metropolis et al. (1953). La méthode a été adaptée à l'optimisation combinatoire par Kirkpatrick et al. (1983). Contrairement à un algorithme de recherche locale, la méthode de recuit simulé permet, dans un problème d'optimisation donné, d'envisager des solutions qui dégradent le coût, quitte à abandonner plus tard lesdites solutions si elles n'offrent toujours pas d'améliorations. Pour ce faire, elle utilise le hasard pour décider si elle accepte ou non une solution qui dégrade le coût. Nous avons comparé notre adaptation de RT à une application de SA au problème d'affectation. L'implémentation de SA utilisée a été développée au Laboratoire de Recherche en Réseautque et Informatique Mobile (LARIM) de l'École Polytechnique de Montréal. Comme la solution finale obtenue par la méthode de SA dépend du hasard, nous avons choisi dans chacune des six séries deux cas de test pour un

nombre de cellules et de commutateurs donnés. L'implémentation de SA a été exécutée 500 fois sur chacun des cas et nous retenons la meilleure évaluation obtenue. Les tableaux 5.11 et 5.12 présentent une comparaison des solutions obtenues pour quelques séries de tests.

Tableau 5.12 Comparaison avec la méthode de recuit simulé (série n°2)

# cellules	# commutateurs	SA	RT	% amélioration RT par rapport SA
15	3	105	103	2
15	3	139	124	11
20	3	189	211	-12
20	3	161	158	2
30	3	359	295	17
30	3	369	364	2
40	3	553	444	20
40	3	611	420	31
50	3	724	597	18
50	3	748	588	21
60	3	832	713	14
60	3	1073	866	19

Tableau 5.13 Comparaison avec la méthode de recuit simulé (série n°6)

# cellules	# commutateurs	SA	RT	% amélioration RT par rapport SA
15	2	124	130	-5
15	2	123	118	4
30	3	405	382	5
30	3	448	324	28
50	4	951	689	28
50	4	851	580	32
100	5	1999	1374	31
100	5	2595	1234	52
150	6	4271	2013	53
150	6	3240	2010	38
200	7	5550	2948	47
200	7	7801	2768	64

La méthode de RT est de manière générale plus performante que l'adaptation de SA. En particulier, pour des problèmes de grande taille, notre adaptation trouve parfois des solutions deux fois meilleures à celles de SA.

En résumé, notre adaptation de RT semble convenir très bien à des problèmes de taille assez élevée. En effet, les solutions obtenues sont en général meilleures par rapport à d'autres méthodes, mais aussi sont assez proches de l'optimum. De plus, le temps de calcul ne croît pas exponentiellement avec la taille des problèmes. Par contre, pour des problèmes de taille plus petite, bien qu'il ait toujours un comportement convenable, notre algorithme ne donne pas toujours les meilleurs résultats. Dans ces cas, il serait préférable de désactiver tous les mécanismes d'intensification et de diversification, ce qui permettrait de gagner en temps de calcul sans trop perdre en qualité de solution.

CHAPITRE 6

CONCLUSION

Le problème d'affectation de cellules est un problème difficile que nous avons essayé de résoudre dans ce mémoire, en lui appliquant la méthode de RT. Les résultats obtenus sont assez concluants. Dans ce chapitre final, nous présentons une synthèse générale des travaux avant d'aborder les limitations de notre adaptation et les indications de recherche future.

6.1 Synthèse des travaux

Le problème d'affectation de cellules à des commutateurs consiste à affecter un nombre n donné de cellules à m emplacements possibles de commutateurs, tout en respectant des contraintes sur la capacité des commutateurs et en affectant chaque cellule à un commutateur unique. Compte tenu de sa complexité, ce problème ne peut, à l'heure actuelle, être résolu de manière exacte, surtout pour des instanciations de taille élevée. On a alors recours à des heuristiques qui fournissent des solutions souvent sous-optimales. Dans ce mémoire, l'heuristique adaptée est celle de recherche taboue. Elle a été introduite en optimisation combinatoire par Glover (1989, 1990a, 1990b) et utilise des techniques d'exploration sophistiquées pour éviter à une autre heuristique le blocage à un minimum local. Elle est de ce fait une méta-heuristique.

Avant d'adapter la méta-heuristique de RT, nous avons proposé une nouvelle formulation du problème, mettant en évidence les similitudes avec le problème bien connu en optimisation combinatoire de localisation d'entrepôts.

L'implémentation de la méthode a consisté à définir une série de mouvements pour les composantes de mémoire à court et à moyen termes. Ces mouvements visent une amélioration du coût de la solution et un rétablissement de sa faisabilité. Ceci nous a amené à définir une structure de gains avec des procédures de mise à jour pour choisir efficacement à chaque itération la meilleure solution dans le voisinage courant. La mise en œuvre des composantes de mémoire à moyen et long termes a nécessité un choix des zones d'intensification et la définition d'une politique de diversification.

Pour évaluer les performances de l'adaptation, nous avons déterminé deux bornes inférieures de l'optimum global. Ces deux bornes nous ont servi d'estimés pour juger de la qualité des solutions obtenues. De manière générale, les résultats sont convaincants et nos solutions sont assez proches de l'optimum global. Notre implémentation a été aussi testée par rapport à différents paramètres de la méthode de RT pour améliorer sa robustesse et son efficacité. Ainsi, nous avons étudié, entre autres, l'influence de la taille de la liste taboue, l'effet de la taille de la zone d'intensification et celui du nombre de cycles de diversification. Enfin, les résultats ont été confrontés à ceux obtenus par d'autres heuristiques appliquées au même problème, ce qui a confirmé l'efficacité et la robustesse de l'adaptation, surtout pour des problèmes de taille assez élevée (à partir de 50 cellules et 3 commutateurs). Cependant, notre adaptation n'est pas parfaite et comporte quelques limitations.

6.2 Limitations des travaux

En dépit des résultats satisfaisants obtenus, les performances de notre approche dépendent fortement du choix des paramètres. Ceux-ci ne sont pas faciles à choisir, et les valeurs utilisées dans le programme sont des valeurs qui donnent globalement de bons résultats, mais qui ne sont pas forcément les meilleures pour un type donné de problème. De plus, les données des tests utilisés pour calibrer ces différents paramètres proviennent de simulations et ne sont donc pas des données réelles.

Notons aussi que la méthode ne garantit pas la faisabilité de la solution finale. On peut donc aboutir dans certains rares cas à une solution finale qui, malgré les pénalités, possède une meilleure évaluation que les solutions faisables trouvées. En ce moment, nous avons préféré la solution non faisable, quitte à ce que le concepteur modifie légèrement le problème pour la rendre faisable si cela n'engendre pas de trop grands coûts.

Les bornes inférieures utilisées comme estimation de l'optimum global se retrouvent, surtout dans le cas de problèmes de taille moyenne, trop loin de l'optimum. L'estimation n'est donc plus très bonne.

Enfin, notre implémentation ne prend pas en compte le problème d'affectation avec domiciliation double des cellules.

6.3 Indications de recherche future

Le problème d'affectation de cellules à des commutateurs demeure un problème ouvert. Les pistes d'amélioration sont nombreuses. Les recherches futures pourraient essayer de déterminer de meilleures bornes inférieures prenant adéquatement en compte les coûts de relève, surtout pour des problèmes de taille moyenne. Il faudrait également envisager une implémentation de la méthode de RT pour le problème avec domiciliation double. Cette implémentation pourrait être directe et utiliser une structure de gains, comme nous l'avons fait pour le cas de domiciliation simple, ou bien elle pourrait être définie comme ensemble de sous-problèmes à domiciliation simple tel que proposé par Merchant et Sengupta (1995). Dans ce cas, les sous-problèmes de domiciliation simple pourraient être résolus avec notre adaptation. Enfin, de nouvelles heuristiques ou des combinaisons d'heuristiques pourraient être proposées et adaptées au problème.

BIBLIOGRAPHIE

Abdinnour-Helm S. « A hybrid heuristic for the uncapacitated hub location problem », *European Journal of Operational Research*, vol. 106, 1998, pp. 489-499.

Beaubrun R., Pierre S. et Conan J. « An efficient Method for Optimizing the Assignment of Cells to MSCs in PCS Networks », *Proceedings 11th Int. Conf. on Wireless Comm., Wireless 99*, vol. 1, July 1999, Calgary (AB), pp. 259-265.

Dell'Amico M., Trubian M. « Solution of large weighted equicut problems », *European Journal of Operational Research*, vol. 106, 1998, pp. 500-521.

Fiduccia C. M. et. Mattheyses R. M. « A Linear-time Heuristic for Improving Network partition », *Proceedings 19th Design Automat. Conf.*, 1982, pp. 175-181.

Garey M. R. et Johnson D. S., *Computers and intractability*, San Francisco, CA, Freeman, 1979.

Glover F. « Tabu Search - Part I », *ORSA Journal on Computing*, vol. 1, No. 3, 1989, pp. 190-206.

Glover F. « Tabu Search : A Tutorial », *INTERFACES*, vol. 20, No. 4, 1990a, pp. 74-94.

Glover F. « Tabu Search - Part II », *ORSA Journal on Computing*, vol. 2, 1990b, pp. 4-32.

Glover F., Taillard E. et de Werra D. « A user's guide to tabu search », *Annals of Operations Research*, vol. 41, No. 3, 1993, pp. 3-28.

Horowitz E. et Sahni S., *Fundamentals of computer algorithms*, Computer science Press, Inc., 1990.

Kernighan B. W. et Lin S. « An Efficient Heuristic Procedure for Partitioning Graphs », *The Bell System Technical Journal*, vol. 49, 1970, pp. 291-307.

Kirkpatrick S., Gelatt Jr. C. D. et Vecchi M. P. « Optimization by Simulated Annealing », *Science*, vol. 220, 1983, pp. 671-680.

Kleinrock L. *Queuing Systems I: Theory*, New York, Wiley, 1975.

Klincewicz J. G. « Heuristics for the p-hub location problem », *European Journal of Operational Research*, vol. 53, 1991, pp. 25-37.

Krishnamurthy B. « Constructing Test Cases for Partitioning Heuristics », *IEEE Transactions on Computers*, vol. C-36, Sept. 1987, pp. 1112-1114.

Kuehn A.A. et Hamburger M. J. « A heuristic program for locating warehouses », *Management Science*, vol. 9, 1963, pp. 643-666.

Metropolis N., Rosenbluth A., Rosenbluth M., Teller A. et Teller E. « Equation for State Calculations by Fast Computing Machines », *Journal of chemical physics*, vol. 21, 1953, pp. 1087-1092.

Merchant A. et Sengupta B. « Multiway graph partitioning with applications to PCS networks », *IEEE Infocom'94*, vol. 2, 1994, pp. 593-600.

Merchant A. et Sengupta B. « Assignment of Cells to Switches in PCS Networks », *IEEE/ACM Transactions on Networking*, vol. 3, No. 5, 1995, pp. 521-526.

O'Kelly M. E. « A quadratic integer program for the location of interacting hub facilities », *European Journal of Operational Research*, vol. 32, 1987, pp. 393-404.

Pierre S. et Elgibaoui A. « A Tabu-Search Approach for Designing Computer-Network Topologies with Unreliable Components », *IEEE Transactions on Reliability*, vol. 46, n°3, Sept. 1997, pp. 350-359.

Pierre S. et Legault G. « A Genetic Algorithm for Designing Distributed Computer Network Topologies », *IEEE Transactions on Systems, Man, and Cybernetics—Part B : Cybernetics*, vol. 28, No 2, April. 1998, pp. 249-258.

Pierre S. « Inferring New Design Rules by Machine Learning : A Case Study of Topological Optimization », *IEEE Transactions on Systems, Man, and Cybernetics—Part A : Systems and Humans*, vol. 28, No 5, Sept. 1998, pp. 575-585.

Sanchis L. A. « Multiple-Way Network Partitioning », *IEEE Transactions on Computers*, vol. 38, No 1, 1989, pp. 62- 81.

Skorin – Kapov J. « Tabu search applied to the quadratic assignment problem », *ORSA Journal on Computing*, vol. 2, No. 1, 1989, pp. 33-45.

Skorin-Kapov D. et Skorin-Kapov J. « On tabu search for the location of interacting hub facilities », *European Journal of Operational Research*, vol. 73, 1994, pp. 502-509.

Sohn J. et Park S. « Efficient solution procedure and reduced size formulations for p-hub location problems », *European Journal of Operational Research*, vol. 108, 1998, pp. 118-126.

ANNEXE A

COMPOSITION DES SÉRIES DE TESTS

Tableau A.1 Composition des séries de tests n°1, 2 et 3

Nombre de cellules	Nombre de cas
15	50
20	50
30	50
40	50
50	50
60	50
70	50
80	50
90	50
100	50
125	50
150	50
175	50
200	50

Tableau A.2 Composition des séries de tests n°4, 5 et 6

Nombre de commutateurs	Nombre de cas
2	50
3	50
4	50
5	50
6	50
7	50

ANNEXE B

TEMPS MOYEN D'EXÉCUTION

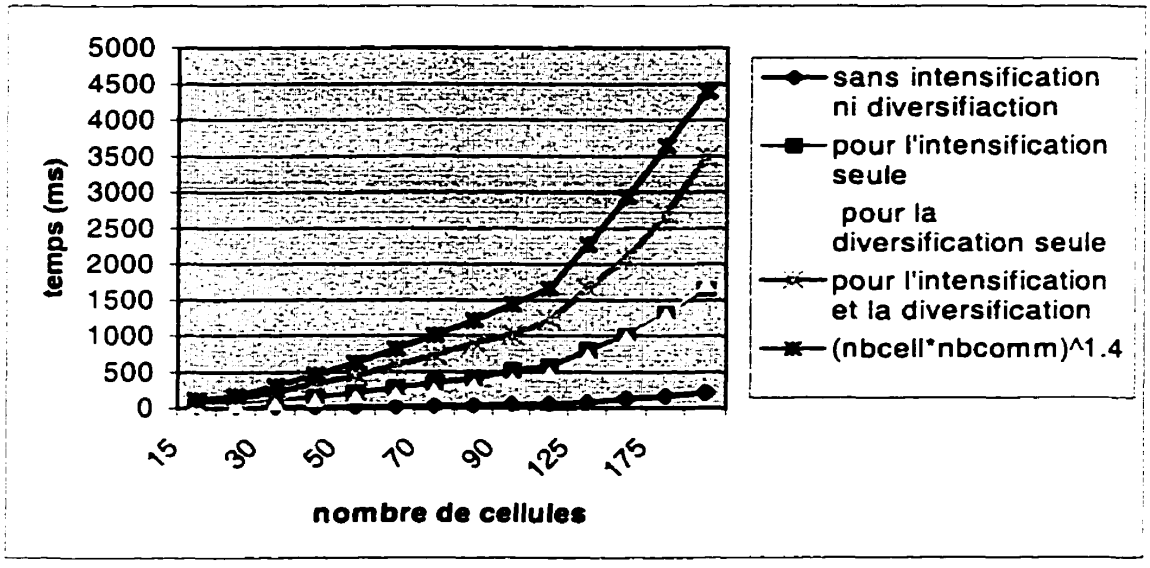

Figure B.1 Temps moyen d'exécution (série n°1)

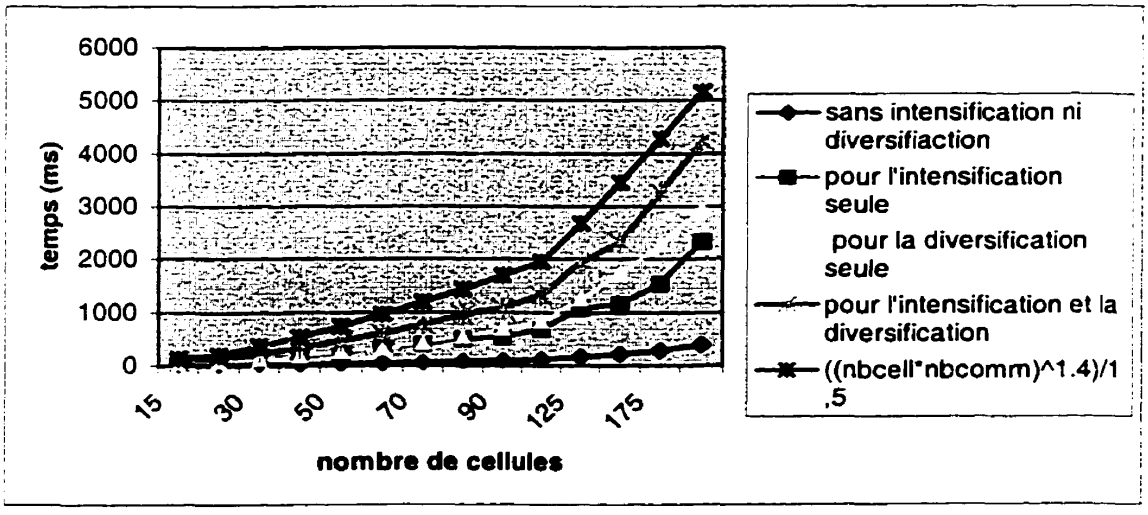

Figure B.2 Temps moyen d'exécution (série n°2)

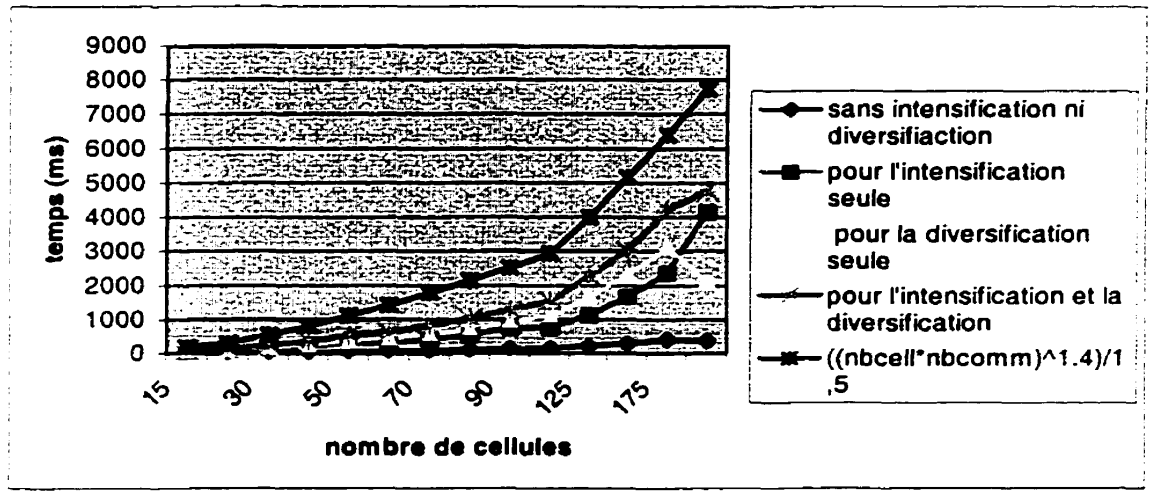

Figure B.3 Temps moyen d'exécution (série n°3)

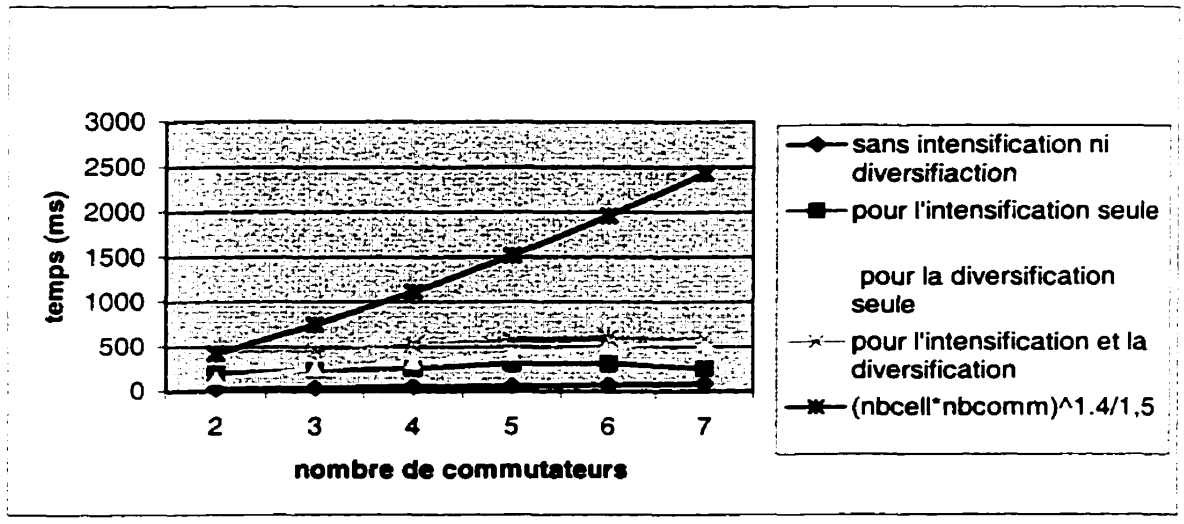

Figure B.4 Temps moyen d'exécution (série n°4)

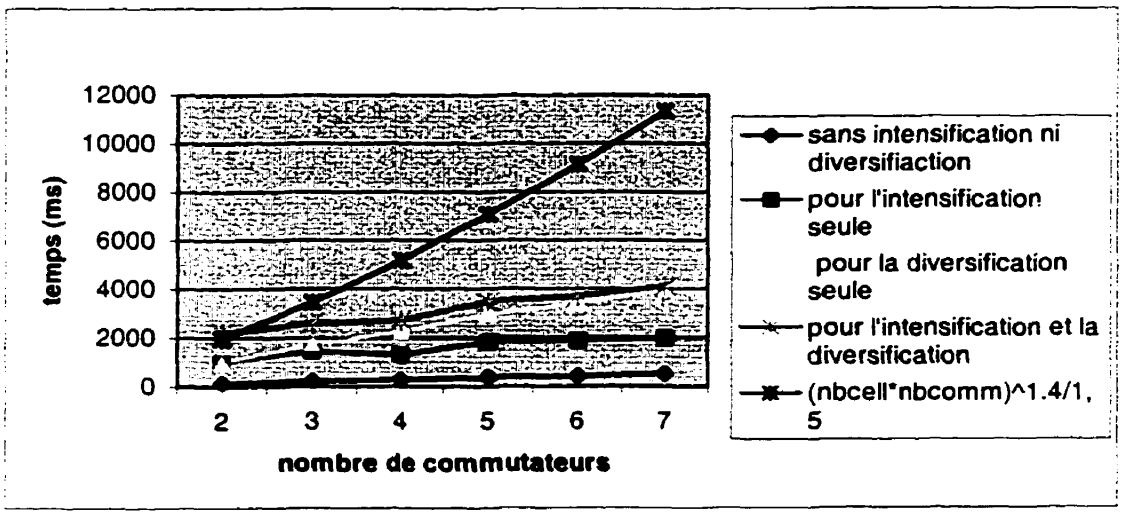

Figure B.5 Temps moyen d'exécution (série n°5)

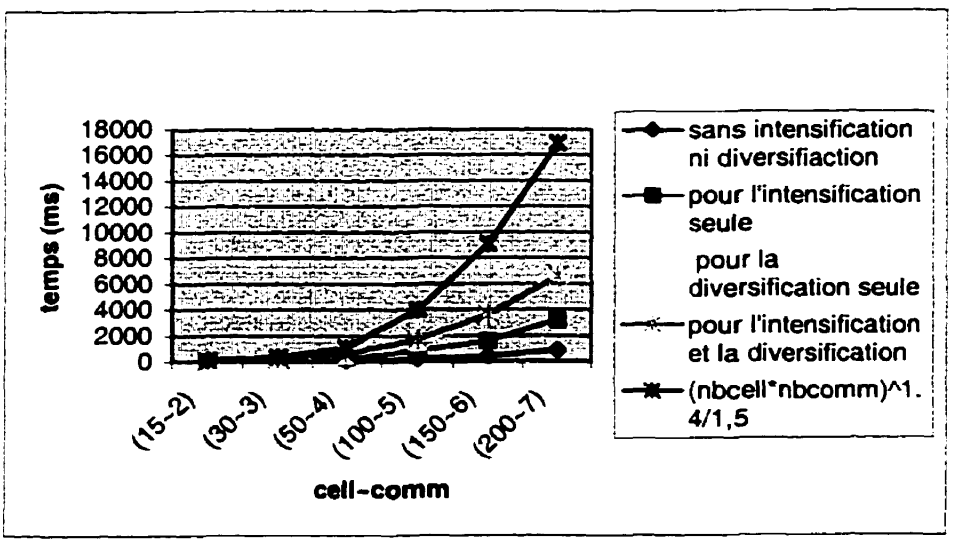

Figure B.6 Temps moyen d'exécution (série n°6)