Titre: Title:	Conception topologique de réseau cellulaire par une approche hybride de programmation par contraintes et de recherche locale
Auteur: Author:	Yanick Pomerleau
Date:	2003
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Pomerleau, Y. (2003). Conception topologique de réseau cellulaire par une approche hybride de programmation par contraintes et de recherche locale [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7153/

In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

UNIVERSITÉ DE MONTRÉAL

CONCEPTION TOPOLOGIQUE DE RÉSEAU CELLULAIRE PAR UNE APPROCHE HYBRIDE DE PROGRAMMATION PAR CONTRAINTES ET DE RECHERCHE LOCALE

YANICK POMERLEAU
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES (M.Sc.A.)
(GÉNIE ÉLECTRIQUE)
MAI 2003

Your file Votre référence
ISBN: 0-612-86429-4
Our file Notre référence
ISBN: 0-612-86429-4

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CONCEPTION TOPOLOGIQUE DE RÉSEAU CELLULAIRE PAR UNE APPROCHE HYBRIDE DE PROGRAMMATION PAR CONTRAINTES ET DE RECHERCHE LOCALE

présenté par: POMERLEAU Yanick

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

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

Mme. CHERIET, Farida, Ph.D., président

M. PESANT Gilles, Ph.D., membre et directeur de recherche

M.CHAMBERLAND, Steven, Ph.D., membre et codirecteur de recherche

M.PIERRE Samuel, M.Sc.A., membre

REMERCIEMENTS

Je remercie d'abord mon directeur de recherche, M. Gilles Pesant, et mon codirecteur de recherche, M. Steven Chamberland. En plus du soutien financier, j'ai énormément appris sous leur supervision et ils ont su me donner le goût de continuer en recherche.

Je remercie également ma conjointe, Mme Chantal Vignola, qui m'a beaucoup inspiré par son aide et son soutien. Je remercie M. Marc Brisson ainsi que tous les membres du Qu'ossé Ça (Québec Optimization and Satisfaction Strategies Exploring Constraint Algorithms) pour leur collaboration et pour l'ambiance agréable du laboratoire.

Je remercie enfin mes parents qui m'ont toujours encouragé à persévérer et à ne pas abandonner dans l'atteinte de mes objectifs.

RÉSUMÉ

Les opérateurs de réseau cellulaire investissent une portion importante de leur budget pour acquérir, installer et entretenir leurs équipements tels que les interfaces radios, les contrôleurs et les commutateurs. Considérant les coûts importants des infrastructures de réseaux cellulaires, la planification et l'optimisation deviennent des éléments primordiaux pour demeurer compétitif.

Dans ce mémoire, nous proposons d'abord un modèle de programmation par contraintes pour le problème de conception topologique de réseau cellulaire. Ce problème consiste à choisir les sites pour installer les contrôleurs de station de base (BSC) et les sites pour installer les centres de commutation radio mobile (MSC) et leur type, ainsi qu'à définir la topologie du réseau et les types des liens et ce, selon la localisation des stations de base (BTS). L'objectif est de concevoir le réseau de coût minimum. Comme ce problème est classé NP-difficile, nous ne pouvons pas résoudre les exemplaires de réseau de taille réelle en un temps raisonnable. Nous proposons donc une heuristique hybride de recherche locale et de programmation par contraintes pour résoudre le problème.

Afin d'évaluer la performance de l'heuristique, nous la comparerons avec une approche tabou proposée par Chamberland et Pierre (2002) et avec une borne inférieure obtenue en résolvant une version relaxée du modèle. L'heuristique est testée avec les jeux de données générés aléatoirement de Chamberland et Pierre (2002). Les solutions trouvées avec notre approche se comparent très bien à la borne inférieure et améliorent par un facteur de plus de 2% les meilleures solutions obtenues jusqu'à maintenant.

ABSTRACT

Cellular wireless network operators set aside an important part of their budget to acquire, install and maintain equipments like radio interfaces, controllers and switches. Considering the cost of a cellular wireless infrastructure, network planning and optimization are important issues for those operators in order to remain competitive.

In this thesis, we first propose a constraint programming model for the design problem of cellular wireless communication networks. It consists of selecting the location of the base station controllers (BSC) and mobile service switching centers (MSC), selecting their types, designing the network topology and selecting the link types, and considering the location of base transceiver stations (BTS). The objective is to find the minimum cost network. Since this problem is NP-hard, it is unlikely that real-size instances of the problem can be solved to optimality within a reasonable amount of time. As a result, we propose a heuristic to find a good solution of the model. This heuristic is based on a local search heuristic combined with constraint programming techniques.

In order to assess the performance of the proposed heuristic, we compare it with a tabu search algorithm proposed by Chamberland and Pierre (2002) and with a lower bound obtained by solving a relaxed version of the model. The heuristic was tested with the same set of randomly generated instances as Chamberland and Pierre (2002). The solutions found with our approach are near the lower bound and improve by more than 2% the best solution found so far.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES NOTATIONS ET DES SYMBOLES

BSC : Contôleur de station de base (Base Station Controller)

BT : Retour arrière (Back Track)

BTS : Station de base (Base Tranceiver Station)

CSOP : Problème d'optimisation de contraintes (Constraint Satisfaction Optimisation Problem)

CSP : Problème de satisfaction de contraintes (Constraint Satisfaction Problem)

GA : Algorithme génétique (Genetic ALgorithms)

LS : Recherche locale (Local Search)

MSC : Centre de commutation radio mobile (Mobile Switching Center)

PC : Programmation par contraintes

PM : Programmation mathématique

SA : Recuit simulé (Simulated Annealing)

TS : Recherche tabou (Tabu Search)

VND : Descente à voisinage variable (Variable Neighborhood Descent)

VNS : Recherche à voisinage variable (Variable Neighborhood Search)

LISTE DES TABLEAUX

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION

Les systèmes de télécommunications mobiles prennent de plus en plus d'importance dans nos vies. En effet, quoi de mieux que de communiquer et naviguer sur Internet sans restriction géographique ou temporelle? Pour répondre à une demande croissante sans en augmenter les coûts d'exploitation (Reed, 1993), les opérateurs de réseaux cellulaires doivent sans cesse améliorer la conception et la gestion de leur réseau. L'allocation de fréquence, la conception topologique, l'acheminement des appels (routage) sont autant de domaines où l'optimisation permet une meilleure gestion des ressources et une réduction des coûts d'exploitation. Dans ce contexte, un des aspects très importants est la conception topologique des réseaux cellulaires qui sera traitée dans ce mémoire. Dans ce premier chapitre, nous introduirons les concepts de base qui définissent un réseau cellulaire, ce qui nous permettra de décrire la problématique. Nous discuterons ensuite des objectifs à atteindre et nous finirons par une description du plan du mémoire.

1.1 Définitions et concepts de base des réseaux cellulaires

Le réseau cellulaire est un réseau informatisé de communication sans fil, constitué de transmetteurs radio circulant sur un territoire divisé en cellules. Chaque cellule est desservie par une station de base (BTS, Base Transceiver Station) servant à relier les utilisateurs au réseau. La cellule est souvent représentée par un hexagone dont la dimension est proportionnelle à sa zone de couverture comme illustré à la figure 1.1. Chaque BTS est connecté à un contrôleur de station de base (BSC, Base Station Controller), qui est un commutateur de grande capacité. Le BSC est responsable de toutes les fonctions liées à la transmission radio,

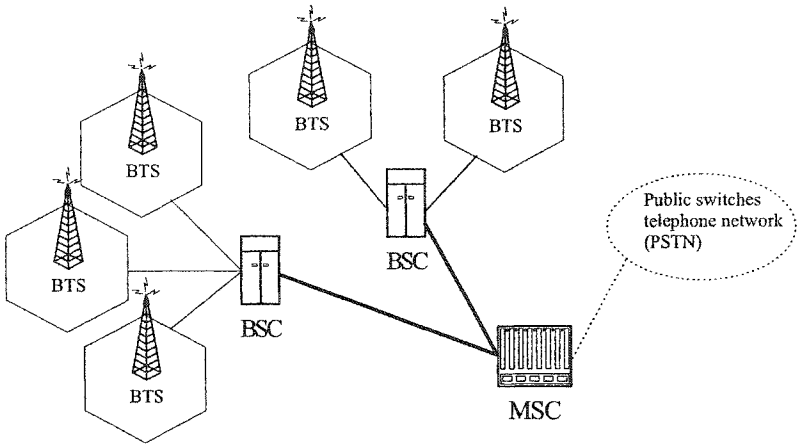

Figure 1.1: Relation entre les composantes d'un réseau cellulaire

comme la relève simple, la gestion des ressources du réseau et les données relatives à la configuration des cellules. Il contrôle également le niveau de puissance des fréquences des BTS. Un transmetteur mobile en déplacement d'une cellule à l'autre doit changer de BTS pour continuer à communiquer, on appelle ce processus la relève (*handoff*). Elle est simple lorsque les deux BTS sont connectés au même commutateur. Chaque BSC est connecté à un niveau hiérarchique supérieur, le centre de commutation radio mobile (MSC, Mobile Switching Center), qui est aussi un commutateur de grande capacité. Le MSC assure le routage des appels reliant deux utilisateurs mobiles ou un utilisateur mobile à un réseau fixe tel que le réseau téléphonique public commuté (RTPC). Les MSC sont donc responsables de la répartition et du transfert des appels et ils s'occupent aussi de la relève complexe. La relève complexe survient lorsqu'un transmetteur se déplace d'une cellule à une autre mais cette fois le nouveau BTS est affecté à un commutateur différent de celui du BTS d'origine.

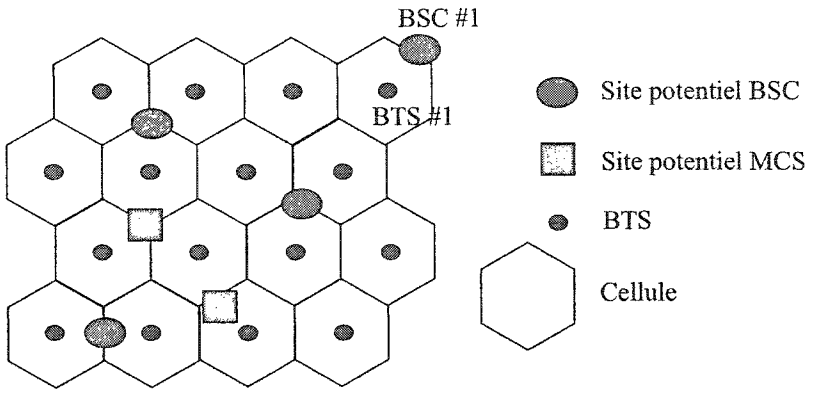

Figure 1.2: Exemple d'un réseau cellulaire typique

1.2 Problématique

Lors de la conception d'un nouveau réseau cellulaire ou lors de la mise à jour de la couverture d'un réseau existant, les opérateurs doivent établir où ils installeront leurs équipements de façon à répondre à la demande et ce, à moindre coût. Ils font d'abord des études pour connaître les caractéristiques du trafic afin de déterminer la localisation et la dimension des cellules. Par la suite, ils identifient les sites potentiels pour installer les BSC et MSC. Leur travail devient plus complexe lorsqu'ils doivent choisir parmi ces sites potentiels les endroits où ils installeront réellement leurs équipements. Ce choix est difficile puisque le nombre de combinaisons croît exponentiellement avec le nombre de sites potentiels. Un mauvais choix de sites peut entraîner une différence de plusieurs millions de dollars lors de la réalisation du réseau. Les opérateurs doivent aussi déterminer quelle sera l'affectation entre les BTS et les sites de BSC utilisés et l'affectation entre les sites de BSC et les sites de MSC utilisés. Ces choix d'affectation sont aussi difficiles que ceux de localisation des équipements et ont encore plus d'importance sur les coûts. Enfin, toutes ces combinaisons sont sujettes aux contraintes de capacité des équipements utilisés.

La difficulté du problème vient du fait que ces choix sont dépendants les uns

des autres. Par exemple, imaginons que le BTS #1 est affecté au site BSC #1. Ce choix se justifie considérant que la distance qui les sépare est la plus petite tel qu'illustré à la figure 1.2. Une des conséquences de cette affectation sera qu'il faudra relier et installer un BSC au site BSC #1 même si ce dernier est loin des sites MSC disponibles. Ce choix contribue à diminuer les coûts d'affectation BTS-BSC mais à augmenter les coûts d'affectation BSC-MSC. De plus, cette affectation a des conséquences sur la localisation des BSC. En effet, il faudra maintenant installer un BSC au site BSC #1 puisqu'il y a au moins un BTS qui sera connecté à celui-ci. Toutes les décisions sont donc dépendantes les unes des autres. Nous devons par conséquent explorer toutes les combinaisons pour trouver la meilleure solution. La complexité du problème vient du fait que le nombre de combinaisons augmente de façon exponentielle avec la taille du problème.

Le problème de conception topologique est en fait une combinaison des problèmes d'affectation, de localisation et de dimensionnement. Il réfère à deux problèmes NP-difficiles notoires : le partitionnement de graphes (Merchant et Sengupta, 1995) et le problème de localisation de commutateurs (Sohn et Park, 1998). Le problème est donc NP-difficile et nous devons par conséquent utiliser une approche heuristique dans le but de trouver de bonnes solutions aux instances de moyenne et de grande taille. Une heuristique ne permet pas nécessairement de trouver la solution optimale au problème mais elle permet de trouver une bonne solution en un temps raisonnable. Par contre, il n'y a généralement pas de garantie sur la qualité de la solution trouvée. Dans le problème de conception topologique de réseau cellulaire on parlera de temps raisonnable lorsque la solution est disponible après quelques heures de travail : étant donné que c'est un problème de planification stratégique, seule la qualité de la solution nous importe.

1.3 Objectifs

Le projet consiste à développer une heuristique hybride de recherche locale et de programmation par contraintes (PC) pour résoudre le problème de conception topologique de réseau cellulaire. Le problème a des caractéristiques intéressantes lui permettant d'être formulé en programmation par contraintes. En effet, il contient de nombreux ensembles (sites potentiels, types des éléments), ce qui est propice à l'utilisation de variables à domaine fini. Les algorithmes de résolution de PC ont démontré par le passé leur efficacité pour résoudre ce type de problème. Par contre, cette recherche de solution exacte ne permettra pas d'obtenir une solution de qualité pour les réseaux de taille réelle. Nous combinerons alors le modèle de programmation par contraintes à une recherche locale pour améliorer les solutions de façon heuristique. Nous commencerons donc par concevoir le modèle de programmation par contraintes pour résoudre le problème de façon exacte. Nous travaillerons ensuite les heuristiques d'affectation de ce modèle pour obtenir la solution optimale le plus rapidement possible pour les instances de petite taille. Par la suite, nous combinerons ce modèle à une heuristique de recherche locale à descente simple pour l'optimisation d'instances de taille réelle. Nous chercherons, plus particulièrement, à définir une transformation locale de la solution qui mène l'heuristique à un minimum local de bonne qualité et ce, en un temps raisonnable. Pour ce faire, nous baserons cette transformation sur le choix des sites de BSC constituant une solution.

1.4 Plan du mémoire

Le mémoire contient six chapitres. Suite à ce premier chapitre d'introduction, le chapitre 2 présente la formulation du problème de conception topologique, la définition des contraintes et des données initiales. Nous discuterons aussi des travaux antérieurs portant sur le sujet. Le chapitre 3 décrit le modèle de programmation par

contraintes conçu pour résoudre le problème de façon exacte. Nous retrouverons également dans ce chapitre les fondements de la PC et l'importance des heuristiques d'affectation de variables. Le chapitre 4 expose les principaux concepts de la recherche locale et présente l'implantation de la recherche locale à descente simple utilisée pour la résolution heuristique du problème. L'évolution de l'heuristique tout au long de ces travaux et les principaux résultats se retrouvent au chapitre 5. Enfin, le chapitre 6 conclut en faisant une synthèse des travaux réalisés et des résultats obtenus, en plus de discuter des possibilités de travaux futurs découlant de cette recherche.

CHAPITRE 2

CONCEPTION TOPOLOGIQUE DE RÉSEAU CELLULAIRE

La conception topologique de réseau cellulaire est un problème d'optimisation très important qui est amplement traité dans la littérature. Cependant, très peu de chercheurs se sont attardés au problème dans son ensemble étant donné sa complexité. Le chapitre débutera par la présentation du problème. Nous discuterons pas la suite des concepts de base nécessaires à la compréhension des travaux reliés à la conception topologique. Nous terminerons par la description des travaux connexes de la littérature.

2.1 Présentation du problème

On considère un problème comportant m BTS, n sites potentiels pour installer les BSC et p sites potentiels pour installer les MSC. Avant de procéder à la conception proprement dite du réseau, les opérateurs ont recueilli plusieurs informations. Ils ont analysé le trafic dans chacune des zones de couvertures des BTS et ils ont consigné les informations suivantes : (I1) la localisation des BTS et le type de chacun; (I2) le trafic entre chaque BTS et le réseau public; (I3) le trafic entre les BTS; (I4) les sites potentiels pour installer les BSC; (I5) les sites potentiels pour installer les MSC; (I6) les types de BSC disponibles, le coût et la capacité de chacun; (I7) les types de MSC disponibles, le coût et la capacité de chacun; (I8) le coût d'installation de chaque type de BSC; (I9) les coûts d'installation de chaque type de MSC; (I10) les coûts des liens et de leur installation.

Le problème de conception topologique de réseau cellulaire est de :

- Déterminer la localisation des sites pour installer les BSC et les MSC;
- Déterminer les types des BSC et des MSC;

- Déterminer la topologie du réseau;
- Choisir le type des liens BSC-MSC.

L'objectif est de minimiser le coût du réseau tout en satisfaisant les demandes de communication. Pour cela, on doit considérer les contraintes suivantes : (C1) chaque BTS doit être connecté à un BSC; chaque BSC doit être connecté à un MSC; (C3) chaque MSC doit être connecté au réseau public; (C4) le nombre de liens BTS-BSC connectés à un BSC doit être plus petit ou égal au nombre maximum d'interfaces BTS pouvant être installé à un BSC; (C5) la somme des capacités des liens BTS-BSC connectés à un BSC doit être plus petite ou égale à la capacité pouvant être installée à un BSC; (C6) le nombre de liens BSC-MSC connectés à un BSC doit être plus petit ou égal au nombre maximum d'interfaces MSC pouvant être installé à un BSC; (C7) le nombre de liens BSC-MSC connectés à un MSC doit être plus petit ou égal au nombre maximum d'interfaces pouvant être installé à un MSC; (C8) la somme des capacités des liens BSC-MSC connectés à un MSC doit être plus petite ou égale à la capacité pouvant être installée à un MSC; (C9) un seul type de BSC peut être installé à un site BSC; (C10) un seul type de MSC peut être installé à un site MSC; (C11) la conservation de flot dans le réseau doit être respectée.

C1, C2 et C3 sont des contraintes d'affectation, C4 à C8 sont des contraintes de capacités des équipements, C9 et C10 sont des contraintes de localisation et C11 impose la conservation de flot.

2.2 Revue des connaissances

Afin de bien comprendre les avantages et inconvénients de l'utilisation de la programmation, par contraintes nous allons dans cette section expliquer et discuter les principaux concepts liés au sujet. Par la suite, nous expliquerons les fondements

de la recherche locale. Nous terminerons par une courte description des métaheuristiques de recherche locale telle que la recherche tabou (Glover et Laguna, 1997).

2.2.1 Fondements de la programmation par contraintes

La programmation par contraintes est un paradigme de programmation qui permet la modélisation d'un problème en faisant abstraction de la résolution. En fait, il s'agit de décrire les relations qui existent entre les divers éléments du problème et non les algorithmes utilisés pour le résoudre. Le modèle ainsi conçu est ensuite résolu par un *solveur de contraintes* qui, lui, possède plusieurs algorithmes de résolution. Ces algorithmes exploitent les caractéristiques du domaine des variables utilisées dans la formulation du problème.

Un avantage indéniable de la PC par rapport aux autres méthodes de modélisation est la facilité avec laquelle les contraintes sont définies. En effet, les contraintes en PC ressemblent beaucoup à la façon d'exprimer des contraintes en langage naturel. Par exemple, l'affirmation "Lorsque la tâche i est suivie de la tâche j, le temps du début de la tâche i plus sa durée doit être inférieur au début de la tâche j" se traduit simplement en PC par "$S_i = j \Rightarrow T_i + D_i \leq T_j$", où S ,D et T représentent respectivement la tâche suivante, la durée et le temps de début de la tâche. Il est intéressant de noter qu'une telle implication est une fonction non linéaire, souvent non convexe très difficile à résoudre en programmation mathématique. Une contrainte est donc une relation logique entre des variables : égalité, inégalité, implication, union, etc. Plus formellement, une contrainte est une formule close bâtie de variables et d'opérateurs définissant une relation.

Un modèle en PC est construit en énonçant toutes les relations entre les variables qui définissent le problème. Lors de la résolution, le modèle est traité par un *solveur* de contraintes, dans notre cas ILOG OPL Studio 3.5 (Ilog, 2001), qui recherche toutes les solutions réalisables du problème. Une solution réalisable est

obtenue lorsque chacune des variables est affectée à une seule valeur et que ces affectations ne violent aucune contrainte. On parlera de problème de satisfaction CSP (*Constraint Satisfaction Problem*) lorsqu'on recherche une solution qui ne viole aucune contrainte. Pour un problème d'optimisation CSOP (*Constraint Satisfaction Optimisation Problem*), l'algorithme de résolution recherche la meilleure solution parmi l'ensemble des solutions réalisables, solution choisie selon un objectif à améliorer. Enfin, il est extrêmement important de bien choisir les variables et les contraintes qui définissent le problème puisque certains modèles se prêtent à une résolution beaucoup plus efficace que d'autres. Il faut donc bien connaître les mécanismes de résolution de la PC afin de bien choisir les variables et les contraintes.

2.2.2 Résolution des problèmes de programmation par contraintes

Un *solveur de contraintes* est spécifique aux types de contraintes et de variables utilisés. Cela prend la forme de contraintes linéaires sur les réels, d'égalités sur les arbres finis, de contraintes booléennes ou encore de contraintes sur domaines finis.

Un de ceux-ci se distingue et obtient de bonnes performances par rapport aux techniques de résolution plus conventionnelles : il s'agit des problèmes avec contraintes sur variables à domaine fini. En effet, les mécanismes de résolution de la PC sont plus efficaces pour ce type de problème. Un domaine est dit fini lorsqu'il est possible d'énumérer toutes les valeurs possibles d'affectation d'une variable. Ces problèmes avec contraintes sur variables à domaine fini (FD) sont parmi les plus communs et les plus difficiles à résoudre.

En programmation mathématique, il est possible de construire des modèles de programmation en nombre entier (PNE) pour résoudre des problèmes sur domaine fini. Par contre, les mécanismes de résolution, tel le *branch and bound*, dépendent souvent de la qualité de la relaxation ou des coupes. Ils sont généralement très coûteux à résoudre et la vitesse de résolution dépend des choix de relaxation du modèle et des coupes. Par exemple, dans le cas d'un *branch and bound* sur une

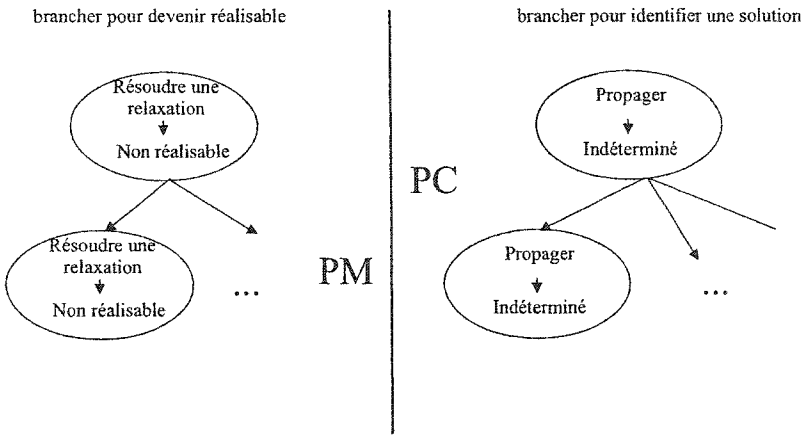

Figure 2.1: *Branch and bound* programmation mathématique (PM) *vs* programmation par contraintes (PC)

version relaxée de modèle, l'objectif est de brancher pour obtenir une solution en nombre entier qui satisfait les contraintes (figure 2.1). On ne peut pas prévoir le nombre d'itérations qu'il faudra pour devenir réalisable. Le temps de recherche sera alors fonction du coût d'évaluer la borne multiplié par 2^N, où N est le nombre de niveaux à explorer. Dans le cas où il y aurait 20 niveaux à explorer avant de devenir réalisable le coût de recherche serait de 2^{20} multiplié par le coût d'évaluer la borne. Ceci est très long, d'où l'intérêt d'exploiter d'autres méthodes de résolution telle que la programmation par contraintes.

Il existe aussi un *branch and bound* en PC (figure 2.1). Il s'agit de fixer une variable à la fois et de calculer le coût de la borne inférieure de la solution partielle ainsi formée. Contrairement à la PNE, la solution est toujours réalisable tout au long de la recherche. On coupe des branches lorsque la solution partielle donne un coût supérieur à la borne préalablement trouvée. Le nombre d'explorations en PC peut aussi devenir très coûteux puisqu'il faut explorer toutes les branches de l'arbre de recherche avant de garantir l'optimum. En PC comme en PE, la conception de bons modèles s'acquiert avec l'expérience car il est impératif de tirer avantage des

mécanismes de résolution propres à chaque technique.

Les problèmes d'optimisation CSOP en PC tel que celui de ce mémoire sont en fait des problèmes de satisfaction auxquels on ajoute une contrainte supplémentaire. Après avoir trouvé la première solution réalisable, on ajoute une contrainte stipulant que les prochaines solutions doivent avoir un coût inférieur à la première solution réalisable, le *branch and bound*. Chaque fois que la solution est améliorée la contrainte est ajustée avec le coût de la solution courante.

Un problème de satisfaction CSP se définit comme suit :

- un ensemble de variables $X_i = \{x_1, ..., x_n\}$;
- un ensemble de contraintes $C_j = \{c_1, ..., c_m\}$ qui régissent les valeurs que peuvent prendre les variables X_i;
- un domaine D qui fait correspondre à chaque variable X_i un ensemble de valeurs possibles $D(X_i)$.

La recherche de solutions pour un problème donné se fait généralement en affectant à chaque variable, une à la fois, une valeur de son domaine. L'ensemble courant des variables fixées ainsi formé se nomme solution partielle. Donc, après chaque affectation, le *solveur de contraintes* vérifie si la solution partielle satisfait bien les contraintes. Il propage l'information découlant de l'affectation courante et filtre ensuite les domaines des variables non fixées. Cette vérification et cette réduction de domaine est nommée la propagation. Dans le cas où la valeur d'affectation courante viole une des contraintes, cette valeur est éliminée du domaine et le *solveur de contraintes* reprend l'affectation à la valeur suivante pour la même variable. Ce mécanisme s'appelle le *retour arrière* (BT). Dans le cas où la valeur d'affectation ne viole aucune contrainte, il existe des mécanismes pour propager l'information et ainsi filtrer les domaines des variables qui ne sont pas encore affectées. Ces mécanismes retirent du domaine des variables non fixées les valeurs qui violeraient

une contrainte, en considérant l'ensemble des affectations courantes (solution partielle). On appelle cohérences de noeud, d'arc et borne ces techniques de filtrage de domaine.

Le choix des variables de modélisation a une importance capitale sur la rapidité de résolution d'un problème. Généralement, minimiser le nombre de variables et de contraintes donne un meilleur modèle. Effectivement, le nombre d'affectations sera plus petit avant d'obtenir une solution et donc le temps résolution est plus court. En fait, l'arbre de recherche est moins profond lorsque le nombre de variables diminue et le temps de propagation diminue puisqu'il y a moins de variables sur lesquelles propager. Dans le même ordre d'idée, le temps de propagation diminue aussi lorsqu'il y a moins de contraintes sur lesquelles propager. Par contre, nous devons tenir compte de la dimension des domaines des variables puisqu'il est possible que la propagation ne puisse pas facilement les réduire. Par conséquent, il est difficile de déterminer si un modèle est bon en terme de variables et de contraintes. Ce n'est qu'un indicateur de qualité.

2.2.2.1 Cohérences de noeud, d'arc et de borne

Les mécanismes de cohérence servent à trouver un CSP équivalent au problème original ayant des variables dont le domaine est réduit. La cohérence de noeud n'est appliquée que sur les contraintes unaires, c'est-à-dire aux contraintes qui n'ont qu'une variable. C'est la technique de réduction de domaine de moindre coût. Elle consiste à parcourir toutes les contraintes, à vérifier s'il n'y a qu'une seule variable dans la contrainte pour finalement enlever du domaine de la variable les valeurs incohérentes. Par exemple, imaginons que la variable X a un domaine $D(X) = \{4, 5, 6, 7\}$, la contrainte $X < 6$ et le mécanisme de cohérence de noeud réduirait le domaine à $D(X) = \{4, 5\}$ puisque l'affectation de X à 6 ou 7 violerait la contrainte.

La cohérence d'arc agit sur toutes les contraintes de deux variables, contraintes

binaires. Considérons les variables X et Y de domaine $D(X)$ et $D(Y)$ respectivement. La cohérence d'arc consiste, pour toutes contraintes impliquant X et Y, à retirer toute valeur de $D(X)$ pour laquelle il n'y a pas de valeur dans $D(Y)$ qui satisfait la contrainte et vice versa. Par exemple, pour la contrainte $X + Y < 6$, les domaines $D(X) = \{4, 5, 6\}$ et $D(Y) = \{1, 2, 3\}$ seraient réduits à $D(X) = \{4, 5\}$ et $D(Y) = \{1, 2\}$, la valeur $X = 6$ n'ayant pas de support dans le domaine $D(Y)$, de même pour $Y = 3$. Il est intéressant de noter que l'hyper-cohérence d'arcs, impliquant plus de deux variables, pourrait être appliquée pour réduire les domaines de plusieurs variables à la fois. Par contre, déterminer l'hyper-cohérence d'arcs est un problème combinatoire qui croît de façon exponentielle avec le nombre de variables; c'est donc un problème NP difficile en soit.

La cohérence de borne est en fait une technique de filtrage particulière aux contraintes arithmétiques. Elle consiste à resserrer les extrémités du domaine des variables impliquées dans une contrainte en déterminant les bornes supérieures et inférieures menant à une solution réalisable. Pour ce faire, nous devons définir des règles de propagation différentes pour chaque type d'opérateur. Par exemple, pour l'addition $X = Y + Z$, en raisonnant sur les valeurs maximum et minimum, la cohérence de borne est résolue comme suit :

- $X \geq minD(Y) + minD(Z)$ et $X \leq maxD(Y) + maxD(Z)$
- $Y \geq minD(X) - maxD(Z)$ et $Y \leq maxD(X) - minD(Z)$
- $Z \geq minD(X) - maxD(Y)$ et $Z \leq maxD(X) - minD(Y)$

Ces réductions de domaines aident la recherche de solution en éliminant des valeurs tout au long de la recherche. Il y a un coût associé à ce travail de réduction de domaine. Il est important que ces coûts soient inférieurs au coût d'énumération et vérification de toutes les combinaisons possibles (algorithme *générer puis tester*), exploration de tout l'*espace de recherche*. En pratique, même pour des problèmes

de petite taille, les coûts de réduction de domaines sont toujours inférieurs. Par contre, des mécanismes de propagation très performants peuvent devenir coûteux si leur gain est petit par rapport à leur temps d'exécution. Dans ce contexte, il est important de bien choisir le moment d'utiliser la propagation (Comme nous le verrons à la section 4.2.4).

2.2.2.2 Événements de propagation

Les contraintes peuvent être classées par type selon l'événement de propagation qui leur est associé. Il est possible de faire la propagation chaque fois qu'une valeur du domaine est retirée (événement *domain*), lorsqu'une borne inférieure ou supérieure change (événement *range*) ou seulement si le domaine est réduit à une valeur unique (événement *value*). Ces événements dont l'occurrence va du plus fréquent, événement *domain*, au moins fréquent, événement *value*, ont leur utilité suivant le contexte. Par exemple, une contrainte de non égalité $X \neq Y$ aura avantage à utiliser l'événement *value* puisqu'on ne peut pas réduire le domaine de X avant que Y soit fixé et vice versa. Il ne sert donc à rien de propager plus souvent sur ce type de contrainte.

Voici un exemple récapitulatif qui permettra de mettre en évidence les caractéristiques propres à la programmation par contraintes. Imaginons un problème de trois variables X, Y et Z de domaine$\{1,2,3,4\}$ impliquées dans trois contraintes, soit :

- $C1 : X = Y$
- $C2 : X < Z$
- $C3 : X + Y = Z$

L'*espace de recherche* est de 4^3 combinaisons, soit la multiplication de tous les domaines. En utilisant la technique *générer puis tester* il y a 64 combinaisons à

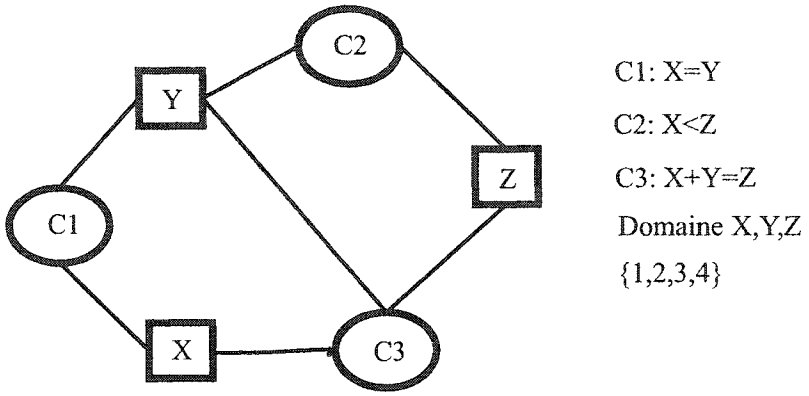

Figure 2.2: Réseau de contraintes

explorer. En PC, les contraintes sont reliées via les variables qu'elles contraignent tel qu'illustré sur la figure 2.2.

Nous constatons d'abord qu'il n'y aura pas propagation par la technique de cohérence de noeuds puisqu'il n'y a aucune contrainte unaire. La contrainte $C1$ est cohérente d'arc puisqu'il existe pour chaque valeur de X une valeur dans Y qui satisfait la contrainte et vice-versa. Par contre, la contrainte $C2$ ne l'est pas puisque pour $X = 4$ il n'y a pas de support dans Z; Z ne pouvant prendre de valeur pour satisfaire la contrainte $C2$ avec $X = 4$. Conséquemment la valeur 4 sera retirée du domaine de X. Comme la contrainte $C1$ fait partie de la classe d'événements *domain*, elle sera réveillée pour propager l'information fournie par la réduction de domaine de X. La valeur 4 sera alors retirée du domaine de Y par la technique de cohérence d'arc sur la contrainte $C1$. La contrainte $C3$ classée dans l'événement *range* s'éveillera à son tour puisqu'une valeur limite du domaine d'une variable qu'elle comporte a changé de valeur. On applique alors à la contrainte $C3$ la technique de cohérence de borne, ce qui réduit le domaine de Z à {2,3,4}, puisque $Z \geq min\ D(X) + min\ D(Y)$. La cohérence de borne n'a pas d'autre effet sur les domaines, les domaines de X et de Y demeurent inchangés à {1,2,3}. Il

est à noter que le *Solveur* procède à une certaine optimisation du modèle avant de traiter les contraintes. Pour ce problème, il aurait éliminé la contrainte $X = Y$ et remplacé Y par X dans toutes les contraintes. Le résultat étant que $C3$ devient une contrainte binaire ($Z = 2X$) où la cohérence d'arc est applicable. La valeur $Z = 3$ serait retirée du domaine de Z n'ayant pas de support dans X et de même pour la valeur $X = 3$. Les domaines des variables sont donc finalement réduits à $Z = \{2, 4\}$ et $X = Y = \{1, 2\}$. Il ne restera plus qu'à fixer une variable pour trouver la valeur des deux autres.

2.2.2.3 Ordre d'affectation des variables et des valeurs

Un autre facteur très important est l'ordre d'affectation des variables. En fait, pour un même modèle, un ordre de choix des variables à affecter peut faire en sorte que le problème soit résolu en quelques secondes ou en plusieurs heures. L'ordre de choix des variables à affecter doit se faire selon les principes de la propagation. Nous cherchons à réduire les domaines le plus rapidement possible et ce, sans se diriger vers une solution non réalisable.

L'ordre de sélection des variables est important puisqu'il influence la topologie de l'arbre de recherche. On parlera d'ordre statique lorsque l'ordre de sélection est fixé à priori. Lorsque l'ordre d'affectation change selon l'état courant de la fouille, il sera question d'ordre dynamique. Il est préférable d'utiliser l'ordre dynamique mais en se rappelant qu'il y a un coût associé à ce choix. Il intervient moins souvent que les coûts de propagation (à chaque affectation de valeur) puisqu'il ne survient qu'au changement de variable.

Il existe des heuristiques dynamiques qui suivent le principe *d'échec d'abord* (first fail). Il s'agit de commencer par les variables les plus difficiles à affecter. On veut savoir le plus rapidement possible si on se dirige vers une impasse. Effectivement, plus les variables susceptibles de mener à un échec sont fixées tard dans la recherche, plus elles seront contraintes et plus il sera difficile de trouver une affec-

tation correcte. Dans le cas d'échec à ce niveau, toutes les affectations des niveaux supérieurs seront à reprendre.

L'heuristique *d'échec d'abord* la plus populaire est celle du *plus petit domaine d'abord* (ppdd). Son principe est que moins il y a de valeurs possibles pour une variable, moins il y a de chances de trouver une affectation pour celle-ci. Il faut donc commencer par affecter les variables dont le domaine est plus petit en premier.

Une autre heuristique intéressante est celle du *moindre regret*. Il s'agit de favoriser l'affectation de la variable dont l'impact sur la fonction de coût entre les deux meilleures affectations est le plus grand. Cette heuristique en est une d'optimisation plutôt qu'une *d'échec d'abord*. Effectivement, plus on retarde l'affectation des variables de grand regret, plus elles risquent un échec lors de son affectation : le coût de la solution partielle sera fort détérioré puisque la deuxième meilleure valeur de son domaine a un impact négatif important. Il est donc intéressant de commencer par affecter les variables de plus grand regret. Ainsi en cas d'échec et de retour arrière, la qualité de la solution sera moins détériorée et le regret moindre.

Le choix de l'ordre des valeurs d'affectation des variables a aussi son importance. Le choix de valeur n'influence pas la topologie de l'arbre de recherche mais seulement l'ordre dans lequel les branches seront explorées. Par contre, un bon choix de valeur peut faire en sorte qu'on se dirige vers une solution réalisable rapidement, ce qui permet de couper dans l'arbre de recherche plus rapidement avec l'ajout de la contrainte de borne supérieure sur la fonction objective pour les problèmes CSOP discutés à la section 2.2.2. Le principe sous-jacent à l'ordre d'attribution de valeur est d'essayer d'abord de réussir. Les heuristiques qui en découlent s'inspirent souvent de la structure même du problème. Il existe quelques heuristiques génériques pour maximiser les chances de réussite, par exemples choisir d'abord :

- la valeur de plus grand support. Il y aura moins de réduction de domaine, ce qui augmente les chances d'être réalisable;

- la valeur avec le plus grand produit des tailles des domaines de variables futures. Le principe est le même soit de conserver un espace de recherche très grand pour avoir plus de chance de demeurer réalisable jusqu'à la fin.
- la valeur minimisant la proportion des domaines des variables futures qui ne seront plus cohérentes. Le but est toujours le même, obtenir une solution réalisable. Cette fois le type des contraintes est analysé pour assurer la cohérence.

Il faut se rappeler que toutes ces heuristiques ont des coûts non négligeables. Il est donc important de bien juger de leur apport avant de les utiliser.

2.2.3 Fondements de la recherche locale

Considérons un ensemble fini S. Un problème d'optimisation combinatoire consiste à trouver $x_{opt} \in X \subseteq S$ avec l'objectif de minimiser une certaine fonction de coût:

$$\min\{f(x) : x \in X, X \subseteq S\} \tag{2.1}$$

S, X, x et f sont respectivement l'espace de recherche, l'ensemble des solutions réalisables, une solution réalisable et la fonction de coût.

Lors de l'utilisation d'une heuristique de recherche locale à descente simple (LS) nous définissons $N(x)$, l'ensemble des solutions voisines de x. Soit x_k la solution courante à l'itération k, la procédure de recherche commence à partir d'une solution réalisable x_1; à chaque itération k, la fonction de coût est évaluée pour chaque voisin $x \in V_k \subseteq N(x_k)$. Lorsqu'une meilleure solution $x_{k+1} \in V_k$ est trouvée, la procédure se poursuit avec x_{k+1} comme solution courante. Dans le cas contraire, la recherche se termine dans ce minimum local: il n'y a pas dans le voisinage $N(x_k)$ de meilleure solution que la solution courante x_k. Il est à noter que $f(x_k)$ peut être beaucoup plus grand que l'optimum global du problème.

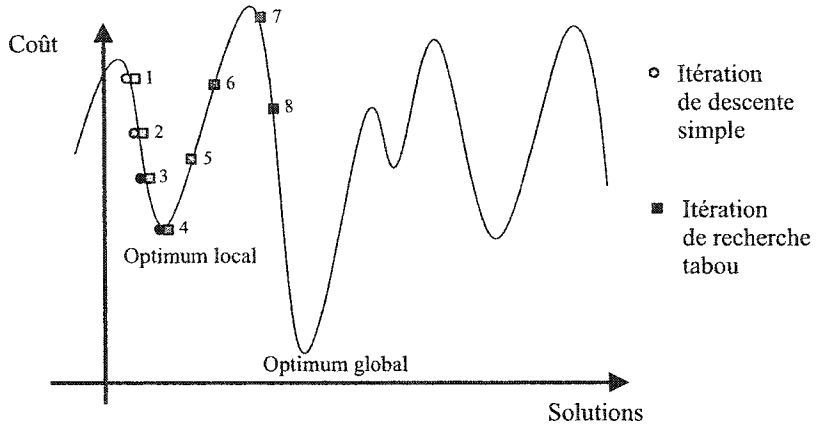

Figure 2.3: Recherche locale à descente simple vs recherche tabou

On appelle cette méthode heuristique, recherche locale à descente simple puisque la recherche se termine lorsqu'on atteint le premier minimum local. En général, il existe plusieurs minimums locaux pour une même fonction de coût. La figure 2.3 représente une fonction de coût qui varie en fonction de la solution courante. La recherche locale permet donc une descente le long de cette courbe. Lorsque le voisinage de la solution courante ne peut plus améliorer la fonction de coût, l'heuristique a atteint un minimum local. Un problème relié à l'utilisation de cette méthode est que le coût de la solution x_k peut-être loin de l'optimum global. En effet, comme on se contente du premier minimum local rencontré, il est fort probable que l'optimum global soit dans un autre minimum local tel qu'illustré à la figure 2.3.

2.2.3.1 Métaheuristiques de recherche

Il existe des mécanismes qui permettent de rechercher au-delà de ce minimum local et ainsi d'explorer d'autres minimums locaux. On parlera alors de métaheuristiques de recherche (Gendreau, Laporte et Potvin, 1994). Parmi les plus populaires, nommons la recherche tabou (TS), le recuit simulé (SA), les algorithmes génétiques (GA) et la recherche à voisinage variable (VNS).

Comme dans le cas d'une heuristique LS, SA et TS débutent leur recherche à partir d'une solution initiale x_1; à chaque itération k, la solution courante change de x_k à une solution x_{k+1} faisant partie du voisinage $N(x_k)$. Par contre, $f(x_{k+1})$ n'est pas nécessairement plus petit que $f(x_k)$. Nous devons donc faire attention de ne pas retomber sur x_k à l'itération suivante. Divers mécanismes ont été développés pour pallier ce problème. Par exemple TS utilise une liste de mouvements tabous. En TS (Soriano et Gendreau, 1997), après les itérations de descente simple, lorsqu'il n'est plus possible d'améliorer la solution x_k, il est alors permis d'effectuer un mouvement local qui dégrade le coût de la solution. Le danger est donc de revenir, à l'itération suivante, sur la solution qu'on vient de quitter puisqu'elle est nécessairement de meilleur coût que la solution courante dégradée (figure 2.3, itération 5). Pour éviter de cycler, TS conserve pendant un certain temps la liste des mouvements récents, nommée la *liste tabou*. En SA (Kirkpatrick, 1983), le mécanisme utilisé est différent. À l'itération k, la solution x est choisie de façon aléatoire parmi $N(x_k)$. Si $f(x) \leq f(x_k)$, alors x_{k+1} égale à x, sinon

$$x_{k+1} = \begin{cases} x & \text{selon une probabilité } p_k \\ x_k & \text{selon une probabilité } 1 - p_k, \end{cases}$$

où, habituellement, p_k est une fonction qui décroît avec k et $f(x) - f(x_k)$. Souvent, p_k est défini comme $\exp(-[f(x) - f(x_k)]/\Theta_k)$, où Θ_k est nommé la température de l'itération k. Typiquement, Θ_k décroît à chaque itération k : il y a refroidissement de la température. La probabilité de choisir une solution de coût plus élevé décroît donc dans le temps ou si l'on préfère avec le refroidissement. Les cycles sont évités puisque SA choisit ses mouvements de façon aléatoire.

GA (Holland, 1975) évalue, à chaque étape, un ensemble de solutions nommé population; chaque population provient des populations précédentes dont on a conservé les meilleurs éléments et rejeté les pires. GA utilise des populations de solutions encodées comme des chromosomes, généralement avec une chaîne de carac-

tères. Considérons une population $X^k = \{x_1^k, ..., x_N^k\}$, dont on a défini une *fonction d'aptitude* f. L'heuristique cherche à optimiser cette fonction. À l'itération k, X^k est transformé en X^{k+1} en produisant de nouvelles solutions (descendants) à partir des vieilles (parents), on remplace ces parents par leurs descendants. Cela est accompli en conservant les caractéristiques des bonnes solutions de génération en génération, le processus de sélection naturel se fait donc via les bonnes solutions. De petites permutations aléatoires sont introduites dans les descendants pour assurer une diversité des solutions d'une population.

Enfin, VNS (Hansen et Mladenovic, 2001) explore plusieurs structures de voisinage N_t $(t = 1, .., t_{max})$ avec $N_t(x)$ l'ensemble des solutions du t^{eme} voisinage de x. Par exemple, la recherche locale à descente simple utilise seulement une structure de voisinage ($t_{max} = 1$). VNS explore un à un les voisinages définis. Pour le voisinage t, l'heuristique choisit d'abord, aléatoirement, une solution voisine x' parmi $N_t(x_k)$. On effectue la recherche locale à descente simple sur ce voisinage en utilisant la solution x' comme solution initiale. Lorsqu'on atteint un minimum local, si la solution x_k est améliorée, on reprend la recherche, $t = 1$, avec cette nouvelle solution à l'itération $k + 1$. Sinon, on continue au voisinage suivant, $t = t + 1$, sur la solution x_k. Plusieurs variantes sont possibles : par exemple, effectuer un changement de voisinage durant la phase de descente de l'algorithme (VND). Ainsi, à chaque itération de descente on explore les voisinages jusqu'à ce qu'il y en ait un qui améliore le coût de la solution.

2.2.3.2 Considérations sur la solution initiale et le voisinage

Idéalement la solution initiale ne devrait pas avoir d'effet sur la qualité de la solution finale. L'heuristique doit être assez robuste pour obtenir le même résultat final indépendamment de la solution initiale utilisée. Par contre, en pratique, étant donné que les espaces de recherche sont très grands, il arrive que ce ne soit pas le cas. Une approche pour résoudre ce problème est de lancer la recherche à partir

de plusieurs solutions initiales. On nomme cette technique la multi-relance (*multi-start*).

La solution initiale peut avoir d'autres impacts sur la qualité des heuristiques. Par exemple, pour la recherche locale à descente simple, une solution initiale très bonne peut être dommageable si elle nous mène dans une région de l'espace de recherche où il est difficile d'améliorer la solution. Dans ce cas, il est mieux d'avoir une solution initiale de moins bonne qualité et de laisser l'heuristique locale diriger la recherche vers une meilleure région de minimum local. Il est à noter qu'il y a autant de directions de descente possibles que de voisins explorés qui améliorent le coût de la solution à chaque itération.

Ce critère de voisinage est très important dans le cas d'une recherche à descente simple puisqu'il n'est pas possible de sortir d'un minimum local. Nous devons donc nous assurer que la descente n'est pas trop rapide pour que la recherche locale ne passe pas outre la direction de recherche qui nous conduirait à une bonne solution. Dans le cas d'une descente simple, il est souvent avantageux d'explorer un grand voisinage, ce qui aide à de meilleures décisions dans le choix de la direction de descente. Dans le cas des métaheuristiques, la philosophie de recherche est un peu différente. Les métaheuristiques recherchent aussi de bonnes directions de descente. Par contre, elles acceptent plus facilement de se commettre puisqu'il est possible de sortir d'un minimum local. La recherche est moins exhaustive à chaque itération mais le nombre d'itérations est plus grand. Ces techniques sont particulièrement efficaces lorsque le temps d'évaluation du coût et de la vérification de la réalisibilité d'une solution est court. La stratégie consiste à visiter le plus de minimums locaux possibles.

2.3 Revue de littérature

Plusieurs chercheurs ont proposé des heuristiques pour résoudre le problème dans son ensemble ou pour résoudre des parties de celui-ci. Nous décrirons dans cette section les méthodologies de recherche qui comportent des éléments de résolution du problème de conception topologique de réseau cellulaire.

2.3.1 Méthodes de résolution du problème global

Chamberland et Pierre (2002) ont étudié un problème comme celui défini dans ce mémoire. Ils ont résolu le problème avec un modèle de programmation mathématique combiné à une heuristique de recherche tabou (Glover et Laguna, 1997). Cox et Sanchez (2002) ont quant à eux considéré un problème d'affectation et de localisation défini de façon légèrement différente : la localisation n'est effectuée que pour les sites de BSC mais ils ajoutent des contraintes de fiabilité sur certains éléments. D'autres auteurs ont résolu le problème de façon exacte dont Ahuja, Magnanti et Orlin (2000). Leurs résultats serviront de base de comparaison pour les réseaux de petite taille.

2.3.1.1 Méthode de recherche tabou

Chamberland et Pierre (2002) ont utilisé une approche tabou. Cette méthode est très importante pour notre recherche, notamment pour la comparaison des résultats: nous avons utilisé les jeux de données de Chamberland et Pierre pour évaluer notre heuristique. L'algorithme débute par la recherche d'une solution initiale, solution qui sera améliorée à l'aide d'une recherche tabou. Une première heuristique commence par fixer les types des BSC et des MSC à installer aux sites potentiels. Après avoir déterminé les types des BSC et des MSC, les affectations BTS-BSC et BSC-MSC deviennent plus faciles et ce, tout comme le calcul du coût associé au réseau. La difficulté est de faire un bon choix de types de BSC

et de MSC à installer aux sites. Afin de calculer le coût d'une solution selon les types choisis, Chamberland et Pierre ont décomposé le problème en deux sous-problèmes. Le premier consiste à affecter les BTS aux BSC tout en respectant les contraintes de degré et de capacité des BSC. Le deuxième sous-problème consiste à affecter les BSC aux MSC tout en respectant les contraintes de degré et de capacité des MSC. Comme ces deux sous-problèmes sont NP-difficiles : ils réfèrent au problème NP-difficile notoire de sac à dos. Chamberland et Pierre ont proposé deux heuristiques correspondant aux deux sous-problèmes pour trouver de bonnes solutions rapidement. Ces heuristiques sont basées sur l'algorithme de résolution de problème d'affectation linéaire de Jonker et Volgenant (1987).

Avant de procéder à la description des mouvements de la recherche tabou, nous devons décrire la recherche de la solution initiale. Soit une *étoile BTS* un sous-réseau qui inclut un BSC (le centre de l'étoile) et les BTS qui lui sont reliés. La grandeur et le coût d'une *étoile BTS* correspondent respectivement au nombre de BTS et à la somme des coûts des liens BTS-BSC de l'*étoile BTS*. De même, une *étoile BSC* est définie comme un sous-réseau de BSC connectés à un MSC (centre de l'étoile). La grandeur et le coût des *étoiles BSC* correspondent respectivement au nombre de BSC et à la somme des coûts des liens BSC-MSC de l'*étoile BSC*.

L'heuristique initiale évalue des solutions pour des *étoiles BTS* de grandeur p allant de 1 au nombre maximal de BTS pouvant être connectés à un BSC. Pour chaque valeur de p considérée, de nouveaux sites de BSC sont choisis en comparant le coût des étoiles BTS de grandeur p et ce, à l'aide d'une méthode vorace. Avec la même méthode, les MSC sont choisis en utilisant le coût des *étoiles BSC*. Une fois les sites de BSC et de MSC choisis, il suffit d'utiliser les deux heuristiques d'affectation BTS-BSC et BSC-MSC afin de déterminer la topologie et le coût du réseau. L'heuristique initiale termine lorsque toutes les valeurs de p ont été explorées. De toutes ces explorations, la meilleure solution est choisie.

La recherche tabou consiste à changer l'état de la solution courante à chaque

itération de l'algorithme. L'heuristique cherche à améliorer une certaine fonction de coût associée au problème. À chaque itération, il s'agit de modifier l'affectation d'un certain nombre de variables de la solution courante (mouvement local) et d'analyser la qualité des solutions ainsi obtenues. Ainsi, l'heuristique décide du meilleur mouvement possible pour le choix des sites de BSC et de MSC. Pour ce faire, on utilise une *liste tabou* de mouvements et un *critère d'aspiration.* Le site choisi demeure tabou pour un nombre déterminé d'itérations à moins que le *critère d'aspiration* ne révèle que ce site nous mènerait à une meilleure solution.

Chamberland et Pierre ont calculé une borne inférieure pour un modèle relaxé des contraintes de conservation de flot. L'heuristique tabou est très rapide (la solution finale est obtenue en quelques minutes) et les résultats se comparent bien à la borne inférieure. Par contre, les mouvements possibles de l'heuristique sont limités. L'heuristique visite un espace de recherche petit et par conséquent la recherche de solution d'optimum local est rapide mais celle-ci peut difficilement être améliorée. De plus, il est plus difficile de se rapprocher de la borne inférieure pour les jeux de données dont le trafic est élevé (réseau de grande taille) puisque la conservation de flot est négligée. Nous chercherons à améliorer les solutions obtenues par Chamberland et Pierre même si cela va au détriment de l'efficacité en temps de calcul. Nous chercherons aussi à améliorer le coût des solutions pour les réseaux de grande taille qui sont les moins bien optimisées avec la recherche tabou.

2.3.1.2 Méthode de recherche tabou combinée au problème de sac à dos

Cox et Sanchez (2000) ont utilisé une heuristique taboue pour un problème défini de façon légèrement différente. Ils étudient l'affectation entre les cellules et les commutateurs (DS1, DS3) et la localisation de ces commutateurs. Ils ajoutent des contraintes de fiabilité de connection dans le réseau. Les commutateurs doivent tous être connectés à au moins deux autres commutateurs pour assurer la fiabilité.

Par contre, le problème ainsi formulé ne considère pas l'affectation et la localisation des MSC. Malgré toutes ces différences, le coeur du problème reste le même soit l'affectation et la localisation de commutateurs. L'heuristique tabou utilise les sous-problèmes de sac à dos et de conservation de flot pour calculer le coût du voisinage exploré à chaque itération.

Pour les petits réseaux, les résultats obtenus en quelques secondes sont les mêmes que les solutions optimales obtenues en plusieurs heures avec le modèle en nombre entier de Ahuja, Magnanti et Orlin (2000). Pour des réseaux de moyenne taille (150 BTS et 25 BSC), les résultats sont obtenus en quelques minutes. Une limitation de ce modèle vient du fait qu'il n'optimise pas la localisation et l'emplacement des MSC. De plus, les coûts obtenus avec ou sans les contraintes de fiabilité sont environ les mêmes. On peut en déduire que le nombre de liens total dans le réseau n'a pas augmenté, seul le nombre d'affectations a augmenté. Cette conclusion est intéressante puisqu'elle confirme que négliger les contraintes de fiabilité a un impact faible sur la fonction de coût.

2.3.2 Méthodes de résolution du problème d'affectation

La majorité des chercheurs se sont concentrés sur des heuristiques pour résoudre le problème d'affectation de cellules aux commutateurs qui est une partie de notre problème. Étant donné que la première étape de notre méthode est de résoudre ce sous-problème, ces recherches deviennent d'autant plus importantes. Les coûts d'affectation de ces heuristiques correspondent aux coûts de liaison additionnés aux coûts de relèves complexes, contrairement à la présente étude où les coûts correspondent aux coûts de liaisons ajoutés aux coûts des équipements. Dans la littérature ce problème est habituellement formulé à l'aide de modèles en nombre entier. Toutefois, du à la complexité du problème, la recherche de solution se fait à l'aide de méthodes heuristiques lorsque le nombre de cellules et de commutateurs augmentent significativement.

2.3.2.1 Méthodes de recherche locale sur un grand voisinage

Les premiers à s'attarder au problème d'affectation furent Merchant et Sengupta (1994). Les auteurs ont utilisé une approche vorace qui tente autant que possible d'éviter les minimums locaux. La méthode consiste à améliorer une solution initiale en effectuant une succession de transformations qui change l'affectation d'une cellule d'un commutateur à un autre. À chaque itération, la méthode visite plusieurs configurations avant de prendre la décision pour la transformation de la solution courante. Nous avons utilisé un mécanisme similaire pour notre recherche mais l'exploration des configurations se fait en programmation par contraintes. Le critère de recherche à chaque itération est différent lui aussi puisque, pour notre problème, il faut considérer deux niveaux d'affectations BTS-BSC et BSC-MSC.

Pour de petits réseaux, les résultats obtenus par cette méthode se comparent avec ceux obtenus par une méthode de programmation en nombre entier. De plus, les temps de calcul ont significativement été réduits. Par contre, les transformations utilisées n'exploitent qu'un nombre limité de possibilités, ce qui rend difficile d'éviter les minimums locaux. Les transformations dépendent de la solution initiale et ne mènent pas nécessairement à une solution finale se rapprochant de la solution optimale pour les réseaux de grande taille. Merchant et Sengupta furent les premiers à utiliser une heuristique locale pour résoudre ce type de problème. Ce qui inspira plusieurs chercheurs à concevoir des heuristiques qui évitent d'être piégées dans les minimums locaux : recherche tabou, recuit simulé, voisinage variable, algorithme génétique ou d'autres heuristiques de recherche locale.

2.3.2.2 Méthode de recherche tabou à voisinage double

Une autre heuristique intéressante d'affectation des cellules aux commutateurs est celle de André, Pesant et Pierre (2002). Cette heuristique utilise un modèle de programmation par contraintes pour trouver une solution initiale et une heuristique

tabou à deux mouvements pour améliorer cette solution. Cette heuristique a permis de réduire les temps de recherche par un facteur important. Un élément intéressant de cette recherche est le double mouvement de l'heuristique tabou. Le principe de base de cette heuristique est que la recherche tabou se fait sur un voisinage variable. Ce double mouvement permet d'éviter les minimums locaux par deux mécanismes distincts. Lorsque le premier ne permet plus d'éviter un minimum, l'heuristique utilise le deuxième mouvement afin de continuer la recherche. La difficulté est de trouver ces deux mouvements complémentaires. Le deuxième aspect intéressant est la recherche de la solution initiale en programmation par contraintes. L'affectation initiale est trouvée en un temps court et la solution est d'excellente qualité. La qualité de la solution initiale est un facteur important en recherche locale afin d'améliorer l'efficacité de l'heuristique. Nous retiendrons cet aspect d'affectation rapide et la qualité de la solution fournie par la programmation par contraintes lors de la conception de l'heuristique proposée dans ce mémoire.

Les auteurs ont comparé leur heuristique à d'autres approches bien connues de la litérature. L'heuristique proposée par André Pesant et Pierre a amélioré les meilleures solutions par un facteur d'environ 1%. De plus, l'efficacité de la recherche de solution a de beaucoup été améliorée, notamment grâce à la qualité de la solution initiale fournit par le modèle de PC.

2.3.2.3 Méthode heuristique basée sur le regoupement de cellules

Saha, Mukherjee et Bhattacharya (2000) se sont fortement inspirés de Merchant et Sengupta. Leur heuristique est basée sur la formation par étape de regroupement de cellules. La cellule où le commutateur est installé est nommée la racine du regroupement. L'heuristique ajoute des cellules aux regroupements pour minimiser le coût global d'affectation.

Les résultats obtenus par cette méthode se comparent bien à la solution optimale pour les exemplaires de petite taille. Dans 95% des cas la solution trouvée

est optimale et pour les 5% restant, celles-ci sont à moins de 3% de l'optimum. De plus, l'heuristique améliore substantiellement les solutions obtenues par Merchant et Sengupta mais elle perd rapidement de son efficacité lorsque la taille des exemplaires augmente.

2.3.2.4 Méthode de recherche locale et de recuit simulé

L'heuristique proposée par Der-Rong et Tseng (2002) s'applique plus particulièrement aux réseaux sans fil ATM (*Asynchronous Tranfer Mode*). Les cellules sont affectées aux commutateurs d'un réseau ATM dont la localisation est fixe et connue.

Les auteurs ont développé deux approches différentes pour résoudre le problème. Ils proposent un algorithme heuristique et un algorithme basé sur une heuristique de recuit simulé. La première approche est divisée en deux phases. Une première qui recherche une solution initiale pour l'affectation de cellules aux commutateurs. Une deuxième phase pour améliorer la solution initiale en exécutant une série de permutations pour lesquelles deux cellules sont affectées à répétition à un commutateur différent.

L'algorithme de recuit simulé utilise trois différentes transformations pour passer d'une solution réalisable à une autre à chaque itération. Le premier type de transformation choisit aléatoirement deux cellules connectées à des commutateurs différents et leur affectation respective est interchangée. Le deuxième type choisit aléatoirement deux commutateurs et réaffecte les cellules du commutateur 1 au commutateur 2 et vice-versa. Finalement, le troisième type de transformation change l'affectation de plusieurs cellules connectées à différents commutateurs.

Les résultats obtenus par ces deux approches se comparent bien avec ceux obtenus par les heuristiques plus conventionnelles. Par contre, ces deux heuristiques n'évitent pas très bien les minimums locaux. En outre, pour l'algorithme de recuit simulé il est très difficile d'ajuster les paramètres pour toutes les transfor-

mations proposées.

2.3.3 Méthodes de résolution du problème de localisation

Plusieurs chercheurs ont aussi étudié le problème de localisation des commutateurs. Ces heuristiques cherchent à minimiser le coût de liaison des cellules aux commutateurs en déterminant la localisation des commutateurs dans le réseau. Nous ne présentons qu'une seule recherche sur le sujet puisque pour notre problème les sites potentiels pour installer les commutateurs sont choisis parmi un ensemble restreint de possibilités. Par contre, pour déterminer l'ensemble des sites potentiels ces recherches sont très intéressantes. En réalité, les opérateurs de réseau cellulaire sont limités dans le choix des sites potentiels puisqu'il est habituellement plus économique d'installer un commutateur dans un endroit appartenant déjà aux propriétaires. Par conséquent, l'ensemble des sites potentiels peut être construit à partir des localisations déjà disponibles.

2.3.3.1 Méthode exacte de programmation en nombre entier

Sohn et Park (1998) ont proposé un modèle en nombre entier pour le problème de localisation de p commutateurs. Le nombre de commutateurs et le trafic entre chaque paire de cellules sont donnés. Le problème consiste à rechercher la localisation des p commutateurs et d'assigner les cellules en minimisant le coût total du réseau.

Deux cas se présentent, celui où chaque cellule peut être connectée à plusieurs commutateurs et celui où chaque cellule peut être connectée à un seul commutateur; respectivement le problème d'affectation multiple et le problème d'affectation simple de p commutateurs. Nous ne discuterons que du problème d'affectation simple puisqu'il correspond à l'affectation utilisée lors de l'implantation de notre heuristique.

Pour le problème d'affectation simple, le modèle proposé par Sohn et Park réduit de plus de la moitié le nombre de contraintes et de variables par rapport à la formulation de Skorin-Karpov et al. (1998). Skorin-Karpov et al. avaient réduit le nombre de contraintes à l'ordre de grandeur de $2n^3$. La vitesse de résolution a grandement profité de cette amélioration lors de la recherche de la solution optimale, seulement 20 des 74000 instances sont des solutions non entières. Pour ces instances, la méthode de *branch and bound* fournit les solutions optimales rapidement. Cependant, comme toute méthode de recherche de solution exacte pour les problèmes NP-difficiles, cette méthode ne s'applique qu'aux instances de petite taille.

CHAPITRE 3

MÉTHODOLOGIE

Dans ce chapitre, nous proposons un modèle de programmation par contraintes (PC) pour résoudre le problème de conception topologique de réseau cellulaire. Nous continuerons par la description de l'heuristique d'affectation de variables du modèle de PC. Nous introduirons le concept de sonde qui nous permet de combiner la recherche locale au modèle, pour terminer par la description du voisinage exploré par la recherche locale, ainsi que l'heuristique de recherche de solution initiale.

3.1 Méthode exacte : Modèle de programmation par contraintes

Ce choix de modlisation se justifie par le fait que plusieurs éléments de la problématique se transposent bien en programmation par contraintes. En effet, pour résoudre un problème de façon efficace en PC, il est primordial de le décrire avec des variables à domaines finis. Ainsi, lors de la résolution du problème on profitera des mécanismes de propagation pour réduire les domaines des variables tout au long de la recherche (Marriott et Stuckey, 1998). Par exemple, supposons qu'après affectation du BTS#1 au BSC #1 il ne reste plus de capacité au BSC #1. Alors toutes les variables d'affectation BTS ne pourront plus prendre la valeur BSC #1, par conséquent il faut enlever de leur domaine la valeur BSC #1. Les mécanismes de propagation contribuent à résoudre le problème sans explorer toutes les combinaisons en éliminant, lors de la recherche, des valeurs du domaine des variables. Ainsi, il n'est pas nécessaire de parcourir toutes les branches de l'arbre de recherche pour trouver l'optimum, les solutions non réalisables ayant été éliminées.

3.1.1 Notation

3.1.1.1 Les ensembles

D'abord définissons I comme l'ensemble des BTS (où α_i le nombre de circuits du BTS i et η_i le nombre de liens partant du BTS i); J, l'ensemble des sites de BSC; et K l'ensemble des sites MSC. Définissons L comme l'ensemble des types de lien (où β_ℓ le nombre de circuit du lien ℓ); et S, l'ensemble des types de BSC (où α_s la capacité en circuit du BSC de type s, η_s^I le nombre d'interfaces BTS du BSC de type s, η_s^T le nombre d'interfaces MSC du BSC de type s). Enfin, définissons T comme l'ensemble des types de MSC (où α_t la capacité en circuit du MSC t, η_t^S le nombre d'interfaces BSC du MSC de type t). Il est à noter que tous ces ensembles sont disjoints.

3.1.1.2 Les demandes en communication

À l'aide des études de demandes de communication entre les cellules, les opérateurs établissent la matrice de trafic entre chaque BTS et le réseau public et entre les BTS. On nommera : $g_{i\,i'}$ le volume d'appels par unité de temps (erlang) entre le BTS i et le BTS i', $g_{i\,P}$ le volume d'appels par unité de temps (erlang) entre le BTS i et le réseau public, $g_{P\,i}$ le volume d'appels par unité de temps (erlang) entre le réseau public et le BTS i.

3.1.1.3 Les coûts

D'abord $a_{i\,j}$ est le coût des liens et des interfaces (incluant le coût d'installation) du BTS i au BSC j; $b_{\ell\,j}$, le coût des liens et des interfaces (incluant le coût d'installation) reliant le site BSC j à un MSC pour des liens de type ℓ. c_j^s est le coût du BSC de type s installé au site j; et d_k^t le coût du MSC de type t installé au site j.

3.1.1.4 Les variables de topologie

Nous utilisons cinq types de variable pour décrire l'ensemble du problème de $|I|$ BTS, $|J|$ sites potentiels de BSC et $|K|$ sites potentiels de MSC:

- v_i : Le site BSC auquel le BTS i est affecté
- w_j : Le site MSC auquel le site BSC j est affecté
- x_j : Le type du BSC installé au site j
- y_k : Le type du MSC installé au site k
- $z_{j\,l}$: Nombre de liens de type l partant du BSC j

3.1.1.5 Les variables de conservation de flot

Ces contraintes fixent la topologie du réseau. Nous introduisons de nouvelles variables afin de vérifier que les capacités des liens et des commutateurs sont suffisantes pour assurer la conservation du flot dans le réseau :

- t_i : le volume d'appels par unité de temps (erlang) du BTS i au BSC auquel il est affecté
- t_j : le volume d'appels par unité de temps (erlang) du BSC j au MSC auquel il est affecté

3.1.2 Les coûts

La fonction de coût est composée des coûts des liens et des interfaces et des coûts des BSC et des MSC. Ces coûts incluent les coûts d'installation.

Le coût des liens et des interfaces, noté $C_L(v, w, z)$, est fourni par l'équation suivante:

$$C_L(v, w, z) = \sum_{i \epsilon I} a_{i\, v_i} + \sum_{j \epsilon J} \sum_{\ell \epsilon L} z_{j\, \ell} b_{\ell\, w_j}. \tag{3.1}$$

Le coût des BSC et des MSC, $C_{B\&M}(x, y)$, est fourni par l'équation suivante:

$$C_{B\&M}(x, y) = \sum_{j \epsilon J} c_j^{x_j} + \sum_{k \epsilon K} d_k^{y_k}. \tag{3.2}$$

3.1.3 Le modèle topologique

Les variables choisies satisfont déjà quatre contraintes du design de réseau cellulaire énoncées à la section 1.2 : les contraintes d'unicité et d'affectation (C1,C2, C9 et C10). Par exemple, la variable v_i garantit que le BTS i ne pourra être affecté qu'à un seul BSC parmi son domaine.

Le modèle de programmation par contraintes pour la conception topologique de réseau cellulaire est formulé comme suit.

$$\min\ C_L(v, w, z) + C_{B\&M}(x, y) \tag{3.3}$$

sujet aux contraintes :

Contraintes de capacité des BSC (interface BTS)

$$\sum_{i \epsilon I} ((v_i = j)\eta_i) \leq \eta^I_{x_j} \qquad (j \epsilon J) \tag{3.4}$$

où $v_i = j$ est une expression logique 0-1 *Contraintes de capacité des BSC (capacité BSC)*

$$\sum_{i \epsilon I} ((v_i = j)\alpha_i) \leq \alpha_{x_j} \qquad (j \epsilon J) \tag{3.5}$$

Contraintes de capacité des BSC (interface MSC)

$$\sum_{\ell \epsilon L} z_{j\,\ell} \leq \eta^T_{x_j} \qquad (j \epsilon J) \tag{3.6}$$

Contraintes de capacité des MSC (interface BSC)

$$\sum_{j \epsilon J}((w_j = k) \sum_{\ell \epsilon L} z_{j\,\ell}) \leq \eta^S_t y_k \qquad (k \epsilon K) \tag{3.7}$$

Contraintes de capacité des MSC (capacité MSC)

$$\sum_{j \epsilon J}((w_j = k) \sum_{\ell \epsilon L} z_{j\,\ell} \beta_\ell) \leq \alpha_{y_k} \qquad (k \epsilon K) \tag{3.8}$$

Contraintes de capacité liens BTS-BSC

$$t_i \leq \alpha_i \qquad (i \epsilon I) \tag{3.9}$$

$$t_j \leq \sum_{\ell \epsilon L} z_{j\,\ell} \beta_\ell \qquad (j \epsilon J) \tag{3.10}$$

Les contraintes de conservation de flot

$$t_j = \sum_{i \epsilon I}((v_i = j) \sum_{o \epsilon I}((v_o \neq j)(g_{i\,o} + g_{o\,i})) + \sum_{i \epsilon I}((v_i = j)(g_{i\,P} + g_{P\,i})) \qquad (j \epsilon J) \tag{3.11}$$

$$t_i = \sum_{o \epsilon I}(g_{i\,o} + g_{i\,P}) + \sum_{o \epsilon I} g_{o\,i} + g_{P\,i} \qquad (i \epsilon I) \tag{3.12}$$

Domaine des variables

$$v_i \epsilon J (i \epsilon I), w_j \epsilon K\ (j \epsilon J), x_j \epsilon S\ (j \epsilon J), y_k \epsilon T\ (k \epsilon K), Z_{j\,\ell} \epsilon N\ (j \epsilon J, \ell \epsilon L) \tag{3.13}$$

La fonction de coût (3.3) représente donc le coût total du réseau. Ainsi, la contrainte (3.4) spécifie que le nombre des liens BTS-BSC connectés au site BSC j doit être inférieur au nombre maximum d'interfaces BTS du BSC installé à ce site. La contrainte (3.5) spécifie que le nombre de communications provenant des BTS connectés au site BSC j doit être inférieur à la capacité du commutateur installé à ce site. La contrainte (3.6) spécifie que le nombre des liens BSC-MSC connectés au site BSC j doit être inférieur au nombre maximum d'interfaces MSC installées à ce site. La contrainte (3.7) spécifie que le nombre des liens BSC-MSC connectés au site MSC k doit être inférieur au nombre maximum d'interfaces BSC installées à ce site. La contrainte (3.8) spécifie

que le nombre de communications provenant des BSC connectés au site MSC k doit être inférieur à la capacité du commutateur installé à ce site. Les contraintes (3.9) et (3.10) sont respectivement les contraintes de capacité des liens BTS-BSC et BSC-MSC. Enfin, les contraintes (3.12) et (3.13) assurent la conservation de flot dans le réseau.

3.2 Méthode heuristique : Recherche Locale

Comme nous l'avons mentionné au premier chapitre, les heuristiques sont des algorithmes efficaces utilisées pour résoudre des problèmes difficiles. Elles ont comme objectif de trouver, en un temps raisonnable, une solution de bonne qualité sans toutefois garantir que celle-ci soit la solution optimale. Les heuristiques de recherche locale ont fait leur apparition au début des années soixante, elles s'appliquent aux problèmes d'optimisation combinatoire tel que celui du présent mémoire. Elles ont, entre autres, démontré leur efficacité lors de la résolution de problèmes difficiles classiques tels que ceux du commis voyageur et du coloriage de graphes (Reeves, 1993).

La méthode choisie pour résoudre le problème de conception topologique de réseau cellulaire est une heuristique hybride combinant la recherche locale et la programmation par contraintes. Avant de décrire la recherche locale nous devons tout d'abord analyser ce qui est résolu avec les heuristiques du modèle de programmation par contraintes.

3.2.1 Stratégies de recherche du modèle de PC

Nous avons mentionné au chapitre précédent que l'ordre d'affectation de variables et de valeurs constituait les heuristiques de recherche du modèle de programmation par contraintes. L'objectif recherché de ces heuristiques est de fournir une bonne estimation du coût pour un réseau qui utilise un sous-ensemble de sites de BSC donné. Nous verrons qu'en séparant le modèle en deux sous-modèles et qu'en utilisant les stratégies d'affectation adéquates, le modèle de PC fournit une bonne estimation du coût. Nous séparons le modèle proposé initialement en deux sous-modèles pour éviter de propager sur la variable t_j lors de l'affectation de la variable x_i. Cette propagation est coûteuse en temps et ne fournit pas d'information supplémentaire. Nous en discuterons plus en

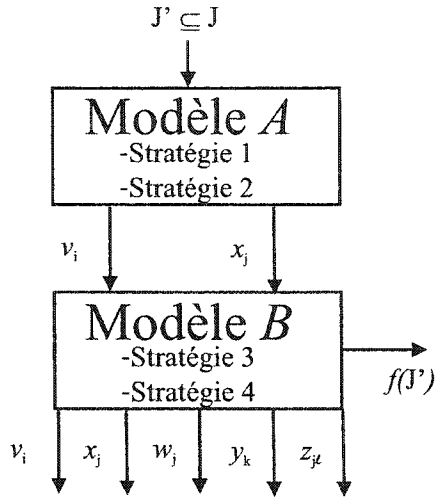

Figure 3.1: Modèles de programmation par contraintes

détail à la section 4.2.4.

Les contraintes du modèle sont conservées mais nous avons séparé celles reliées à l'affectation BTS-BSC (modèle A) de celles reliées à l'affectation des sites BSC-MSC (modèle B). Le modèle A reçoit en paramètre un sous-ensemble J' de sites de BSC qui seront utilisés pour constituer la solution (figure 3.1). Le modèle A commence par l'affectation BTS-BSC (v_i) : puisqu'elle détermine environ 80% du coût du réseau, il est justifié de commencer par cette étape. Le choix du type à installer à chacun des sites de BSC (x_j) est étroitement lié à l'affectation BTS-BSC, cette étape fera donc partie du modèle A.

Le modèle B reçoit en paramètre les résultats du modèle A (figure 3.1) et détermine les affectations BSC-MSC (w_j), ainsi que le type à installer à chacun des sites de MSC (y_k) et finalement le nombre de liens BSC-MSC ($z_{j\ell}$) de chacun des types.

3.2.1.1 Modèle A : affectation BTS-BSC

Les contraintes (3.4) et (3.5) sont utilisées dans ce modèle, le coût à minimiser est composé du coût des liens BTS-BSC et du coût d'installation des BSC aux sites choisis.

L'heuristique d'affectation de variables commence par affecter les BTS aux sites de BSC (voir stratégie 1). On essaie d'abord d'affecter chacun des BTS au site de BSC le plus près pour minimiser les coûts des liens. Il est possible qu'un BSC n'ait pas assez de capacité pour recevoir la demande de tous les BTS qui aimerait s'y connecter. Dans ce cas, les derniers BTS à être affectés prendront la valeur du deuxième site de BSC le plus près. En ordonnant les variables v_i en ordre décroissant de demande, on favorise les BTS les plus coûteux en premier, ceux qui ont plus de liens (η_i). Il est possible que quelques réaffectations BTS-BSC améliorent le coût de la solution mais nous verrons un peu plus loin que ce gain est petit.

Stratégie 1 : Affectation des BTS aux sites BSC

$$
\begin{array}{l}
forall_{i \epsilon I \text{ ordered by decreasing } \eta_i} \\
\quad tryall_{j \epsilon J \text{ ordered by increasing } d_{i\,j}} \\
\quad\quad v_i = j
\end{array}
$$

Cette étape complétée, nous pouvons déterminer les types à installer aux sites de BSC, ceux de moindre coût respectant les demandes des BTS (voir stratégie 2). En commençant par les types de moindre coût, la première solution qui satisfait toutes les contraintes est celle de plus faible coût pour les affectations BTS-BSC déterminées par la stratégie 1. En effet, ces affectations sont indépendantes les unes des autres. Nous pouvons donc installer, à chaque site de BSC, le plus petit type qui satisfait les demandes des BTS connectés sans égard à ce qui sera installé aux autres sites. On arrête donc la recherche après l'obtention de la première solution (*first solution*).

Stratégie 2 : Affectation des types aux sites BSC

$$
\begin{array}{l}
firstSolution(1) \\
\quad forall_{j \epsilon J} \\
\quad\quad tryall_{s \epsilon S \text{ ordered by increasing } cout_s} \\
\quad\quad\quad x_j = s
\end{array}
$$

3.2.1.2 Modèle B : affectation BSC-MSC

Le modèle B est sujet à l'ensemble des contraintes et à la fonction de coût définis dans le modèle de la section 3.1.3. Il est à noter que les variables v_i et x_j sont déjà fixées par le modèle A. Dans le modèle B, la stratégie d'affectation commence par déterminer le type à installer à chacun des sites de MSC. L'objectif est de trouver un sous-ensemble de sites de MSC qui minimisera le coût des liens BSC-MSC. On classe les variables y_k en ordre décroissant de distance avec l'ensemble des sites de BSC (voir stratégie 3). On commence par affecter les sites de MSC les plus loins de plusieurs sites de BSC. Étant donné que le type de moindre coût est le type $NULL$ (pas de BSC installé à ce site), les sites les plus loin recevront cette affectation de valeur. Les sites les plus près de plusieurs BSC seront affectés en dernier et auront comme affectation le plus petit type qui satisfait les demandes des BSC. Nous voulons ainsi favoriser les sites de MSC qui minimisent le coût de plusieurs liens BSC-MSC. On commence par installer à chaque site de MSC le type de moindre coût pour que la première solution trouvée soit celle de moindre coût.

Stratégie 3 : Affectation des types aux sites MSC

$$forall_{k \epsilon K \text{ ordered by decreasing } \sum_{j \epsilon J} d_{j\,k}}$$
$$tryall_{t \epsilon T \text{ ordered by increasing } cout_t}$$
$$y_k = t$$

En déterminant le nombre de liens de chaque type entre les sites de BSC et les sites de MSC, nous obtiendrons également les affectations BSC-MSC. En effet, les mécanismes de propagation réduiront les domaines représentant les affectations BSC-MSC à une seule valeur puisqu'il ne pourra y avoir d'affectation aux endroits où il n'y pas de liens. On essaie d'abord de ne pas installer de lien entre chacun des sites (voir stratégie 4). Lorsqu'il y a insatisfaction de la contrainte de flot, on ajoute un lien de moindre coût. Pour les jeux de données considérés, on n'utilisera pas plus de deux liens de moindre coût. En effet, les deux types de lien, DS1 et DS3, provenant de nos jeux de données ont respectivement des coûts

d'installation de 2000\$/km et de 4000\$/km, des coûts d'interfaces de 1000\$ et de 5000\$ et des capacités de 96 circuits et de 2688 circuits. On limite les affectations possibles pour les DS1 à deux puisqu'il est plus cher d'installer trois DS1 (6000\$ + 3000\$/km)$qu'un DS3$(4000\$ + 5000\$/km) pour plus d'un kilomètre. De plus, la capacité de trois DS1 est de beaucoup inférieure à celle d'un DS3. Il est alors plus avantageux d'utiliser des DS3 lorsque le nombre de liens requis dépasse deux DS1. C'est pourquoi nous introduisons max_ℓ qui spécifie le nombre maximum de liens pour chacun des types. La première solution qui satisfait les contraintes est celle de moindre coût puisqu'on commence par les liens de moindre coût. En effet, les types à installer sont indépendants entre eux. Il suffit de connaître les demandes des BSC pour en déduire le plus petit type à installer. Cette fois, contrairement à la stratégie 2, on ne peut pas spécifier d'arrêter la recherche à la première solution trouvée puisqu'il est possible qu'il n'y en ait pas étant donné les affectations des stratégies précédentes. On utilise alors une limite de temps obtenue par expérimentation (0.15 sec.). Ce temps a été ajusté pour permettre de trouver une première solution, s'il y en a une, même pour les plus gros réseaux.

Stratégie 4 : Affectation du nombre de liens de chaque type entre les BSC-MSC

$$
\begin{array}{l}
timeLimit(0.15) \\
\quad forall_{j \in J} \\
\quad\quad tryall_{\ell \in L \text{ ordered by increasing } cout_\ell} \\
\quad\quad\quad tryall_{nombre \in 0..max_\ell} \\
\quad\quad\quad\quad z_{k\,\ell} = nombre
\end{array}
$$

On se rappellera que l'objectif recherché de ces heuristiques est de fournir une bonne estimation du coût d'une solution pour un sous-ensemble de site de BSC.

Dans le modèle A, seule l'affectation BTS-BSC n'est pas optimale. Ce manque d'optimalité est dû au fait que ce ne sont pas tous les BTS qui peuvent être affectés au site de BSC de leur choix, faute de capacité à ce site de BSC. Il est donc possible qu'en enlevant les derniers BTS affectés à un site de BSC pour en réaffecter d'autres, nous améliorions le coût global d'affectation BTS-BSC. C'est à ce niveau

que réside notre plus grande incertitude sur la qualité de la solution trouvée par le modèle A. Par contre, pour des réseaux réels, les BTS et les sites de BSC sont géographiquement bien répartis et donc ils ne sont pas tous au même endroit. Dans ce contexte, il est possible qu'on dépasse la capacité d'un site de BSC mais pas de façon démesurée. Le petit nombre de BTS qui n'ont pu se connecter au site le plus près se connecteront au deuxième site le plus près. De plus, en donnant priorité aux sites de plus fortes demandes nous favorisons les choix des BTS les plus coûteux en premier. Le gain possible en permutant des affectations BTS-BSC par d'autres est très petit par rapport au coût de l'ensemble du réseau (moins de 1% en moyenne). Enfin, le coût du modèle B est très peu influencé par la perte d'optimalité du modèle A. Comme toutes les solutions possèdent le même ensemble J' de sites de BSC (ceux de l'ensemble de départ) les affectations BSC-MSC sont à peu près les mêmes indépendamment des choix du modèle A. Quelques fois, les demandes des BSC découlant des affectations du modèle A nous conduisent à un choix d'affectation BSC-MSC différent pour un même sous-ensemble de sites mais la plupart du temps ce n'est pas le cas.

Le modèle B peut aussi causer une perte d'optimalité si on ne lui laisse pas le temps nécessaire pour parcourir tout l'arbre de recherche. Pour éviter ce problème nous avons décidé de rechercher les meilleures affectations possibles pour les variables du modèle B, ce qui se traduit par le fait de parcourir tout l'arbre de recherche. Cette technique est très longue mais elle garantit la qualité des affectations des variables du modèle B. Nous sommes alors certains que la solution trouvée est la meilleure considérant les choix du modèle A.

3.2.2 Combiner le modèle de PC et la recherche locale

Le modèle de programmation par contraintes fournit donc une bonne estimation de la qualité d'une solution pour un sous-ensemble de sites de BSC. Nous devons maintenant déterminer quel sous-ensemble de sites de BSC conduira à la meilleure

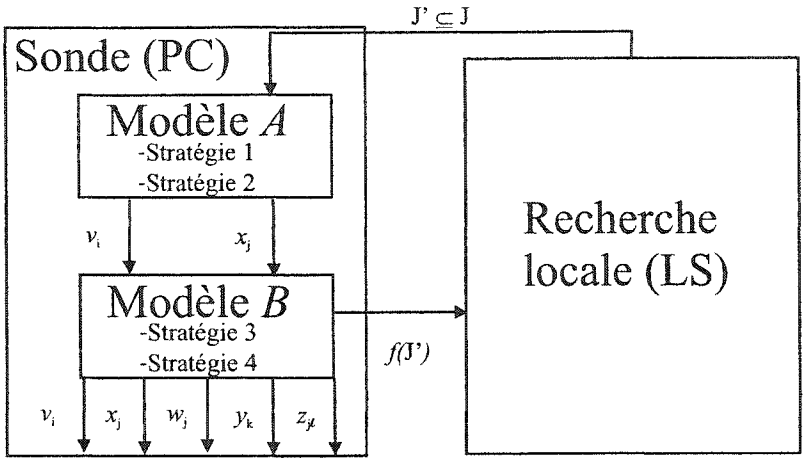

Figure 3.2: Combinaison de la recherche locale et du modèle de PC

solution. Nous utilisons une recherche locale à descente simple pour le faire. Le problème résolu à l'aide de la recherche locale est un sous-problème du problème global, soit de rechercher le sous-ensemble de sites de BSC qui conduira à un optimum local.

Du point de vue de la recherche locale, le modèle de programmation par contraintes ne sert qu'à évaluer le coût des différents voisins de la solution courante. La recherche locale fait abstraction de toutes les décisions prises dans le modèle de PC pour ne considérer que le coût associé au sous-ensemble de sites de BSC donné. Le modèle de PC fournit donc un estimé de qualité d'une solution pour un sous-ensemble de sites de BSC. C'est en fait comme si la recherche locale lançait une sonde de t secondes sur le modèle de PC pour obtenir un estimé de la qualité d'un sous-ensemble de sites de BSC (figure 3.2). Par exemple, imaginons que nous voulons connaître le coût obtenu d'une solution composée de 3 sites de BSC donnés. La sonde utilise le modèle de programmation par contraintes pour rechercher, pour ces 3 sites, la meilleure affectation possible après t secondes et retourne le coût de cette solution.

3.2.2.1 Concept de sonde

Nous utilisons un modèle de PC pour explorer le voisinage $N(x_k)$ de la solution x_k. Un avantage propre aux techniques de résolution en PC est l'élimination de toute solution non réalisable au cours de la recherche. Il n'est donc pas nécessaire d'affecter toutes les variables avant de s'apercevoir que la solution partielle n'est pas réalisable. La recherche locale avec sondes exploite ce principe en laissant le modèle de PC évaluer le coût d'un voisin suite à un mouvement local en complétant une solution partiellement spécifiée. Nous comparons les sondes entre elles pour déterminer le mouvement local qui conduit au meilleur voisin estimé. Il est à noter que le temps d'évaluation du coût d'un voisin est long comparativement au temps pris par une recherche locale sans sonde.

Habituellement, lors d'une recherche locale sans sonde, les variables qui ne sont pas impliquées dans le mouvement conservent leur affectation. Le temps requis pour évaluer le coût et la réalisabilité d'une solution est court puisque les affectations sont complètes. En outre, il suffit de vérifier qu'aucune contrainte n'est violée et le cas échéant d'en calculer le coût. Par contre, il est possible dans un tel cas que l'heuristique se bute sur de longues séries de solutions qui ne satisfont pas les contraintes. La recherche est alors dans une zone de solutions non réalisables voisines. Par exemple, pour un mouvement local qui enlève un site de BSC de la solution courante, il est certain que la solution voisine n'est pas réalisable puisque tous les BTS qui étaient connectés à ce site violent des contraintes de capacité.

Un premier avantage du concept de sonde est qu'il garantit de trouver un voisin qui satisfait toutes les contraintes, s'il existe. Un deuxième avantage est qu'il réduit l'espace de recherche aux variables sur lesquelles le mouvement local est appliqué. Par exemple pour notre problème, la sonde réduit l'espace de recherche aux sites de BSC faisant partie de la solution. Le coût de chacune des solutions correspond donc à une certaine configuration de variables locales. Un problème de cette méthode

est que la direction de descente dépend de la qualité du coût évalué pour un voisin et donc du temps de recherche de la sonde. Le temps de recherche de la sonde doit être assez long pour assurer que le coût calculé pour un voisin est représentatif du coût de la solution optimale pour ce voisin. Nous souhaitons donc qu'un temps de recherche plus long à l'itération k nous mène au même choix de voisin pour l'itération $k+1$.

L'espace de recherche du problème est de $|S|^{|J|}*|T|^{|K|}*|J|^{|I|}*|K|^{|J|}$ combinaisons de solutions possibles; ce qui correspond respectivement aux combinaisons des types pouvant être installés aux sites de BSC, des types pouvant être installés aux sites de MSC, des affectations BTS-BSC et des affectations BSC-MSC. Par exemple, pour un réseau typique de taille de 100 BTS, 30 sites de BSC et 10 sites de MSC, ce nombre s'élève à 10^{202} combinaisons : même si chaque solution s'évalue en une micro seconde, le temps de calcul serait 10^{179} milliards d'années pour comparer le coût de chaque combinaison. Il est à noter que ce calcul néglige la conservation de flots. La recherche locale avec sonde diminue le nombre de combinaison à $2^{|J|}$, soit à utiliser ou non chacun des sites de BSC dans la solution courante. Pour le problème de 30 sites de BSC cité plus haut, il y a 10^9 combinaisons possibles. L'espace de recherche de l'heuristique locale avec sonde est donc diminué par un facteur très important de l'ordre de 10^{192}. Par contre, le temps d'évaluer un voisin est augmenté par un facteur de 10^3. Il sera ainsi plus facile de diriger la recherche dans une bonne direction de descente lors de l'exploration mais chaque mouvement prendra beaucoup de temps à évaluer.

3.2.3 Heuristique de recherche locale

Le concept de sonde diminue l'espace de recherche de l'heuristique locale mais le temps pour évaluer un voisin est augmenté. Comme l'évaluation d'un voisinage prend un temps considérable, nous avons donc décidé de limiter notre heuristique à une recherche locale à descente simple. Conséquemment, nous utiliserons un

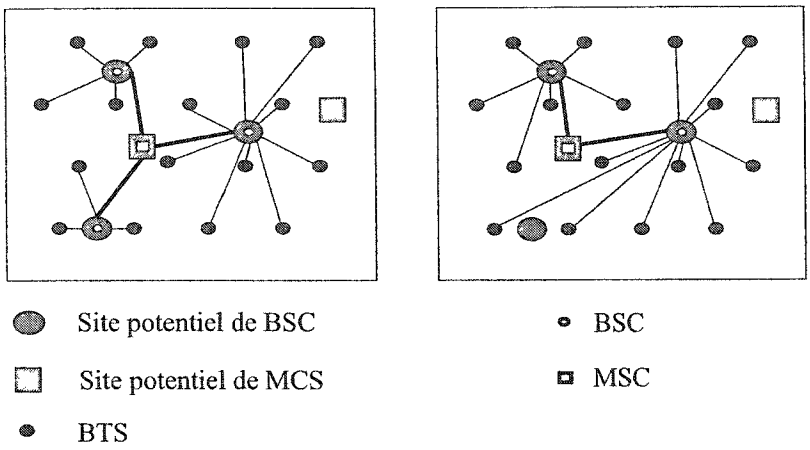

Figure 3.3: Mouvement de la recherche locale : le site de BSC le plus faiblement connecté est enlevé de solution courante

voisinage assez grand pour favoriser la prise de bonnes décisions à chaque itération. Le choix d'un bon voisin est primordial puisqu'il décide de la direction de descente.

3.2.3.1 Mouvement de la recherche locale

Le choix du voisinage réfère directement au problème étudié. Nous cherchons à améliorer la solution courante à chaque itération afin d'assurer une descente vers un optimum local de qualité. Dans le cas du problème de conception topologique de réseau cellulaire résolu à l'aide de la recherche locale avec sonde, nous utilisons une technique se basant sur les sites de BSC utilisés dans la solution courante. À chaque itération, un site de BSC est enlevé de la solution courante soit pour faire place à un site de BSC qui était exclu ou soit pour tout simplement enlever ce site. Ce qui caractérise une solution par rapport à une autre est donc les sites de BSC utilisés.

Il est trop long de permuter tous les sites de la solution courante à chaque itération. Pour accélérer la recherche dans le voisinage, nous permutons seulement les sites de BSC les moins utilisés (en terme de nombre de BTS connectés) par

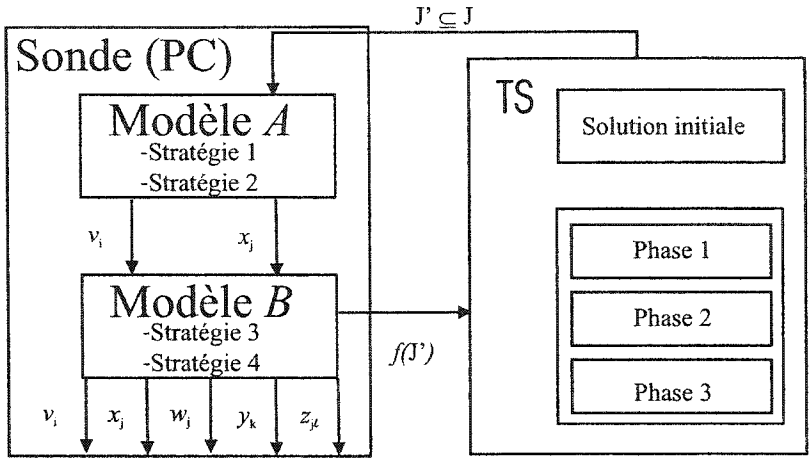

Figure 3.4: Combinaison de la recherche locale et du modèle de PC

d'autres sites qui ne font pas partie de la solution tel qu'illustré à la figure 3.3. Les sites de la solution x_k les plus fortement connectés sont bons puisqu'ils minimisent le coût des liens de plusieurs BTS, chaque BTS étant affecté au site de BSC le plus près. Nous cherchons donc les sites de BSC parmi ceux qui ne sont pas dans la solution qui compléteront adéquatement les sites de BSC les plus fortement connectés. L'heuristique de recherche locale est divisée en trois phases tel qu'illustré à la figure 3.4.

Phase 1

L'heuristique de recherche locale commence par classer les sites de BSC de la solution x_k en ordre croissant du nombre de BTS connectés (voir algorithme phase 1). Les n sites les plus faiblement connectés sont ensuite remplacés, un à un, avec les m sites qui ne font pas partie de la solution. De plus, l'heuristique essaie de simplement enlever de la solution x_k, un à un, les sites les plus faiblement connectés. Nous lançons la sonde pour chacune des permutations et nous conservons la meilleure solution pour l'itération subséquente. Il y a donc $n * (m + 1)$ relances à chaque itération. Il est important de noter que le nombre de sites utilisés ne peut pas

augmenter. La décision de diminuer le nombre de sites de BSC a des conséquences très grandes puisqu'elle élimine toutes les solutions qui contiendraient plus de sites de BSC.

ALGORITHME PHASE 1 (*void*)

1 $o \leftarrow$ nombre de sites utilisés par le modèle
2 $J \leftarrow$ tous les sites de BSC disponibles
3 $n \leftarrow$ nombre de sites à enlever
4 $m \leftarrow$ nombre de sites à ajouter
5 **repeter**
6 trier J en ordre croissant de nombre de BTS connectés
7 $J' \leftarrow$ les o premiers sites de J
8 $J'' \leftarrow$ les autres sites de J
9 **for** $i \leftarrow 1 \ldots n$ **do**
10 enlever le i^{eme} site de J'
11 lancer la sonde pour déterminer le coût du voisin
12 **for** $j \leftarrow 1 \ldots m$ **do**
13 remplacer le i^{eme} site de J' par le j^{eme} de J''
14 lancer la sonde pour déterminer le coût du voisin
15 **endfor**
16 **endfor**
17 comparer les sondes entre elles et conserver la solution de meilleur coût
18 **tant que** $\exists x \epsilon N(x_k)$ **telle que** $f(x) \leq f(x_k)$

Phase 2

La phase 2 commence lorsque l'heuristique a atteint un premier minimum local. Nous augmentons alors le voisinage de la recherche pour sortir de ce minimum local (voir algorithme phase 2). Le deuxième voisinage exploré se fait sur les mêmes critères que le premier, soit sur les sites de BSC, mais le nombre de voisins visités est augmenté de façon importante. En effet, la recherche ne se fait plus que sur les

sites les plus faiblement connectés mais sur tous les sites. Nous explorons donc un deuxième voisinage exploitant le même mouvement mais dont le nombre de voisins visités est augmenté. Comme nous avons deux voisinages visités durant la descente, la recherche s'apparente donc à un VND bien que les mouvements locaux soient du même type. La méthode proposée tente de sauter directement dans une autre zone de minimum en mettant en doute les décisions prises pour les sites de BSC les plus fortement connectés. Pour ce faire, l'heuristique permute, un à un, les sites faisant partie de la solution x_k avec tous ceux qui n'en font pas partie, en plus de tout simplement enlever le site de la solution x_k. Par conséquent, n et m prendront respectivement la valeur du nombre de sites dans la solution x_k et du nombre de sites qui ne sont pas dans la solution x_k. Nous croyons qu'il y a probablement un site plus fortement connecté qui ne va plus avec l'ensemble des sites de la solution x_k. Comme cette étape est très longue, l'heuristique n'utilise cette méthode qu'à la première itération de la phase 2 et ce, pour changer de minimum local. Pour les itérations subséquentes, la recherche continue avec les mêmes paramètres que ceux énoncés lors de la description de la phase 1. La phase 2 se termine lorsqu'on atteint un deuxième minimum local.

ALGORITHME PHASE 2 (*void*)

1 $o \leftarrow$ nombre de sites utilisés par le modèle
2 $J \leftarrow$ tous les sites de BSC disponibles
3 $n \leftarrow o$
4 $m \leftarrow$ **card**$(J) - o$
5 **repeter**
6 trier J en ordre croissant de nombre de BTS connectés
7 $J' \leftarrow$ les o premiers sites de J
8 $J'' \leftarrow$ les autres sites de J
9 **for** $i \leftarrow 1 \ldots n$ **do**
10 enlever le i^{eme} site de J'
11 lancer la sonde pour déterminer le coût du voisin

12 **for** $j \leftarrow 1 \ldots m$ **do**

13 permuter le i^{eme} site de J' par le j^{eme} de J''

14 lancer la sonde pour déterminer le coût du voisin

15 **endfor**

16 **endfor**

17 $n \leftarrow$ nombre de sites à enlever de la phase 1

18 $m \leftarrow$ nombre de sites à ajouter de la phase 1

19 comparer les sondes entre elles et conserver la solution de meilleur coût

20 **tant que** $\exists x \epsilon N(x_k)$**telle que** $f(x) \leq f(x_k)$

Phase 3

La phase 3 sert à améliorer la dernière solution trouvée à la phase 2. Les modèles de PC utilisés trouvent de bonnes solutions pour un ensemble de sites donnés mais on se rappelle que les solutions ne sont pas optimales. La phase 3 est particulièrement utile pour les grands réseaux, ceux pour lesquels il n'est pas possible de trouver le choix optimal de sites de MSC à utiliser pour une affectation BTS-BSC donnée. Il s'agit d'augmenter le temps de recherche de la sonde du modèle A pour améliorer le coût d'affectation BTS-BSC et surtout d'augmenter le temps de recherche de la sonde du modèle B pour assurer de trouver les sites de MSC à utiliser de façon optimale. Il s'agit donc de relancer une grande sonde sur le dernier sous-ensemble de sites de BSC pour obtenir une solution de meilleur coût.

3.2.4 Solution initiale

Il y a deux considérations importantes dans le choix de la solution initiale. D'abord, plus elle est de bonne qualité, plus la recherche est rapide puisqu'on est déjà près de la solution finale. Par contre, la qualité de la solution initiale a un impact sur la descente de la recherche locale. Effectivement, une solution comportant un nombre de sites de BSC avoisinant le nombre de sites de BSC

de la solution d'optimum local aura un très bon coût. Par contre, il sera très difficile d'améliorer cette solution, la recherche locale terminera rapidement avec un très petit gain. Limiter le nombre de sites de BSC de la solution initiale c'est aussi limiter le nombre d'itération de la recherche locale à descente simple puisque le nombre de sites utilisés ne peut pas augmenter. Dans ce contexte, meilleur est le coût de la solution initiale, plus il y a de décisions sur lesquelles il n'est plus possible de revenir étant donné une recherche locale limitée. L'objectif est d'obtenir une solution initiale d'assez bonne qualité pour ne pas perdre de temps dans une recherche locale triviale. Par contre, la solution initiale ne doit pas prendre des décisions importantes sur lesquelles nous ne pourrons plus revenir lors de la recherche locale.

La recherche de la solution initiale se fait à l'aide de la sonde fournie par les modèles de PC et des stratégies de recherche décrites dans la section précédente. Pour obtenir une première solution, nous devons déterminer un ensemble comportant o sites de BSC à fournir aux modèles. Plus le nombre de sites utilisés se rapproche du nombre de sites totaux, moins la solution initiale confine le domaine de la recherche locale. En fait, déterminer les sites utilisés est moins important que de déterminer leur nombre. Les moins bons sites de la solution initiale pourront être permutés avec ceux qui n'en font pas partie lors de la recherche locale. Par contre, l'heuristique de recherche locale ne permet pas d'augmenter leur nombre. On fixe donc le nombre de sites de BSC utilisés à un nombre assez grand pour s'assurer d'être au-dessus du nombre de sites utilisés par la solution d'optimum locale. Ce nombre a été déterminé par expérimentation comme nous le verrons à la section 4.2.5.

Les sites de BSC qui composent la solution initiale ont aussi leur importance puisque certains resteront dans la solution jusqu'à la fin de la recherche, la recherche locale ne s'effectuant que sur les sites les moins utilisés. Nous devons donc nous assurer de bien choisir les sites les plus utilisés. Nous classons d'abord tous les

sites de BSC en ordre croissant de somme de distances avec les p BTS les plus près (voir algorithme initial). On choisit ainsi les sites de BSC où environ p BTS se connecteront à moindre coût. Nous conservons les o sites de plus petite somme de distances avec p BTS et lançons une première sonde. Les sites de BSC ainsi choisis ont une importance relative puisqu'on essaiera par la suite de les permuter avec ceux qui ne sont pas dans la configuration initiale, q sites à la fois.. On relance la sonde à chaque permutation de q sites et on choisit la solution de moindre coût comme solution initiale. Cela assure de lancer la sonde pour que chacun des sites de BSC se retrouve quelques fois dans l'ensemble des solutions évaluées. On remarque que les sites faisant partie de la première configuration, ceux de meilleure somme de distance avec les p BTS les plus près, ont plus de chance de se retrouver dans la solution initiale puisqu'on ne permute que q sites à la fois. Il y aura donc $(o - q)$ sites de BSC de la première configuration dans la solution initiale. Une discussion sur le choix des valeurs pour les paramètres o, p et q se trouve à la section 4.2.5

ALGORITHME SOLUTION INITIALE (*void*)

1 $o \leftarrow$ nombre de sites utilisés par le modèle
2 $p \leftarrow$ nombre de BTS
3 $q \leftarrow$ nombre de sites à permuter
4 trier J en ordre croissant de distance avec p BTS
5 $J' \leftarrow$ les o premiers sites de J
6 lancer la sonde sur l'ensemble J' pour déterminer le coût de la configuration
7 **repeter**
8 permuter q sites de J' avec q sites qui ne sont pas dans J'
9 lancer la sonde sur l'ensemble J' pour déterminer le coût de la configuration
10 **tant que** $\exists$ des permutations distinctes possibles
11 retourner J' de meilleur coût

CHAPITRE 4

RÉSULTATS ET DISCUSSION

Nous discuterons dans ce chapitre des principales constatations découlant de l'analyse des résultats lors du développement de l'heuristique. Ces constatations ont mené à des décisions importantes guidant nos travaux vers l'heuristique hybride de recherche locale et de programmation par contraintes proposée. Nous débuterons le chapitre par la présentation des données utilisées pour évaluer la performance de l'heuristique. Par la suite, nous discuterons des choix de stratégies de recherche du modèle de PC et de l'ajout de la recherche locale. Pour terminer, nous comparerons les résultats finaux obtenus par notre heuristique avec ceux obtenus avec l'heuristique tabou proposée par Chamberland et Pierre (2002).

L'heuristique a été implantée avec ILOG OPL Studio 3.5, un langage script fournissant une interface pour l'utilisation du solveur de programmation par contraintes, ILOG Solver 5.2. Les tests ont été effectués sur un processeur Athlon de ADM (1.6 GHz et 512 Mo de RAM).

4.1 Description des données

Nous avons utilisé pour nos tests trois types de BTS, trois types de BSC et trois types de MSC. Leurs caractéristiques sont présentées respectivement dans les tableaux 4.1, 4.2 et 4.3. De plus, nous avons utilisé des DS1 pour les liaisons BTS-BSC alors que des DS1 et des DS3 ont été utilisés pour les liaisons BSC-MSC. Les coûts des interfaces sont donnés dans le tableau 4.4 et le coût des types de lien dans les tableaux 4.5 et 4.6.

Pour l'ensemble des tests, les problèmes ont été générés en utilisant $|I|$ coordonnées correspondant aux localisations de BTS, $|J|$ coordonnées correspondant

Tableau 4.1: Caractéristiques des types de BTS

	Type A	Type B	Type C
Capacité (circuit)	96	288	576
Nombre d'interfaces DS1	1	3	6

Tableau 4.2: Caractéristiques des types de BSC (incluant les coûts d'installation)

	Type A	Type B	Type C
Capacité (circuit)	5000	10 000	15 000
Nombre maximal d'interfaces BTS	15	30	60
Nombre maximal d'interfaces MSC	15	30	60
Coût ($)	50 000	90 000	120 000

Tableau 4.3: Caractéristiques des types de MSC (incluant les coûts d'installation)

	Type A	Type B	Type C
Capacité (circuit)	100 000	200 000	300 000
Nombre maximal d'interfaces BSC	50	100	150
Coût ($)	200 000	350 000	500 000

Tableau 4.4: Caractéristiques des types d'interfaces (incluant les coûts d'installation)

Type d'interface	Capacité (circuit)	Coût ($)
DS1	96	500
DS3	2688	2500

Tableau 4.5: Caractéristiques des liens BTS-BSC (incluant les coûts d'installation)

Type de BSC	Nombre de liens DS1	Capacité (circuit)	Coût ($)
A	1	96	2000
B	3	288	3000
C	6	596	4000

Tableau 4.6: Caractéristiques des liens BSC-MSC (incluant les coûts d'installation)

Type de lien	Capacité (circuit)	Coût ($/km)
DS1	96	2000
DS3	2688	4000

aux localisations des BSC et $|K|$ coordonnées correspondant aux localisations des MSC. Ces coordonnées ont été générées aléatoirement à l'intérieur d'une région de 10 000 km^2, selon une distribution uniforme. Les types des BTS ont été déterminés de façon aléatoire entre les types fournis au tableau 4.1, selon une distribution uniforme. Finalement, les demandes entre chaque paire de BTS et entre les BTS et le réseau public ont été générées aléatoirement dans un intervalle de [0, 0.2] erlang, selon une distribution uniforme. Nous avons utilisé un ensemble de problèmes générés selon ces caractéristiques par Chamberland et Pierre (2002).

Tous nos résultats sur ces données seront comparés avec une borne issue d'une relaxation des contraintes de conservation de flot et avec l'heuristique tabou proposée par Chamberland et Pierre (2002).

4.2 Évolution de l'heuristique hybride

Nous avons d'abord développé un modèle de PC en omettant les contraintes de conservation de flot. Déjà nous avions pris une décision importante, celle de définir la topologie du réseau pour ensuite vérifier que celle-ci satisfaisait bien les contraintes de conservation de flot.

4.2.1 Stratégies de recherche

Nous avons décrit au chapitre précédent (section 3.2.1) les stratégies de recherche que nous avons utilisées pour la résolution du problème. Par contre, nous n'avons pas décrit les constatations qui ont mené à ce choix.

Rappelons qu'un facteur important lors de la résolution de problèmes en PC est de déterminer quelle variable affecter en premier (Section 2.2.2.3, Ordre d'affectation des variables et des valeurs). De ce choix découle la topologie de l'arbre de recherche, la réduction des domaines et donc le temps de recherche avant d'obtenir une solution. En outre, la première variable à être affectée a une importance capitale puisque l'ordre d'affectation des autres variables se fera conformément à ce choix. Nous pouvions commencer par déterminer (ordre des stratégies de recherche):

- (1) les types installés aux sites de BSC (x_j);
- (2) les types installés aux sites de MSC (y_k);
- (3) les affectations des BTS aux BSC (v_i);
- (4) les affectations des BSC aux MSC (w_j).

4.2.1.1 Ordre d'affectation de variables entre les BTS et les BSC

Comme environ 80% du coût d'un réseau est relié à la section entre les BTS et les BSC, les variables les plus intéressantes à fixer en premier sont celles reliées à cette section du réseau : v_i (affectation BTS-BSC) et x_j (type installé au site de BSC). De plus, ces variables sont étroitement liées. Par exemple, lorsque l'affectation BTS-BSC est complétée, il devient plus facile de déterminer les types à installer aux sites de BSC. Pour chaque site de BSC, le plus petit type (celui de moindre coût) qui satisfait les demandes des BTS sera installé. Le contraire est aussi vrai, lorsqu'on connaît les types installés aux sites de BSC, il devient plus facile de déterminer les affectations BTS-BSC. Elles se font selon les capacités des types installés aux sites de BSC, ce qui limite grandement le nombre de combinaisons à explorer. Chamberland et Pierre (2002) ont exploité cet aspect en fixant en premier les sites utilisés. Pour de petits réseaux, la méthode consistant à fixer les types des

BSC en premier (stratégie 1) est la meilleure stratégie (tableau 4.7) pour trouver la solution optimale mais pour des réseaux de taille plus réaliste cette méthode ne fournit pas de bonnes solutions dans un délai raisonnable (tableau 4.8).

Les résultats des tests du tableau 4.7 correspondent à une moyenne de six essais sur des réseaux de taille de dix BTS, de quatre sites de BSC et de quatre sites de MSC. Il est à noter qu'au moment des tests, nous comparions aussi les stratégies correspondant à la section du réseau entre les BSC et les MSC (stratégies 2 et 4). Comme nous avions associé les stratégies 1 avec 2 et 3 avec 4, nous pouvons quand même différencier l'effet de placer la stratégie 1 avant la stratégie 3 et vice-versa. Lorsque l'ordre d'affectation n'est pas mentionné pour une variable, cela indique que nous utilisons l'ordre par défaut pour les variables et les valeurs. Par exemple, la stratégie 2-1-*d*-*d* (le *d* représente l'utilisation de stratégies par défaut) correspond à affecter la variable y_k, suivie par x_j et à utiliser l'ordre par défaut pour les autres variables. L'ordre par défaut correspond à l'ordre d'apparition des variables dans le modèle, soit la stratégie 1-2-3-4. Dans notre cas, lorsque nous ne le mentionnons pas, la stratégie 1 sera avant 2 et la stratégie 3 avant 4. Il y a aussi un ordre pour affecter les variables de même type. Par exemple pour la stratégie 3, les variables v_i seront choisies en ordre décroissant du nombre de liens partant du BTS i. L'ordre par défaut pour les variables de même type est celui de plus petit domaine d'abord. Il est à noter que l'ordre d'affectation de valeur a aussi un impact sur l'efficacité de la recherche. L'ordre d'affectation de valeurs que nous utilisons est basé sur le coût : pour la stratégie 1 par exemple, nous commençons par affecter aux sites de BSC le type (valeur) de moindre coût. Tout cela explique la différence entre les résultats obtenus pour les stratégies 2-1-*d*-*d* et 2-1-3-4. Le détail sur les heuristiques des choix de variables et de valeurs que nous avons utilisées se retrouve à la section 3.2. Nous remarquons que les tests qui débutent en fixant le type à installer aux sites (stratégie 1) sont plus rapides pour obtenir la solution optimale, moins de 4 secondes comparativement à plus de 10 secondes lorsqu'on commence

Tableau 4.7: Comparaison des stratégies de recherche pour des réseaux de petite taille pour le modèle sans les contraintes de conservation de flot

|I|	|J|	|K|	Optimum (k$)	Stratégie	Temps (sec)
10	4	4	1333	2-1-d-d	1.26
10	4	4	1333	2-1-3-4	2.08
10	4	4	1333	1-2-d-d	2.48
10	4	4	1333	1-2-3-4	3.38
10	4	4	1333	3-4-d-d	10.52
10	4	4	1333	3-4-1-2	215.08

par l'affectation BTS-BSC (stratégie 3).

Les résultats des tests du tableau 4.8 correspondent à la meilleure solution obtenue par la stratégie de recherche après une heure. Nous comparons ces résultats avec la borne inférieure (BI) relaxée des contraintes de conservation de flot. Nous remarquons que les tests qui impliquent la stratégie 3 en premier se rapprochent davantage de la solution relaxée (moins de 12%) que ceux qui impliquent la stratégie 1 en premier (à plus de 100%) et ce, après une heure.

Une difficulté avec la méthode qui fixe les types des BSC (stratégie 1) en premier en programmation par contraintes est de déterminer un ensemble de types qui conduit à une solution qui satisfait l'ensemble des demandes des BTS. Effectivement, puisqu'on débute par affecter les plus petits types (capacités) aux sites pour minimiser le coût, toutes les premières solutions n'auront pas assez de capacités pour satisfaire la demande des BTS. Par exemple, la première solution partielle ainsi formée fixera tous les sites de BSC au type de plus faible coût (de plus faible capacité). La capacité résultante pour l'ensemble des BSC sera loin d'être suffisante pour satisfaire les demandes des BTS. Il s'ensuivra une longue série d'échecs avant d'obtenir la première solution réalisable, solution qui satisfait les demandes des BTS. Cette perte de temps est bien visible dans les résultats des réseaux de plus grande taille (tableau 4.8).

Tableau 4.8: Comparaison des stratégies de recherche pour des réseaux de taille réelle pour le modèle sans les contraintes de conservation de flot

|I|	|J|	|K|	Borne inf.(k$)	Stratégies	Coûts (k$)	Temps(sec)	Écart(%)
100	10	10	6860.9	3-4-d-d	6934.5	3600	1.07
100	10	10	6860.9	4-3-d-d	7682.5	3600	11.98
100	10	10	6860.9	2-1-d-d	14396.5	3600	109.83
100	10	10	6860.9	2-d-d-d	15036.5	3600	119.16
100	10	10	6860.9	1-2-d-d	15148.5	3600	120.79

Pour améliorer la méthode qui commence par fixer le type en premier (stratégie 1) nous avons déterminé la capacité minimale à installer pour l'ensemble des sites de BSC. Ainsi, l'heuristique coupe l'arbre de recherche aussitôt que la somme des capacités de la solution partielle conduit vers un échec. Nous avons ajouté une contrainte redondante pour mettre en place ce mécanisme. Une contrainte redondante ne contraint pas davantage le problème : le problème demeure le même mais, lors de sa résolution, les mécanismes de propagation utilisent cette contrainte pour propager de l'information supplémentaire. Nous avons donc ajouté une contrainte stipulant que la somme des demandes de l'ensemble des BTS doit être plus petite ou égale à la somme des capacités des types installés aux sites BSC.

Contrainte Redondante

$$\sum_{i \epsilon I} \alpha_i \leq \sum_{j \epsilon J} \alpha_{x_j} \tag{4.1}$$

Cet ajout améliore considérablement (par un facteur de 10) l'efficacité de la recherche lorsqu'on commence par la stratégie 1. Par contre, cette amélioration n'est pas suffisante pour les réseaux de grande taille. Utiliser la stratégie 3 en premier demeure la meilleure méthode pour obtenir une solution de bonne qualité dans un délai raisonnable.

4.2.1.2 Ordre d'affectation de valeurs entre les BTS et les BSC

Le problème qui demeure, même avec l'ajout de la contrainte redondante, est que nous n'avons pas de critère pour guider le choix des types à installer aux sites. Il est difficile de définir les sites auxquels nous devons affecter une plus grande capacité. Nous n'avons donc pas d'heuristique de choix de valeur pertinente pour les variables x_j. Au chapitre 3, nous avions discuté de l'importance de ces heuristiques.

Nous obtenons une meilleure solution (tableau 4.8) lorsque nous déterminons les affectations BTS-BSC en premier (stratégie 3) pour des réseaux de taille réelle. La raison est que nous avons un critère intéressant pour attribuer les valeurs aux variables. On fixe chaque BTS au site de BSC le plus près, ce qui aide grandement la fonction coût de la solution. De plus, commencer avec les variables d'affectation BTS-BSC (v_i) contraint moins les autres variables : du moins les implications sont moindres pour les autres variables. C'est notamment le cas, pour la variable x_j, puisque dans le pire des cas l'affectation v_i contraint les variables x_j à prendre le type de plus grande capacité. Par exemple, lorsque plusieurs BTS choisissent le même site de BSC, ce dernier est contraint, dans le pire des cas, à la valeur du type de plus grande capacité. Contrairement, lorsqu'on commence par fixer le type à installer en premier (stratégie 1), ce choix influence grandement le domaine des variables d'affectation BTS-BSC x_i. Par exemple, les derniers BTS à être affectés prendront la valeur du site de BSC où il y a encore de la capacité et ce, même si ce dernier est à une grande distance. La fonction de coût en sera grandement affectée.

Un problème avec la méthode qui débute par la stratégie 3 est qu'on utilisera presque tous les sites de BSC. En effet, lorsqu'on affecte à chaque BTS le site de BSC le plus près, il y a de fortes chances que chaque site de BSC ait au moins un BTS assez près pour s'y connecter. La solution a donc un coût relativement élevé puisqu'elle utilise tous les sites de BSC. Par contre, la solution est obtenue rapidement et le coût d'affectation BTS-BSC est minimisé. Dans le cadre d'une

recherche heuristique un critère très important est la rapidité de l'évaluation d'une solution. Nous avons donc choisi de commencer par la stratégie 3 suivie de la stratégie 1.

4.2.1.3 Ordre d'affectation de variables entre les BSC et les MSC

La première partie des affectations, celles entre les BTS et les sites de BSC, est terminée. Nous avons donc un nouveau problème beaucoup plus petit. Ce problème consiste à déterminer l'affectation BSC-MSC et la localisation des MSC connaissant les BSC utilisés et leurs demandes. Ce problème est le même que celui que nous venons de discuter entre les BTS et les BSC. Par contre, les instances pour cette partie du réseau sont plus petites. En effet, il y a toujours plusieurs BTS pour un BSC et plusieurs BSC pour un MSC dans un réseau. Les affectations BSC-MSC sont par conséquent plus faciles que les affectations BTS-BSC. Il est même généralement possible de trouver la solution optimale en un temps raisonnable pour cette partie du problème. Comme il faut peu de MSC pour satisfaire les demandes de tous les BSC, il y aura peu de solutions non réalisables lorsqu'on commence par affecter les types aux sites de MSC (stratégie 2). La capacité n'est pas un facteur limitatif puisque souvent un ou deux MSC du type le plus petit suffira pour satisfaire l'ensemble de la demande des BSC. Contrairement à ce que nous avions lorsque nous déterminions les types à installer aux sites de BSC (stratégie 1) avant les affectations BTS-BSC (stratégie 3), il n'y a pas de longues séries d'échec lorsqu'on commence par déterminer les types à installer aux sites de MCS (stratégie 2) avant les affectations BSC-MSC (stratégie 4). On constate expérimentalement sur de petits réseaux du tableau 4.7 que fixer le type en premier est plus rapide que de fixer l'affectation en premier pour atteindre la solution optimale pour le sous-problème entre les BTS et les sites de BSC. Il en va de même avec la partie du réseau entre les BSC et les sites de MSC : pour obtenir la solution optimale nous commençons donc par fixer le type à installer aux sites MSC (stratégie 2) avant les affectations

BSC-MSC (stratégie 4).

L'ordre des stratégies d'affectation sera donc de déterminer les affectations BTS-BSC (stratégie 3), les types à installer aux sites de BSC (stratégie 1), les types à installer aux sites de MSC (stratégie 2) et enfin les affectations BSC-MSC (stratégie 4).

4.2.2 Ajout d'une heuristique combinée au modèle de PC

Nous avons mentionné que les stratégies de recherche trouvaient la solution optimale plus rapidement en fixant en premier les sites BSC pour les petits réseaux (tableau 4.7). Nous avons rejeté cette stratégie parce que la première solution était longue à obtenir pour des réseaux de taille plus réaliste. Le problème, même après l'ajout de la contrainte redondante 4.1, était que nous n'avions pas de bonne heuristique de choix de valeur pour les variables x_j. Par contre, fixer le type des BSC (stratégie 1) en premier permet de limiter le nombre des sites utilisés dans la solution contrairement à fixer les affectations BTS-BSC (stratégie 3) en premier. Nous aimerions retrouver cet aspect lors de la recherche de solution. Le moyen que nous avons employé fut d'ajouter une heuristique au-dessus du modèle de PC pour limiter le nombre de sites utilisés. Ainsi, nous exploitons le fait que fixer les sites BSC utilisés en premier nous mène vers l'optimum plus rapidement tout en évitant le problème de choix de valeur que nous avions en implantant cette stratégie dans le modèle de PC. Il est important de noter que la recherche heuristique choisit les sites de BSC utilisés dans une solution mais ne choisit pas leur capacité (type). Ceci est important puisque le nombre de BTS affectés à un site de BSC ne sera contraint que par la capacité maximale pouvant être installée à ce site. Les BTS peuvent donc être affectés au site de BSC le plus près jusqu'à concurrence de la capacité maximale d'un BSC.

Nous avons d'abord lancé la recherche sur le modèle de PC en diminuant de un le nombre de sites de BSC permis dans la solution à chaque itération. Les résultats

Tableau 4.9: Résultats lorsque le nombre de sites de BSC constituant une solution est limité par l'heuristique hybride utilisant le modèle sans les contraintes de conservation de flot

|I|	|J|	|K|	BI(k$)	# de BSC	Coûts (k$)	Temps(sec)	Écart(%)
50	10	10	4051.50	6	4080.87	5000	0.72
50	10	20	4147.98	9	4365.77	5000	5.25
100	10	10	6860.90	9	7040.98	5000	2.62
50	20	10	3316.61	10	3560.46	10 000	7.35
50	20	20	3557.43	10	3746.25	10 000	5.31
100	20	10	6238.35	11	6684.98	10 000	7.15
100	20	20	6928.86	13	7272.43	10 000	4.96
50	30	20	3583.75	11	3893.79	15 000	8.65
100	30	10	6114.51	11	7279.26	15 000	19.05
150	30	10	7711.31	18	8869.44	15 000	15.02

du tableau 4.9 montrent que la technique est prometteuse puisque les résultats sont assez près de la borne inférieure (BI) calculée sans les contraintes de conservation de flot.

4.2.3 Ajout de l'heuristique de recherche locale

Nous décrivons dans cette section les constatations qui nous ont mené au choix du voisinage exploré. Nous avons étudié les résultats obtenus (tableau 4.9) pour constater que les sites de BSC les plus faiblement connectés (en terme de nombre de BTS) étaient les moins intéressants d'une solution. Inversement, les sites qui ont le plus de BTS connectés sont de bons candidats. En effet, comme chaque BTS se connecte au site de BSC le plus près, plus le nombre de BTS connectés à un site de BSC est élevé plus il y aura de BTS qui auront un bon coût de liaison BTS-BSC. Les sites de BSC fortement connectés ont donc un impact positif sur la fonction de coût. Il devient alors intéressant de réaffecter les BTS des sites de BSC les plus faiblement connectés. Nous utilisons une heuristique locale à descente simple pour explorer ce voisinage. Le mouvement entre deux solutions voisines se définit comme

Tableau 4.10: Résultats de l'heuristique hybride de recherche locale à descente simple combinée au modèle de PC sans les contraintes de conservation de flot

|I|	|J|	|K|	BI(k$)	TS (k$)	# de BSC	Coûts (k$)	Écart (%)	
							vs BI	vs TS
50	20	10	3316.61	3321.10	7	3323.71	0.21	0.08
50	20	20	3557.43	3583.20	8	3590.07	0.92	0.19
100	20	10	6238.35	6651.10	9	6431.09	3.09	-3.31
100	20	20	6928.86	7150.90	10	7096.34	2.42	-0.76
50	30	20	3583.75	3591.70	8	3628.38	1.25	1.02
100	30	10	6114.51	6217.30	10	6412.45	4.87	3.14
150	30	10	7711.31	8417.60	18	8620.26	11.79	2.41

suit : soit permuter le site faiblement connecté par un autre qui ne fait pas partie de la solution ou soit, enlever le site faiblement connecté de la solution. Il est à noter qu'avec ce voisinage le nombre de sites utilisés ne peut pas augmenter.

Les résultats du tableau 4.10 démontrent bien que la recherche proposée améliore substantiellement les résultats du tableau 4.9. Ce sont les premiers tests qui se rapprochent très bien de la borne inférieure (BI) et de la recherche tabou (TS) de Chamberland et Pierre (2002). Par contre, nous devons rappeler que les résultats obtenus font fi des contraintes de conservation de flot.

4.2.4 Ajout des contraintes de conservation de flot

Les contraintes de conservation de flot sont négligées dans le modèle développé. Pour en tenir compte, nous devons ajouter les contraintes (3.10) et (3.11) au modèle. On remarque que la propagation sur ces contraintes se fait lorsque les variables v_i (affectation BTS-BSC) sont fixées. À chaque affectation de valeur de la stratégie 3, il y aura propagation de l'information pour diminuer le domaine de t_j (le nombre de communications partant du BSC j vers le MSC auquel il est connecté). Le lien entre la variable v_i et la variable t_j n'étant pas très évident, voici un exemple pour l'expliquer. Soient deux BTS qui ne sont pas connectés au même site de BSC :

puisque la structure du réseau est en arbre, nous pouvons déduire que leur communication devra nécessairement être transmise via des liens BSC-MSC. Autrement dit, lorsque deux BTS sont connectés au même site de BSC, leur communication se fera sans lien BSC-MSC. Nous pouvons alors compter les communications partant des BSC sachant quels BTS lui sont connectés. Par contre, cette propagation est extrêmement coûteuse et n'améliore pas la recherche. Elle est coûteuse puisqu'elle implique une double sommation à chaque affectation de v_i. De plus, la propagation sur la variable t_j n'améliore pas l'efficacité de la recherche puisque aucune solution ne sera éliminée. En effet, t_j ne sera pas contraint puisque aucune limite de capacité n'est encore affectée dans le réseau. Nous devons donc empêcher la propagation sur ces contraintes tant que les variables v_i ne sont pas complètement fixées. Suite à cette affectation, nous pourrons calculer la valeur de tous les t_j connaissant v_i.

Pour pallier ce problème nous avons décidé de diviser le modèle en deux, un premier modèle pour fixer v_i et x_j et un deuxième modèle qui utilise le résultat du premier pour compléter le réseau en fixant w_j, y_k et t_j. Ainsi on évite de propager sur la contrainte coûteuse lors de l'instanciation de v_i puisque ces contraintes se retrouvent dans le deuxième modèle. Lorsque la recherche sur le deuxième modèle est lancée, les variables t_j sont calculées connaissant les valeurs de v_i. Le tableau 4.11 démontre que cette méthode utilisant deux sous-modèles incorpore bien les contraintes de conservation de flot. Les résultats se comparent bien aux résultats obtenus avec le modèle sans les contraintes de conservation de flot.

4.2.5 Ajustement de la solution initiale

Nous avons finalement fait des tests pour déterminer les caractéristiques d'une bonne solution initiale. L'objectif est de choisir le nombre et quels sites de BSC seront utilisés dans la solution initiale. Deux faits sont à considérer pour l'ajustement de la solution initiale. Premièrement, il est difficile de faire une corrélation entre les sites utilisés dans la solution initiale et la qualité de solution finale obtenue

Tableau 4.11: Résultats de l'heuristique hybride de recherche locale à descente simple combiné aux modèles de PC

|I|	|J|	|K|	BI(k$)	TS (k$)	# de BSC	Coûts (k$)	Écart (%)	
							vs BI	vs TS
50	20	10	3316.61	3321.10	7	3323.71	0.21	0.08
50	20	20	3557.43	3583.20	8	3561.60	0.92	-0.61
100	20	10	6238.35	6651.10	9	6602.28	6.61	-0.73
100	20	20	6928.86	7150.90	10	7119.67	4.11	-0.44
50	30	20	3583.75	3591.70	8	3594.34	1.98	0.07
100	30	10	6114.51	6217.30	10	6254.49	5.19	0.59
150	30	10	7711.31	8417.60	18	8453.82	15.02	0.43

par la recherche locale. Deuxièmement, nous savons que l'heuristique de recherche locale converge souvent vers la même solution finale indépendamment de la solution initiale utilisée. Il est donc difficile de faire une corrélation, ce qui rend l'ajustement de l'heuristique de recherche de solution initiale difficile. Par contre, comme l'heuristique de recherche locale converge bien, cet ajustement prend moins d'importance. L'ajustement de la solution initiale sera donc important pour l'efficacité de la résolution mais pas pour la qualité de la réponse finale.

L'heuristique initiale classe les sites de BSC en ordre croissant de somme des distances avec les o BTS les plus près. Les p sites de BSC ayant les plus faibles sommes de distance avec les o BTS sont conservés dans la configuration initiale. La configuration initiale est définie comme l'ensemble des p sites de BSC dont la somme des distances avec les o BTS sont les plus petites. Nous avons testé plusieurs valeurs de o afin de déterminer le bon nombre de sites à sommer. Cette valeur représente le nombre moyen de BTS connectés à un site de BSC dans une solution. Imaginons une solution où tous les sites de BSC ont environ le même nombre de BTS qui leur sont connectés. En sommant le bon nombre o de BTS, les sites de BSC ainsi choisis correspondraient aux meilleurs sites à utiliser. Ces essais ont été peu concluants puisque le nombre de BTS connectés varie beaucoup

en réalité pour chaque site de BSC. La valeur o choisie n'a donc d'impact que sur un nombre restreint de sites. Nous avons fixé o à quatre puisque, typiquement, le nombre de BTS par site de BSC est plus grand ou égal à cette valeur. Ainsi, pour les sites choisis, l'heuristique minimisera, en moyenne, le coût d'affectation de quatre BTS.

Le choix des sites de BSC de la configuration initiale n'est donc pas optimal. Par exemple, il y a peut-être un site de BSC qui minimiserait le coût de liaison de huit BTS que nous avons exclus de la configuration initiale. Rappelons que pour réduire le temps de recherche il est important d'avoir la meilleure solution initiale pour qu'elle converge plus rapidement. Il est donc intéressant de choisir de bons sites dans la solution initiale. Comme nous n'avons pas d'indice pour déterminer quels sites auront un impact positif sur la fonction de coût, nous donnons autant de chance à chacun des sites de se faire valoir. Pour mettre en place ce mécanisme, nous permutons q sites consécutifs de la configuration initiale par q sites qui n'en font pas partie et ce, jusqu'à ce qu'il n'y ait plus de permutation possible. Par exemple, pour une configuration initiale de 14 (p) sites sur les 20 (|J|) disponibles et de q égale 2, il y aura alors 21 $(14/2 * (20 - 14)/2)$ configurations évaluées. Ainsi, chacun des sites de la configuration initiale se retrouve dans p/q configurations évaluées et ceux qui n'en faisaient pas partis dans $(|J| - p)/q$ configurations. La configuration finale (solution initiale) est celle de meilleur coût de toutes les configurations explorées.

Nous devons aussi déterminer le nombre p de sites à utiliser dans la configuration initiale. Des essais ont montré que lorsqu'on utilise un plus grand nombre de sites dans la solution initiale, la recherche locale est plus longue et converge moins rapidement. La raison est qu'on est très loin du nombre de sites utilisés dans la solution finale, celle de l'optimum local. Par contre, lorsque le nombre de sites est en dessous d'un certain seuil, il est possible qu'on ne puisse plus converger vers ce minimum local. Choisir le nombre p de sites à utiliser est difficile puisqu'il varie d'un réseau à l'autre. Nous sommes donc conservateurs dans ce choix, il est mieux

Tableau 4.12: Paramètres de l'heuristique initiale établis selon le nombre de BTS dans le réseau

$\|I\|$	o	p	q
50	4	12	2
100	4	18	2
150	4	24	3
200	4	28	3

de prendre plus de temps et d'obtenir une meilleure solution finale. Ce nombre est fixé selon le nombre de BTS présent dans le réseau (tableau 4.12).

4.3 Résultats finaux

Nous comparons dans cette section l'heuristique hybride (HS) de recherche locale et de programmation par contraintes que nous proposons à la borne inférieure (BI) et à l'heuristique de recherche tabou (TS) de Chamberland et Pierre (2002). La borne inférieure a été calculée sur un modèle en nombre entier résolu avec CPLEX Solver 7.1 (Ilog, 2001).

Pour évaluer l'heuristique, 28 tests ont été générés tel que défini au début du chapitre. Les résultats sont présentés au tableau 4.13. Les colonnes 1 à 3 présentent respectivement le nombre de BTS, de sites de BSC et de sites de MSC. La colonne 4 présente le coût de la borne inférieure relaxée (BI). Les colonnes 5 et 6 présentent respectivement le coût et le temps CPU de résolution obtenu par l'heuristique tabou de Chamberland et Pierre (2002). Enfin, les colonnes 7 à 10 présentent les résultats obtenus par notre approche hybride; respectivement le coût, le temps CPU de résolution, l'écart avec la borne inférieure et l'écart avec l'heuristique tabou.

Mentionnons d'abord que les résultats sont assez près de la borne inférieure. De plus, les résultats obtenus avec notre approche confirment les solutions trouvées par l'heuristique tabou de Chamberland et Pierre (2002). En fait, nous améliorons

leurs résultats en moyenne de 2%, seuls les coûts de deux réseaux n'ont pas été améliorés. Pour les réseaux de grande taille, plus de 150 BTS, cette amélioration peut atteindre plus de 6%. D'autre part, nous constatons que la différence entre la borne inférieure et nos solutions s'accentue avec le nombre de BTS. Cela est normal puisque la borne inférieure est relaxée des contraintes de conservation de flot. Il n'y aura donc qu'un seul lien de coût et capacité minimale (DS1) pour représenter chaque affectation BSC-MSC. En réalité, il devrait y avoir la capacité nécessaire sur ces liens pour assurer la conservation de flot, soit un ou plusieurs DS1 et DS3. Le coût relié à la conservation de flot s'accroît donc avec la demande en communication des BTS. Il est donc normal que plus le nombre de BTS est élevé plus le coût négligé par la relaxation est élevé. Ceci explique l'écart grandissant en fonction du nombre de BTS entre le coût de nos solutions et la borne inférieure. Enfin, nous observons que pour un nombre fixe de BTS, les écarts ne changent pas beaucoup avec le nombre de sites de BSC et de MSC disponibles.

Il est difficile de faire une comparaison des temps de calcul entre l'heuristique de Chamberland et Pierre et celle que nous proposons. En effet, les tests ont été effectués sur des ordinateurs de puissance différente: il y a environ un facteur de quatre entre la vitesse du processeur Athlon que nous avons utilisé et le Sun Ultra 5 utilisé par Chamberland et Pierre. Par contre, on remarque que la solution initiale est souvent très bonne (tableau 4.14) et qu'elle se compare assez bien avec la solution finale obtenue avec l'heuristique tabou. Comme cette solution est améliorée dans le temps, il est possible d'avoir une bonne idée du coût final d'un réseau assez rapidement lors de la recherche.

Tableau 4.13: Résultats finaux de l'heuristique hybride de recherche locale à descente simple combiné au modèle de PC

|I|	|J|	|K|	BI	TS		LS			
			OBJ (k$)	OBJ (k$)	CPU (sec)	OBJ (k$)	CPU (sec)	Écart (%)	
								vs BI	vs TS
50	10	10	4051.5	4083.5	233.4	4051.5	360.0	0.00	-0.78
50	10	20	4148.0	4218.0	462.8	4177.7	681.0	0.72	-0.96
100	10	10	6860.9	7192.8	335.3	7136.3	718.0	4.01	-0.79
100	10	20	6566.8	6935.9	536.3	6861.0	517.0	4.48	-1.08
50	20	10	3316.6	3321.1	323.6	3321.1	2033.0	0.14	0.00
50	20	20	3557.4	3583.2	462.8	3559.1	2445.0	0.05	-0.67
100	20	10	6238.4	6651.1	477.2	6576.9	4485.0	5.43	-1.12
100	20	20	6928.9	7150.9	787.0	7112.8	4730.0	2.65	-0.53
150	20	10	8581.4	9847.1	696.0	9330.1	5374.8	8.73	-5.25
150	20	20	8181.6	9298.6	1099.3	8924.9	6210.0	9.09	-4.02
200	20	10	10562.6	12576.7	1030.3	11811.4	13827.0	11.82	-6.09
200	20	20	11049.6	12873.8	1402.2	12295.7	7533.0	11.28	-4.49
50	30	10	3518.3	3525.2	460.0	3518.3	6147.0	0.00	-0.20
50	30	20	3583.8	3591.7	615.8	3591.7	4092.0	0.22	0.00
100	30	10	6114.5	6217.3	889.9	6157.9	14049.0	0.71	-0.96
100	30	20	5600.4	5773.0	1117.5	5737.9	11756.0	2.46	-0.61
150	30	10	7711.3	8417.6	1391.5	8289.4	13903.0	7.50	-1.52
150	30	20	7530.5	8394.0	1753.3	8109.3	17120.0	7.69	-3.39
200	30	10	9233.2	10921.0	1430.4	10384.6	40764.0	12.47	-4.91
200	30	20	10247.6	11631.6	1847.3	11312.0	25234.0	10.39	-2.75
50	40	10	3267.9	3304.2	611.0	3284.5	9968.0	0.51	-0.60
50	40	20	3318.8	3366.1	858.9	3337.6	12076.0	0.57	-0.85
100	40	10	5398.1	5558.8	1342.7	5586.6	16400.0	3.49	0.50
100	40	20	5249.4	5322.8	1821.5	5364.8	18807.0	2.20	0.79
150	40	10	8218.2	8928.1	1342.73	8739.4	60179.0	6.34	-2.11
150	40	20	7345.3	8161.7	1878.9	8064.9	41513.0	9.80	-1.19
200	40	10	9064.9	10872.4	2277.4	10123.4	87318.0	11.68	-6.89
200	40	20	9561.1	11249.1	2474.5	10549.0	55555.0	10.33	-6.22

Tableau 4.14: Comparaison de la solution initiale (SI) pour notre méthode à la solution finale (TS) de l'heuristique tabou

|I|	|J|	|K|	TS	SI		
			OBJ	OBJ	CPU	Écart
			(k$)	(k$)	(sec)	%
50	10	10	4083.5	4199.8	7	2.85
50	10	20	4218.0	4281.3	7	1.50
100	10	10	7192.8	7154.2	12	-0.54
100	10	20	6935,9	6897.4	12	-0.56
50	20	10	3321.1	3501.4	168	5.43
50	20	20	3583.2	3635.8	168	1.47
100	20	10	6651.1	6832.8	108	2.73
100	20	20	7105.9	7150.9	108	0.00
150	20	10	9847,1	9680.2	27	-1.69
150	20	20	9298.6	9305.6	27	0.08
200	20	10	12576.7	12304.0	42	-2.17
200	20	20	12873.8	12625.2	42	-1.93
50	30	10	3525.2	3796.9	378	7.71
50	30	20	3591.7	3852.9	378	7.27
100	30	10	6217.35	6625.1	648	6.56
100	30	20	5773.0	5990.9	648	3.77
150	30	10	8417.6	8426.4	432	0.10
150	30	20	8394.0	8581.4	432	2.23
200	30	10	10921.0	10743.5	1176	-1.63
200	30	20	11631.6	11958.2	1176	2.81
50	40	10	3304.2	3491.0	588	5.65
50	40	20	3366.1	3650.2	588	8.44
100	40	10	5558.8	6103.3	1188	9.80
100	40	20	5322.8	5900.5	1188	10.85
150	40	10	8928.1	9530.2	1080	6.74
150	40	20	8161.7	8326.4	1080	2.02
200	40	10	10871.4	10675.9	1512	-1.81
200	40	20	11249.1	11615.7	1512	3.26

CHAPITRE 5

CONCLUSION

Dans ce mémoire nous avons étudié le problème de conception topologique de réseau cellulaire qui consiste à déterminer la localisation des commutateurs (BSC et MSC) d'un réseau et leur type, ainsi qu'à définir la topologie du réseau et les types des liens. Ce problème combinatoire est classé NP-difficile. Nous avons proposé une approche hybride de recherche locale et de programmation par contraintes pour le résoudre.

Nous avons commencé par développer une méthode exacte pour la résolution du problème à l'aide d'un modèle de PC. Les stratégies d'affectations du modèle ont été conçues pour que la solution trouvée, lors d'un arrêt de la recherche avant sa fin, reflète bien le coût d'utiliser tous les sites de BSC dans la solution. Le modèle de PC peut ainsi servir à évaluer une fonction de coût qui varie en fonction des sites de BSC utilisés par le modèle. Nous avons baptisé ce concept du nom de sonde. La sonde sert donc à estimer, en quelques secondes, le coût d'une solution comportant un ensemble de sites de BSC donné. Nous avons ensuite ajouté une recherche locale à descente simple qui utilise le concept de sonde pour déterminer l'ensemble des sites de BSC qui conduit à la meilleure solution. La recherche s'effectue donc sur les sites de BSC utilisés et non sur l'ensemble des variables. L'espace de recherche a ainsi été diminué par un facteur très important. Par contre, le temps pour évaluer une solution du voisinage a été augmenté et nous devons considérer que le coût associé à cette solution n'est qu'un estimé.

Les résultats ont démontré que notre méthode donne d'excellentes solutions comparativement aux bornes inférieures issues de la relaxation des contraintes de conservation de flot et aussi aux résultats obtenus par l'approche tabou de Chamberland et Pierre (2002). En moyenne, l'heuristique hybride que nous avons pro-

posée améliore les meilleures solutions obtenues jusqu'à maintenant par un facteur de plus de 2%. Ces résultats montrent aussi que la sonde estime bien le coût des voisins lors la recherche locale.

Deux instances n'ont pu être améliorées, toutes deux comptant 100 BTS et 40 sites de BSC. Le nombre de sites de BSC est très élevé comparativement au nombre de BTS pour ces instances. L'espace de recherche locale est donc très grand puisqu'il faudra peu de sites de BSC pour satisfaire l'ensemble des BTS. L'ordre de grandeur de l'espace de recherche correspond à choisir environ 15 sites parmi 40, soit 10^{22} combinaisons. Nous avons analysé l'évolution de la solution lors de la recherche locale pour ces deux instances. Dès le début de la recherche, l'heuristique a fait un mauvais choix de direction de descente en limitant rapidement le nombre de sites de BSC dans la solution. Limiter le nombre de sites confine la recherche dans une zone de minimum local où il n'est plus possible de sortir puisqu'on ne peut pas augmenter le nombre de sites lors de la recherche. Pour la première instance fautive, les résultats font état d'une solution initiale utilisant 18 sites de BSC contre 14 sites lors de la première itération de recherche locale. Il est effectivement possible que le nombre de sites utilisés diminue de plus de un site à une itération. On se rappelle que nous devons définir un sous-ensemble de sites à fournir au modèle. La stratégie de recherche du modèle affecte, à chaque BTS, le site de BSC le plus près. Cette méthode ne garantit pas l'utilisation de tous les sites de BSC. En effet, dans le cas où il n'y aurait pas de BTS qui se connecte à un des sites de BSC celui-ci ne fera pas partie de la solution. Il est fort probable, surtout lorsque le nombre de sites fournis au modèle est grand, que les sites ne soient pas tous utilisés dans la solution retournée. Dans le cas de la deuxième instance, le nombre de sites de BSC de la solution initiale passe de 18 sites à 16 sites dès la première itération.

Pour éviter ce problème, l'heuristique locale pourrait visiter des solutions avec plus de sites de BSC lors de l'exploration du voisinage. Le nombre de sites de BSC pourrait ainsi diminuer ou augmenter d'une itération à l'autre. Par contre, la

recherche serait beaucoup plus longue et n'améliorerait probablement pas la solution finale. En effet, lorsqu'un site serait ajouté à la solution courante, il est fort probable que ce site ait un effet négatif sur le coût de la solution visitée. Dans ce contexte, l'heuristique ne choisirait jamais une solution avec plus de sites de BSC du voisinage. Pour améliorer les chances de choisir la solution avec plus de sites, nous pourrions utiliser une permutation et un ajout de site lors de l'exploration du voisinage. Ceci augmenterait considérablement le temps d'une itération et donc serait néfaste pour le temps de recherche et la convergence de l'heuristique locale. Il serait tout de même intéressant de valider ces affirmations avec quelques expérimentations. Un moyen plus pratique pour résoudre ce problème est de permettre de n'enlever qu'un seul site de BSC par itération. L'heuristique visiterait ainsi plus de voisins avant de prendre les décisions de descente qui diminuent le nombre de sites. Pour explorer plus de solutions en début de recherche, nous pourrions ajouter un effet aléatoire lors du choix d'un voisin. Ce processus est inspiré par les heuristiques de recuit simulé. L'heuristique ne choisirait pas nécessairement le voisin de meilleur coût pour l'itération subséquente mais un autre qui a de meilleures caractéristiques telles qu'un nombre de sites de BSC plus élevé. Plus l'heuristique s'approcherait de la solution finale, plus l'effet aléatoire serait diminué.

Outre les améliorations proposées précédemment, plusieurs autres aspects pourraient être perfectionnés. D'abord, un ajustement plus adéquat des temps de recherche pour les modèles et leurs stratégies selon la taille des instances améliorerait substantiellement l'efficacité de l'heuristique locale. En effet, comme ces temps sont ajustés pour le réseau de plus grande taille, il serait possible de réduire le temps de recherche pour toutes les tailles inférieures de réseaux. Ce gain à chaque itération pourrait nous aider à introduire efficacement une recherche à voisinage variable, par exemple. Il suffirait de définir un deuxième mouvement local qui complète bien le premier pour permettre de sortir d'un minimum local et ainsi visiter plusieurs minimums locaux. Il serait aussi intéressant de modifier

la structure en arbre utilisée dans ce projet par une structure plus réaliste afin de garantir une certaine fiabilité aux topologies des réseaux solutionnés.

Finalement, mentionnons que notre modèle et le paradigme de programmation par contraintes sont suffisamment flexibles pour permettre d'adapter le problème à des réseaux cellulaires très différents. En outre, les réseaux cellulaires de troisième génération seraient des candidats idéaux.

BIBLIOGRAPHIE

[1] Ahuja, R., Magnanti, T., et Orlin, J.,"Network Flows", Prentice Hall, 1992.

[2] André, M., Pesant, G. et Pierre, S., "A Variable Neighborhood Search Algorithm For Assigning Cells to Switches in Wireless Networks", soumis pour publication, 2002.

[3] Bhattacharya, P.S., Saha, D. et Mukherjee, A., "Heuristics for assignment of cells to switches in a PCSN: a comparative study", IEEE International Conference on Personal Wireless Communications, Jaipur, Inde, pp. 331-334, 1999.

[4] Chamberland, S. et Pierre, S., "On the Design Problem of Cellular Wireless Networks", publication Centre de recherche sur les transports CRT-2002-38, 2002.

[5] Cox, L.A. Jr. et Sanchez, J.R., "Designing least-cost survivable wireless backhaul networks", Journal of Heuristics, vol. 6, no. 4, pp. 525-540, 2000.

[6] Der-Rong, D. et Tseng, S.S.,"Heuristic and simulated annealing algorithms for solving extended cell assignment problem in wireless ATM networks", International Journal of Communication Systems, vol. 15, pp. 47-65, 2002.

[7] Gendreau, M., Laporte, G. et Potvin, J.Y., "Metaheuristics for the Vehicle Routing Problem", publication Centre de recherche sur les transports CRT-963, 1994.

[8] Glover, F. et Laguna, M., "Tabu Search", Kluwer Academic Publisher, 1997.

[9] Hansen, P., Mladenovic, N. et Perez-Brito, D., "Variable Neighborhood Decomposition Search", Journal of Heuristics, vol. 7, no. 4, pp. 335-350, 2001.

[10] Holland, J.H.,"Adaptation in Natural and Artificial Systems", The University of Michigan Press, Ann Arbor, Michigan, 1975.

[11] Hung, W.N.N. et Song, X., "On Optimal Cell Assignments in PSC Networks", IEEE Conference on Performance, Computing, and Communications, Phoenix, USA, pp. 225-232, 2002.

[12] ILOG, "ILOG OLP STUDIO 3.5 - User's Manual", 2001.

[13] ILOG, "ILOG OLP STUDIO 3.5 - The Optimisation Language", 1999.

[14] ILOG, "ILOG CPLEX 7.1 - Advanced Reference Manual", 2001.

[15] Jonker, R. et Volgenant T., "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems", Computing, vol. 38, pp. 325-340, 1987.

[16] Kirpatrick, S., Gelatt, C.D. Jr. et Vecchi, M.P.,"Optimization by Simulated Annealing", Science, vol. 220, 1983, pp. 671-680.

[17] Klincewicz, J.G., "Heuristic for the p-hub location problems", European Journal of Operational Research, vol. 53, pp. 25-37, 1991.

[18] Marriott, K. et Stuckey, P.J., "Programming with Constraints : An Introduction", The MIT Press, 1998.

[19] Merchant, A., et Sengupta, B., "Multiway graph partitioning with applications to PCS networks", IEEE INFOCOM'94, vol. 2, pp. 593-600, 1994.

[20] Merchant, A., et Sengupta, B., "Assignment of cells to switches in PCS networks", IEEE/ACM Transactions on Networking, vol. 3, no. 5, pp. 521-526, 1995.

[21] O'Kelly, M.E., "The location of interacting hub facilities", Transportation Science, vol. 20, no. 2, 92-106, 1986.

[22] Pierre, S. et Houéto, F., "Assigning Cells to Switches in Cellular Mobile Networks Using Taboo Search", IEEE/ACM Transactions on System, Man and Cybernetics, vol. 32, no. 3, 2002.

[23] Quintero, A., Pierre, S., "A Memetic Algorithm for Assigning Cells to Switches in Cellular Mobile Networks", IEEE Communications Letter, vol. 6, no. 11, 2002.

[24] Saha, D., Mukherjee, A. et Bhattacharya, P., "A Simple Heuristic for Assigment of Cell to Switches in a PCS Network", Wireless Personal Communication, vol. 12, pp. 209-224, 2000.

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

[26] Reed, D., "The Cost Structure of Personal Communication Services", IEEE Communications Magazine, vol. 31, pp. 102-108, 1993.

[27] Reeves, C.R., "Modern Heuristic Techniques for Combinatorial Problems", Halsted Press, 1993.

[28] Skorin-Kapov, D. et Skorin-Kapov, J., "On Tabu-Search for the location of interacting hub facilities", European Journal of Operational Research, vol. 73, pp. 502-509, 1994.

[29] Soriano, P. et Gendreau, M., "Fondements et applications des méthodes de recherche avec tabous", R.A.I.R.O., vol. 31, no.2, pp. 133-159, 1997.

Annexe I

Modèle A

```
setting searchStrategy = DFS;

Open int+ Nombre_BSC_NULL;

int NombreBSC = ...;

%DONNEES

%i
enum BTS = ...;
%alpha
int Cap_Circuit_BTS [BTS] = ...;
%n
int Nombre_Link_Requis [BTS] = ...;

%J
enum BSC_Site = ...;
%K
enum MSC_Site = ...;

Open int+ ForceNull[int+];
BSC_Site truc[0..NombreBSC-1] = ...;

%DONNEES FIXENT
%l
enum Link_Types = DS1, DS3;
%beta
int Cap_Circuit_Link [Link_Types] = [96, 2688];
```

```
%BSC
%S
enum BSC_Types = {BSC_NULL, BSC_A, BSC_B, BSC_C};
%m bts
int BSC_Max_Interface_BTS [BSC_Types] = [0,15,30,60];
%m msc
int BSC_Max_Interface_MCS [BSC_Types] = [0,15,30,60];
%n bsc
int Cap_Circuit_Stwich_BSC [BSC_Types] = [0,5000,10000,15000];

%MSC
%T
enum MSC_Types = {MSC_NULL, MSC_A, MSC_B, MSC_C};
%m bsc
int MSC_Max_Interface_BCS [MSC_Types] = [0, 50, 100, 150];
%n msc
int Cap_Circuit_Stwich_MSC [MSC_Types] = [0, 100000, 200000, 300000];

%COUTS
%Liens BTS-BSC
int+ Cout_Unitaire_Link_BTS_BSC [BTS] = ...;
int+ Dist_Link_BTS_BSC [BTS, BSC_Site] = ...;

%Interfaces
int+ Cout_Unitaire_Interface [Link_Types] = [ 500, 2500 ]; %DS1 et DS3

%Liens BSC-MSC
int+ Cout_Unitaire_Link_Type [Link_Types] = [ 2000, 4000];
int+ Dist_Link_BSC_MSC [BSC_Site, MSC_Site] = ...;

%Couts intallation BSC et MSC respectivement
```

```
int+ Cout_Unitaire_BSC_Type [BSC_Types] = [0, 50000, 90000, 120000];
int+ Cout_Unitaire_MSC_Type [MSC_Types] = [0, 200000, 350000, 500000];

%VARIABLES

%A quel BSC est associe le BTS
var BSC_Site BTS_Connection [BTS];

%Quel est le type de BSC a ce site (6) y
var BSC_Types BSC_Site_Types [BSC_Site];

var int+ Cout_Link_BTS_BSC [BTS] in 0..1000000;

%CONVENTION : i pour BTS, j pour BSC, k pour MSC et l type de lien

minimize
%Cout des liens entre BTS et BSC_Site
    sum(i in BTS) (Cout_Link_BTS_BSC [i])

%Cout des liens entre MSC_Site et BSC_Site
    +sum (j in BSC_Site) Cout_Link_BSC_MSC [j]

%Cout d'installation d'un tel type au BSC_SITE
    +sum (j in BSC_Site) Cout_Unitaire_BSC_Type
        [BSC_Site_Types [j]]/100

%Cout d'installation d'un tel type au BSC_SITE
    +sum (k in MSC_Site) Cout_Unitaire_MSC_Type
        [MSC_Site_Types [k]]/100

subject to {
forall (i in BTS)
    Cout_Link_BTS_BSC [i]=(Cout_Unitaire_Link_BTS_BSC [i] *
    Dist_Link_BTS_BSC [i,BTS_Connection [i]]/1000 +
```

```
    Nombre_Link_Requis[i] * Cout_Unitaire_Interface[DS1])/100;

forall (j in BSC_Site)
    Cout_Link_BSC_MSC [j] = (sum(l in Link_Types)
    (Nombre_Link_BSC_MSC_Type [j,l]* Cout_Unitaire_Link_Type [l]
    * Dist_Link_BSC_MSC [j,BSC_Connection [j]]/1000 +
    (Nombre_Link_BSC_MSC_Type [j,l] * 2 *
    Cout_Unitaire_Interface [l])))/100;

%Somme des liens requis de BTS connectes a un BSC
%<= Nombre interfaces BTS a ce BSC
forall(j in BSC_Site)
    sum (i in BTS) ((BTS_Connection [i] = j) * Nombre_Link_Requis [i])<=
    BSC_Max_Interface_BTS [BSC_Site_Types [j]];

%Somme des capacites des BTS connectes a un BSC
%<= Nombre switch capacity BTS a ce BSC
forall(j in BSC_Site)
    sum (i in BTS) ((BTS_Connection [i] = j)* Cap_Circuit_BTS [i])<=
    Cap_Circuit_Stwich_BSC [BSC_Site_Types [j]];

    sum (j in BSC_Site) (BSC_Site_Types [j] = BSC_NULL)
    >= Nombre_BSC_NULL;

forall(no in 0..Nombre_BSC_NULL-1)
    BSC_Site_Types [truc[ForceNull[no]]] = BSC_NULL;

%CONTRAINTES REDONDANTES

%La somme des capacites des BTS doit etre plus petite
%que la somme des capacites des BSC
    sum (i in BTS) Cap_Circuit_BTS[i] <= sum (j in BSC_Site)
```

Cap_Circuit_Stwich_BSC [BSC_Site_Types [j]];

sum (i **in** BTS) Nombre_Link_Requis [i] <= **sum** (j **in** BSC_Site)
BSC_Max_Interface_BTS [BSC_Site_Types [j]];

};

Annexe II

Modèle B

```
setting searchStrategy = DFS;
%DONNEES
%i
import enum BTS;
%alpha
int Cap_Circuit_BTS [BTS] = ...;
%n
int Nombre_Link_Requis [BTS] = ...;

%J
import enum BSC_Site;
%K
import enum MSC_Site;

%DONNEES FIXENT
%l
import enum Link_Types;
%beta
int Cap_Circuit_Link [Link_Types] = [96, 2688];

%BSC
%S
import enum BSC_Types;
%m bts
int BSC_Max_Interface_BTS [BSC_Types] = [0,15,30,60];
%m msc
```

```
int BSC_Max_Interface_MCS [BSC_Types] = [0,15,30,60];
%n bsc
int Cap_Circuit_Stwich_BSC [BSC_Types] = [0,5000,10000,15000];

%MSC
%T
import enum MSC_Types;
%m bsc
int MSC_Max_Interface_BCS [MSC_Types] = [0, 50, 100, 150];
%n msc
int Cap_Circuit_Stwich_MSC [MSC_Types] = [0, 100000, 200000, 300000];

Open int+ NbMaxMSC;

Open float+ tempsRecherche;

%COUTS
%Liens BTS-BSC
int+ Cout_Unitaire_Link_BTS_BSC [BTS] = ...;
int+ Dist_Link_BTS_BSC [BTS, BSC_Site] = ...;

%Interfaces
int+ Cout_Unitaire_Interface [Link_Types] = [ 500, 2500 ];
%DS1 et DS3 %Liens BSC-MSC

int+ Cout_Unitaire_Link_Type [Link_Types] = [ 2000, 4000];
int+ Dist_Link_BSC_MSC [BSC_Site, MSC_Site] = ...;

%Couts intallation BSC et MSC respectivement
int+ Cout_Unitaire_BSC_Type [BSC_Types] = [0, 50000, 90000, 120000];
int+ Cout_Unitaire_MSC_Type [MSC_Types] = [0, 200000, 350000, 500000];

%TRAFIC DAT
```

```
%Goi
int+ Communication_BTS_BTS [BTS, BTS] = ...;

%Gip
int+ Communication_BTS_NET [BTS] = ...;

%Gpi
int+ Communication_NET_BTS [BTS] = ...;

%A quel BSC est associe le BTS
Open BSC_Site BTS_Connection[BTS];

%A quel MCS est associe le BSC
var MSC_Site BSC_Connection[BSC_Site];

%Quel est le type de BSC a ce site
Open BSC_Types BSC_Site_Types [BSC_Site];

%Quel est le type de MSC a ce site
var MSC_Types MSC_Site_Types [MSC_Site];

%VARIABLES

var int+ Nombre_Link_BSC_MSC_Type [BSC_Site, Link_Types] in 0..60;

%ATTENTION LIMITE SUPERIEURE
var int+ Cout_Link_BTS_BSC [BTS] in 0..1000000;
var int+ Cout_Link_BSC_MSC [BSC_Site] in 0..24000000;

%TRAFIC
%Fij o
var int+ Trafic_BTS_BSC_FROM_BTS [BTS] in 0..20000;
%Fji o
```

```
var int+ Trafic_BSC_BTS_FROM_BTS [BTS, BTS] in 0..20000;
%Fji p
var int+ Trafic_BSC_BTS_FROM_NET [BTS] in 0..20000;

var int+ Trafic_BSC_MSC [BSC_Site] in 0..10000000;

%CONVENTION : i pour BTS, j pour BSC, k pour MSC et l type de lien

minimize
%Cout des liens entre BTS et BSC_Site
    sum(i in BTS) (Cout_Link_BTS_BSC [i])

%Cout des liens entre MSC_Site et BSC_Site
    +sum (j in BSC_Site) Cout_Link_BSC_MSC [j]

%Cout d'installation d'un tel type au BSC_SITE
    +sum (j in BSC_Site) Cout_Unitaire_BSC_Type
        [BSC_Site_Types [j]]/100

%Cout d'installation d'un tel type au BSC_SITE
    +sum (k in MSC_Site) Cout_Unitaire_MSC_Type
        [MSC_Site_Types [k]]/100

subject to {
forall (i in BTS)
    Cout_Link_BTS_BSC [i]=(Cout_Unitaire_Link_BTS_BSC [i] *
    Dist_Link_BTS_BSC [i,BTS_Connection [i]]/1000 +
    Nombre_Link_Requis[i] * Cout_Unitaire_Interface[DS1])/100;

forall (j in BSC_Site)
    Cout_Link_BSC_MSC [j] = (sum(l in Link_Types)
    (Nombre_Link_BSC_MSC_Type [j,l]* Cout_Unitaire_Link_Type [l]
    * Dist_Link_BSC_MSC [j,BSC_Connection [j]]/1000 +
```

```
    (Nombre_Link_BSC_MSC_Type [j,l] * 2 *
    Cout_Unitaire_Interface [l]))))/100;

%Somme du nombre liens de chacun des MSC connectes a un BSC
%<= Nombre interfaces MSC a ce BSC
forall(j in BSC_Site)
    sum (l in Link_Types) Nombre_Link_BSC_MSC_Type [j,l]<=
    BSC_Max_Interface_MCS [BSC_Site_Types [j]];

%Somme du nombre de liens de chaque type (Lien) des BSC connectes a un MSC
%<= Nombre interfaces BSC a ce MSC (Inverse de 9)
forall(k in MSC_Site)
    sum (j in BSC_Site) ((BSC_Connection [j] = k) * (sum (l in Link_Types)
    Nombre_Link_BSC_MSC_Type [j,l]))<=
    MSC_Max_Interface_BCS [MSC_Site_Types [k]];

%Somme du nombre de lien de chaque type associe a un MSC multiplie par la
%capacites en circuits de ce type de lien<=Capacite en circuits de ce MSC
forall(k in MSC_Site)
    sum (j in BSC_Site) ((BSC_Connection [j] = k)*(sum (l in Link_Types)
    Nombre_Link_BSC_MSC_Type [j,l]*Cap_Circuit_Link[l])) <=
    Cap_Circuit_Stwich_MSC [MSC_Site_Types [k]];

    MSC_Site_Types[MSC_Site_NULL]=MSC_NULL;

    sum (k in MSC_Site) (MSC_Site_Types[k] <> MSC_NULL)
    <= NbMaxMSC;

forall(j in BSC_Site)
    Trafic_BSC_MSC [j] = sum (i in BTS)sum (o in BTS)
    ((BTS_Connection[i]=j)*(BTS_Connection[o]<>j)*
```

```
    (Communication_BTS_BTS [i,o]+Communication_BTS_BTS[o,i]))
    + sum(i in BTS)((BTS_Connection[i]=j) *
    (Communication_BTS_NET [i] + Communication_NET_BTS [i]));

forall(i in BTS)
    Trafic_BTS_BSC_FROM_BTS [i] +
    sum ( o in BTS ) Trafic_BSC_BTS_FROM_BTS [i,o] +
    Trafic_BSC_BTS_FROM_NET [i] <= Cap_Circuit_BTS [i]*1000;

forall(j in BSC_Site)
    Trafic_BSC_MSC [j] <= sum (l in Link_Types)
    (Nombre_Link_BSC_MSC_Type [j,l]*Cap_Circuit_Link[l]*1000);

forall (i in BTS)
    Trafic_BTS_BSC_FROM_BTS [i] =sum (iprim in BTS)
    (Communication_BTS_BTS [i, iprim]) + Communication_BTS_NET [i];

forall (i in BTS, o in BTS)
    Trafic_BSC_BTS_FROM_BTS [i,o] = Communication_BTS_BTS [o,i];

forall(i in BTS)
    Trafic_BSC_BTS_FROM_NET [i] = Communication_NET_BTS [i];

};
```