UNIVERSITÉ DE MONTRÉAL

ALGORITHME GÉNÉTIQUE
POUR L'AFFECTATION DE CELLULES À DES COMMUTATEURS
DANS LES RÉSEAUX MOBILES

CONSTANTIN HEDIBLE

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)

SEPTEMBRE 2000

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

ALGORITHME GÉNÉTIQUE POUR L'AFFECTATION DE CELLULES À DES COMMUTATEURS DANS LES RÉSEAUX MOBILES

présenté par: HEDIBLE Constantin

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., Directeur

Mme BELLAICHE Martine, M.Sc., Membre

REMERCIEMENTS

Je désire remercier mon directeur de recherche, le professeur Samuel Pierre, pour sa patience, ses conseils pertinents et son soutien tout au long de ma recherche.

Je désire ensuite remercier tous mes collègues du Laboratoire de Recherche en Réseautique et Informatique Mobile (LARIM), pour leurs critiques pertinentes et constructives tout au long de ce travail.

Je désire également remercier mes parents et mes frères pour leur soutien moral et leurs prières.

Je désire enfin remercier tous ceux qui m'ont soutenu et aidé dans l'accomplissement de cette noble tâche.

RÉSUMÉ

Durant les dernières décennies, les télécommunications ont connu un développement prodigieux. Le besoin d'échanger des informations avec un usager, quels que soient le moment et l'endroit, a entraîné l'apparition des Réseaux de Communications Personnelles (RCP) qui sont des réseaux sans-fil intégrant plusieurs services tels que la voix, la vidéo, le courrier électronique, etc. La gestion de ces réseaux a entraîné plusieurs problèmes parmi lesquels celui de l'optimisation des coûts de relève et de liaison par rapport à l'affectation des cellules aux commutateurs dont les localisations potentielles sont connues, tout en respectant la contrainte d'affectation unique des cellules aux commutateurs. La configuration obtenue doit aussi être faisable, c'est-à-dire qu'elle doit tenir compte de la capacité limitée des commutateurs.

Ce mémoire a pour but essentiel la mise au point d'une adaptation de la métaheuristique des *algorithmes génétiques* pour la résolution du problème décrit plus haut. Pour y parvenir, nous avons proposé une formulation mathématique du problème qui inclut la fonction objectif à minimiser et les contraintes à respecter. Comme contributions de ce mémoire, nous avons conçu une procédure efficace pour la génération d'une population initiale sans doubles. Nous avons ensuite défini et implémenté des opérateurs génétiques spécifiques au contexte de notre problème, et qui sont appliqués par la suite à cette population initiale dans le but d'atteindre la solution qui minimise le coût du réseau, tout en respectant les contraintes.

Comme autre contribution, nous avons évalué la performance de l'algorithme génétique, en le comparant à trois autres méthodes alternatives qui ont été appliquées au même problème. Cette comparaison place l'adaptation proposée en bonne position parmi les méthodes concurrentes. Nous avons enfin proposé une version parallèle préliminaire de notre algorithme génétique, qui sera exécutable dans un environnement avec plusieurs processeurs dans le cadre des travaux futurs.

Les résultats obtenus nous ont permis d'aboutir à certaines conclusions que nous résumons en quatre points :

- Une grande taille permet d'avoir une population assez diversifiée et augure d'une bonne couverture de l'espace de recherche. Néanmoins, elle ne mène pas nécessairement vers une solution optimale ;
- Une augmentation de la probabilité de croisement accroît les chances d'aboutir à une bonne solution. En particulier, les meilleurs résultats sont fournis par une probabilité de croisement de 0.9 ;
- De même, une diminution de la probabilité de mutation entraîne une baisse des coûts des solutions obtenues. Les meilleures solutions sont obtenues en particulier pour une probabilité de mutation de 0.05 ;
- Enfin, dans la classification établie à partir de l'analyse comparative effectuée sur quatre méthodes appliquées au même problème, l'algorithme génétique proposé se classe derrière la recherche taboue et devant le recuit simulé et l'heuristique proposée par Beaubrun, Pierre et Conan (1999).

ABSTRACT

During the last decades, telecommunications field experienced an extraordinary development. The need to exchange information with a user, whatever the moment and the place, led to the appearance of the Personal Communication Systems (PCS) which are wireless networks integrating several services such as voice, video, electronic mail, etc. PCS management generate several problems of which the one of the optimization of handoffs and cabling costs, while respecting switches capacities and assigning each cell to one switch.

The purpose of this thesis is essentially the development of a *genetic algorithm* adaptation for the resolution of the previously described problem. For that purpose, we proposed a mathematical formulation of the problem, which includes the objective function to be minimized and the constraints to be respected. As contributions to this thesis, we designed an effective procedure for the generation of an initial population without doubles. Then we defined, implemented and applied specific genetic operators in the context of our problem, to this initial population in order to reach a minimum cost, while respecting the problem constraints. The results obtained are satisfactory in spite of the execution time, which is rather prohibitory. As other contribution, we also evaluated the performance of the genetic algorithm, by comparing it with three other alternative methods which were applied to the same problem. This comparison places our adaptation in good position among the compared methods. We finally designed a preliminary parallel version of our genetic algorithm, executable in an environment with several processors.

The results obtained enabled us to arrive at certain conclusions which we summarize in four points:

- a big size makes it possible to have diversified an enough population and forecasts of a good cover of the space of search. Nevertheless, it does not necessarily carry out towards an optimal solution ;

- an increase of the probability of crossover increases the chances to lead to a good solution. In particular, the best results are provided by a probability of crossover of 0.9 ;
- In the same way, a reduction in the probability of mutation leds to a fall in the costs of the solutions obtained. The best solutions are obtained in particular for a probability of mutation transfer of 0.05 ;
- Finally, in the classification established starting from the comparative analysis carried out on four methods applied to the same problem, the genetic algorithm proposed is classified behind tabu search and above simulated annealing and the heuristic proposed by Beaubrun, Pierre and Conan (1999).

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE I
INTRODUCTION

Depuis les dernières décennies, les télécommunications ont connu un développement prodigieux. Elles sont devenues un instrument omniprésent dans la société moderne. D'autre part, les individus étant en perpétuel mouvement, la nécessité de les joindre quels que soient le moment et l'endroit où ils se trouvent s'est fait sentir. Les communications se devaient donc d'accéder à une autre étape de leur évolution, celle de la mobilité. Ce concept de mobilité a permis le développement de l'outil de plus en plus répandu que constitue le téléphone cellulaire. Cependant, au delà des facilités créées comme la possibilité de joindre une personne à un endroit et à un moment quelconques, la notion de mobilité a engendré plusieurs problèmes de gestion de réseau, dont celui de l'affectation des cellules aux commutateurs dans les réseaux de communications personnelles (RCP) sur lequel portera ce mémoire. Dans ce chapitre, nous allons d'abord introduire les concepts de base qui permettent de définir notre problématique de recherche ; par la suite, nous exposerons les objectifs visés et les principales contributions qui en découlent, pour enfin esquisser le plan du mémoire.

1.1 Définitions et concepts de base

Un *réseau de communications personnelles (RCP)* est un réseau de communications sans-fil qui intègre différents services (voix, vidéo, courrier électronique, etc.) accessibles d'un terminal unique et pour lesquels l'usager obtient une facturation unique. Ces différents services sont offerts dans une région appelée *zone de couverture* qui est divisée en *cellules*. Chaque cellule comporte une *station de base* qui gère toutes les communications à l'intérieur de cette cellule. Dans la zone de couverture, les cellules sont reliées à des unités spéciales appelées *commutateurs*.

On désigne par *cellule* la surface géographique sur laquelle est assurée la disponibilité d'un canal de transmission donné; ce canal est constitué d'une voie radioélectrique caractérisée par une fréquence donnée ou un couple de fréquences

donné, selon le service assuré. Cette surface peut être de forme quelconque, mais elle est souvent modélisée par un cercle ou par un hexagone pour des raisons de calcul. Le rayon de ce cercle varie en fonction du nombre d'usagers par unité de surface (1, 2, 4, 8, 16, 32 kilomètres).

En radiocommunication mobile, on se sert des canaux radios pour acheminer les communications. Afin d'éviter les interférences, les groupes de canaux radios utilisés par deux cellules adjacentes sont différents. Donc, lorsqu'un usager en communication passe d'une cellule à une autre, la station de base de la nouvelle cellule doit allouer un canal radio libre au mobile afin d'assurer une bonne qualité de service. L'acte de supporter le transfert de la communication d'un mobile d'une station de base à une autre se nomme *relève entre cellules* (handoff).

Dans le cas de la Figure 1.1 par exemple, lorsque l'usager passe de la cellule B à la cellule A, on parle de *relève simple* car ces deux cellules sont reliées au même commutateur. Le commutateur qui supervise les deux cellules reste le même et le coût induit est faible. Par contre, lorsque l'abonné passe de la cellule B à la cellule C, on parle de *relève complexe*. Le coût induit est élevé car les commutateurs 1 et 2 restent actifs pendant toute la procédure de relève et la base de données contenant les informations sur l'abonné doit être mise à jour. Il apparait donc intuitivement plus judicieux de relier les cellules B et C au même commutateur si la fréquence des relèves entre ces deux cellules est élevée.

Du point de vue de la procédure, on distingue également deux sortes de relève : la *relève «dure»* (hard handoff) et la *relève «douce»* (soft handoff). Pour effectuer la *relève dure*, la communication entre le mobile et la station de base de la première cellule est coupée (cette coupure reste transparente aux utilisateurs) avant que ne s'établisse une communication entre le mobile et la station de base de la deuxième cellule; pour la *relève douce*, le mobile reste en liaison avec la station de base de la première cellule jusqu'à ce que s'établisse une communication avec la station de base de la deuxième cellule.

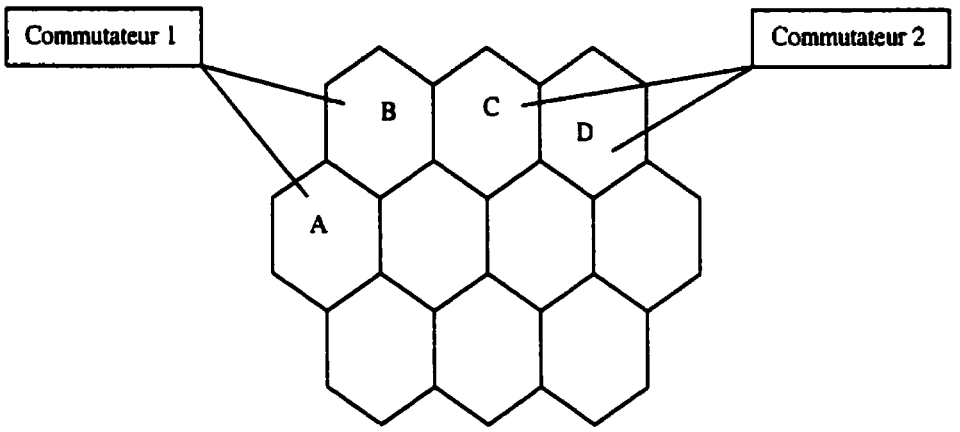

Figure 1.1 Découpage géographique de la zone de couverture en cellules

Une *station de base* est constituée essentiellement d'une antenne qui gère toutes les communications à l'intérieur d'une cellule. Elle gère et alloue les canaux radioélectriques aux usagers dans la mesure de leur disponibilité. Elle est souvent installée au centre de la cellule afin de permettre une couverture globale de celle-ci.

Encore appelé Mobile Switching Center (MSC), un *commutateur* est une unité qui gère, en fonction de sa capacité en volume d'appels, un certain nombre de cellules par l'intermédiaire de leurs stations de base. Il est relié physiquement à ces dernières, surtout au réseau public fixe, afin d'assurer le transfert des appels provenant du réseau fixe vers le réseau mobile et vice versa.

Le coût induit par le phénomène de relève comporte deux composantes : un coût fixe lié à la liaison physique entre la station de base de la cellule et le commutateur auquel elle est associée, et un coût variable lié à la relève qui survient entre deux cellules. Le but visé est donc de réaliser l'affectation des cellules aux commutateurs de façon à minimiser le coût total engendré.

1.2 Éléments de problématique

L'affectation des cellules aux commutateurs dans un Réseau de Communications Personnelles consiste essentiellement à trouver la configuration des liaisons

commutateur-cellule qui minimise le coût total du réseau en tenant compte d'un certain nombre de contraintes dont la configuration du réseau basée sur la fréquence des relèves entre cellules, la capacité de chaque commutateur et le trafic acheminé par l'intermédiaire du réseau..

Considérons l'aspect de la configuration de réseau. Le problème revient, à partir de n cellules et m commutateurs, à trouver un schéma d'affectation des n cellules aux m commutateurs qui minimise le coût total du réseau tout en respectant certaines contraintes. Si nous devions explorer de façon exhaustive tous les schémas d'affectation possibles (m^n dans notre cas) pour en choisir le meilleur, nous déboucherions rapidement sur un problème d'explosion combinatoire que même les ordinateurs les plus performants mettraient un temps excessivement long à résoudre.

Considérons maintenant l'aspect de la capacité des commutateurs. Il s'agit alors de réaliser les affectations en tenant compte du fait qu'un commutateur a une capacité limitée, c'est-à-dire que seul un nombre limité de cellules peut lui être raccordé. Cette capacité s'exprime en volume d'appels par unité de temps.

Considérons enfin l'aspect du trafic acheminé par l'intermédiaire du réseau. Ce trafic peut varier considérablement à deux moments de la journée. Ainsi, un schéma d'affectation efficace à un moment de la journée peut s'avérer inefficace à un autre moment. Il s'agit alors de réaliser deux affectations correspondant chacune à un moment de la journée et ensuite de les implanter toutes deux. Il peut alors arriver qu'une cellule soit reliée à deux commutateurs différents, et cela doit être pris en compte dans le calcul du coût global du réseau.

Tout compte fait, la nature du problème commande une approche heuristique de résolution, ce que certains chercheurs ont déjà préconisé (Merchant et Sengupta, 1994). Dans ce mémoire, l'approche heuristique que nous préconisons est celle des algorithmes génétiques.

1.3 Objectifs de recherche et esquisse méthodologique

Compte tenu des difficultés énoncées précédemment, nous visons dans ce mémoire cinq objectifs principaux :

- construire une population initiale formée de configurations ou schémas d'affectation que nous souhaitons faisables ;
- définir, dans le contexte spécifique de ce problème, un ensemble d'opérateurs génétiques qui inspireront un certain nombre de perturbations ;
- définir un algorithme génétique applicable à cette population initiale, de manière à atteindre une solution minimisant le coût total du réseau, tout en respectant les contraintes ;
- comparer les performances des algorithmes génétiques à celles de trois autres méthodes qui ont été appliquées au même problème ;
- concevoir une version parallèle de l'algorithme génétique exécutable sur une grappe (cluster) d'ordinateurs distribués.

Pour y parvenir, nous ferons subir, par application des opérateurs génétiques que nous définirons, des perturbations à la population initiale de façon à réduire le coût total du réseau. Les configurations qui seront retenues de la nouvelle population ainsi générée seront, nous l'espérons, parmi les meilleures de leur génération. Les solutions qui en seront extraites correspondront donc à des schémas d'affectation faisables ou réalisables. Nous espérons ainsi que les résultats que nous obtiendrons seront meilleurs, comparés à ceux obtenus par d'autres métaheuristiques tels que le recuit simulé et la recherche taboue.

1.4 Principales contributions

Dans ce mémoire, nous ferons une étude comparative de quatre méthodes qui ont été développées au Laboratoire de Recherche en Réseautique et Informatique mobile (LARIM) pour la résolution du problème d'affectation des cellules aux commutateurs dans les réseaux de communication personnelle. Ces méthodes sont : la recherche taboue, le recuit simulé, l'heuristique proposée par Beaubrun, Pierre et Conan (1999) et

bien sûr les algorithmes génétiques. Nous proposerons aussi une version parallèle d'algorithme génétique assortie d'une analyse de complexité mettant en évidence le gain d'efficacité qui pourrait être obtenu par rapport à la version séquentielle développée dans ce mémoire.

1.5 Plan du mémoire

Le chapitre 2 analyse le problème d'affectation de cellules aux commutateurs, les problèmes qui s'y apparentent et les différentes méthodes ayant servi à les résoudre. Le chapitre 3 décrit les fondements des algorithmes génétiques, les opérateurs et le fonctionnement de l'approche; on y aborde aussi les applications des algorithmes génétiques dans le domaine de la conception topologique des réseaux. Le chapitre 4 présente la méthode proposée, une adaptation de l'algorithme génétique appliqué au problème d'affectation de cellules aux commutateurs; nous y décrivons également le choix et le codage des paramètres, ainsi que les opérateurs génétiques retenus. Le chapitre 5 relate les détails d'implantation et de mise en œuvre de notre adaptation. Le chapitre 6 analyse les résultats obtenus et en évalue la qualité par rapport à d'autres méthodes. Nous y proposons aussi une version parallèle de notre algorithme. En guise de conclusion, le chapitre 7 résume les principaux résultats obtenus, fait état des limitations de l'implantation de la méthode et achève sur une indication pour des recherches futures.

CHAPITRE II
ANALYSE DU PROBLÈME D'AFFECTATION DES CELLULES

L'introduction du concept de mobilité dans les RCP a été une grande révolution dans le monde des télécommunications. Toutefois, la planification de ces réseaux soulève de nombreux défis qui ont alimenté la recherche durant les dernières décennies. Dans ce chapitre, nous nous proposons d'analyser plus spécifiquement le problème d'affectation des cellules aux commutateurs dans les réseaux de communications personnelles. Dans un premier temps, nous formulerons le problème; nous l'analyserons ensuite en étudiant sa similarité avec certaines classes de problèmes et enfin nous exposerons les différentes méthodes déjà proposées pour sa résolution.

2.1 Formulation du problème

Le problème d'affectation de cellules aux commutateurs dans un réseau mobile tel que décrit par Merchant et Sengupta (1995) peut se formuler comme suit :

Étant donné

n cellules et m commutateurs,

une matrice des coûts de relèves entre cellules,

une matrice des coûts de liaison cellule-commutateur,

Minimiser

le coût total du réseau

Par choix

du schéma d'affectation

Sous contraintes

de l'affectation unique des cellules aux commutateurs, et

de la capacité du commutateur.

Certaines formulations suggèrent la double affectation, c'est-à-dire qu'une cellule peut être affectée à deux commutateurs dépendamment du trafic à certains moments de la journée. Nous allons étudier séparément chacun des aspects du problème d'affectation des cellules aux commutateurs. Supposons que l'on ait *n* cellules à affecter à *m* commutateurs, les localisations potentielles des commutateurs étant connues; cette affectation peut se réaliser de deux manières : par affectation unique, par affectation double.

2.1.1 Affectation unique

Soit c_{ik} le coût du câblage entre une cellule i et un commutateur k. Soit λ_i le volume d'appels générés par unité de temps dans la cellule i et M_k la capacité du commutateur k. L'objectif visé est donc d'affecter chaque cellule à un commutateur en minimisant le coût total par unité de temps. Ce coût inclut celui des relèves entre cellules par unité de temps et celui du câblage entre la cellule et le commutateur. L'affectation doit être effectuée en respectant la contrainte de capacité de chaque commutateur. Le problème d'affectation des cellules à des commutateurs peut être posé comme un problème de programmation en nombre entiers. Pour cela, définissons la variable

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

Étant donné qu'une cellule ne peut être affectée qu'à un seul commutateur, nous avons la contrainte suivante :

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

De plus, la contrainte sur la capacité des commutateurs s'exprime:

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leq M_k, \; k = 1, \ldots, m \qquad (2.2)$$

Le coût total du câblage entre une cellule i et un commutateur k est représenté par la relation:

$$\sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik} x_{ik} \tag{2.3}$$

Soit H_{ij} et H'_{ij} les coûts des relèves respectivement si les cellules i et j sont affectées au même commutateur et si elles sont affectées à des commutateurs différents. Le coût induit par ces relèves est difficile à cerner. Nous définissons donc les variables additionnelles :

$$z_{ijk} = x_{ik} * x_{jk}\ ,\ i, j = 1, \ldots, n \text{ et } k = 1, \ldots, m \tag{2.4}$$

$$z_{ijk} = \begin{cases} 1 \text{ si les cellules i et j sont affectées au commutateur k} \\ 0 \text{ sinon} \end{cases}$$

et

$$y_{ij} = \sum_{k=1}^{m} z_{ijk}\ ,\ i, j = 1, \ldots, n \tag{2.5}$$

$$y_{ij} = \begin{cases} 1 \text{ si les cellules i et j sont affectées au même commutateur} \\ 0 \text{ sinon} \end{cases}$$

A partir de ces définitions, le coût des relèves par unité de temps est donné par :

$$\sum_{i=1}^{n}\sum_{j=1}^{n} H_{ij} y_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} H'_{ij}(1 - y_{ij})$$

Le coût total du réseau à minimiser étant la somme des coûts de câblage et de relèves, il peut alors s'écrire :

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

On peut ignorer le coût des relèves entre cellules affectées au même commutateur devant celui des relèves entre cellules affectées à des commutateurs différents. On pose alors :

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

La relation précédente devient :

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

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

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

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

En négligeant la partie constante, on aboutit à la relation :

$$F = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j=1}^{n} h_{ij}(1-y_{ij}) \qquad (2.6)$$

Nous avons alors un problème de programmation mathématique défini par la relation (2.6) et les contraintes (2.1), (2.2), (2.4), (2.5). La contrainte additionnelle

$$x_{ik} = 0 \text{ ou } 1,\ i = 1, \ldots, n \text{ et } k = 1, \ldots, m \qquad (2.7)$$

est aussi nécessaire. La difficulté dans la formulation donnée ci-dessus est que la contrainte (2.4) n'est pas linéaire. Donc, une solution ne peut être trouvée à partir des techniques classiques de résolution de programmation linéaire. Cependant, il est possible de convertir le problème en un problème de programmation en nombres entiers en linéarisant la contrainte (2.4).

$$z_{ijk} \leq x_{ik} \qquad (2.8)$$

$$z_{ijk} \leq x_{jk} \qquad (2.9)$$

$$z_{ijk} \geq x_{ik} + x_{jk} - 1 \qquad (2.10)$$

$$z_{ijk} \geq 0 \qquad (2.11)$$

Il est facile de vérifier que (2.4) implique (2.8)-(2.11) et vice-versa. La fonction (2.6) et les contraintes (2.1), (2.2), (2.5), (2.7), (2.8)-(2.11) constituent ainsi un programme linéaire entier valide et peuvent être résolues comme tel.

2.1.2 Affectation double

Dans cette version du problème, il y a deux différents moments de la journée où les échantillons de volume d'appels et de relèves sont différents. Les définitions de λ_i et h_{ij} précédentes restent les mêmes, sauf qu'elles ne sont applicables qu'à un moment de la journée. À l'autre moment de la journée, soit λ'_i et h'_{ij} respectivement le volume d'appels par unité de temps et le coût de la relève de la cellule i à la cellule j. Les définitions de M_k et c_{ik} demeurent les mêmes. Cette fois-ci, une cellule peut être affectée à deux commutateurs à la fois, si cela s'avère économique, et la solution de la programmation en nombres entiers devrait l'indiquer. Les deux modèles d'affectation correspondent aux deux volumes d'appels et coûts de relève. Les deux modèles d'affectation sont les résultats de deux affectations simples avec des paramètres différents. Le seul «*risque*» est que si une cellule est affectée au même commutateur dans les deux modèles, son coût du câblage risquerait d'être compté deux fois.

Pour le premier modèle d'affectation, les variables x_{ik}, y_{ij} et z_{ijk} sont définies comme précédemment et doivent respecter les contraintes (2.1), (2.2), (2.5), (2.7), (2.8)-(2.11). Pour le modèle 2, définissons les variables correspondantes x'_{ik}, y'_{ij} et z'_{ijk}. Ces variables doivent respecter les mêmes contraintes, sauf que λ_i doit être remplacé par λ'_i pour tout i. Pour s'assurer que le coût du câblage n'est pas compté deux fois, nous définissons :

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

où le symbole « v » représente l'opérateur « OU ». La fonction à minimiser devient alors :

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

Cette fonction définit un problème de programmation mathématique valide, mais ne constitue pas encore un problème de programmation en nombres entiers à cause de la contrainte (2.12). Comme précédemment, cette contrainte non linéaire sera remplacée par l'ensemble de contraintes linéaires suivantes :

$$w_{ik} \geq x_{ik} \tag{2.14}$$

$$w_{ik} \geq x_{ik} \tag{2.15}$$

$$w_{ik} \leq 1 \tag{2.16}$$

$$w_{ik} \leq x_{ik} + x_{ik} \tag{2.17}$$

Il est facile de vérifier que (2.12) implique (2.14)-(2.17) et vice versa. Cela complète la formulation des problèmes d'affectation simple et double.

2.2 Similitude entre le problème d'affectation de cellules et d'autres problèmes

Le problème d'affectation de cellules à des commutateurs est, de par sa formulation, un problème d'assignation quadratique (Chiang. et al., 1998; Zhou et Gen, 1998; Skorin-Kapov, 1990). Sa résolution par une méthode d'exploration exhaustive peut entraîner une explosion combinatoire, ce qui conduit à une croissance exponentielle du temps d'exécution. Ce problème appartient donc à la classe des problèmes NP-complets qui sont bien connus en recherche opérationnelle. En effet, il s'apparente en particulier aux problèmes de localisation d'entrepôts et de partitionnement de graphes.

Le problème de localisation d'entrepôts ou de concentrateurs peut être considéré comme un ensemble de deux sous-problèmes s'énonçant comme suit (Skorin-Kapov et Skorin-Kapov, 1994 ; O'Kelly, 1987 ; Abdinnour-Helm, 1998): étant donné un ensemble de n positions potentielles (ou nœuds), il s'agit d'abord de localiser p concentrateurs et ensuite d'affecter les $(n - p)$ nœuds restants à ces concentrateurs en respectant certaines contraintes. Ce problème très complexe est bien connu dans le domaine de l'optimisation et plusieurs heuristiques ont été mises en œuvre pour sa résolution. La similitude de ce problème avec celui de l'affectation des cellules à des commutateurs provient du fait que les p nœuds concentrateurs peuvent être assimilés à des commutateurs et les n-p restants à des cellules. La seule différence est que dans le cas du problème d'affectation, la localisation potentielle des commutateurs est connue d'avance.

Le problème d'affectation de cellules à des commutateurs s'apparente aussi à un problème de partitionnement de graphe. En effet, un réseau est souvent modélisé comme un graphe dans lequel les nœuds représentent des terminaux et les arcs des liaisons entre

les différents terminaux. La façon la plus simple de construire un réseau comprenant n nœuds est de les relier 2 à 2. Cette méthode, quoique très fiable, est très coûteuse et difficile à implémenter. L'une des méthodes utilisées est alors le partitionnement des graphes qui consiste à répartir les nœuds d'un graphe en différents lots appelés dans ce cas sous-graphes. Cette répartition s'effectue suivant un ou des critères donnés. Les sous-graphes sont formés généralement de façon à minimiser les liaisons entre eux tout en assurant la fiabilité et la robustesse du réseau. Le plus souvent, on choisit un nœud servant de serveur parmi les nœuds du sous-graphe (Merchant et Sengupta, 1994). Parfois, en plus de la constitution des partitions, on minimise aussi les longueurs des liaisons entre les nœuds d'un sous-graphe et leur serveur (Guttmann-Reck et Hassin, 1998).

Les problèmes exposés ci-dessus et équivalents à celui de l'affectation des cellules à des commutateurs demeurent assez complexes et sont eux aussi NP-complets. Ainsi, seules des méthodes heuristiques tendant à trouver des solutions optimales ou proches de l'optimum sont recommandées. Parmi les heuristiques couramment appliquées à ces problèmes, nous avons le recuit simulé (Kouvelis et al., 1992), la recherche taboue (Sun et al., 1995) et bien sûr les algorithmes génétiques (AG) que nous proposons dans ce mémoire.

2.3 Méthodes traditionnelles de résolution

Même si plusieurs méthodes ont été développées pour résoudre les différents problèmes qui s'y apparentent, très peu de méthodes existent spécifiquement pour la résolution du problème d'affectation des cellules aux commutateurs. Dans cette section, nous exposerons celles qui concernent directement l'affectation des cellules aux commutateurs.

2.3.1 Méthode de Merchant et Sengupta

Merchant et Sengupta (1995) ont proposé un algorithme qui consiste à trouver une solution initiale qu'on tente d'améliorer progressivement jusqu'à ce qu'aucune

amélioration ne soit plus possible. Cet algorithme intègre les trois grandes étapes suivantes : le choix de la solution initiale, l'affectation simple et l'affectation double.

1. Choix de la solution initiale

La faisabilité d'une solution dépend de la satisfaction de la contrainte sur la capacité des commutateurs. Ainsi, une affectation est dite faisable si elle satisfait la contrainte de capacité du commutateur.

Pour choisir la solution initiale, on ordonne les cellules dans l'ordre décroissant des volumes d'appel. Cette partie comporte n étapes. À la $k^{ième}$ étape ($k \leq n$), l'affectation des k-1 premières cellules est réalisée. On choisit alors au plus les b meilleures solutions faisables résultant de l'affectation de la $k^{ième}$ cellule. L'optimisation est ainsi faite progressivement et à la fin des n étapes, on obtient les b meilleures solutions faisables. La solution initiale est alors la meilleure des b solutions retenues, c'est-à-dire celle dont le coût total est le plus bas (b est choisi aléatoirement, par exemple 10).

2. Affectation simple

- On recherche une solution initiale.

- On réalise les meilleurs déplacements possibles (c'est-à-dire qu'on choisit la cellule i et le commutateur k tel que le déplacement de i vers k soit celui qui réduise le plus le coût total) successivement jusqu'à ce qu'aucun déplacement ne réduise le coût total.

Après chaque étape, la cellule pour laquelle on vient de changer de commutateur est fixée et on conserve la topologie obtenue.

- À la fin la solution retenue est celle qui présente le coût le plus bas.

3. Affectation double

Pour obtenir un schéma d'affectation double, deux affectations simples sont réalisées : la première tenant compte des données λ_i, h_{ij}, M_k, c_{ik}, et la seconde tenant

compte des données λ'_i, h'_{ij}, M_k, c_{ik}. La particularité introduite par cette heuristique est que la solution initiale de la seconde affectation est la solution finale trouvée par la première. S'il s'avère qu'une cellule est affectée au même commutateur que dans la première affectation, le coût du câblage n'est pas ajouté.

Cet algorithme a été implémenté et les résultats obtenus ont été comparés avec ceux obtenus par une méthode de programmation mathématique. Il en ressort que pour des problèmes de petite taille (15 ou 30 noeuds), la méthode de programmation mathématique donne de meilleurs résultats que l'heuristique proposée car elle aboutit à des solutions optimales en un temps relativement court. Néanmoins, l'heuristique proposée est applicable à des réseaux de différentes tailles et les solutions obtenues sont très satisfaisantes. Pour pallier à la grande variance entre les charges des commutateurs, des variantes de cet algorithme permettent de trouver des solutions qui satisfont à la contrainte de répartition équitable des charges.

Merchant et Sengupta (1994) ont également présenté une heuristique fondée sur l'algorithme de partitionnement d'un graphe en deux sous-graphes pour résoudre le problème d'affectation de cellules aux commutateurs. Les grandes lignes de cet algorithme sont décrites ci-après :

Soit un graphe comprenant n nœuds à partitionner en deux partitions S_1 et S_2 de tailles maximales respectives n_1 et n_2.

1. *Ajouter $n_1 + n_2 - n$ nœuds factices sans arcs incidents au graphe.*
2. *Diviser les nœuds arbitrairement en deux partitions S_1 et S_2 de taille respectives n_1 et n_2.*
3. *Répéter les meilleurs déplacements sur le partitionnement initial jusqu'à ce qu'aucun déplacement ne réduise le coût total $W_2(S_1,S_2)$ des arcs de liaison entre les deux partitions obtenues. À chaque déplacement :*
   a. *Trouver les nœuds $v_1 \in S_1$ et $v_2 \in S_2$ tel que déplacer v_1 vers S_2 et v_2 vers S_1 réduise le plus possible $W_2(S_1,S_2)$.*

b. *Échanger les deux nœuds v_1 et v_2 et les marquer temporairement, ce qui les empêchera d'être déplacés à nouveau avant la fin des déplacements. Noter la nouvelle partition.*

c. *Répéter (a) et (b) jusqu'à ce que tous les nœuds aient été marqués. Dans toute la séquence de partitions ainsi obtenue, trouver la paire ayant un coût $W_2(S_1,S_2)$ minimum. Cela achève les déplacements.*

4. *Enlever tous les nœuds factices.*

Le problème d'affectation de cellules est un peu plus complexe que l'algorithme décrit ci-dessus. D'abord, il requiert un partitionnement en plusieurs sous-graphes. Ensuite, le volume d'appels n'est pas représenté correctement car l'algorithme précédent considère que tous les nœuds ont le même volume d'appels. Enfin, le coût de la liaison physique entre la station de base et le commutateur a été ignoré.

Des solutions ont été apportées à ces différents problèmes et un algorithme plus adapté à la résolution du problème d'affectation de cellules à des commutateurs a été proposé par Merchant et Sengupta (1994). Les grandes étapes en sont les suivantes :

1. *Représenter chaque cellule i avec son volume d'appel λ_i par un nœud principal et $\lfloor K\lambda_i \rfloor$-1 nœuds auxiliaires interconnectés par des liens de coûts élevés (le facteur d'expansion K est choisi arbitrairement). Pour chaque paire de nœuds, ajouter une liaison de coût $h_{ij} + h_{ij}$ entre les nœuds primaires. Le commutateur k est représenté par un nœud fixe comme étant une partition P_k de taille $\lfloor KM_k \rfloor + 1$. Pour représenter le coût du câblage, on ajoute un arc entre le nœud principal de chaque cellule et le nœud représentant le commutateur k de coût* $\sum_{j=1}^{m} C_{ij} \Big/ (m-1) - C_{ik}$.

2. *Cela entraîne une décomposition en m partitions d'un graphe de taille* $m + \sum_{i=1}^{n} K\lambda_i$ *nœuds, sous contrainte de la taille d'une partition*

$1+\lfloor KM_k \rfloor$ pour k de 1 à m. La fonction à minimiser, $W_m(P_1, P_2, \ldots, P_m)$ est la somme des coûts des arcs assurant la liaison entre les partitions. Diviser successivement le (sous)-graphe obtenu par $V_k \equiv (P_1 \cup P_2 \cup \ldots \cup P_k)$ en (V_{k-1}, P_k), pour k = m, m-1, ..., 2, en utilisant l'algorithme exposé ci-dessus pour minimiser $W_2(V_{k-1}, P_k)$.[$V_m = V$ et $V_1 = P_1$]

3. *Si, pour une cellule i, les nœuds correspondants ne sont pas assignés à la même partition, alors l'algorithme a échoué. Sinon, assigner la cellule i au commutateur k.*

Les résultats obtenus montrent cependant que cet algorithme est inefficace et les auteurs ont déconseillé son utilisation.

Sanchis (1989) a présenté une adaptation du concept de niveau de gain décrit par Krishnamurthy (1984) et a exposé une autre méthode d'amélioration progressive des partitions obtenues. Après avoir choisi une partition de base, on l'optimise de façon uniforme de la manière suivante :

À chaque étape, on considère tous les déplacements possibles d'une cellule de sa partition vers toutes les autres partitions, et on choisit celui de meilleur coût. Comme précédemment, on réalise les déplacements jusqu'à ce qu'aucun déplacement ne minimise le coût des liaisons entre partitions.

Les résultats obtenus confirment l'efficacité de l'algorithme. En effet, il a été montré que l'algorithme de partitionnement uniforme est capable de trouver les coûts minima avec un taux d'amélioration variant selon les réseaux.

2.3.2 Méthode basée sur la recherche taboue

La méthode de la recherche taboue est une technique adaptative introduite en optimisation combinatoire par Glover(1989) pour résoudre les problèmes difficiles.

Cette méthode est une amélioration de l'algorithme général de descente en ce sens qu'elle essaie d'éviter le piège des minima locaux. Pour cela, elle accepte de temps en temps des solutions qui n'améliorent pas la fonction objectif, en espérant ainsi parvenir à de meilleures solutions. Cependant, le fait de vouloir accepter des solutions pas toujours meilleures introduit un risque de cycle, autrement dit un retour vers des solutions déjà visitées. D'où l'idée de conserver une liste des k dernières solutions visitées; cette liste est appelée *liste taboue*. Cette stratégie permet ainsi d'éviter des cycles de longueur inférieure ou égale à k. L'algorithme s'arrête quand aucune amélioration n'est intervenue depuis un nombre *kmax* d'itérations, ou quand toutes les solutions voisines candidates sont taboues.

Pierre et Houéto (2000) ont adapté la méthode de recherche taboue à la résolution du problème d'affectation de cellules aux commutateurs. Selon l'adaptation proposée, l'espace de recherche est libre des contraintes sur la capacité des 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. À chaque solution sont associées 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 non-respect des contraintes de capacité. Cette sanction comprend une partie fixe et une partie variable par unité de violation. À chaque étape, la solution ayant la meilleure évaluation est retenue. Une fois qu'une solution initiale est construite à partir des données du problème, la composante de mémoire à court terme essaie d'améliorer itérativement cette solution tout en évitant les cycles. Une composante de mémoire à moyen terme cherche à intensifier la recherche dans des voisinages précis, tandis que la composante de mémoire à long terme vise à diversifier l'exploration du domaine.

- **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 avec une solution initiale obtenue tout simplement en affectant chaque cellule au commutateur le plus proche selon une métrique de distance euclidienne. Le but de cette mémoire est d'améliorer la solution courante, soit en diminuant son coût, soit en diminuant les pénalités encourues.

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 :

$$G_s(a,b)=\begin{cases} \sum_{i=1,i\neq a}^{n}(h_{ai}+h_{ia})x_{ib_0}-\sum_{i=1,i\neq a}^{n}(h_{ai}+h_{ia})x_{ib}+c_{ab}-c_{ab_0} & \text{si } b\neq b_0\ ; \\ M & \text{sinon.} \end{cases} \tag{2.18}$$

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, le mouvement ayant le gain le plus faible parmi tous les mouvements possibles est sélectionné. 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 :

$$f(S') = f(S) + G_S(a,b) \tag{2.19}$$

Au début, on génère tous les gains $G_S(i,k)$ dans un tableau $n \times m$, où n désigne le nombre de cellules et m le nombre de commutateurs. 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 :

$$G_{S'}(p,q) = G_S(p,q) \qquad \text{si } C_p \neq b',\ C_p \neq b,\ q \neq b',\ q \neq b \tag{2.20}$$

$$G_{S'}(p,q) = M \qquad \text{si } C_p = q \tag{2.21}$$

$$G_{S'}(p,q) = G_S(p,q) - 2(h_{ap} + h_{pa}) \qquad \text{si } C_p = b',\ q = b \tag{2.22}$$

$$G_{S'}(p,q) = G_S(p,q) - (h_{ap} + h_{pa}) \qquad \text{si } C_p = b',\ q \neq b',\ q \neq b \tag{2.23}$$

$$G_{S'}(p,q) = G_S(p,q) + 2(h_{ap} + h_{pa}) \qquad \text{si } C_p = b,\ p \neq a,\ q = b' \tag{2.24}$$

$$G_{S'}(p,q) = G_S(p,q) + (h_{ap} + h_{pa}) \qquad \text{si } C_p = b,\ q \neq b',\ q \neq b \tag{2.25}$$

$$G_{S'}(a,b') = -G_S(a,b) \tag{2.26}$$

$$G_{S'}(a,q) = G_S(a,q) - G_S(a,b) \qquad \text{si } q \neq b',\ q \neq b \tag{2.27}$$

À chaque itération, on définit un voisinage de la solution courante composée des solutions engendrées par tous les mouvements $m(a,b)$, puis on choisit le mouvement qui améliore le plus le coût de la solution, c'est-à-dire celui ayant le gain le plus faible. 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 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.

Intuitivement, 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 donné 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 et d'une partie variable, au gain des mouvements qui ajoutent une cellule à un commutateur ayant déjà une capacité résiduelle négative. À chaque solution non faisable, ce multiplicateur est incrémenté jusqu'à une certaine valeur maximale. 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.

Quant à la liste taboue, elle 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 est donc annulé si ce dernier conduit à une solution dont l'évaluation est inférieure à celle de la meilleure solution actuellement connue. Pour avoir de meilleurs résultats, on ajoute à la composante de mémoire à court terme, une composante à moyen terme.

- **Mémoire à moyen terme**

La composante de mémoire à moyen terme 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.

Définir des régions prometteuses pour y intensifier la recherche n'est pas chose aisée. Nous avons choisi de conserver dans une liste les dernières meilleures solutions, ainsi que les valeurs associées des gains permettant ainsi 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^* est substituée à la plus ancienne meilleure solution de la liste. 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 dont l'idée 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 :

- $i_1(a,b)$: *permutation des cellules a et b 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.*

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 *kmax* dernières itérations ou s'il n'y a plus de mouvements disponibles. On peut alors enchaîner avec des mécanismes de diversification.

- **Mémoire à long terme**

Pour diversifier la recherche, nous utilisons une structure de mémoire à long terme visant à orienter la recherche vers 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.

Des expériences visant à tester les différents paramètres mis en jeu par chacune des composantes ont été effectuées (Houéto et Pierre, 2000). Des résultats obtenus, il ressort que :

- plus la taille de la liste est grande, plus l'algorithme tend à obtenir de meilleurs résultats ;
- un délai de déclenchement de deux solutions consécutives non faisables donne de meilleurs résultats que des délais de 5 ou 8 solutions ;
- l'écart entre les résultats obtenus pour différents nombres de redémarrages n'est pas grand.

L'adaptation de la recherche taboue réalisée par Houéto et Pierre (2000) a été comparée avec l'heuristique mise au point par Merchant et Sengupta (1995). Cette comparaison révèle que, pour un grand nombre de cellules, cette adaptation est plus rapide que l'heuristique de Merchant et Sengupta, mais un peu plus lente quand le nombre de cellules est faible (moins de 50).

CHAPITRE III
ALGORITHMES GÉNÉTIQUES

Ce chapitre présente les algorithmes génétiques en tant que méthode de résolution de problèmes. Nous en exposerons d'abord les principes de base. Nous caractériserons ensuite ces algorithmes qui reposent, entre autres, sur les notions d'opérateurs génétiques et de gabarits de similitude. Après une explication approfondie des mécanismes de base, nous aborderons les stratégies classiques permettant d'accroître l'efficacité des différents opérateurs. Nous mentionnerons enfin deux exemples d'application de ces algorithmes dans le domaine des télécommunications.

3.1 Principes des algorithmes génétiques

Les algorithmes génétiques (AG) introduits par John Holland (1975) et ses étudiants à l'Université de Michigan sont fondés sur la théorie de la survie des espèces de Charles Darwin. Comme dans la nature où les êtres se reproduisent, dans le modèle des algorithmes génétiques, les spécimens se reproduiront aussi; en particulier ceux jugés les plus forts se reproduiront à un rythme plus rapide. Des opérateurs génétiques seront appliqués sur des candidats en espérant engendrer ainsi de nouveaux candidats plus performants.

En biologie, on manipule des gènes et des chromosomes; il en va de même dans le modèle des AG. Les problèmes et les solutions seront encodés. L'encodage prend souvent la forme d'une *chaîne de bits*. Ces chaînes de bits sont comparables aux *chromosomes* des systèmes biologiques, tandis que les bits ou caractères qui composent ces chaînes sont comparables aux *gènes*. L'ensemble de ces chaînes forme une *population*, alors qu'en biologie on parle de *génotype*.

Dans la recherche de solutions à un problème, les AG utilisent une grande part de hasard. En effet, les candidats à la reproduction sont choisis de façon probabiliste; les chromosomes de la population sont croisés de façon aléatoire dans la progéniture; et les gènes d'un chromosome sont mutés selon une certaine probabilité. En appliquant ainsi

de génération en génération les opérateurs génétiques sur des candidats jugés performants, on cherche à obtenir une progéniture plus performante que celle de la génération précédente, ce qui permet de s'approcher ainsi d'une solution optimale.

3.2 Caractéristiques des algorithmes génétiques

Les algorithmes génétiques se caractérisent par quatre aspects : le codage des paramètres du problème, l'espace de recherche, la fonction d'évaluation servant à sélectionner les chromosomes parents, et le hasard qui joue un rôle important dans l'évolution des chromosomes de génération en génération. Nous allons passer en revue ces différents aspects.

3.2.1 Le codage et l'espace de recherche des solutions

La première particularité des AG réside dans le fait qu'ils n'agissent pas directement sur l'espace des solutions : les solutions sont codées sous la forme de chaînes de longueur finie à partir d'un alphabet fini (Pirlot, 1996). C'est grâce aux codes et aux similitudes de ces codes que l'on arrive à trouver une solution satisfaisante à un problème et à comprendre pourquoi cette solution fonctionne. De génération en génération, les chromosomes qui forment la population ont une valeur d'aptitude de plus en plus élevée, et ont des gènes de même valeur aux mêmes positions dans la chaîne.

La plupart des méthodes d'optimisation effectuent une recherche point à point. Les règles de transition d'un point à un autre sont souvent déterministes et la solution trouvée est souvent un optimum local au lieu d'être un optimum global. Les AG, quant à eux, effectuent la recherche à partir d'une population de chaînes générées aléatoirement. Dans cette population, on retrouvera à la fois des candidats très performants et d'autres qui le sont moins. Le parallélisme induit est un avantage évident car l'approche de la recherche à partir d'une population peut être perçue comme une recherche locale dans un sens généralisé. Ce n'est pas le voisinage d'une seule solution qui est explorée, mais le voisinage de toute la population; ce qui ne devrait pas être assimilé à une simple union des voisinages individuels (Pirlot, 1996). Ainsi donc, une population initiale

diversifiée offre plus de chances de bien cerner la recherche et de mieux se rapprocher de la solution optimale, sinon on risque d'obtenir des espèces dégénérées et la probabilité de converger vers un minimum global est ainsi fortement réduite.

3.2.2 La fonction d'évaluation et le hasard

Contrairement à bon nombre de méthodes qui requièrent beaucoup d'informations pour pouvoir fonctionner efficacement, les AG nécessitent peu d'informations : ils fonctionnent essentiellement de manière aveugle. Pour effectuer une recherche de solutions meilleures, ils n'ont besoin que des valeurs des fonctions objectives associées aux chaînes individuelles. Ces valeurs sont déterminées par une fonction objectif et serviront au processus de sélection des candidats aptes à la reproduction et au processus de survie des espèces.

Par ailleurs, les AG utilisent des règles de transition probabilistes plutôt que déterministes pour guider leur recherche. Le choix des chromosomes à perturber est réalisé de façon probabiliste. Dans le processus de croisement, le lieu de croisement est choisi aléatoirement à l'intérieur du chromosome. De même, le gène devant subir une mutation à l'intérieur d'un chromosome est choisi selon une certaine probabilité. Le hasard occupe donc une place importante dans le fonctionnement des AG.

3.3 Mécanismes de base

Les mécanismes usuels sur lesquels repose la méthode des algorithmes génétiques sont principalement la représentation des chromosomes et les opérateurs génétiques. Un bon choix des paramètres de ces chromosomes augure de la convergence vers une bonne solution.

3.3.1 Représentation des chromosomes

L'algorithme génétique de base repose sur la représentation *binaire* des chromosomes. Ce choix le rend intuitivement applicable à tous les problèmes dont les solutions sont transposables en binaire. Les chromosomes sont alors représentés par une

chaîne de bits. Cette représentation est indépendante du problème traité et rend l'algorithme génétique d'autant plus robuste. Cependant, elle demande un effort supplémentaire : l'espace des solutions potentielles doit être transposé dans un espace de solutions binaires en entrée de l'algorithme, et les solutions obtenues en sortie doivent être converties en solutions réelles pour pouvoir être interprétées. La Figure 3.1 montre un chromosome en représentation binaire.

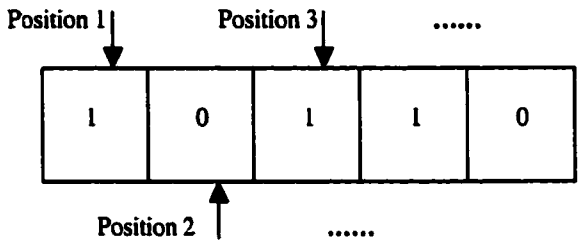

Figure 3.1 Représentation binaire d'un chromosome

La lecture d'un chromosome se fait de gauche à droite. Le gène le plus à gauche occupe la première position : c'est le bit 1. Le gène qui le suit immédiatement occupe la deuxième position, et ainsi de suite.

Les chromosomes peuvent être aussi représentés à partir d'un alphabet plus élargi. Ainsi, dans le cas du problème du voyageur de commerce (Chatterjee et al., 1996), ou celui d'arbre de recouvrement minimum (Zhou et Gen, 1998), ils sont représentés respectivement par un alphabet comprenant autant d'éléments que de villes à visiter ou de nœuds dans le réseau.

3.3.2 Les opérateurs génétiques

Trois mécanismes composent essentiellement les opérateurs génétiques : la sélection, le croisement et la mutation. Ces opérateurs se retrouvent dans la littérature sous plusieurs variantes.

La sélection

La sélection est le processus selon lequel des chaînes de la population sont choisies pour une nouvelle génération suivant leur valeur par la fonction objectif. Plus la valeur de la fonction objectif est élevée, plus cette chaîne a des chances d'être sélectionnée. Une partie des gènes des chromosomes choisis contribuera à une solution-chromosome dans les générations futures. Une des techniques les plus utilisées pour réaliser la sélection est celle de la roulette de casino. Selon cette technique, la probabilité d'être choisie est directement liée à la valeur d'aptitude du parent. Le Tableau 3.1 illustre le principe de fonctionnement de la roulette : d'abord on calcule la valeur d'aptitude de chaque chromosome, puis on calcule l'aptitude totale en faisant la somme des valeurs d'aptitude de chaque individu de la population; enfin, on calcule le pourcentage de l'aptitude de chaque chromosome par rapport à l'aptitude totale.

Tableau 3.1 Exemple de sélection selon le principe de la roulette

Chromosome	1	2	3	4	5	Total
Chaîne	01110	01010	10111	00101	11100	
Aptitude	14	10	23	5	28	80
% du total	17,5	12,5	28,75	6,25	35	100,0

Comme le montre la Figure 3.2, les quartiers de la roulette sont proportionnels à la valeur d'aptitude de chaque chromosome. Nous remarquons ainsi qu'en faisant tourner la roulette, nous avons plus de chances de choisir les chromosomes 3 et 5.

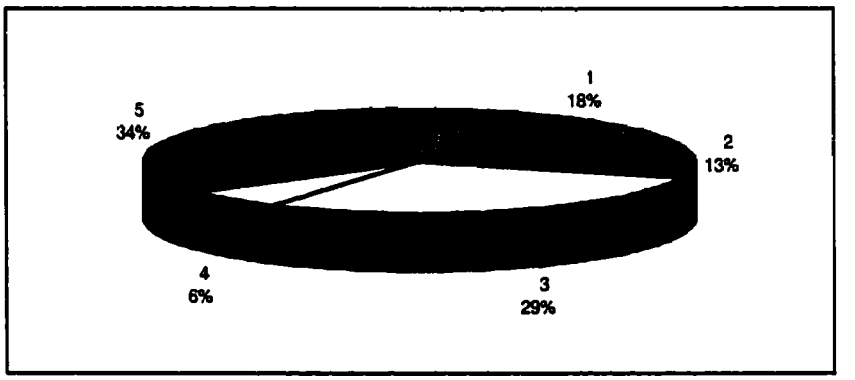

Figure 3.2 Roulette avec « slots » proportionnels à l'aptitude

Le croisement

Le croisement est le processus selon lequel les bits de deux chaînes sélectionnées sont interchangées : dans le langage génétique, on dira que ces chaînes sont croisées. Pour exécuter le croisement, des chaînes de la population sont accouplées au hasard. Chaque paire de longueur t subit le croisement comme suit : une position entière k est choisie uniformément entre 1 et (t-1). Deux nouvelles chaînes sont créées en échangeant tous les gènes entre les positions (k+1) et t de chaque paire considérée. Les nouvelles chaînes peuvent être totalement différentes de leurs parents. Il faut toutefois remarquer que le croisement n'aura aucun effet sur un gène dont les parents ont la même valeur à la même position.

Soient les chromosomes A_1 et A_2 définis comme suit :

$$A_1 = 0\,1\,0\,1\downarrow 1\,0\,1$$

$$A_2 = 1\,1\,1\,0\downarrow 1\,0\,0$$

Les deux chaînes ont une longueur t = 7 et pour alphabet {0,1}. Le symbole ↓ représente l'endroit à partir duquel aura lieu le croisement, appelé lieu de croisement. Après le croisement, nous obtenons les deux chaînes suivantes :

$$A'_1 = 0\,1\,0\,1\downarrow 1\,0\,0$$

$$A'_2 = 1\,1\,1\,0\downarrow 1\,0\,1$$

Comme nous pouvons le constater, la valeur des bits des positions 5 et 6 n'a nullement changé. Nous pouvons aussi remarquer que, dans ce cas-ci, le croisement a très légèrement affecté les chaînes initiales. Tout comme pour le croisement sexuel, les candidats devraient être assez dissemblables; en effet, une trop grande similarité dans le bagage génétique peut entraîner l'appauvrissement de l'espèce (Davis, 1991).

On retrouve aussi des AG avec plusieurs lieux de croisement (Nguyen et Moon, 1996). Ce sont alors les gènes entre les lieux de croisement qui sont interchangés. Syswerda (1989) a également introduit le concept de croisement uniforme : pour chaque bit des deux enfants, on choisit aléatoirement à chaque position le parent qui affectera sa valeur de bit à l'enfant.

La mutation

La mutation est le processus selon lequel la valeur d'un gène choisi au hasard dans un chromosome est régénérée. C'est un processus qui ne survient qu'occasionnellement dans un algorithme génétique. En modifiant aléatoirement la valeur d'une position dans une chaîne, la mutation est utile pour ramener du matériel génétique qui aurait été oublié par les opérateurs de sélection et de croisement. Certaines implantations vont automatiquement changer la valeur du bit choisi (Goldberg, 1989), d'autres vont générer aléatoirement la nouvelle valeur du bit, ce qui n'entraîne à l'occasion aucune mutation (Davis, 1991).

Soit la chaîne :

$$A = 0\,1\,\mathbf{0}\,1\,1\,0\,1$$

Après mutation, nous pouvons obtenir la chaîne :

$$A' = 0\,1\,\mathbf{1}\,1\,1\,0\,0$$

La valeur du troisième bit a donc été modifiée.

3.3.3 Les grandes lignes d'un algorithme génétique

De façon générale, un AG fonctionne de la façon suivante :

Étape 1: On génère une population initiale de taille n représentant le nombre de chromosomes, puis on choisit au hasard les gènes qui composent chaque chromosome: c'est la première génération de chromosomes.

Étape 2: On évalue chaque chromosome par la fonction objectif, ce qui permet de déduire sa valeur d'aptitude.

Étape 3: Commence alors le cycle de génération des populations, chaque nouvelle remplaçant la précédente. Le nombre x de générations est déterminé au départ. Dans chaque génération, n chromosomes sont choisis pour se voir appliquer les différents opérateurs génétiques. Après chaque génération, les n nouveaux chromosomes créés remplacent la génération précédente. Après la création de la $x^{ième}$ génération, les chromosomes auront évolué de telle façon que cette dernière génération contienne des chromosomes meilleurs à ceux des générations précédentes.

3.4 Mécanismes avancés

Les algorithmes génétiques avancés surclassent les algorithmes de base en termes de temps de convergence vers une solution. Ils emploient à cet effet des mécanismes reposant essentiellement sur la représentation réelle des chromosomes.

3.4.1 Représentation réelle des chromosomes

Les nombres binaires étant pour nous moins évocateurs que les nombres réels, des difficultés surviennent pour exprimer la fonction objectif et traiter les problèmes à plusieurs variables. En outre, les opérations de conversion des solutions potentielles (réelles) en chaînes de bits et des solutions obtenues en une forme réelle facilitant leur interprétation sont coûteuses en temps-machine. De plus, elles sont répétées un grand nombre de fois à chaque génération. La représentation réelle propose un compromis intéressant : elle élimine toutes les opérations de conversion, mais en contrepartie elle rend les algorithmes génétiques avancés plus dépendants des problèmes. Dans ce type

de représentation, un chromosome est un *n*-uplet de nombres réels, chacune des composantes correspondant alors à une inconnue du problème.

3.4.2 Stratégies d'accroissement de l'efficacité des opérateurs

Jusqu'à présent, tous les chromosomes générés par les opérateurs génétiques étaient directement insérés dans la population de la génération suivante. C'est en effet la stratégie de l'algorithme de base. Il existe cependant d'autres alternatives qui contribuent à réduire le temps de convergence de l'algorithme génétique avancé. Parmi elles, mentionnons la stratégie de l'élitisme, et celle de la population sans doubles.

La stratégie de l'élitisme consiste à recopier le meilleur chromosome de la population actuelle dans la population de la génération suivante, et de compléter par d'autres chromosomes générés de manière traditionnelle, jusqu'à obtenir le nombre de chromosomes nécessaires. Il devient alors impossible pour les générations suivantes que leur meilleur chromosome soit inférieur à celui des générations précédentes.

La stratégie de la population sans doubles consiste à n'insérer le chromosome généré dans la population suivante que s'il est différent de tous les chromosomes présents dans celle-ci. Des chromosomes différents composent ainsi la population obtenue. Il devient alors impossible d'appliquer l'opérateur de croisement à une paire de chromosomes identiques (cas défavorable n'entraînant aucune amélioration dans la population suivante).

3.4.3 Opérateurs génétiques réels

A la différence des algorithmes de base, les algorithmes génétiques avancés offrent plusieurs possibilités concernant l'utilisation de leurs opérateurs :

- les opérateurs sont considérés comme « équiprobables » ;
- les opérateurs ont des poids différents mais constants au cours des générations ;
- les opérateurs ont des poids différents et variant au cours des générations.

Il en résulte une nouvelle classe d'opérateurs qui sont : le croisement réel, le croisement arithmétique, la mutation réelle et la mutation uniforme.

Le croisement réel ne se différencie du croisement binaire que par la nature des éléments qu'elle altère : ce ne sont plus les bits qui sont échangés à droite du lieu de croisement, mais des variables réelles.

Le croisement arithmétique est propre à la représentation réelle. Il s'applique à une paire de chromosomes et se résume à une moyenne pondérée des variables des deux parents :

Soient $[a_i, b_i, c_i]$ et $[a_j, b_j, c_j]$ deux parents, et un poids p appartenant à l'intervalle [0, 1]. Les enfants sont donc : $[pa_i + (1-p)a_j, pb_i + (1-p)b_j, pc_i + (1-p)c_j]$ et $[(1-p)a_i + a_j, (1-p)b_i + b_j, (1-p)c_i + c_j]$

Le mécanisme de la mutation réelle est semblable à celui de la mutation binaire. Mais ici, ce n'est plus un bit qui est inversé, mais une variable réelle qui est de nouveau tirée au hasard sur son intervalle de définition et qui remplace l'ancienne valeur.

La mutation uniforme possède la particularité de retirer les éléments qu'elle altère dans des intervalles de plus en plus petits. Plus nous avançons dans les générations, moins la mutation écarte les éléments de la zone de convergence. Cette mutation adaptative offre un bon équilibre entre l'exploration du domaine de recherche et un raffinement des individus. Le coefficient d'atténuation de l'intervalle est un paramètre de cet opérateur.

On peut se demander ce qu'ont en commun les chaînes dont la valeur par la fonction objectif est élevée. Remarque-t-on certaines valeurs particulières à des positions particulières dans ces chaînes? La recherche d'une solution optimale est guidée par l'exploitation des similitudes dans les chaînes.

3.5 Gabarits de similitude

Un gabarit de similitude est un patron commun à plusieurs chaînes et qui permet de guider la recherche vers les espaces prometteurs. Il est principalement lié à la notion de schéma et est exploité au niveau des opérateurs génétiques.

3.5.1 Le schéma

Tous les chercheurs dans le domaine des AG s'accordent à dire que le succès de ces algorithmes est dû à la prolifération progressive d'un bon *schéma* dans la population. Un *schéma* est un gabarit de similitude décrivant un sous-ensemble de chaînes pour lesquelles on retrouve des gènes de même valeur à des positions particulières dans la chaîne. En enrichissant l'alphabet de base par le symbole spécial * ou « peu importe », le schéma devient un mécanisme de «pattern matching». Soit l'alphabet des schémas V = {0,1,*} avec des chaînes de longueur 5, le schéma *110* décrit un sous-ensemble de 4 éléments. Ce sous-ensemble est {01100, 01101, 11100, 11101}.

Proposition 3.1

En général, pour des alphabets de cardinalité k avec des chaînes de longueur t, il y a $(k+1)^t$ schémas, alors qu'il n'y a que k^t chaînes différentes. Chaque chaîne est donc un élément de 2^t schémas.

Preuve: Soit le schéma $(b_1, b_2, b_3, \ldots, b_{t-1}, b_t)$; b_i, avec $i = 1, \ldots, t$, peut prendre k valeurs différentes + 1 (le symbole spécial *). D'où $((k+1) \times (k+1) \times (k+1) \ldots (k+1) \times (k+1))$ ou $(k+1)^t$ schémas différents.

Soit le schéma $(b_1, b_2, b_3, \ldots, b_{t-1}, b_t)$; b_i, avec $i = 1, \ldots, t$, peut prendre k valeurs différentes + 1 (les chaînes ne sont jamais constituées du symbole *). D'où $(k \times k \times k \ldots k \times k)$ ou k^t chaînes différentes.

Soit le schéma $(b_1, b_2, b_3, \ldots, b_{t-1}, b_t)$; b_i, avec $i = 1, \ldots, t$, peut prendre sa valeur actuelle ou le symbole *. D'où $(2 \times 2 \times 2 \ldots 2 \times 2)$ ou 2^t. Chaque chaîne appartient à 2^t schémas différents.

La chaîne $A = 01010$ est élément du schéma $H_1 = *1010$, du schéma $H_2 = 0*010$, du schéma $H_3 = **010$, et ainsi de suite. En fait, dans cet exemple, il y a exactement 32 schémas différents pour cette chaîne.

John Holland (1975) nomme les schémas des *blocs de construction.* Tel un enfant qui construit des châteaux avec des blocs, les AG construisent la solution sub-optimale à partir des schémas courts et d'aptitude élevée contenus dans la population. Nous allons donc analyser la croissance et la décroissance des schémas dans la génération. Tout d'abord, définissons deux propriétés qui seront utiles plus tard :

- l'ordre d'un schéma H, noté $o(H)$, est le nombre de positions fixes présentes dans le gabarit, ainsi o(111**1*) = 4.
- La longueur d'un schéma H, notée $\delta(H)$, est la distance entre la première et la dernière position fixe dans le schéma; ainsi δ(111**1*) = 5 puisque la première position fixe est 1 et la dernière 6.

Les schémas et leurs propriétés constituent l'outil de base pour l'analyse de l'effet de la sélection, du croisement et de la mutation sur les membres d'une population.

3.5.2 La sélection et les schémas

Le théorème des schémas de Holland (1975) s'énonce comme suit :

> *«Si on utilise une technique de sélection proportionnelle à l'aptitude de chaque chromosome, alors on peut prédire la croissance ou la décroissance relative d'un schéma H dans la prochaine génération».*

Soit $f(H)$ l'aptitude moyenne d'une chaîne représentant le schéma H. Soit $m(H)$ le nombre de chromosomes contenant le schéma H dans la population.

Soit $\bar{f} = \sum f_j / n$ l'aptitude moyenne de toute la population, n étant sa taille.
Alors, le nombre prévu d'occurrences de H dans la prochaine génération est égal à $m(H) \times f(H)/f$. Ainsi, les schémas ayant des valeurs d'aptitude supérieures à la moyenne de la population auront un nombre croissant d'échantillons dans la génération suivante, alors que les schémas avec des valeurs inférieures à la moyenne de la population recevront des échantillons en nombre décroissant. Tous les schémas de la population croissent ou décroissent selon les moyennes des schémas avec l'opérateur de sélection.

3.5.3 Le croisement et les schémas

Soient le chromosome A et les deux schémas H_1 et H_2 :

$$A = 0\,1\,0\,1\,0\,0\,1$$

$$H_1 = *\,1\,*\,*\,*\,*\,1$$

$$H_2 = *\,*\,*\,1\,0\,*\,*$$

Si la chaîne A a été choisie pour un croisement, et que le lieu de croisement se situe à la position 3, le croisement aura un effet différent sur les deux schémas.

$$A = 0\,1\,0 \downarrow 1\,0\,0\,1$$

$$H_1 = *\,1\,* \downarrow *\,*\,*\,1$$

$$H_2 = *\,*\,* \downarrow 1\,0\,*\,*$$

En effet, le schéma 1 sera détruit car le 1 de la position 1 et celui de la position 7 seront placés dans des progénitures différentes. Par contre, on peut voir que H_2 survivra puisque les valeurs aux positions 4 et 5 seront transférées intactes dans une même progéniture. Si A est croisé avec la chaîne B, on retrouve le schéma H_2 dans la chaîne B', alors que H_1 n'est ni dans A' ni dans B'.

$$A = 0\,1\,0\,1\,0\,0\,1$$

$$B = 1\,0\,0\,0\,0\,1\,0$$

$$A' = 0\,1\,0\,0\,0\,1\,0$$

$$B' = 1\,0\,0\,1\,0\,0\,1$$

$$H_1 = *\,1\,*\,*\,*\,*\,1$$

$$H_2 = *\,*\,*\,1\,0\,*\,*$$

Comme nous pouvons le voir, un schéma H survit quand le lieu de croisement est à l'extérieur de l'intervalle de longueur $\delta(H)$.

Le croisement peut s'effectuer à n'importe quelle position entre 1 et t - 1. Dans le cas du schéma H_1 précédent, il peut s'effectuer à 6 positions différentes. Le schéma H_1 sera détruit si le lieu de croisement est en position 2, 3, 4, 5 ou 6. Le schéma H_1 a une probabilité d'être détruit de 5/6, ou plus généralement $\delta(H)/(t\text{-}1)$. La probabilité de survivre à un croisement est donc :

$$p_s = 1 - \delta(H)/(t\text{-}1).$$

En supposant l'indépendance des opérations de sélection et de croisement, et sachant que le nombre prévu d'occurrences dans la prochaine génération par suite de la sélection est égale à $m(H)\times f(H)/\bar{f}$, et qu'un schéma survit avec une probabilité de $1-\delta(H)/(t-1)$, nous obtenons un estimé du nombre de schémas particuliers N(H) prévu dans la prochaine génération :

Les schémas avec aptitude supérieure à la moyenne et de longueur $\delta(H)$ petite vont être échantillonnés à un taux exponentiellement croissant.

$$N(H) = m(H)\times\frac{f(H)}{\bar{f}}\left[1-\frac{\delta(H)}{(t-1)}\right]$$

3.5.4 La mutation et les schémas

La mutation est une altération aléatoire d'une seule position avec probabilité p_m. Pour qu'un schéma survive, il faut que chacune des positions fixes de $o(H)$ dans le schéma survive. La probabilité qu'un schéma survive à une mutation est : $1 - o(H) \times p_m$ lorsque p_m est petit. Le nombre de copies $C(H)$ prévu dans la prochaine génération après sélection, croisement et mutation est :

$$C(H)=m(H)\times\frac{f(H)}{\bar{f}}\left[1-\frac{\delta(H)}{(l-1)}-o(H)p_m\right]$$

3.6 Applications des algorithmes génétiques

Les AG ont été utilisés pour la résolution de nombreux problèmes dont celui du recouvrement (Beasley et Chu, 1996), celui de la formation de cellules (Gravel et al., 1998) ou celui du voyageur de commerce (Chatterjee et al., 1998). Ici, nous exposerons deux exemples d'application, un de conception de réseaux et un autre d'optimisation d'un réseau de distribution.

3.6.1 Conception topologique de réseaux téléinformatiques

Pierre et Legault (1996a, 1996b, 1998) ont appliqué la méthode des algorithmes génétiques au problème de conception topologique de réseaux téléinformatiques à commutation de paquets. Leur but était de minimiser le coût du réseau sous les contraintes de la localisation des commutateurs, du trafic acheminé et du délai. Pour ce problème, l'implantation proposée était caractérisée par la génération d'une population initiale suivie d'une succession de générations. Les solutions étaient codées sous forme de chromosomes faites de séquences de 1 et de 0 qui symbolisent l'existence ou non d'une liaison. La longueur des chromosomes était directement liée au nombre de nœuds composant le réseau. Ainsi, un réseau composé de n nœuds se représente par un chromosome de longueur $n(n - 1)/2$, ce qui correspond au maximum de liaisons pouvant constituer un réseau de n nœuds. Le premier gène dans le chromosome représentait la liaison 1 qui reliait les nœuds 1 et 2, le second représentait la liaison 2 reliant les nœuds 1 et 3, et ainsi de suite.

La population initiale est générée en deux phases. D'abord, une topologie initiale de degré de connexité k est créée; ensuite les autres chromosomes sont créés par modification de la topologie initiale. La création de la première topologie est effectuée de façon déterministe car elle est basée sur la distance la plus courte entre nœuds. Durant la formation des générations, le premier chromosome de la population actuelle est le meilleur de la génération précédente, le choix des autres étant basé sur le principe de la roulette. Un opérateur particulier, l'inversion, est associée au croisement dans l'algorithme. En vue des tests, les paramètres utilisés sont les suivants : taille moyenne du paquet : *1000 bits*; trafic uniforme de *5 paquets/s*; degré de connexité : *3*; taille de la population : *80 chromosomes*; probabilité de croisement : *0,95*; probabilité de mutation : *0,12*; probabilité d'inversion : *0,23*; nombre de générations : *40*.

Des résultats obtenus, il ressort que :

- un nombre élevé de générations tend à produire de bonnes solutions;

- l'algorithme est plus performant pour les réseaux de plus de 15 nœuds et déconseillé pour les réseaux de petite taille pour lesquels il dégénère parfois la solution;
- une grande taille de la population est gage d'une bonne diversité des solutions, mais pas forcément de l'obtention des meilleures solutions;
- une probabilité de croisement de 0,6 produit généralement de très bonnes solutions car les solutions jugées performantes des parents ont plus de chances de se retrouver intactes dans la population-enfant; par contre une probabilité de 0,95 produit une grande diversité des solutions obtenues;
- les solutions générées par la variation de la probabilité de mutation entre 0,05 et 0,2 se situent généralement dans le même intervalle de valeurs;
- une probabilité d'inversion faible de l'ordre de 0,1 offre de meilleures solutions par rapport à une probabilité plus élevée.

Ces résultats ont été comparés à ceux obtenus par la méthode de la coupe saturée et celle du recuit simulé. Il s'ensuit que pour des réseaux de taille inférieure à 15 nœuds, la coupe saturée offre de meilleures solutions. Par contre, pour des réseaux de taille plus grande, les solutions obtenues par l'algorithme génétique sont meilleures. Toutefois, le délai moyen obtenu par la coupe saturée est inférieur à celui de l'algorithme génétique. Dans le cas du recuit simulé, les solutions obtenues, quelle que soit la taille du réseau, sont moins performantes que celles obtenues par l'algorithme génétique.

Il faut toutefois préciser que l'obtention de bonnes solutions par l'algorithme génétique dépend en grande partie de la qualité de la solution initiale pour ce type de problème.

3.6.2 Optimisation d'un réseau de distribution

Le problème de conception d'un réseau de distribution est très courant dans le domaine des transports de marchandises ou de fluides. Ce problème s'apparente beaucoup à celui d'affectation de capacité aux liaisons d'un réseau de communications. Castillo et Gonzalez (1998) ont proposé une heuristique basée sur l'algorithme

génétique pour l'optimisation de tels réseaux de distribution. Cette heuristique doit tenir compte de certaines contraintes telles que le diamètre des conduits, la pression des fluides, le débit et le flot. Les solutions à ce problème ont été codées sous forme réelle, chaque gène représentant le coût final d'une liaison entre deux nœuds. Les résultats obtenus montrent que cette heuristique conduit à la solution optimale dans 80 à 90 % des cas expérimentés.

CHAPITRE IV

ALGORITHME GÉNÉTIQUE D'AFFECTATION DE CELLULES

Ce chapitre expose l'adaptation des AG que nous proposons pour la résolution du problème d'affectation des cellules à des commutateurs dans les RCP. Notre heuristique tentera de trouver, à partir d'une population initiale de chromosomes, la meilleure affectation, c'est-à-dire celle qui minimise le coût du réseau tout en respectant la contrainte sur la capacité des commutateurs et celle d'affectation unique des cellules aux commutateurs. Après un bref rappel du principe des AG, nous exposerons l'encodage des solutions en tenant compte de la spécificité de notre problème, le processus de génération de la population initiale et les opérateurs génétiques choisis. Nous achèverons par une analyse des différentes étapes du processus génétique décrit par notre algorithme.

4.1 Adaptation de la méthode des algorithmes génétiques

Les AG suivent tous un même principe et l'adaptation présentée ici n'y fait pas exception. Comme l'indique la Figure 4.1 qui représente l'organigramme de notre heuristique, on distingue une phase de création de la population initiale, suivie de la perturbation des éléments de cette population par l'application de différents opérateurs génétiques, et enfin une phase d'évaluation de cette population. Ces mécanismes sont répétés pendant un certain nombre de générations. Chaque génération est supposée contenir des éléments plus performants que ceux de la génération précédente. Intuitivement, plus le nombre de générations est grand, plus la solution se raffinera et on espère ainsi obtenir une bonne solution, mais pas forcément la meilleure.

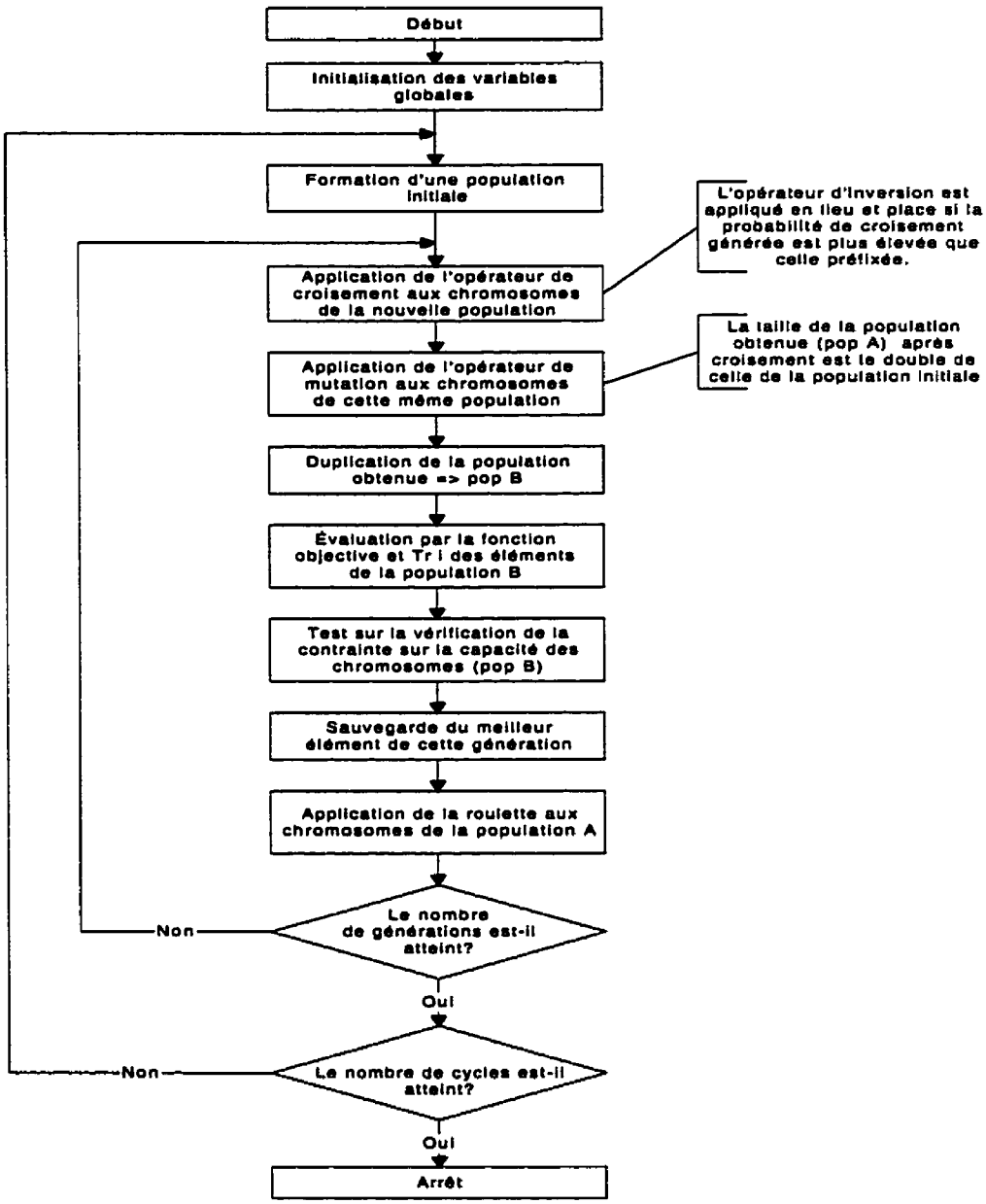

Figure 4.1 Algorithme général du processus génétique

Il convient de rappeler ici que la méthode des algorithmes génétiques est essentiellement aléatoire. En effet, tous les mécanismes de création de chromosomes, de détermination de lieux de croisement ou de mutation, de choix des candidats auxquels

appliquer les opérateurs génétiques sont basés sur le hasard. Seuls un bon codage et un choix judicieux des différents opérateurs permettent (non sans l'œuvre du hasard) de converger vers une bonne solution quasi-optimale.

4.2 Codage des chromosomes

Les solutions du problème sont codées sous forme de chromosomes qui sont souvent représentés par des chaînes d'éléments entiers ou réels. La forme la plus répandue est la chaîne binaire où les gènes ne prennent que les valeurs 0 et 1. Le problème que nous avons à résoudre en est un d'affectation, sujet à deux principales contraintes : la première sur l'affectation unique et la seconde sur la capacité des commutateurs. Pour notre adaptation, nous avons retenu une représentation non binaire des chromosomes (Gondim, 1996). Chaque chromosome représente un schéma d'affectation spécifique. Les éléments de la chaîne représentant un chromosome sont des entiers. Ces entiers représentent les différents commutateurs qui sont numérotés de 1 à m. La longueur des chaînes représentant les chromosomes est égale au nombre de cellules et reste inchangée, car toutes les cellules du réseau doivent être affectées. De même, la valeur maximale d'un gène de ces chromosomes est égale au nombre maximal de commutateurs. Certaines valeurs peuvent ne pas faire partie d'une chaîne, ce qui implique que le commutateur portant le numéro manquant ne figure pas dans le schéma d'affectation, donc n'est pas utilisé. Un réseau composé de 8 cellules pour 3 commutateurs potentiels est représenté par un chromosome de longueur 8, chaque gène pouvant prendre la valeur 1, 2 ou 3. La lecture des chromosomes se fait de gauche à droite. Ainsi, le chromosome représenté à la Figure 4.2 s'interprète de la façon suivante : la cellule 1 est reliée au commutateur 1, la cellule 2 au commutateur 2, la cellule 3 au commutateur 2, la cellule 4 au commutateur 1, la cellule 5 au commutateur 2, la cellule 6 au commutateur 1, la cellule 7 au commutateur 1, la cellule 8 au commutateur 2. Le commutateur 3 étant absent de ce schéma d'affectation, seuls les commutateurs 1 et 2 sont effectivement utilisés dans le réseau.

1	2	2	1	2	1	1	2

Figure 4.2 Représentation non binaire d'un chromosome

Le codage que nous avons adopté ici nous permet de satisfaire une contrainte : celle de l'affectation unique des cellules aux commutateurs, car un gène d'un chromosome ne peut pas prendre simultanément plus d'une valeur. La seule contrainte à satisfaire reste alors celle sur la capacité des commutateurs.

4.3 Génération de la population initiale

Comme dans les algorithmes génétiques classiques, nous avons d'abord créé une population initiale où tous les chromosomes sont générés de façon aléatoire. Nous avons alors remarqué que deux ou plusieurs chromosomes identiques peuvent ainsi se retrouver dans la population initiale. Cette situation entraîne la dégénération des populations subséquentes et la régression de la solution finale. Pour éviter ce piège, nous avons adopté la stratégie de la population initiale sans doubles. Cette stratégie permet de s'assurer que deux chromosomes identiques ne se retrouveront pas dans la population initiale. Ainsi, elle maintient la diversité de la population et donc une bonne couverture de l'espace de recherche. Comme le décrit l'algorithme de création de la population initiale de la Figure 4.3, la création de la population initiale se fait en deux temps. Dans un premier temps, le premier chromosome de la population est créé. Ce chromosome est celui obtenu si toutes les cellules étaient reliées au commutateur qui leur est le plus proche selon la métrique de la distance euclidienne. Si une cellule est équidistante de deux ou plusieurs commutateurs, elle est affectée au premier commutateur dans l'ordre spécifié par les données du problème. La formation du premier élément de la population initiale est donc déterministe. Ce chromosome ne respecte pas forcément la contrainte de capacité sur les commutateurs, mais respecte celle d'affectation unique des cellules aux commutateurs.

La seconde étape consiste en la création des autres chromosomes de façon aléatoire. Par la suite, à chaque fois qu'un nouveau chromosome est créé, on vérifie sa ressemblance avec tous ses prédécesseurs dans la population et on le recrée le cas échéant. Il faut noter que, plus la taille de la population initiale est grande, plus le temps de formation de la population sans doubles augmente, car le nombre de comparaisons au moment de la formation de chaque chromosome devient de plus en plus grand. Mais compte tenu de la proportion du nombre d'éléments de la population par rapport au nombre total d'éléments possibles, la probabilité de recréer le même élément plusieurs fois en est plus réduite. Le codage des chromosomes induisant une relaxation de la contrainte sur la capacité des commutateurs, la faisabilité des solutions n'est donc pas garantie. Mais, étant donné que la taille de la population est relativement grande, les chances d'aboutir à une bonne solution sont grandes.

```
Initialiser le premier chromosome de la population initiale
Pour i := 1 à Taille_max
        Créer le chromosome
        Pour j := 0 à i-1
                Tant que le chromosome créé est identique à l'un de ses prédécesseurs
                        Créer un chromosome
                Fin Tant que
        Fin Pour
Fin Pour
```

Figure 4.3 Algorithme de création de la population initiale

4.4 Les opérateurs génétiques

- **Opérateur de croisement**

Plusieurs options s'offrent à nous dans le choix de l'opérateur de croisement. Nous avons le croisement à un lieu, le croisement à deux lieux et celui à *n* lieux. Nous

avons effectué des tests sur des réseaux de différentes tailles et nous avons observé les résultats obtenus et les temps d'exécution pour les différents types de croisement. Les croisements à plusieurs lieux sont plus *gourmands* en temps d'exécution, tandis que les résultats qu'ils fournissent n'en sont pas pour autant meilleurs que ceux obtenus avec un croisement à lieu unique. Notre choix s'est donc porté sur ce dernier dans le souci de réduire le temps d'exécution. Les chaînes seront donc interchangées autour d'un lieu de croisement généré aléatoirement. Les chromosomes sur lesquels on appliquera l'opérateur de croisement sont choisis de façon aléatoire dans la population. Après avoir été choisis, les deux chromosomes parents sont insérés dans une nouvelle population. Un mécanisme est mis en place pour qu'à la prochaine itération, le choix se fasse parmi les chromosomes non encore choisis. On applique ensuite l'opérateur de croisement et on insère ensuite les deux chromosomes enfants obtenus dans la population. L'opérateur d'inversion est introduit lorsque la probabilité de croisement générée serait supérieure à celle préfixée. Cet opérateur inverse alors tous les gènes d'un chromosome. Pour un chromosome de longueur t, les gènes des positions 1 et t sont permutés, ceux des positions 2 et t-1 le sont, et ainsi de suite. On introduit alors les deux chromosomes enfants obtenus après inversion des deux chromosomes parents dans la nouvelle population. La probabilité de croisement préfixée est souvent proche de 1.

La Figure 4.4 décrit les grandes lignes de l'algorithme d'application de l'opérateur de croisement. La taille de la population obtenue après l'application de cet opérateur est le double de celle de la population initiale, car chromosomes parents et enfants se retrouvent tous dans la nouvelle population. Un chromosome-parent qui représente une bonne solution a donc plus de chances de survivre aux perturbations introduites dans la population par cet opérateur. On se rapproche ainsi du principe du meilleur des deux mondes.

- **Opérateur de mutation**

Étant donné que l'opérateur de mutation, compte tenu de la faible valeur de probabilité qui lui est associée, n'influe pas beaucoup sur les résultats, nous l'avons

choisi le plus simple possible. Un lieu est choisi aléatoirement à l'intérieur du chromosome et une nouvelle valeur est générée pour le gène à ce lieu. Le chromosome obtenu après mutation peut rester inchangé si la nouvelle valeur générée pour le gène est identique à l'ancienne. La probabilité de mutation est souvent très faible (proche de 0). La Figure 4.5 montre l'organigramme de ce mécanisme.

4.5 La fonction d'évaluation

Un des éléments clés de l'algorithme génétique est la fonction d'évaluation. Sur cette fonction repose le choix des candidats à la prochaine génération. Dans le cadre de notre adaptation, le chromosome est évalué suivant les critères de coût dans un premier temps par la formule suivante :

$$F = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik} x_{ik} + \sum_{i=1}^{n}\sum_{j=1}^{n} h_{ij}(1-y_{ij})$$

où c_{ik} représente le coût de la liaison entre la cellule i et le commutateur k, x_{ik} une variable qui vaut 1 si la cellule i est affectée au commutateur k et 0 sinon, y_{ij} une variable qui vaut 1 si les cellules i et j sont affectées au même commutateur et 0 sinon, et enfin h_{ij} le coût des relèves entre les cellules i et j quand elles sont affectées à des commutateurs différents, celui entre cellules affectées à un même commutateur étant négligeable. Cette fonction associe ainsi à chaque chromosome une valeur qui représente le coût de la configuration du réseau qu'il représente.

La seconde étape d'évaluation consiste à vérifier si les chromosomes respectent la contrainte sur la capacité des commutateurs et à déterminer le chromosome ayant le coût le moins élevé tout en respectant cette contrainte. À cette étape, nous avons introduit un mécanisme qui conserve des chromosomes violant la contrainte sur la capacité des commutateurs. Les capacités des commutateurs qui figurent dans ces chromosomes ont un écart d'au plus 10% par rapport à la capacité maximale autorisée. Le concepteur du réseau pourra ainsi faire un compromis sur le choix des chromosomes solutions car les chromosomes non faisables à priori pourraient devenir de très bonnes solutions par une légère modification.

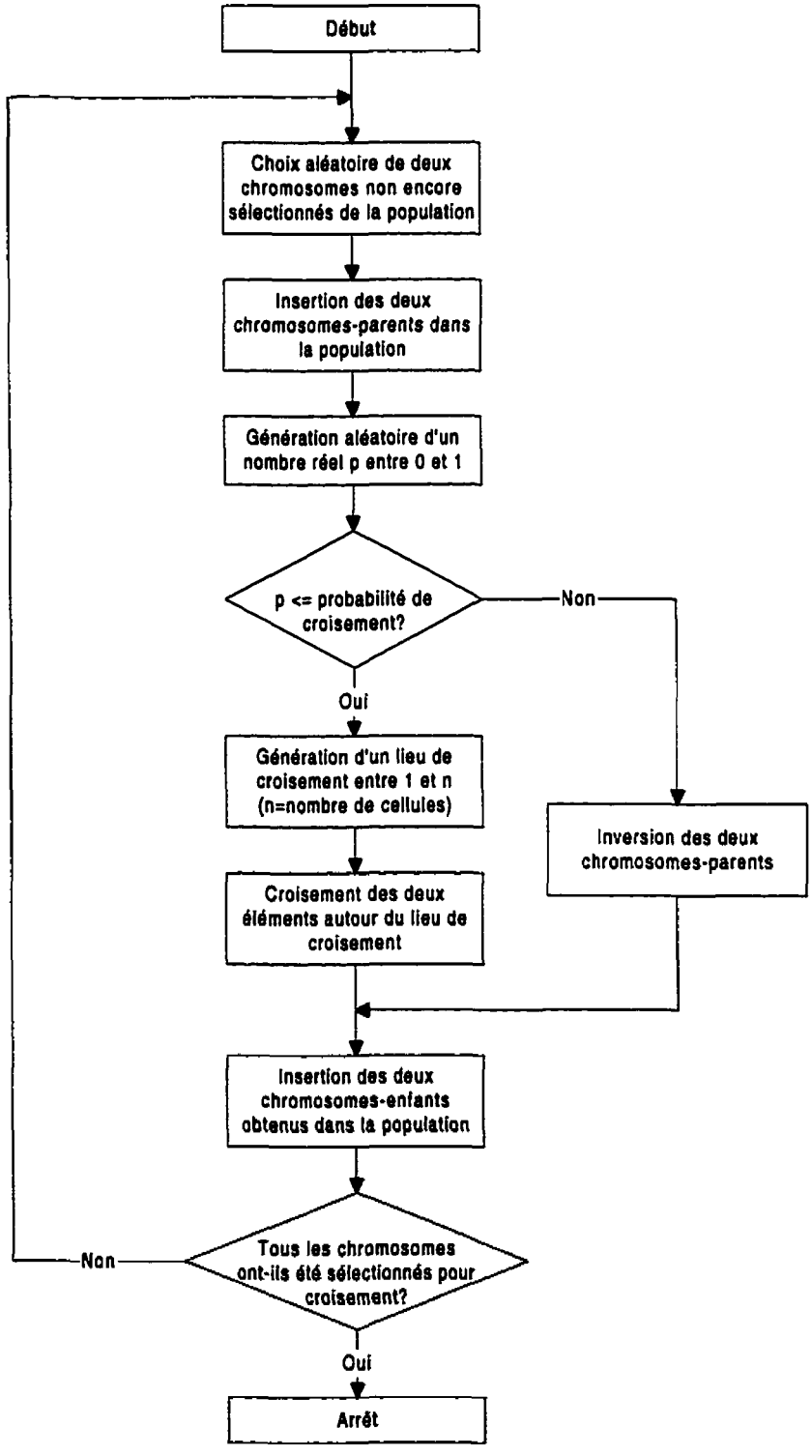

Figure 4.4 Organigramme du mécanisme de croisement

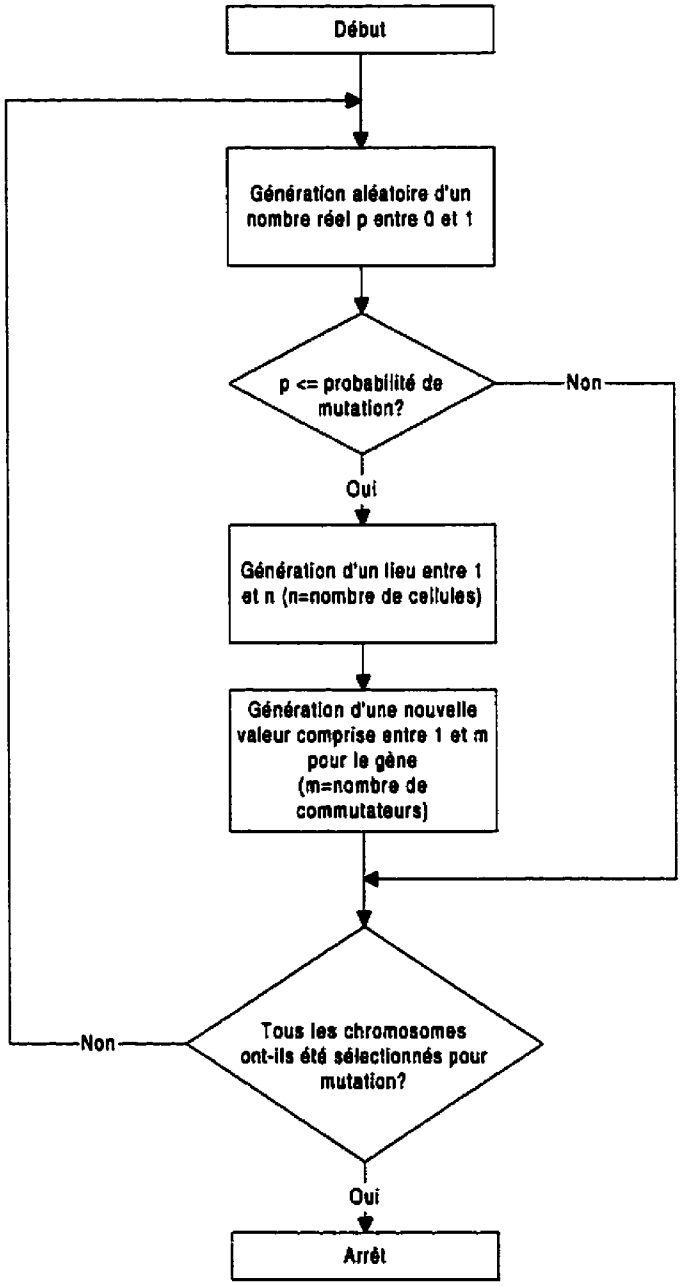

Figure 4.5 Organigramme du mécanisme de mutation

4.6 Formation d'une nouvelle population

- **Opérateur de sélection**

Pour déterminer les chromosomes qui feront partie d'une nouvelle génération, plusieurs stratégies sont disponibles : celle de la roulette de casino, celle dite élitiste ou celle dite d'insertion directe se basant sur les valeurs objectives des différents éléments. La stratégie de choix direct reposant sur les valeurs objectives des éléments de la population a été écartée, car elle ne prend en compte que les meilleures solutions de la génération présente. Cette stratégie conduit souvent à l'exploration d'une partie seulement de l'espace de recherche et entraîne du coup l'appauvrissement de l'espèce. Pour les mêmes raisons, la stratégie élitiste qui ne conserve que le meilleur élément de la génération présente a aussi été écartée, car elle rejette toutes les bonnes solutions issues de l'application des différents opérateurs génétiques. Nous avons choisi d'utiliser pour notre adaptation une combinaison de la stratégie de la roulette de casino et celle de la population sans doubles. En effet, vu que chromosomes parents et enfants se retrouvent dans la population obtenue après croisement, une duplication d'éléments conduirait à coup sûr à une régression des solutions subséquentes. La population formée à la fin d'une génération est donc une population sans doubles, mais qui contient en majorité – du moins on l'espère – les bonnes solutions des générations précédentes.

4.7 Processus génétique

Le processus génétique qui est mis en œuvre par notre adaptation et qui est décrit par la Figure 4.1 précédente est globalement semblable au processus classique. Au début de l'exécution, les variables communes à tous les modules sont initialisées. Ces variables sont essentiellement la taille de la population, le nombre de générations d'un cycle, le nombre de cycles du processus, les probabilités associées aux différents opérateurs. On conserve également les différentes données du problème dans des structures appropriées. Une population initiale est générée; on applique ensuite les différents opérateurs sur cette population dans l'espoir d'obtenir des éléments plus

performants. Un mécanisme de sélection permet, à la fin d'une génération, de former une nouvelle population pour la prochaine génération.

4.7.1 Concept de cycle

Afin d'accroître les chances d'atteindre la solution optimale, nous avons introduit le concept de *cycle*. Un cycle est composé d'un certain nombre de générations. Il débute par une population initiale et, à la fin, une solution (la meilleure du cycle) est déterminée. L'originalité de ce concept de cycle réside dans le fait qu'à chaque début de cycle, une nouvelle population initiale est générée. Si éventuellement avec la population initiale précédente et malgré l'application des différents opérateurs l'algorithme s'enlisait dans une seule direction ou autour d'un minimum local, cette régénération de la population initiale permettra d'éviter le piège de l'optimum local et d'orienter la recherche vers d'autres directions. La meilleure solution de tous les cycles est retenue comme étant la meilleure du processus génétique.

4.7.2 Altération de la population

Pendant le processus génétique, deux populations existent en parallèle : la nouvelle population générée au début de chaque génération, d'où sont puisés les chromosomes parents sur lesquels seront appliqués les opérateurs, et une population en attente d'où seront sélectionnés les chromosomes enfants pour la prochaine génération.

Les opérateurs génétiques définis dans les sections précédentes sont appliqués à cette population initiale. Nous avons d'abord le croisement. Précisons ici que la taille de la population en attente devient le double de celle de la population initiale car les deux chromosomes parents et les deux chromosomes enfants se retrouvent dans la population en attente. Nous avons ensuite la mutation. Ce mécanisme est appliqué sur chaque élément de la population en attente. Mais, compte tenu de la faible probabilité associée à cet opérateur, de grands changements ne seront pas observés dans la population. À cette étape, on duplique la population en attente. On a alors une population en attente A et une population en attente B. Cette duplication nous permet de sauvegarder l'homogénéité de

la population obtenue après l'application des opérateurs et, dans le même temps, de déterminer à partir des différentes évaluations de la même population la meilleure solution de la génération présente.

On associe alors à chaque élément de la population B ainsi formée une valeur dite *objective*. Cette valeur est déterminée par l'application d'une fonction d'évaluation appelée *fonction objectif*. On effectue ensuite un tri par valeur croissante de tous les éléments de B. Intuitivement, les meilleurs éléments de cette génération se retrouveront ainsi au début de la population. Vient ensuite l'évaluation de tous les chromosomes pour la vérification de la contrainte sur la capacité des commutateurs. On renvoie alors à la fin de la population tous les chromosomes qui violent cette contrainte. Conformément à une structure de file, on a ainsi en tête de la population l'élément représentant la meilleure solution de la génération. On conserve cette solution.

Une autre originalité introduite par notre adaptation est le mécanisme du *repêchage*. C'est un mécanisme qui permet de récupérer et de conserver toutes les solutions violant la contrainte sur la capacité des commutateurs de 10% ou moins. Ces solutions seront soumises à un examen plus judicieux par le concepteur et des compromis seront effectués. Notons que ces solutions peuvent devenir faisables par une légère modification, par exemple une modification des capacités maximales des commutateurs.

4.7.3 Création d'une nouvelle population

À la fin d'une génération, on applique l'opérateur de sélection pour déterminer ceux des éléments de la population en attente A qui feront partie de la prochaine génération. Comme nous l'avons précisé plus haut, la stratégie adoptée est une combinaison de la roulette et de la population sans doubles. Le problème que nous avons à résoudre étant un problème de minimisation de coût, la roulette sera appliquée non sur les valeurs objectives des différents chromosomes mais sur leurs inverses. On calcule ainsi les inverses des valeurs objectives de tous les chromosomes de la population. On normalise la somme de ces inverses à 1 et on associe à chaque

chromosome une borne supérieure et une borne inférieure qui serviront à déterminer la portion de la roulette qu'il occupera. Seule la moitié de ces chromosomes, éléments de la population A, sera sélectionnée pour faire partie de la prochaine génération. Ces nouveaux éléments ne sont pas forcément les meilleurs, ni faisables. Nous espérons donc que le brassage occasionné par l'application des opérateurs de croisement et de mutation permettra d'orienter la recherche vers d'autres régions de l'espace de recherche qui sont plus fécondes et d'accroître ainsi les chances d'obtenir de bonnes solutions.

À la fin de chaque cycle, on conserve la meilleure solution faisable; à la fin de tous les cycles, la meilleure de toutes les meilleures solutions trouvées au cours des générations est retournée comme meilleure solution trouvée par l'algorithme.

CHAPITRE V
IMPLÉMENTATION ET MISE EN ŒUVRE DE LA MÉTHODE

Ce chapitre expose l'implémentation et la mise en œuvre de la méthode que nous proposons pour la résolution du problème d'affectation de cellules. Nous y présenterons d'abord le format des fichiers d'entrée et le diagramme des classes. Nous décrirons par la suite les différentes structures de données ainsi que les principaux algorithmes. Enfin, en guise de mise en œuvre, nous appliquerons notre méthode à un exemple numérique pour illustrer le fonctionnement de celle-ci.

5.1 Format des fichiers d'entrée et diagramme des classes

Pour implanter l'adaptation proposée, nous avons écrit un programme en langage C++, d'une complexité de l'ordre de mn^2 (m étant le nombre de commutateurs et n le nombre de cellules), et intégrant quatre grandes classes. La Figure 5.1 montre le diagramme UML de ces différentes classes.

Le programme reçoit essentiellement deux fichiers en entrée. Le premier fichier contient : le nombre de cellules et de commutateurs sur la première ligne; ensuite les coûts de liaison cellule-commutateur sous forme d'une matrice $n \times m$, n et m étant respectivement le nombre de cellules et celui des commutateurs; et enfin les coûts des relèves entre cellules adjacentes sous la forme d'une matrice $n \times n$ (n étant le nombre de cellules). Le second fichier contient les volumes d'appel des cellules sur la première ligne et les capacités maximales des commutateurs sur la seconde ligne.

Chacune des classes définies possède un constructeur et un destructeur qui libère toute la mémoire allouée aux attributs de cette classe. Les quatre grandes classes sont : la classe *Donnee*, la classe *Chromosome*, la classe *Population* et la classe *Result*.

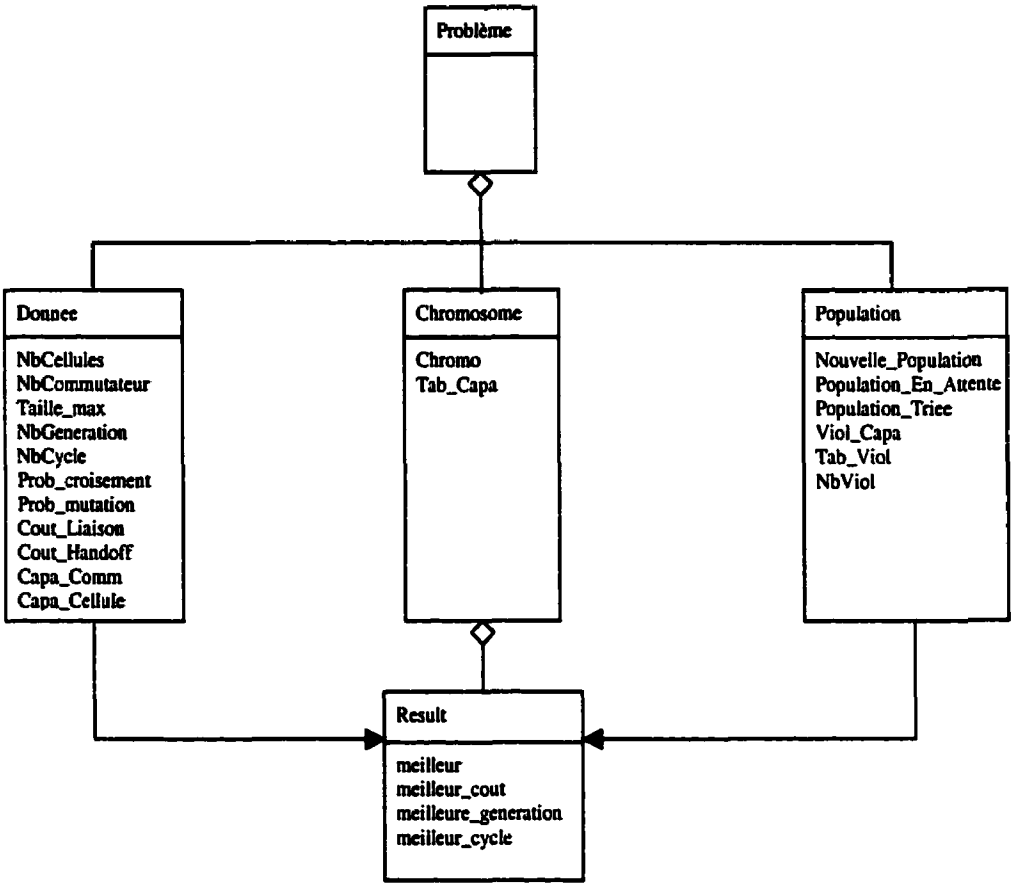

Figure 5.1 Diagramme UML des différentes classes

- La classe *Donnee*

La classe *Donnee* comprend toutes les données en entrée du problème. Elle intègre aussi les principaux paramètres qui seront définis par l'utilisateur. On y retrouve les attributs suivants :

NbCellules : nombre de cellules

NbCommutateur : nombre de commutateurs

Cout_Liaison : tableau des coûts de liaison cellule-commutateur

Cout_Handoff : tableau des coûts de relèves entre cellules

Capa_Comm : tableau des capacités maximales des commutateurs

Capa_Cellule : tableau des volumes d'appel des cellules

Taille_max : taille maximale d'une population
NbGeneration : nombre de générations au cours d'un cycle
NbCycle : nombre de cycle au cours du processus génétique
Prob_croisement : probabilité de croisement préfixée
Prob_mutation : probabilité de mutation préfixée

Les principales méthodes sont :

- Lire_Fichiers() qui lit les données provenant des fichiers et les conserve dans les différentes structures ;
- GenerateurAleatoire() qui génère aléatoirement des nombres entiers compris entre 1 et une limite que l'on fournit à la fonction.

- La classe *Chromosome*

La classe *Chromosome* comprend outre les attributs d'un chromosome, à savoir le tableau représentant le chromosome *Chromo*, et le tableau des capacités de chaque commutateur dans ce chromosome *Tab_Capa*, différentes méthodes qui serviront à manipuler ces chromosomes. Les principales méthodes sont :

- *InitialiserChromosome()* qui initialise un chromosome en créant le tableau représentant les affectations entre cellules et commutateurs et en créant le tableau des capacités correspondant ;
- *Plus_Courte_Distance()* qui permet de créer le chromosome résultant de l'affectation des cellules aux commutateurs le plus proche ;
- *EvaluerChromosome()* qui évalue un chromosome selon la fonction objectif;
- EvaluerCapaciteChromosome() *qui évalue un chromosome par rapport à la contrainte de capacité sur les commutateurs ;*
- *InverserChromosome()* qui inverse un chromosome ;
- *MuterChromosome()* qui mute un gène d'un chromosome ;
- *ChromosomeIdentique()* qui compare deux chromosomes ;

- *ChromosomeFaisable()* qui vérifie si un chromosome satisfait la contrainte sur la capacité des commutateurs ;
- *CopierChromosome()* qui copie un chromosome dans un autre.

- La classe *Population*

La classe *Population*, quant à elle, regroupe toutes les populations utilisées dans le programme et les différentes méthodes servant à les gérer. Les principaux attributs sont :

Nouvelle_Population : Nouvelle population sur laquelle seront appliqués les différents opérateurs ;

Population_EnAttente : Population obtenue après l'application des opérateurs ;

Population_Triee : Population obtenue après le tri par ordre croissant des valeurs objectives des différents chromosomes ;

ViolCapa : Population de chromosomes non faisables et qui pourraient le devenir par suite d'une légère modification.

Deux tableaux de réels, *Tableau_Valeurs* et *Tab_Viol*, qui contiennent respectivement les valeurs objectives des différents chromosomes d'une population et celles des chromosomes repêchés y font aussi partie. Les principales méthodes de cette classe sont :

- *InitialiserPopulation()* qui initialise les différents chromosomes d'une population ;
- *Former_Population_Initiale()* qui forme une population initiale sans doubles ;
- *Evaluer_Population()* qui évalue les différents chromosomes d'une population ;
- *EvaluerCapacitePopulation()* qui évalue les chromosomes d'une population par rapport à la contrainte de capacité sur les commutateurs ;
- *Moyenne()* qui calcule et retourne la moyenne des valeurs objectives de tous les chromosomes d'une population ;

- *Croisement()* qui réalise le croisement à un lieu de deux chromosomes sous une certaine probabilité et les inverse si nécessaire ;
- *Mutation()* qui réalise la mutation des éléments d'une population ;
- *Trier_Population()* qui trie par ordre croissant les valeurs objectives des chromosomes d'une population ;
- *Selectionner_Nouvelle_Population_Roulette()* qui détermine les chromosomes de la prochaine génération en appliquant la roulette sur les inverses des valeurs objectives de ceux de la génération présente ;
- *CopierPopulation()* qui copie une population dans une autre.

- La classe *Result*

Enfin, la classe *Result* conserve à l'aide des différentes méthodes les principaux résultats obtenus. Les attributs de cette classe comprennent un objet chromosome *meilleur*, deux variables *meilleure_generation* et *meilleur_cycle* pour conserver la génération et le cycle auxquels le meilleur chromosome a été obtenu. Ses principales méthodes sont :

- *Sauvegarde1()* qui sauvegarde pour une exécution les principaux paramètres ;
- *Sauvegarde2()* qui sauvegarde pour une exécution les meilleures solutions obtenues à chaque cycle ;
- *Sauvegarde3()* qui sauvegarde pour une exécution les meilleures solutions obtenues à chaque génération.

Différentes structures de données ont été utilisées pour implémenter ces différentes classes.

5.2 Structures de données

- **Classe *Donnee***

Pour conserver le nombre de cellules, le nombre de commutateurs et les différents paramètres définis par l'utilisateur tels que la taille maximale de la population initiale, le nombre de générations d'un cycle, le nombre de cycles du processus, des

variables entières sont utilisées. Les probabilités de croisement et de mutation sont des variables réelles. Pour les coûts de liaison cellule-commutateur et les coûts de relèves entre cellules adjacentes, nous avons choisi d'utiliser des tableaux à deux dimensions de taille respectives $n \times m$ et $n \times n$ (n étant le nombre de cellules et m celui de commutateurs). Deux autres tableaux à une dimension sont utilisés : le premier d'une taille égale au nombre de cellules sert à conserver les volumes d'appels des cellules et le second d'une taille égale au nombre de commutateurs sert à conserver les capacités maximales des commutateurs.

Nous avons choisi d'utiliser essentiellement des tableaux pour implémenter cette classe car la plupart des fonctions des autres classes y accèdent. L'accès étant direct, le temps de recherche des données s'en trouve ainsi réduit, ce qui rencontre l'un de nos objectifs qui est de trouver en un temps relativement court une solution quasi-optimale. L'inconvénient majeur de cette structure est la mémoire utilisée qui augmente avec le nombre de cellules et de commutateurs.

- **Classe *Chromosome***

Les deux attributs de cette classe sont implémentés sous forme de tableaux à une dimension, le premier représentant le chromosome dont la taille est égale au nombre de cellules et le second dont la taille est égale au nombre de commutateurs représentant les capacités obtenues à partir du schéma d'affectation induit par le premier tableau. Ici encore, nous utilisons des tableaux pour cause de rapidité d'accès.

Les différentes méthodes de cette classe entraînent diverses altérations sur les chromosomes auxquels elles sont appliquées. La fonction *Plus_Courte_Distance()* crée le premier chromosome de la population initiale qui est celui obtenu quand toutes les cellules sont affectées au commutateur le plus proche. La fonction *EvaluerChromosome()* détermine la valeur objective de la solution décrite par un chromosome. La fonction *EvaluerCapaciteChromosome()* détermine les capacités des commutateurs pour un schéma d'affectation spécifique. La fonction *InverserChromosome()* inverse tous les gènes d'un chromosome. Pour ce faire, le

chromosome passé en argument est dupliqué, puis inversé. La fonction *MuterChromosome()* détermine un lieu de mutation à l'intérieur du chromosome, puis génère ensuite une nouvelle valeur pour le gène à ce lieu. La fonction *ChromosomeIdentique()* compare deux chromosomes et retourne *Vrai* si ces deux chromosomes sont identiques et *Faux* sinon. La fonction *ChromosomeFaisable()* vérifie si un chromosome satisfait la contrainte sur la capacité des commutateurs en comparant le tableau de capacités issu du schéma d'affectation obtenu à partir du chromosome au tableau des capacités maximales des commutateurs.

- **Classe *Population***

Les attributs de cette classe sont essentiellement des tableaux d'objets *Chromosome*. Le premier, *Nouvelle_Population*, représente la population initiale. Sa taille est définie par l'utilisateur. Le second, *Population_en_attente*, est celle obtenue après l'application des différents opérateurs. Sa taille est le double de celle de la population initiale. Le troisième, *Population_triee*, est une duplication du précédent, mais trié cette fois-ci. Le quatrième, *Viol_Capa*, contient des chromosomes non faisables qui sont conservés selon un certain critère pour une étude plus minutieuse.

Deux autres tableaux de réels, *Tableau_Valeurs* et *Tab_Viol*, viennent compléter la liste et contiennent respectivement les valeurs objectives des éléments contenus dans la population en attente et celles des solutions violant la contrainte sur la capacité des chromosomes, mais qui sont conservées pour une étude plus attentive par le concepteur.

Les méthodes de cette classe agissent directement sur les populations. La fonction *InitialiserPopulation()* initialise de façon aléatoire tous les chromosomes de toutes les populations. La fonction *Former_Population_Initiale()* crée une population initiale sans doubles. Pour cela, chaque chromosome créé est comparé à tous ses prédécesseurs. S'il est identique à l'un d'entre eux, il est détruit puis régénéré, sinon il est maintenu dans la population. On arrête quand tous les chromosomes de la population sont créés.

Les fonctions *Evaluer_Population()* et *EvaluerCapacitePopulation()* évaluent tous les chromosomes de la population respectivement par rapport à la fonction objective et à la contrainte sur la capacité des commutateurs. Pour cela, elles se servent des fonctions *EvaluerChromosome()* et *EvaluerCapaciteChromosome()* de la classe Chromosome. La fonction *EvaluerCapacitePopulation()* renvoie à la fin du tableau tous les chromosomes qui ne respectent pas la contrainte sur la capacité des commutateurs. La fonction *Moyenne()* calcule la moyenne des valeurs objectives de tous les chromosomes obtenus à une génération donnée. La fonction *Trier_Population()* trie par ordre croissant des valeurs objectives toutes les solutions obtenues dans une génération donnée.

La fonction *Croisement()* réalise l'opération de croisement entre les éléments d'une population. Pour cela, on sélectionne deux éléments de façon aléatoire entre 1 et la taille de la population. On génère alors un lieu de croisement à l'intérieur du chromosome et une probabilité de croisement. On effectue le croisement si la probabilité générée est inférieure ou égale à la probabilité préfixée. Sinon, on effectue une inversion des deux chromosomes sélectionnés. On introduit les deux parents sélectionnés et les deux enfants obtenus dans une population que l'on nomme Population_En_Attente. On renvoie ensuite les deux chromosomes sélectionnés à la fin de la population et on réduit la taille de la population de deux pour ne plus sélectionner ces deux éléments au cours de la génération présente. On arrête quand tous les éléments de la population sont sélectionnés.

La fonction *Mutation()* réalise l'opération de mutation sur chacun des chromosomes de la population en attente obtenue. Pour ce faire, une probabilité de mutation est générée pour chaque chromosome. Si cette probabilité est inférieure à la probabilité de mutation préfixée, on choisit aléatoirement un lieu de mutation à l'intérieur du chromosome et on génère une nouvelle valeur pour le gène à ce lieu. Sinon, on réalise la même opération sur le chromosome suivant. La fonction *Selectionner_Nouvelle_Population_Roulette()* applique la roulette sur les éléments d'une population pour en déterminer ceux qui feront partie de la prochaine génération.

Pour cela, deux tableaux sont créés. Le premier contient les inverses de toutes les valeurs objectives des éléments de la population. On en fait alors la somme que l'on normalise à 1. On calcule ensuite dans le second tableau les proportions de ces éléments sur la base de normalisation à 1. Les éléments ayant une valeur objective peu élevée auront ainsi plus de chances d'être choisis.

- **Classe *Result***

Les attributs de cette classe sont : un objet de type *Chromosome* nommé *meilleur*, qui conserve le meilleur chromosome de la présente génération, une variable réelle *meilleur_cout* qui contient le coût du meilleur chromosome de la génération, deux variables entières *meilleure_generation* et *meilleur_cycle* qui contiennent la génération et le cycle auxquels la meilleure solution a été obtenue. Les méthodes de cette classe sont essentiellement des méthodes de sauvegarde. La fonction *Sauvegarde1()* conserve dans un fichier les différents paramètres d'une exécution. La fonction *Sauvegarde2()* conserve dans un fichier le meilleur élément du cycle courant. La fonction *Sauvegarde3()* conserve dans un fichier les valeurs objectives des meilleurs éléments de chaque génération.

5.3 Mise en œuvre de la méthode proposée

Afin de tester l'efficacité de l'adaptation que nous proposons, nous l'avons appliqué à un réseau de 3 commutateurs et 14 cellules. Le Tableau 5.1 montre les coûts des liaisons cellule-commutateur. Par exemple, la liaison entre la cellule 6 et le commutateur 3 a un coût de 1.73 unités.

De même, le Tableau 5.2 montre les volumes d'appels par cellule et les capacités maximales par commutateur pour le même réseau. Ainsi, la cellule 12 a un volume d'appel de 1 unité et le commutateur 2 a une capacité maximale de 8 unités.

Tableau 5.1 Coûts des liaisons cellule-commutateur

Cellule	Commutateur 1	Commutateur 2	Commutateur 3
1	0	2	2
2	1	1.73	1.73
3	1	1	2.65
4	1	1.73	3
5	1	2.65	2.65
6	1	3	1.73
7	1	2.65	1
8	1.73	1	2.65
9	1.73	1	3.61
10	2	0	3.46
11	2	2	2
12	1.73	2.65	1
13	2	3.46	0
14	1.73	3.61	1

Tableau 5.2 Volumes d'appel des cellules et capacités des commutateurs

Numéro	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Cellule	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Commutateur	7	8	7											

Les paramètres utilisés sont les suivants :

Taille maximale de la population	80
Nombre de générations d'un cycle génétique	40
Nombre de cycles d'un processus génétique	20
Probabilité de croisement	0.90
Probabilité de mutation	0.15

Une population initiale est formée selon le principe décrit à la section 4.3. Le premier élément de cette population, qui constituera une base de comparaison pour nous, est le schéma d'affectation quand toutes les cellules sont assignées au commutateur le plus proche. Comme le montre la figure 5.2, ce chromosome n'est pas faisable : il viole la contrainte sur la capacité des commutateurs, car le volume d'appels total affecté au

commutateur 1 est de 8 unités alors que sa capacité maximale est de 7 unités. Son évaluation est de 132 unités.

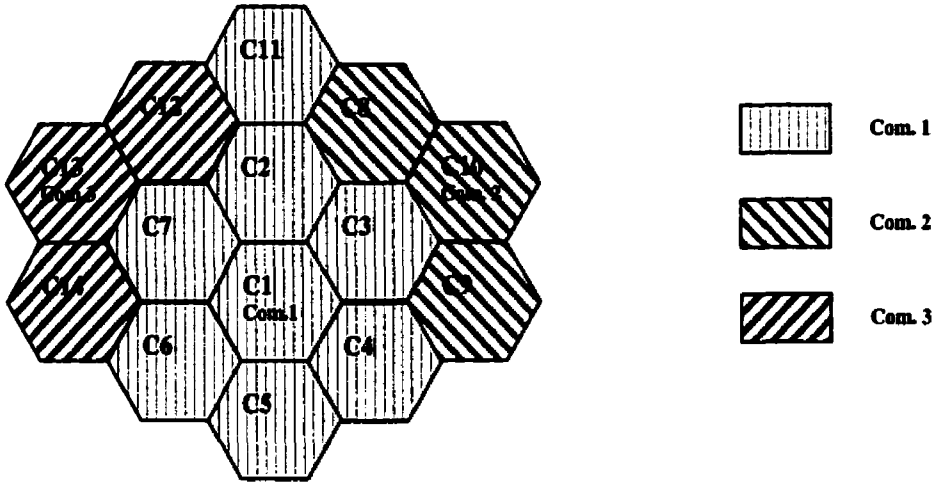

Figure 5.2 Solution initiale

Les coûts des relèves entre cellules adjacentes figurent au Tableau 5.3. On y retrouve un grand nombre de valeurs nulles car une cellule n'a que 6 voisins.

Tableau 5.3 Coûts des relèves entre cellules adjacentes

Cellule	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	3	8	3	18	15	7	0	0	0	0	0	0	0
2	3	0	1	0	0	0	4	2	0	0	4	1	0	0
3	8	1	0	6	0	0	0	5	10	1	0	0	0	0
4	3	0	6	0	6	0	0	0	7	0	0	0	0	0
5	18	0	0	6	0	13	0	0	0	0	0	0	0	0
6	15	0	0	0	13	0	6	0	0	0	0	0	0	11
7	7	4	0	0	0	6	0	0	0	0	0	9	2	4
8	0	2	5	0	0	0	0	0	0	11	3	0	0	0
9	0	0	10	7	0	0	0	0	0	4	0	0	0	0
10	0	0	1	0	0	0	0	11	4	0	0	0	0	0
11	0	4	0	0	0	0	0	3	0	0	0	5	0	0
12	0	1	0	0	0	0	9	0	0	0	5	0	7	0
13	0	0	0	0	0	0	2	0	0	0	0	7	0	6
14	0	0	0	0	0	11	4	0	0	0	0	0	2	0

Le meilleur chromosome faisable obtenu au cours des deux premiers cycles (voir Figure 5.3) a une évaluation de 142.61 unités, soit une augmentation de 8.04% par rapport à notre base de comparaison.

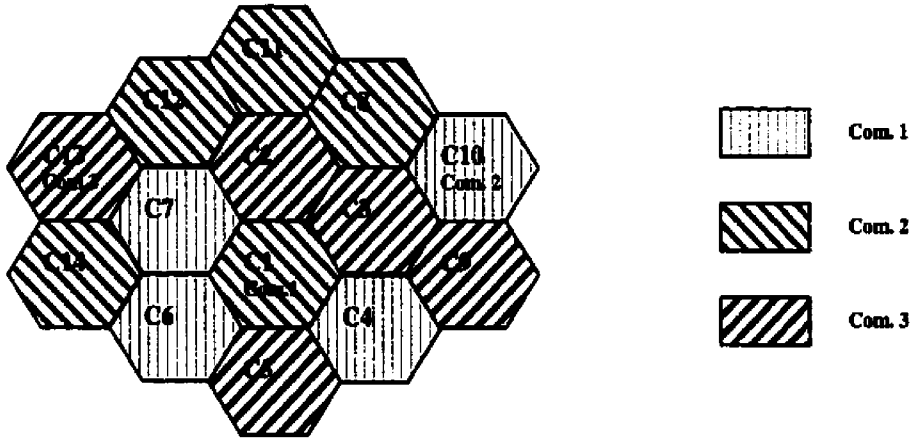

Figure 5.3 Première solution faisable obtenue

Du troisième au sixième cycle, on note une amélioration négligeable avec le meilleur chromosome qui est évalué à 142.03 unités, soit une amélioration de 0.5% par rapport à la première solution faisable obtenue et un écart de 7.6% par rapport à la base de comparaison. Aux cycles 7 et 8 respectivement, on obtient une évaluation de 141.86 unités et de 133.66 unités soit une amélioration de 0.6% et 6.43% par rapport à la première solution faisable obtenue. Du cycle 9 au cycle 13, la meilleure solution obtenue a une évaluation de 119.57 unités, soit une amélioration de 16.3% par rapport à la première solution faisable obtenue. Il faut remarquer que tous les commutateurs ont été utilisés pour les meilleures solutions obtenues jusque là. La meilleure solution du processus génétique est obtenue au quatorzième cycle. Son évaluation est de 75.92 unités, ce qui donne une amélioration de 46.85% par rapport à la première solution faisable obtenue et un écart de 42.48% par rapport à notre base de comparaison. La Figure 5.4 montre le schéma d'affectation correspondant à cette solution. Remarquons que, pour cette solution, seuls les commutateurs 1 et 2 sont utilisés.

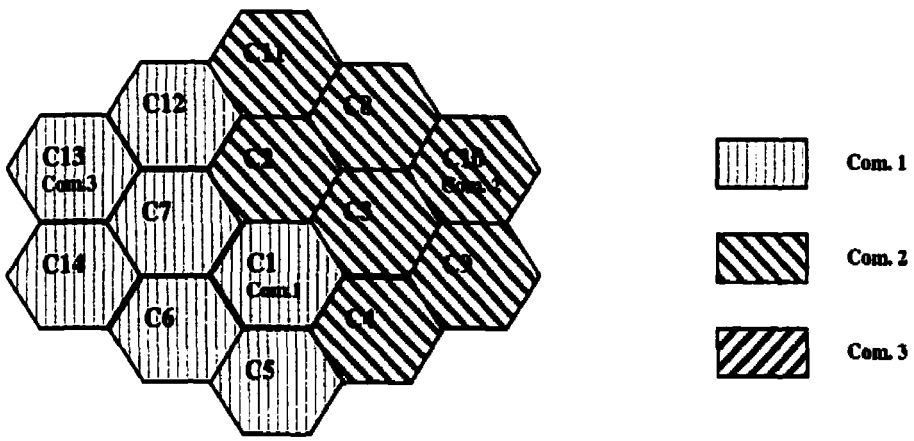

Figure 5.4 Meilleure solution obtenue au cours du processus génétique

Signalons toutefois qu'aucun chromosome n'a été repêché dans le cas de cet exemple, car la plus petite violation de la contrainte sur la capacité des chromosomes est de 12.5%, ce qui est supérieur à la limite de 10% que nous nous sommes fixée au départ.

Au vu des résultats précédents, nous pouvons affirmer que notre algorithme génétique se comporte bien. Dans le prochain chapitre, nous mettrons en œuvre une série de tests dans le but de prouver sa robustesse et l'influence des différents paramètres sur la qualité des solutions obtenues.

CHAPITRE VI
ANALYSE DES RÉSULTATS

Pour mesurer l'efficacité de l'algorithme que nous avons proposé et la qualité des solutions qu'il fournit, nous l'avons soumis à une série de tests intensifs. Dans ce chapitre, nous présenterons les résultats que nous avons obtenus à partir de la mise en œuvre de notre algorithme. Ensuite, nous effectuerons des analyses de sensibilité afin de déterminer l'influence de chaque paramètre génétique sur la qualité des résultats et la performance de l'algorithme proposé. Nous procèderons aussi à une analyse comparative de nos résultats avec ceux obtenus à partir d'autres méthodes telles que la recherche taboue, le recuit simulé, etc.

6.1 Génération des tests et environnement d'exécution

Les fichiers que nous avons utilisés ont été générés à partir du logiciel de calcul *Matlab®*. Nous avons 20 fichiers tests pour les réseaux de petite taille (15 cellules-2 commutateurs, 30 cellules-3 commutateurs), 15 fichiers tests pour les réseaux de taille moyenne (50 cellules-4 commutateurs, 100 cellules-5 commutateurs), et 5 fichiers pour les réseaux de grande taille (150 cellules-6 commutateurs, 200 cellules-7 commutateurs). Pour chaque niveau d'un paramètre, nous avons effectué 5 exécutions quant aux réseaux de petite taille, 3 exécutions quant à ceux de taille moyenne et 2 exécutions pour les réseaux de grande taille. Nous avons opéré ces choix compte tenu du temps d'exécution qui est assez élevé pour les réseaux de grande taille. Les expériences ont été réalisées sur un micro-ordinateur Pentium III IBM compatible, 450 MHz.

Les valeurs suivantes ont été retenues pour la configuration de base.

- Taille de la population 100
- Nombre de générations 40
- Nombre de cycles 20
- Probabilité de croisement 0.9
- Probabilité de mutation 0.08

Étant donné le grand nombre d'expériences, nous avons décidé de ne faire varier qu'un paramètre à la fois, les autres conservant leurs valeurs de référence. Nous avons aussi choisi de n'utiliser que quelques valeurs discrètes. L'inconvénient majeur de l'algorithme génétique est le temps de calcul. Dans l'environnement de calcul et de mise en œuvre considéré, ce temps varie d'environ 15 secondes pour un réseau de 15 cellules et 2 commutateurs, à environ 50 minutes pour un réseau de 200 cellules et 7 commutateurs, ce pour 20 cycles, le nombre de cycles étant le nombre de fois que nous répétons un problème avec des paramètres identiques. Les paramètres considérés pour ces temps de calcul ont les valeurs de référence. Une population de 100 chromosomes représente en réalité, 100 schémas d'affectation possibles pour lesquelles il faut déterminer le coût du réseau et le statut de la solution – faisable ou non. En répétant cela 40 fois par cycle pour les 40 générations et 20 fois par exécution pour les 20 cycles, nous obtenons un temps de calcul plutôt prohibitif.

6.2 Effet du nombre de générations

Nous avons voulu connaître l'influence du nombre de générations sur les résultats obtenus. Pour cela, nous avons suivi l'étalement des solutions obtenues en variant le nombre de générations. Le nombre de générations varie de 20 à 100 par intervalle de 20 unités. Comme le montre la Figure 6.1 qui porte sur l'étalement des meilleurs résultats obtenus par cycle en fonction du nombre de générations, l'écart entre les deux solutions extrêmes, c'est-à-dire la différence entre la solution ayant le coût le moins élevé et celle ayant le coût le plus élevé, augmente au fur et à mesure que le nombre de générations augmente. Ce phénomène s'observe pour les réseaux de moyenne et grande taille. Pour ces réseaux, l'écart considéré croît généralement jusqu'à 80 générations par cycle; parfois cette croissance s'observe jusqu'à 100 générations par cycle, mais souvent pour ce dernier cas, cet écart est légèrement réduit par rapport à celui du cycle comportant 80 générations. Par contre, pour les réseaux de petite taille, ce phénomène ne s'observe pas vraiment car l'intervalle des coûts des solutions est généralement assez réduit.

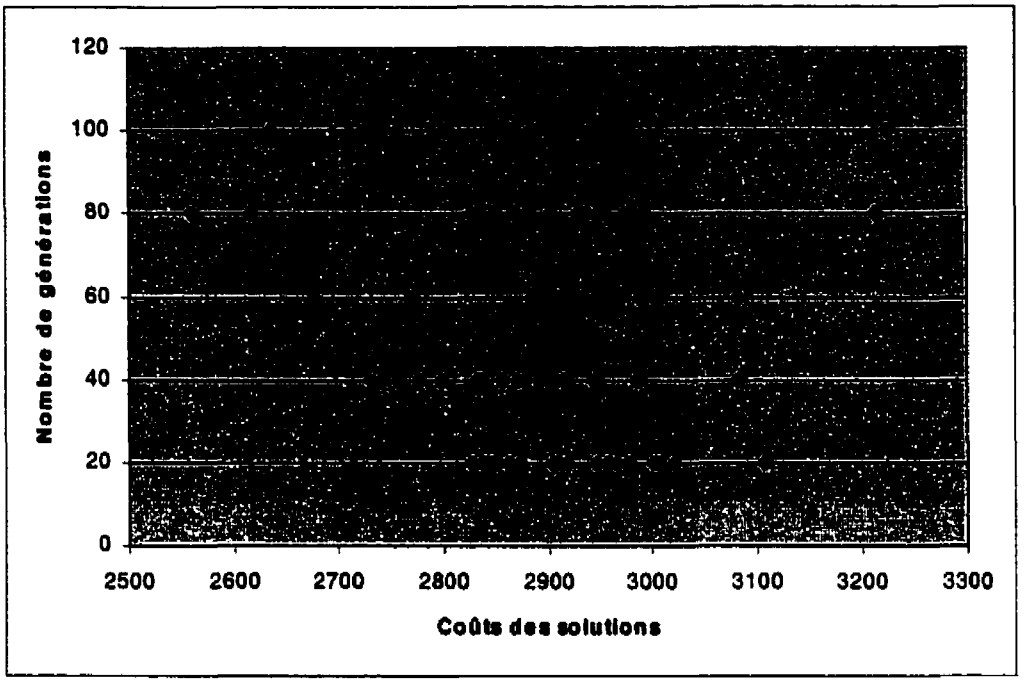

Figure 6.1 Dispersion des résultats selon le nombre de générations

6.3 Effet de la taille de la population

En général, la taille de la population n'a pas d'effet sur les réseaux, quelle que soit leur taille. Pour confirmer ces résultats, les courbes de la Figure 6.2 montrent la tendance que suivent les résultats obtenus pour différentes tailles de réseaux. Pour chaque taille de réseau, 5 fichiers ont été choisis. Ces courbes représentent les écarts types entre la meilleure solution obtenue et une borne inférieure représentant le schéma d'affectation obtenu à partir de la liaison des cellules au commutateur selon la métrique de la plus courte distance. Il faut néanmoins faire remarquer que cette borne inférieure est souvent non faisable.

Les courbes ci-dessous (Figure 6.2) montrent clairement que les meilleurs résultats peuvent être obtenus pour n'importe quelle taille de la population. Ce résultat est contre-intuitif car une grande taille de la population devrait offrir plus de chances de tomber sur une meilleure solution, étant donné que l'exploration porte sur un plus grand

nombre d'éléments. Souvent, pour les réseaux comportant 2 ou 3 commutateurs, la solution optimale se rapproche assez du premier élément de la population initiale - fut-il non faisable - celle où les cellules sont raccordées au commutateur le plus proche. Cette meilleure solution s'obtient en général à partir d'une légère modification. Ceci s'explique par le fait que le nombre de commutateurs est réduit et, par le jeu des altérations dues aux opérateurs génétiques, la probabilité de tomber rapidement sur une bonne solution augmente, même quand la taille de la population est petite. Pour les plus grands réseaux, le facteur du hasard peut entraîner l'obtention de meilleures solutions pour n'importe taille de la population. Mais intuitivement une grande taille de la population offrirait plus de chances de tomber sur la meilleure solution très rapidement.

Soulignons que la dégradation de la solution que l'on observe parfois avec l'augmentation de la taille de la population est surtout due au fait que toutes les opérations du processus génétique (création d'une population initiale, application des opérateurs, sélection des chromosomes pour une nouvelle population) sont basées sur le facteur incontrôlable du hasard. Néanmoins, pour quelques cas, l'écart type diminue au fur et à mesure que la taille de la population augmente. Ceci s'explique par le fait que notre algorithme n'a aucun contrôle sur les populations initiales. Elles sont totalement aléatoires et le seul lien entre elles est le premier élément qui partout est identique. De plus, de mauvais croisements ou mutations peuvent survenir à n'importe quel moment du processus génétique, ce qui pourrait dégrader la solution.

Il faut par ailleurs noter que, de façon générale, pour les grands réseaux, la courbe des valeurs objectives des meilleures solutions décroît plus rapidement quand la taille de la population augmente (Figure 6.3). Intuitivement, on pourrait alors conclure que pour une grande taille de la population, la réduction du nombre de cycles entraînerait l'obtention de résultats similaires.

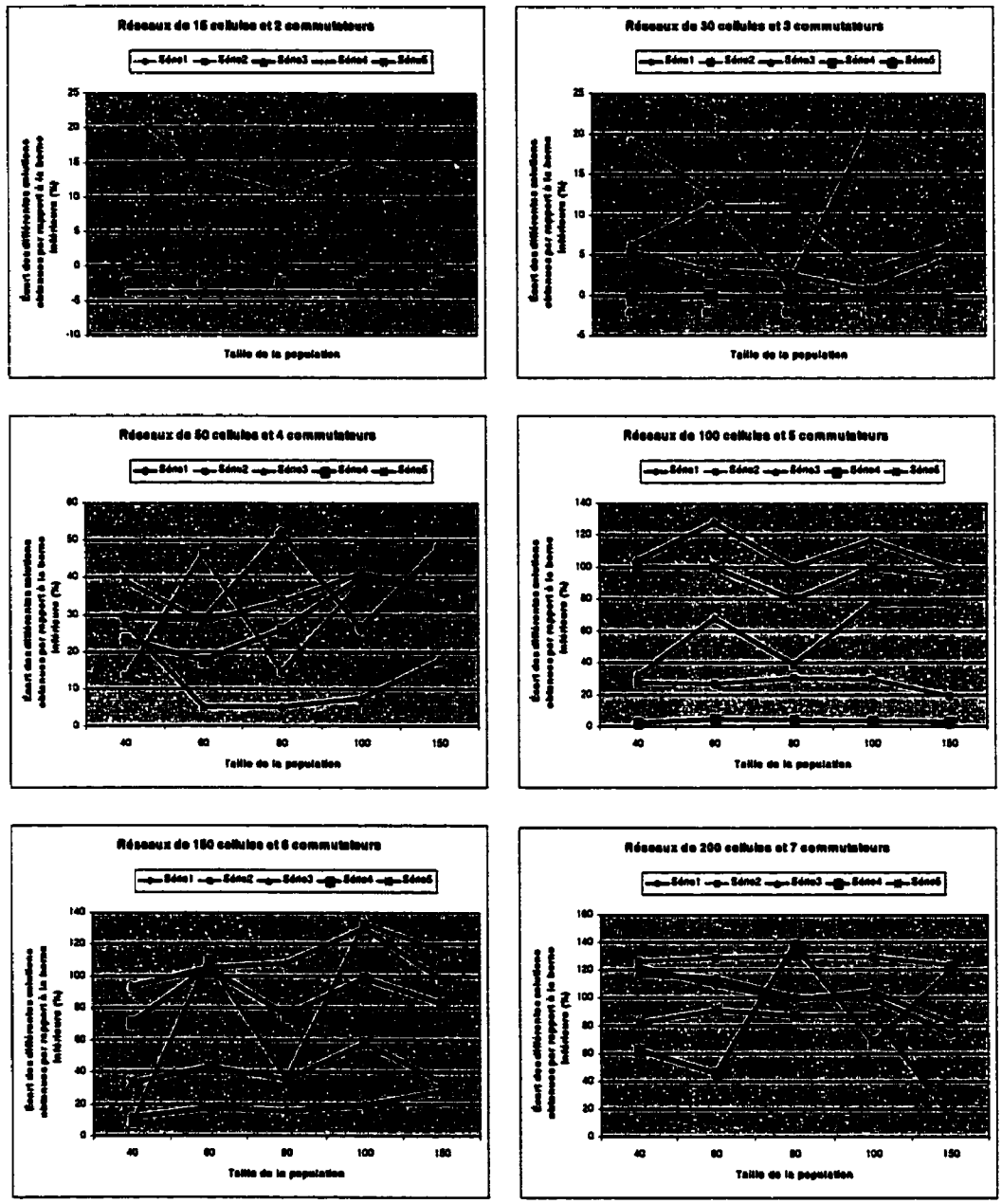

Figure 6.2 Écarts types entre la meilleure solution et la borne inférieure en fonction de la taille de la population

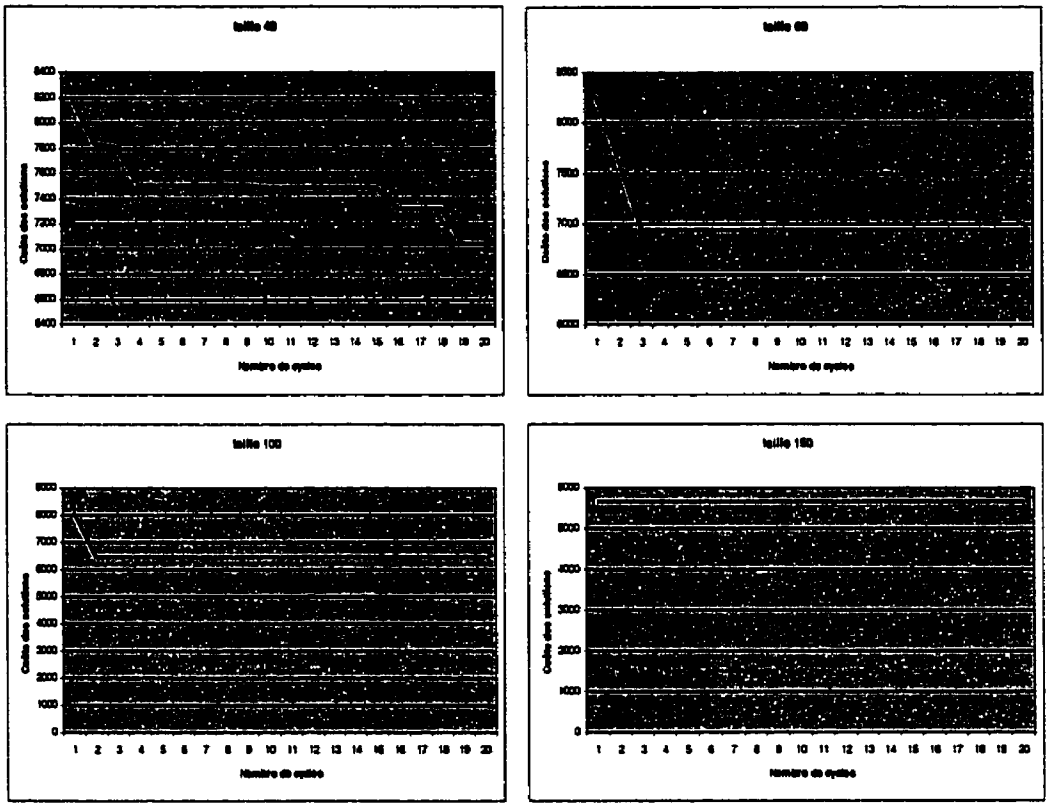

Figure 6.3 Valeurs objectives des meilleures solutions en fonction du nombre de cycles

6.4 Effet des probabilités associées aux différents opérateurs génétiques

La première série d'expériences a consisté à faire varier la probabilité de croisement, les autres paramètres étant fixes. Afin de connaître l'effet de ce paramètre sur les résultats obtenus, nous avons effectué trois séries de tests qui concernent respectivement les réseaux de petite taille (15 cellules – 2 commutateurs, 30 cellules – 3 commutateurs), les réseaux de taille moyenne (50 cellules – 4 commutateurs, 100 cellules – 5 commutateurs) et les réseaux de grande taille (150 cellules – 6 commutateurs, 200 cellules – 7 commutateurs). Pour chacun de ces réseaux, nous avons choisi 5 fichiers de façon arbitraire sur lesquels nous avons basé nos analyses. Les

valeurs de probabilité de croisement considérées sont 0.6, 0.7, 0.8, 0.9, 0.95. Les valeurs des autres paramètres demeurent celles de référence. Nous avons répété 5 fois l'exécution de chaque problème et ce sont les moyennes qui sont représentées à la Figure 6.4. À partir de celle-ci, nous pouvons remarquer qu'une probabilité de croisement de 90% fournit en général de meilleurs résultats surtout pour les réseaux de moyenne et grande taille. Il faut rappeler ici que nous avions opté pour un mode de sélection qui favorise l'émergence tant des parents jugés performants que des chromosomes enfants résultant du croisement de deux parents. Ce mode de sélection se rapproche ainsi du principe du meilleur des deux mondes.

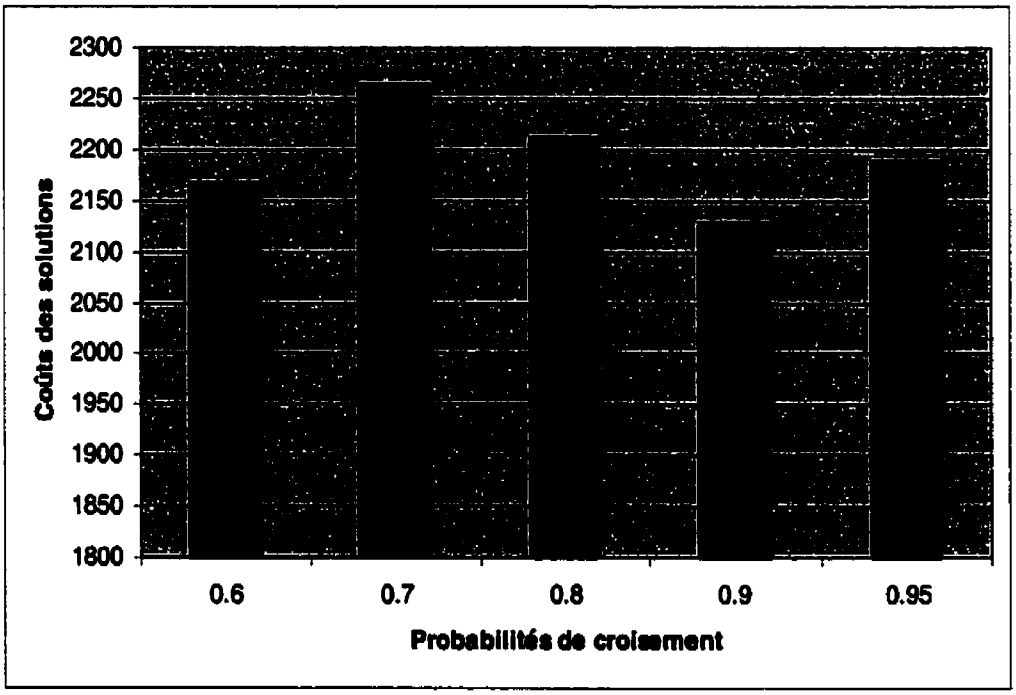

Figure 6.4 Coûts du réseau en fonction des probabilités de croisement

Pour les réseaux de petite taille, les valeurs des coûts demeurent dans un intervalle de coûts très réduit; ceci ne permet pas de déduire une tendance par rapport aux probabilités de croisement.

La Figure 6.5, quant à elle, représente la dispersion des résultats.

- **Réseaux de petite taille**

Pour chaque fichier, nous avons calculé la moyenne des valeurs objectives obtenues pour 5 exécutions. À partir de ces données, nous avons tracé la courbe des valeurs obtenues en fonction de la probabilité de croisement. On observe en général un plus grand étalement des résultats pour les probabilités de croisement de 0.9 et de 0.95. Aussi, les meilleures valeurs objectives ont été obtenues avec ces valeurs de probabilité de croisement. Il faut toutefois noter que les valeurs obtenues, quelle que soit la valeur de la probabilité, demeurent dans le même intervalle et sont très peu dispersées.

- **Réseaux de taille moyenne**

Pour chaque fichier, nous avons calculé la moyenne des valeurs objectives obtenues pour 3 exécutions. À partir de ces données, nous avons tracé la courbe des valeurs obtenues en fonction de la probabilité de croisement. En général, les valeurs obtenues se situent pour la plupart dans le même intervalle. Mais la tendance observée pour les réseaux de petite taille se confirme pour les réseaux de taille moyenne. Ainsi, on remarque une plus grande dispersion pour les probabilités de croisement de 0.9 et parfois 0.95. Aussi, note-t-on que les meilleures valeurs de la série sont souvent obtenues à ces valeurs de probabilité.

- **Réseaux de grande taille**

Pour chaque fichier nous avons calculé la moyenne des valeurs objectives obtenues pour 2 exécutions. À partir de ces données, nous avons tracé la courbe des valeurs obtenues en fonction de la probabilité de croisement. On note des valeurs assez groupées, mais la tendance observée pour les réseaux de petite ou moyenne taille se confirme pour les réseaux de grande taille. Ainsi donc, comme le montre la Figure 6.5 plus la probabilité de croisement augmente plus les valeurs obtenues sont dispersées.

Somme toute, plus la probabilité de croisement est grande, plus les valeurs obtenues sont dispersées. Ceci s'explique, car une forte probabilité de croisement permet d'avoir une bonne diversité dans les solutions obtenues et d'effectuer ainsi une plus grande exploration de l'espace de recherche, d'où une grande dispersion des valeurs obtenues.

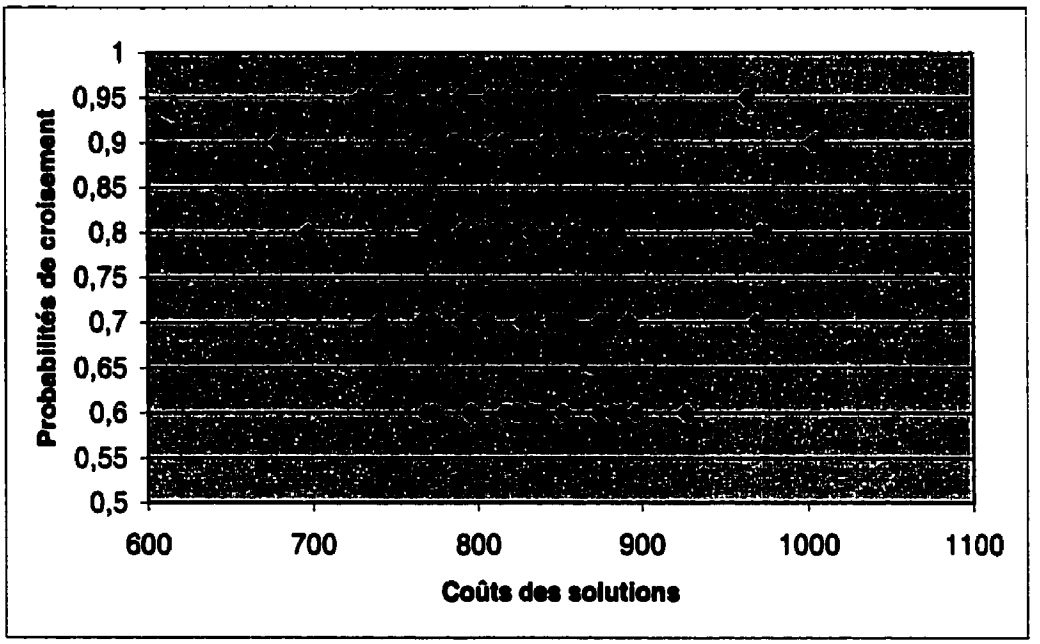

Figure 6.5 Dispersion des résultats en fonction de la probabilité de croisement

La deuxième série de tests a consisté à faire varier la probabilité de mutation, les autres paramètres conservant leur valeur de référence. Ces tests visaient à déterminer la meilleure valeur de probabilité de mutation. Afin de connaître l'effet de ce paramètre sur les résultats obtenus, nous avons aussi effectué trois séries de tests qui concernent respectivement les réseaux de petite taille (15 cellules – 2 commutateurs, 30 cellules – 3 commutateurs), les réseaux de taille moyenne (50 cellules – 4 commutateurs, 100 cellules – 5 commutateurs) et les réseaux de grande taille (150 cellules – 6 commutateurs, 200 cellules – 7 commutateurs). Pour chacun de ces réseaux, nous avons

choisi 5 fichiers de façon arbitraire sur lesquels nous avons basé nos analyses. Les valeurs de probabilité de croisement considérées sont 0.05, 0.1, 0.15, 0.2. Les valeurs des autres paramètres demeurent celles de référence. Nous avons répété 5 fois l'exécution de chaque problème et ce sont les moyennes qui sont représentées à la Figure 6.6. À partir de celle-ci, nous pouvons remarquer qu'une probabilité de croisement de 5% fournit en général de meilleurs résultats, surtout pour les réseaux de moyenne et grande taille. Ce résultat est prévisible car, pour une probabilité de 5%, très peu de mutants seront créés, ce qui empêche l'exploration des solutions de s'orienter vers les zones peu favorables. Notons qu'une probabilité de 10% fournit les moyennes de coût les plus élevées.

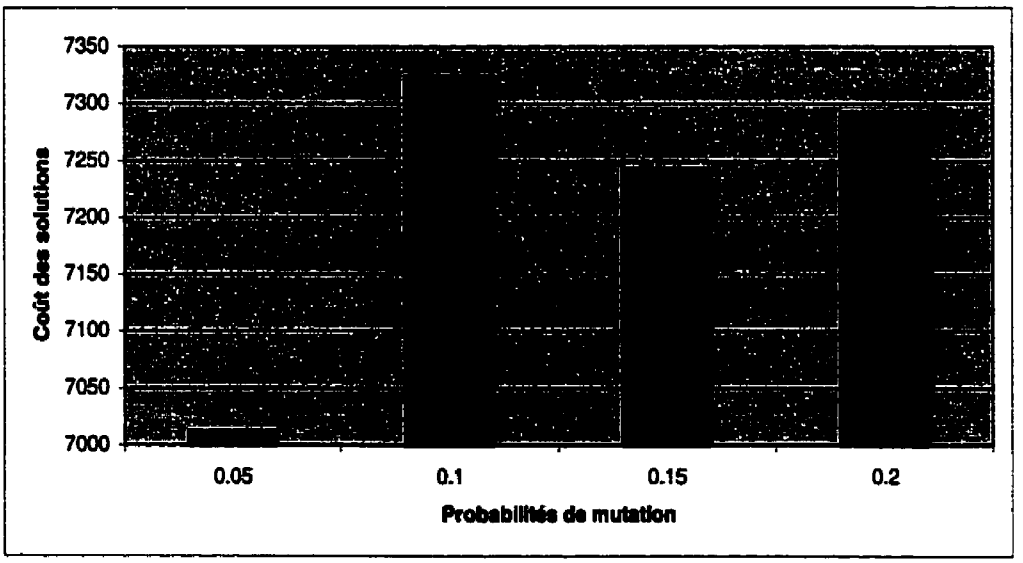

Figure 6.6 Coûts du réseau en fonction de la probabilité de mutation

En général, les résultats obtenus sont plus étalés pour les probabilités de mutation de 15% et 20%. Mais comme le montre la Figure 6.7, on note la présence de quelques rares mutants qui souvent ont des valeurs extrêmes.

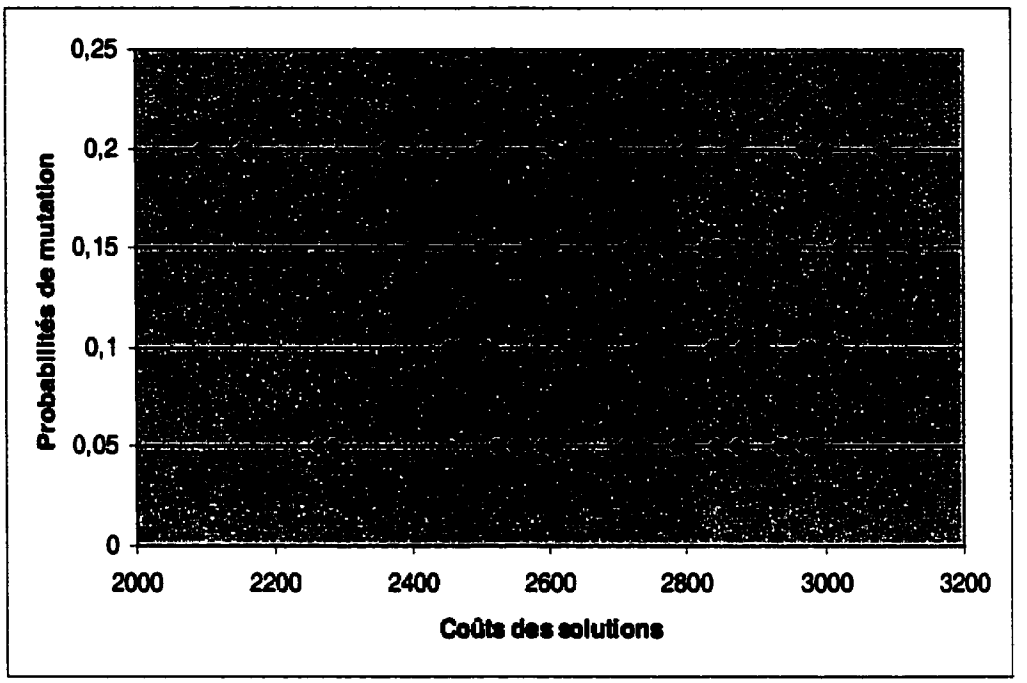

Figure 6.7 Dispersion des résultats en fonction de la probabilité de mutation

L'opérateur d'inversion constitue un opérateur complémentaire à celui de croisement et est seulement appliqué en l'absence de ce dernier. L'analyse de l'influence de ce paramètre sur les résultats obtenus se déduit donc facilement de celle effectuée pour la probabilité de croisement.

6.5 Comparaison avec d'autres méthodes

Nous avons procédé à une étude comparative des résultats obtenus avec les AG et ceux obtenus par l'application de trois méthodes qui ont toutes été appliquées au problème d'affectation de cellules à des commutateurs dans les réseaux de communication personnelle. Ces méthodes sont l'heuristique proposée par Beaubrun, Pierre et Conan (1999) que nous noterons HB, la méthode du recuit simulé, la méthode de la recherche taboue.

Nous avons d'abord effectué une comparaison de la recherche taboue, du recuit simulé et des AG. Nous avons effectué les tests sur deux séries de données. Les résultats figurent dans les tableaux 6.1 et 6.2.

Tableau 6.1 Pourcentage d'amélioration de AG par rapport à RT et SA (nombre variable de commutateurs)

#cell.	#comm.	AG	RT	SA	% d'amélioration de AG par rapport à RT	% d'amélioration de AG par rapport à SA
15	2	114	118	123	3.51	7.89
30	3	394	382	405	-3.05	2.79
50	4	697	580	851	-16.79	22.09
100	5	2265	1374	1999	-39.34	-11.74
150	6	4980	2013	4271	-59.58	-14.24
200	7	3721	2768	7801	-25.61	109.65

Tableau 6.2 Pourcentage d'amélioration de AG par rapport à RT et SA (nombre fixe de commutateurs)

#cell.	#comm.	AG	RT	SA	% d'amélioration de AG par rapport à RT	% d'amélioration de AG par rapport à SA
15	3	133	124	139	-6.77	4.51
20	3	238	211	189	-11.34	-20.59
30	3	395	364	369	-7.85	-6.58
40	3	424	420	611	-0.94	44.1
50	3	600	588	748	-2.00	24.67
60	3	917	713	832	-22.25	-9.27

Nous avons ensuite effectué une comparaison de la recherche taboue, de l'heuristique proposée par Beaubrun, Pierre et Conan (1999) et des AG. Nous avons également effectué les tests sur deux séries de données.

Les résultats figurent dans les tableaux 6.3 et 6.4.

Tableau 6.3 Pourcentage d'amélioration de AG par rapport à RT et HB (nombre variable de commutateurs)

#cell.	#comm.	% d'amélioration de AG par rapport à RT	% d'amélioration de AG par rapport à HB
15	2	3.51	34.51
30	3	-3.05	32.95
50	4	-16.79	25.21
100	5	-39.34	10.88
150	6	-59.58	-3.58
200	7	-25.61	33.39

Tableau 6.4 Pourcentage d'amélioration de AG par rapport à RT et HB (nombre fixe de commutateurs)

#cell.	#comm.	% d'amélioration de AG par rapport à RT	% d'amélioration de AG par rapport à HB
15	3	-6.77	27.23
20	3	-11.34	22.66
30	3	-7.85	26.15
40	3	-0.94	37.06
50	3	-2.00	36
60	3	-22.25	16.75

Nous avons résumé les résultats de cette étude comparative dans les tableaux 6.5 et 6.6. Ces derniers nous permettent d'apprécier les améliorations de l'adaptation des AG par rapport aux différentes méthodes sus-citées.

Tableau 6.5 Pourcentage d'amélioration de AG par rapport à RT, SA et HB (nombre variable de commutateurs)

#cell.	#comm.	% d'amélioration de AG par rapport à RT	% d'amélioration de AG par rapport à SA	% d'amélioration de AG par rapport à HB
15	2	3.51	7.89	34.51
30	3	-3.05	2.79	32.95
50	4	-16.79	22.09	25.21
100	5	-39.34	-11.74	10.88
150	6	-59.58	-14.24	-3.58
200	7	-25.61	109.65	33.39

Tableau 6.6 Pourcentage d'amélioration de AG par rapport à RT, SA et HB (nombre fixe de commutateurs)

#cell.	#comm.	% d'amélioration de AG par rapport à RT	% d'amélioration de AG par rapport à SA	% d'amélioration de AG par rapport à HB
15	3	-6.77	4.51	27.23
20	3	-11.34	-20.59	22.66
30	3	-7.85	-6.58	26.15
40	3	-0.94	44.1	37.06
50	3	-2.00	24.67	36
60	3	-22.25	-9.27	16.75

À partir des tableaux 6.5 et 6.6, nous avons déduit les figures 6.8 et 6.9. Ces deux graphiques nous permettent de tirer quelques conclusions :

1. Comme le montre la Figure 6.8, la méthode de recherche taboue fournit de meilleurs résultats que celle des AG, quelle que soit la taille du réseau. En particulier, cette amélioration va de 3.05% pour les réseaux de 30 cellules et 3 commutateurs à 59.58% pour les réseaux de 150 cellules et 6 commutateurs. Cela confirme la tendance notée plus haut que l'adaptation des AG se révèle plus efficace pour les réseaux de petite taille.
2. Les résultats fournis par les AG sont toujours meilleurs à ceux obtenus par l'application de l'heuristique HB, comme le montrent les figures 6.8 et 6.9. Sur cette dernière, on remarque que les améliorations sont en général supérieures à 20% pour les réseaux ayant 3 commutateurs et un nombre de cellules inférieur ou égal à 60.
3. En général, les AG fournissent de meilleurs résultats par rapport au recuit simulé, mais parfois les résultats obtenus sont dégradés par rapport à ceux du recuit simulé comme le montre la Figure 6.8 où les solutions fournies par les AG sont moins bonnes que celles fournies par le recuit simulé pour les réseaux de 100 et 150 cellules.

En tenant compte de la performance des différentes méthodes utilisées, la classification nous donnerait donc l'ordre suivant : la méthode de la recherche taboue,

suivie de l'adaptation des algorithmes génétiques, de la méthode du recuit simulé et enfin l'heuristique proposée par Beaubrun, Pierre et Conan (1999).

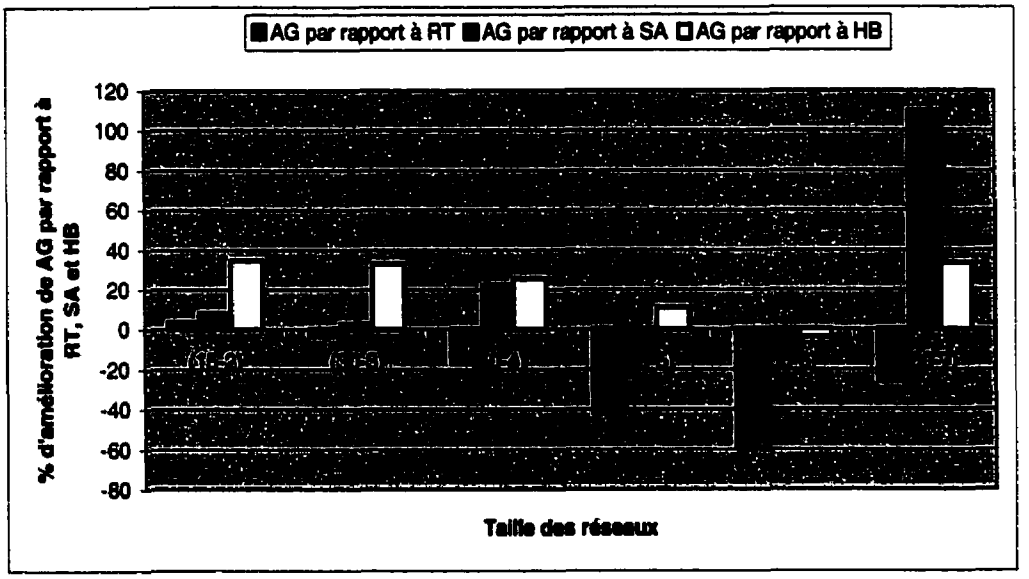

Figure 6.8 Étude comparée des résultats obtenus pour des réseaux de taille variable

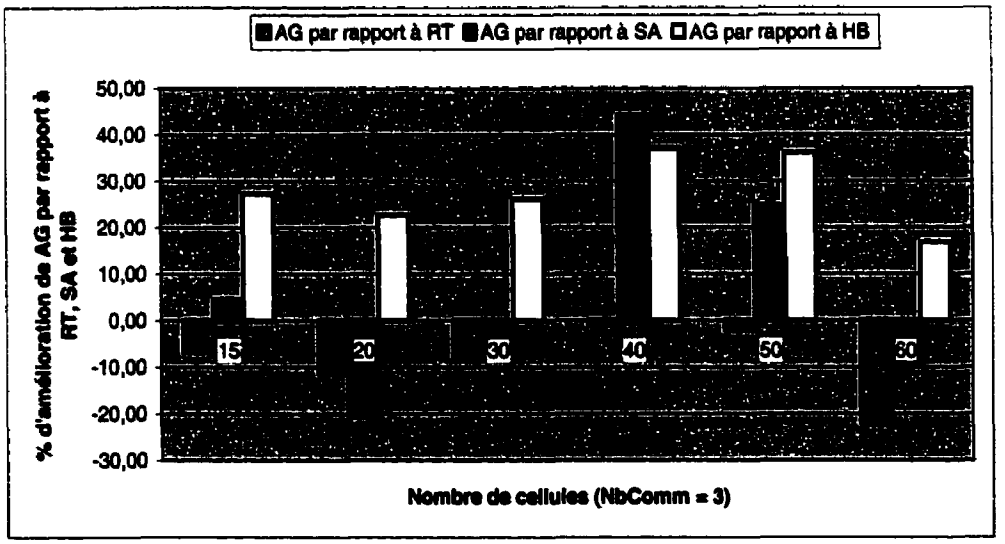

Figure 6.9 Étude comparée des résultats obtenus pour des réseaux de 3 commutateurs et de n cellules ($n \leq 60$)

6.6 Version parallèle de l'algorithme génétique

Dans le but de réduire le temps d'exécution tout en augmentant le nombre d'exécutions, nous présentons ici une version parallèle de notre algorithme génétique pour la résolution du problème d'affectation des cellules aux commutateurs dans les réseaux de communication personnelle.

6.6.1 Présentation de la structure des programmes parallèles

Une machine parallèle peut être considérée comme une agglomération d'unités de traitement semblables à celles que l'on rencontre dans une machine classique. Elle se distingue fondamentalement d'une machine séquentielle par son réseau d'interconnexion, indispensable pour faire coopérer les unités entre elles.

Les techniques utilisées pour l'exploitation du parallélisme dans les machines parallèles sont appelées *sources*. On distingue 3 sources principales :

- le parallélisme de contrôle ;
- le parallélisme de données ;
- le parallélisme de flux.

Le *parallélisme de contrôle* provient du fait qu'une application informatique est constituée de tâches qui peuvent être exécutées en même temps et de manière plus ou moins indépendante sur différentes unités de calcul appelées *processeurs*. Dans le cas où les actions sont indépendantes, il suffit d'associer un processeur à chacune d'entre elles pour obtenir un gain de temps linéaire. Une dépendance intervient lorsqu'une action doit être terminée pour qu'une autre commence. On distingue deux cas de dépendance :

- la dépendance de contrôle de séquence qui correspond au séquencement dans un algorithme classique ;
- la dépendance de contrôle de communication dans le cas où une action envoie des informations à une autre.

L'exploitation du parallélisme de contrôle consiste à gérer les dépendances entre les tâches d'une application pour obtenir une allocation des ressources aussi efficace que possible.

Le *parallélisme de données* fait référence au fait que la même action est répétée régulièrement sur plusieurs données. On associe donc plusieurs processeurs qui exécutent cette tâche de manière simultanée sur plusieurs données souvent vectorielles. Mais, lorsqu'une donnée scalaire intervient dans le système, l'algorithme parallèle perd en efficacité car plusieurs processeurs restent inactifs.

Le *parallélisme de flux* provient du fait que certaines applications fonctionnent selon le mode de travail à la chaîne. Ce mode d'exécution est aussi appelé *pipeline*. On dispose d'un flux de données, généralement similaires, sur lesquelles on doit effectuer une suite d'opérations en cascade. Le flux de données peut provenir de deux sources :

- données de type vectoriel placées en mémoire ;
- données de type scalaire provenant d'un dispositif d'entrée; cette source peut être considérée comme infinie pendant la durée de l'application.

Cette distribution de travail possède certains inconvénients. En effet, lors de l'initialisation du travail, certains processeurs sont inactifs. Il en est de même à la fin du travail (le premier processeur aura terminé avant les autres, il sera donc inactif pendant que les autres achèvent). De plus, Si un processeur travaille plus lentement (supposons que le travail ne soit pas bien réparti; que sa tâche soit plus complexe), il va ralentir tout le travail. Supposons qu'un processeur soit plus lent, il va forcer les autres à travailler à son rythme.

Plusieurs paramètres entrent en jeu dans la conception d'un algorithme parallèle. Le paramètre de *contrôle* fait référence à la présence ou à l'absence d'une unité globale de contrôle. Les architectures parallèles de contrôle de flux avec seulement une unité de contrôle appartiennent à la classe des SIMD (Single Instruction Multiple Data), alors que les systèmes avec plusieurs unités de contrôle (généralement un par processeur) appartiennent à la classe des MIMD (Multiple Instruction Multiple Data).

La *synchronisation* réfère à la présence ou à l'absence d'une horloge centrale utilisée pour synchroniser les opérations entre processeurs. Un système est dit *synchrone* quand il n'y a qu'une seule horloge, et *asynchrone* quand il y a plusieurs horloges

(généralement un par processeur). Les ordinateurs SIMD sont synchrones par définition alors que les MIMD sont pour la plupart asynchrones.

La *granularité* d'un système indique la quantité de données manipulable par chaque processeur du système. Dans les systèmes à *faible granularité*, on dispose d'un grand nombre de processeurs de faible puissance, ne pouvant manipuler qu'une petite quantité de données, tandis que les systèmes à *forte granularité* sont caractérisés par la présence d'un nombre restreint de processeurs très puissants, pouvant traiter simultanément de grandes quantités de données.

Le paramètre de la *communication* fait référence à la façon dont l'information est échangée entre processeurs. Deux possibilités majeures existent: les processeurs peuvent écrire et lire dans une zone mémoire commune accessible à tous dans le cas des *systèmes à mémoire partagée*, ou peuvent échanger des messages dans le cas des *systèmes à passage de messages*.

6.6.2 Topologies de « *cluster* »

Il est rare que les différents processus qui composent un programme parallèle soient complètement indépendants. Ils se doivent de communiquer entre eux pour accomplir leur travail. La fonction de communication peut se faire de deux manières :

- par le partage d'une ressource : donnée commune en mémoire commune ;
- par la copie d'une information et son expédition : envoi de message.

Dans le premier cas, il faut relier physiquement les organes parallèles à la mémoire partagée. Dans le deuxième cas, il faut relier physiquement les processeurs entre eux pour pouvoir réaliser l'envoi de messages. On constate donc que toutes les machines parallèles ont besoin d'un réseau de communication qui réalise l'échange d'information.

On distingue deux grandes classes de réseaux dans les machines parallèles : les réseaux statiques et les réseaux dynamiques. Les réseaux statiques sont ceux dont la topologie est définie une fois pour toutes, lors de la construction de la machine parallèle. Ils sont surtout utilisés dans les machines à passage de messages. Dans ce cas, le réseau relie les processeurs entre eux en formant un réseau d'intercommunication. Les

processeurs s'envoient des messages : l'information de la source est copiée puis routée vers la destination en passant par les dispositifs d'entrées-sorties et le réseau; on dit que *le couplage est faible*. On demande peu de débit au réseau car les communications ne concernent que les échanges d'informations entre les différentes tâches. Chaque processeur dispose d'une mémoire locale.

Les réseaux dynamiques sont ceux dont la topologie peut varier au cours de l'exécution du programme parallèle. Ils sont souvent utilisés dans les multiprocesseurs à mémoire partagée et on parle de réseau d'interconnexion. Les processeurs sont tous reliés à la mémoire partagée et doivent obligatoirement faire passer leurs données à travers le réseau; on dit que *le couplage est fort*.

6.6.3 Considérations topologiques

Le choix et l'implantation d'une topologie dépendent en partie de la complexité des traitements qui composent le programme parallèle à exécuter. Le Tableau 6.7 décrit les principaux traitements qui composent celui que nous avons à implémenter et leur complexité.

Tableau 6.7 Tâches du programme parallèle et leur complexité

Tâches	Complexité
Génération d'une population initiale	n^2+mn
Croisement des éléments de la population	n^2+m
Mutation des éléments de la population	constante
Duplication des éléments d'une population	$m+n$
Évaluation des coûts des éléments de la population	mn^2
Trier les éléments d'une population	$m+n$
Évaluation des capacités des éléments de la population	$m+n$
Sélection des éléments d'une nouvelle population	$m+n$

Dans le tableau précédent, *n* désigne le nombre de cellules et *m* le nombre de commutateurs. M est de l'ordre d'une dizaine ou moins tandis que n peut atteindre quelques centaines.

Chaque traitement fournit au traitement suivant la donnée d'entrée sur laquelle il effectuera ses opérations. Pour chaque traitement, cette donnée est une population de chromosomes. Ces populations étant différentes les unes des autres, elles constituent ainsi des ressources locales pour chaque processeur. Les tampons dans lesquels seront conservés ces données locales auront la taille des différentes populations. Les données globales que constituent le nombre de cellules et le nombre de commutateurs seront propagées dans tous les processeurs avant le début de l'exécution du programme. Il n'y aura donc pas de donnée commune partagée. On aura alors un réseau statique d'intercommunication entre les processeurs qui formeront le système parallèle.

Étant donné que les traitements s'effectueront en chaîne sur la même population (même si elle est modifiée), on peut en générer plusieurs jusqu'à ce que la toute première population insérée dans le système passe par tous les processeurs. Ces différentes populations initiales générées feront le tour des processeurs pendant un nombre de générations préfixé. Ainsi, le système d'intercommunication le plus adéquat pour l'architecture de notre programme se révèle être un *réseau de connexion en anneau* avec mémoire locale à chaque processeur. À partir des complexités des différents traitements et pour assurer un certain équilibre entre les charges des différents processeurs, nous avons effectué une allocation des tâches et un réseau d'interconnexion comme le montre la Figure 6.10.

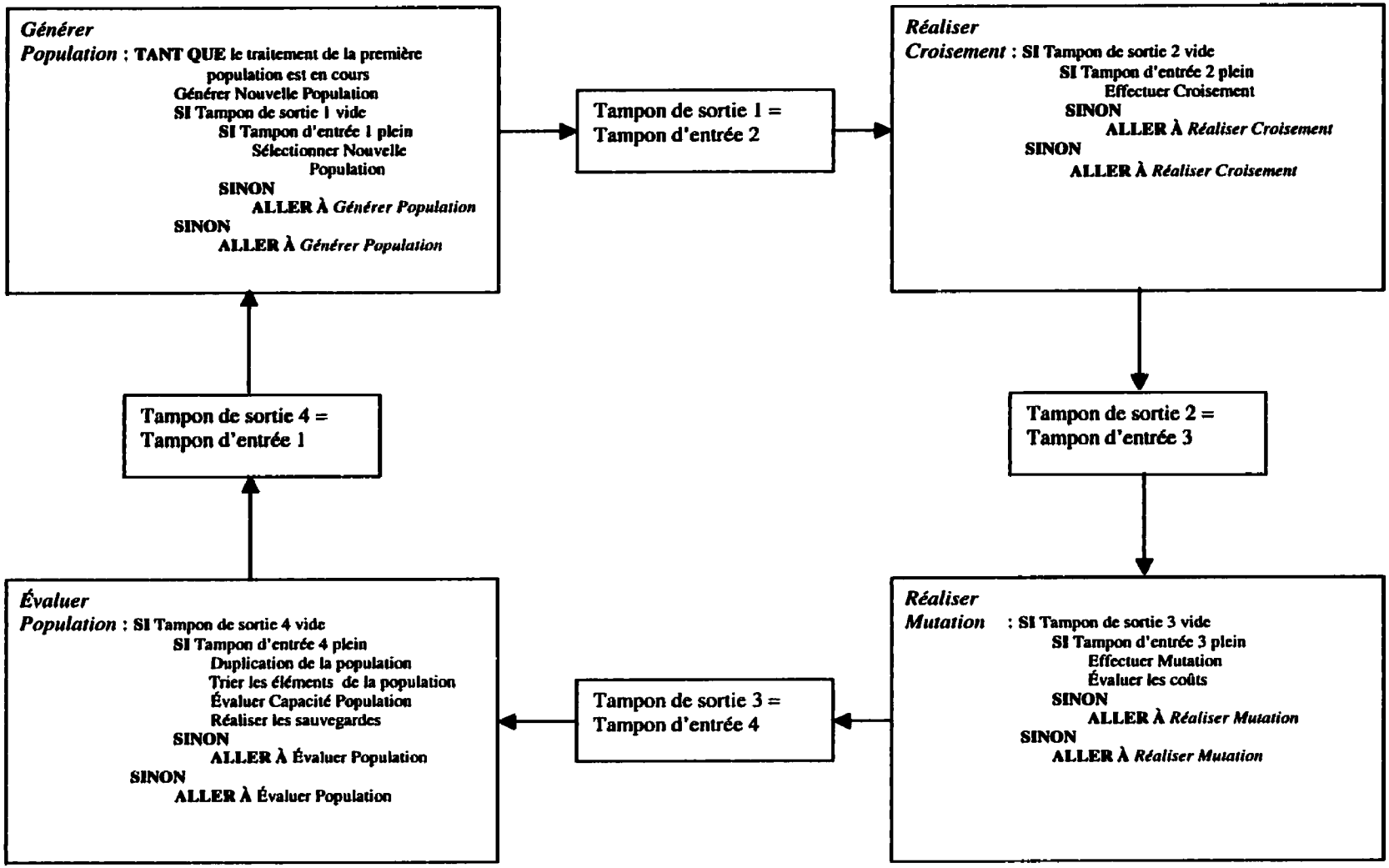

Figure 6.10 Schéma général du système d'entrées-sorties – Communication entre processus

6.6.4 Impact du nombre de processeurs dans l'anneau

En général, dans un réseau d'intercommunication en anneau, le nombre de processeurs est un facteur très important car un grand nombre de processeurs peuvent être inactifs à un instant donné. Dans notre cas, comme le montre le Tableau 6.7, seulement trois traitements ont une complexité en n^2. Tous les autres ont une complexité de $m+n$. Cela nous amène à dire que, outre ces trois traitements, tous les autres mettront moins de temps pour compléter leur exécution. Comme nous pouvons le constater dans la Figure 6.10, les trois traitements qui ont les temps d'exécution les plus élevés sont répartis sur trois processeurs différents. Tous les autres processeurs que l'on ajoutera si l'on décide de partitionner encore plus notre programme parallèle seront inactifs la plupart du temps. Dans ces conditions, nous pensons que le nombre de quatre processeurs est optimal pour la structure parallèle que nous souhaitons implanter. Le Tableau 6.2 montre la répartition des différentes tâches entre les quatre processeurs.

Tableau 6.8 Répartition des tâches entre les quatre processeurs

# du processeur	Tâches	Complexité
1	- Génération d'une population initiale ou - Sélection des éléments d'une nouvelle population	n^2+mn *ou* $m+n$
2	- Croisement des éléments de la population	n^2+m
3	- Mutation des éléments de la population - Évaluation des coûts des éléments de la population	*Constante+* $m+n$
4	- Duplication des éléments d'une population - Tri des éléments d'une population - Évaluation des capacités des éléments de la population	$mn^2+2m+2n$

6.6.5 Gain de performance par rapport à la version séquentielle

Afin de quantifier le gain obtenu en parallélisant notre algorithme, nous avons calculé le temps de traitement en nombre d'opérations pour le programme séquentiel et pour le programme parallèle.

Pour le traitement séquentiel, soit S le temps de traitement et g le nombre de générations; on a :

$$S = n^2+mn+g[n^2+mn^2+5m+4n+1]-m-n$$

$$Soit\ K = n^2+mn^2+5m+4n+1;$$

K représente le temps d'application des opérateurs et d'évaluation des populations et on peut faire l'approximation :

$$K = n^2+mn^2+5m$$

On a alors : $S = n^2+mn+gK-(m+n)$

En général n»m, on peut alors négliger (m+n) par rapport au reste de l'expression et on obtient :

$$S = n^2+mn+gK$$

Pour le traitement parallèle, on suppose que g_i est le nombre de populations initiales générées avant que la première ne revienne sous forme de population sélectionnée et P le temps de traitement. La quantité gi sera alors le rapport entre le temps d'une exécution et le temps du traitement le plus long. On a alors :

$$g_i = \left\lceil \frac{K}{mn^2} \right\rceil$$

Pour effectuer la comparaison, il faudrait calculer le temps nécessaire pour traiter g_i populations de façon séquentielle, soit g_iS.

Le nombre de générations est aussi désigné par g. La Figure 6.11 montre les temps d'exécution en nombre d'opérations en faisant ressortir la notion d'exécution simultanée des différentes tâches composant le programme parallèle. Sur cette figure et dans les calculs, nous négligeons le temps de communication entre les processeurs. Ce temps de communication ne sera pris en compte que si les processeurs sont situés sur plusieurs

machines. Dans le cas d'une machine multiprocesseurs, l'accès aux données est local et le temps de communication ne diffère pas de celui d'une exécution séquentielle. La durée n^2+mn représente le temps de génération d'une nouvelle population initiale, K représente le temps de traitement des autre tâches. La durée mn^2 représente le temps de traitement le plus long dans tout le processus.

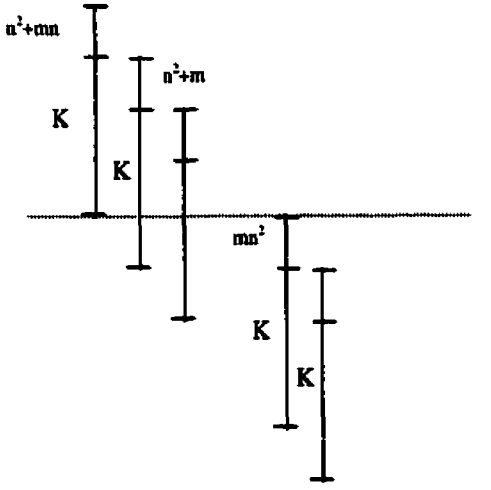

Figure 6.11 Diagramme des temps de l'exécution parallèle

$$P = n^2+mn+K+m+n+K+g_i(mn^2)(g - 1)$$
$$P = n^2+mn+2K+gg_i(m+n)-(g_i-1)(mn^2)$$

Dans le programme parallèle, nous avons g_i populations qui sont traitées simultanément. Nous devons alors prendre en compte le temps correspondant à g_i exécutions de façon séquentielle.

L'efficacité ε d'un algorithme parallèle est un indice de performance qui est déterminé par le rapport entre l'accélération due à la parallélisation et le nombre de processeurs. On a :

$$\varepsilon = \frac{\frac{T_{séquentiel}}{T_{parallèle}}}{Nb\_Processeurs}$$

Un programme sera dit efficace si ε est proche de 1. La perte d'efficacité due à la parallélisation est obtenu par 1-ε. En prenant $g = 40$, les gains obtenus pour des tailles de réseaux variables ainsi que la perte d'efficacité due à la parallélisation se résument dans le Tableau 6.9 ci après.

Tableau 6.9 Gain obtenu en parallélisant l'algorithme génétique pour diverses tailles de réseau

#cell.	#comm.	% de gain en temps d'exécution	Perte d'efficacité
15	2	50.51	0.62
30	3	33.04	0.67
50	4	24.66	0.69
100	5	19.66	0.70
150	6	16.38	0.71
200	7	14.05	0.71

Il faut préciser que les calculs sont approximatifs car ils sont essentiellement basés sur la complexité en nombre d'instructions. Néanmoins, ces valeurs se rapprochent assez de la réalité. Les gains obtenus sont très minimes et la perte d'efficacité due à la parallélisation est grande. Une autre approche de parallélisation serait de découper l'espace de recherche en de petits sous-espaces que l'on pourrait traiter simultanément sur des processeurs différents. Nous pensons que cela augmenterait les chances d'atteindre de bonnes solutions tout en réduisant le temps d'exécution.

CHAPITRE VII
CONCLUSION

7.1 Synthèse des travaux

Dans ce mémoire, nous avons utilisé la méthode de l'algorithme génétique pour résoudre le problème d'affectation des cellules aux commutateurs dans les réseaux de communications personnelles. Le problème revenait à affecter n cellules à m commutateurs potentiels dont les emplacements sont connus, tout en respectant des contraintes d'affectation unique et de capacité des commutateurs. Étant donné la complexité du problème, une approche heuristique de résolution nous a paru plus appropriée.

Nous avons donc conçu, implémenté et testé un algorithme génétique applicable à une population initiale, dans le but d'obtenir la solution qui minimise le coût du réseau, tout en respectant la contrainte d'affectation unique des cellules aux commutateurs et celle sur la capacité des commutateurs. Pour cela, nous avons d'abord conçu une procédure efficace pour la génération d'une population initiale sans doubles. Nous avons ensuite défini, implémenté et appliqué des opérateurs génétiques spécifiques au contexte de notre problème, opérateurs qui ont été appliqués par la suite à cette population initiale.

Nous avons également évalué la performance de l'algorithme génétique, en le comparant à trois autres méthodes alternatives qui ont été appliquées au même problème. Nous avons enfin conçu une version parallèle préliminaire de notre algorithme génétique qui sera exécutable dans un environnement avec plusieurs processeurs.

Les opérateurs génétiques sont des paramètres importants de l'algorithme génétique. Un choix judicieux des valeurs de ces paramètres offre plus de chances d'orienter les recherches dans une bonne direction. Nous parlons ici de chance, car le hasard a une grande part de responsabilité en ce qui a trait aux résultats obtenus, étant donné qu'il intervient à chaque étape du processus génétique.

Nous avons aussi effectué une analyse de sensibilité de l'algorithme génétique par rapport aux divers paramètres, en le soumettant à différents problèmes. Certaines hypothèses fondées sur l'intuition se sont confirmées, d'autres se sont infirmées. Nous avons ainsi constaté qu'un grand nombre de générations ne garantit pas forcément l'obtention d'une meilleure solution. Il suffit en fait d'un croisement adéquat, qui peut d'ailleurs se produire à n'importe quelle génération, pour obtenir de bonnes solutions.

Nous pouvons confirmer qu'une grande taille de la population offre une diversité de chromosomes, et par suite une diversité de solutions. Néanmoins, contrairement à nos attentes, elle ne mène pas nécessairement vers une solution optimale. Par ailleurs, les opérateurs génétiques se sont comportés selon nos hypothèses intuitives. Ainsi, plus la probabilité de croisement augmente, plus les résultats obtenus s'améliorent. De même, de meilleurs résultats sont obtenus au fur et à mesure que la probabilité de mutation diminue.

Nous avons aussi étudié les performances des AG par rapport à d'autres méthodes qui ont été appliquées au même problème. De cette étude, il ressort que la méthode de recherche taboue surclasse celle des AG en termes de qualité des solutions fournies. Mais, cette dernière se classe devant le recuit simulé et l'heuristique proposée par Beaubrun, Pierre et Conan (1999).

Enfin, nous avons proposé une version parallèle de notre adaptation des AG dans le souci de procéder à des tests plus intensifs tout en réduisant le temps d'exécution.

7.2 Limitations des travaux et indications de recherche future

Malgré les résultats globalement satisfaisants que nous avons obtenus, les performances de notre algorithme génétique dépendent fortement des paramètres choisis. Ces derniers ne sont pas faciles à choisir, et les valeurs utilisées dans notre implémentation sont celles qui donnent de bons résultats, mais qui ne sont pas forcément meilleures pour un autre type de problème. De plus, les données utilisées pour effectuer les différents tests proviennent de simulations, plutôt que de la réalité.

Il faut aussi noter que la méthode ne garantit pas toujours l'obtention de solutions faisables. Dans ce cas, le concepteur peut y appliquer une légère modification pour la rendre faisable, tout comme dans le cas du mécanisme du repêchage si cela n'engendre pas des coûts énormes.

Si le temps ne constitue pas une contrainte, on pourrait augmenter le nombre de cycles pour chaque exécution afin d'accroître les chances d'atteindre ou tout au moins d'approcher le plus possible l'optimum. Pour finir, notre algorithme génétique ne prend pas en compte le problème d'affectation double.

En guise de travaux futurs, il est envisagé d'implémenter la version parallèle que nous avons proposée. On pourrait aussi envisager, pour la version parallèle proposée, un découpage de l'espace de recherche en des sous-espaces plus petits, ce qui permettrait d'exécuter l'algorithme séquentiel sur chacun des espaces ainsi obtenus et qui augmenterait possiblement le gain en temps d'exécution. Cela nous donnerait une base de comparaison additionnelle pour situer les résultats fournis par la version séquentielle.

BIBLIOGRAPHIE

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

Beasley J. E. et Chu P. C., «A genetic algorithm for the set covering problem». *European Journal of Operational Research*, Vol. 94, No. 2 1996, pp. 392-404.

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.

Castillo L. et Gonzalez A. «Distribution network optimization: finding the most economic solution by using genetic algorithms». *European Journal of Operational Research*, Vol. 108, No. 3, 1998, pp. 527-37.

Chatterjee S., Carrera C. et Lynch L. A. «Genetic algorithm and traveling salesman problems». *European Journal of Operational Research*, Vol. 93, No. 3, 1996, pp. 490-510.

Chiang Wen-Chuyan, Chiang Chi. «Intelligent local search strategies for solving facility layout problems with the quadratic assignment formulation». *European Journal of Operational Research*, Vol. 106, 1998, pp. 457-488.

Davis L. (dir. Publ.). «Handbook of genetic algorithms». *Van Nostrand Reinhold*, New-York, 1991.

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

Goldberg D.E., «Genetic Algorithms in Search, Optimisation, and Machine Learning». *Addison-Wesley, Don Mills*, 1989.

Gondim, R. L. Paulo. «Genetic algorithms and the location area partitionning problem in cellular networks». *Proceedings of the Vehicular Technology Conference '96*, Atlanta, (VA) April 29-30, May 1, 1996, pp. 1835-1838.

Gravel M., Nsakanda A.L., Price W. «Efficient solutions to the cell-formation problem with multiple routings via a double-loop genetic algorithm». *European Journal of Operational Research*, Vol. 109, No. 2, 1998, pp. 286-298.

Guttmann-Reck N., Hassin R. «Approximation algorithms for minimum tree partition». *Discrete Applied Mathematics*, Vol. 87, 1998, pp 117-137.

Hockney R. W. et Jesshope C. R.«Parallel computers 2». Adam Hilger, Bristol and Philadelphia, 1998.

Holland J. H. «Genetic algorithms and the optimal allocation of trials». *SIAM Journal of Computing*, Vol. 2, No. 2, 1973, pp. 88-105.

Holland J. H. «Adaptation in Natural and Artificial Systems». *The University of Michigan Press*, Ann Arbor, Michigan 1975.

Houéto F. et Pierre S. «Affectation heuristique de cellules à des commutateurs dans les réseaux cellulaires mobiles». *Annales des télécommunications*, À paraître.

Kernighan B. W. et Lin S. «An Efficient Heuristic Procedure for Partitionning Graphs». *The Bell Systems Technical Journal*, Vol. 49, 1970, pp. 291-301.

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

Kouvelis P., Chiang W.-C., Fitzsimmons J. «Simulated annealing for machine layout problems in the presence of zoning constraints». *European Journal of Operational Research*, Vol. 57, 1992, pp. 203-223.

Krishnamurthy B. «An improved min-cut algorithm for partitionning VLSI networks». *IEEE Transactions on Computers*, Vol. C-33, 1984, pp. 438-446.

Merchant A. et Sengupta B. «Multiway graph partitionning with applications to PCS networks». *Proceedings IEEE INFOCOM '94. The Conference on Computer Communications*, 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.

Nguyen B. T. et Moon B. R. «Genetic algorithm and graph partitioning». *IEEE Transactions on Computers*, Vol. 45, No. 7, 1996, pp. 841-855.

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 Legault G. «A Genetic Algorithm for Designing Distributed Computer Network Topologies », *IEEE Transactions on Man, Systems, and Cybernetics*, Vol. 28, No. 2, 1998, pp. 249-258.

Pierre S. et Legault G. «Une méthode heuristique de dimensionnement de réseaux téléinformatiques », *Annales des Télécommunications, CNET*, Vol. 51, No. 3-4, 1996, pp. 143-157.

Pierre S. et Legault G. «An Evolutionary Approach for Configuring Economical Packet-Switched Computer Networks», *Artificial Intelligence in Engineering, Elsevier Science*, Vol. 10, No. 2, 1996, pp. 127-134.

Pirlot M. «General local search methods». *European Journal of Operational Research*, Vol. 92, No. 3, 1996, pp. 493-511.

Sanchis A. L. «Multiple-Way Network Partitionning». *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, 1990, 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.

Sun D., Lin L. et Batta R. «Cell formation using Tabu Search». *Computer-Integrated Manufacturing*, Vol. 28, No. 3, 1995, pp. 485-494.

Syswerda G. «Uniform Crossover in genetic algorithms». *Proceedings of the third Conference on Genetic Algorithms*. Morgan Kaufmann Publisher.

Zhou G. et Gen M. «An effective genetic algorithm approach to the quadratic minimum spanning tree problem». *Computers & Operations Research*, Vol. 25, No. 3, pp. 229-237.