UNIVERSITÉ DE MONTRÉAL

DÉVELOPPEMENT DE MÉTHODES PARALLÈLES POUR DES PROBLÈMES DE GRANDE TAILLE

JOSÉ MANUEL PIRES
DÉPARTEMENT DE MATHÉMATIQUES
ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(MATHÉMATIQUES APPLIQUÉES)
AOÛT 1997

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

DÉVELOPPEMENT DE MÉTHODES PARALLÈLES POUR DES PROBLÈMES DE GRANDE TAILLE

présenté par : PIRES José Manuel

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

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

Mme SANSÓ Brunilde, Ph.D., présidente

M. SOUMIS François, Ph.D., membre et directeur de recherche

M. DESAULNIERS Guy, Ph.D., membre

REMERCIEMENTS

Je remercie François Soumis pour m'avoir supporté financièrement durant ma maîtrise et donné accès aux ressources informatiques du GERAD.

Un gros merci à Daniel Villeneuve pour avoir tenu le rôle (non officiel) de codirecteur.

Je tiens à souligner la contribution d'Éric Gélinas au développement de DG, de même que Daniel Villeneuve à l'ensemble des travaux, par leur participation active à des discussions.

Je me dois aussi de mentionner la collaboration d'Éric Gélinas, de June Lavigne et l'équipe PAIRING de la compagnie ADOPT pour m'avoir fourni des problèmes tests.

RÉSUMÉ

Ce mémoire a pour sujet le développement de méthodes parallèles pour résoudre par génération de colonnes des problèmes de grande taille dans le cadre d'un modèle unifié fondé sur une extension du principe de décomposition de Dantzig-Wolfe. Il comporte trois contributions à l'avancement des connaissances dans le domaine de la génération de colonnes.

En premier lieu, une *analyse* du potentiel parallèle d'un logiciel générique de génération de colonnes, la bibliothèque GENCOL. Cette analyse a pour but de cerner les composantes du logiciel pouvant le plus bénéficier du parallélisme. Deux de ces composantes sont retenues et font chacune l'objet d'une implantation informatique.

En deuxième lieu, l'implantation informatique, nommée DG, de la première composante : *résolution de la relaxation linéaire provenant de la décomposition de Dantzig-Wolfe* en résolvant en parallèle le problème-maître et les sous-problèmes de façon asynchrone. La difficulté d'intégration de la méthode dans des applications commerciales et les résultats numériques peu satisfaisants pour plusieurs classes de problèmes font en sorte que cette méthode ne sera pas intégrée à la bibliothèque GENCOL.

En troisième lieu, l'implantation informatique, nommée STA, de la deuxième composante : *résolution du problème de plus court chemin avec contraintes de ressources* où les nœuds des réseaux sont traités en parallèle. Les résultats numériques obtenus sont encourageants et justifient l'utilisation de la méthode. La robustesse de la méthode et le caractère modulaire de son implantation informatique font en sorte que STA fait maintenant partie intégrante de la bibliothèque GENCOL.

ABSTRACT

The subject of this master's thesis is the development of parallel methods to solve large scale problems by column generation in the context of a unified model based on an extension of the Dantzig-Wolfe principle. It provides three contributions to the advancement of knowledge in the field of column generation.

First, an *analysis* of the parallel potential of a generic column generation software, the GENCOL library. The goal of this analysis is to identify the components of the library that could benefit the most from parallelism. Two of those components are chosen and lead to computer implementations.

Next, a computer implementation, named DG, of the first component: *solving the linear relaxation of the Dantzig-Wolfe decomposition* by optimizing the master problem and the subproblems asynchronously. The difficulty to integrate the method in commercial applications and the unsatisfactory numerical results are such that this method is not incorporated in the GENCOL library.

Finally, a computer implementation, named STA, of the second component: *solving of a resource constrained shortest path problem* where the networks' nodes are processed in parallel. The numerical results obtained during tests are encouraging and justify the use of the method. The robustness of the method and the modular aspect of its computer implementation give STA a place in the GENCOL library.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ALGORITHMES

INTRODUCTION

Les problèmes de tournées de véhicule et d'horaires d'équipage sont des problèmes d'optimisation importants et ils font l'objet de recherches intensives depuis les quinze dernières années. L'évolution rapide des outils informatiques permet depuis quelques années de traiter des problèmes de taille utile en pratique. Toutefois, le fait que les problèmes proviennent majoritairement d'applications commerciales complexes a contribué à la multiplication de modèles spécialisés taillés sur mesure. Afin de contrer cette expansion du nombre de modèles, certains groupes de chercheurs ont entrepris de développer des modèles plus généraux, dont un modèle unifié par Desaulniers *et al.* (1994).

Le but du projet de recherche est de faire l'étude du potentiel parallèle des méthodes de résolution s'appliquant au modèle unifié et d'en tirer des implantations informatiques afin de fournir aux développeurs et aux chercheurs de nouveaux moyens de résolution basés sur des techniques parallèles. Ces dernières tirent profit de la puissance des architectures parallèles et des réseaux locaux d'ordinateurs afin de faciliter le traitement de problèmes de plus grande taille et de modèles plus riches. La bibliothèque de procédures GENCOL, qui est une implantation informatique du modèle unifié, fera office de version séquentielle de référence. Le deuxième chapitre montre que GENCOL est une implantation générique d'un optimiseur pour le modèle unifié, ce qui permet d'étendre les conclusions de l'étude de GENCOL au modèle unifié. Le mémoire se divise en trois parties : l'analyse des différents modules de la bibliothèque GENCOL en vue d'évaluer leur potentiel parallèle et deux implantations informatiques, soit la résolution de la relaxation linéaire en parallèle et la résolution d'un sous-problème en parallèle. Le lecteur notera que la revue de la littérature est effectuée de façon ponctuelle, les sujets traités étant hétérogènes.

Le premier chapitre délimite la portée des travaux au niveau théorique, soit le modèle unifié décrit par Desaulniers *et al.*, en mettant l'accent sur le caractère général du modèle. Il met ensuite en évidence la pertinence de développer des méthodes parallèles pour les problèmes de grande taille. Le choix des classes de problèmes retenues pour valider les implantations informatiques y est justifié. De plus, quelques concepts théoriques du domaine du parallélisme y sont présentés. Ce chapitre introduit aussi la notion de potentiel parallèle, qui sera utilisée ultérieurement au chapitre 3 pour analyser un logiciel d'optimisation. Finalement, la dernière section définit les critères d'évaluation qui serviront à mesurer quantitativement les performances des implantations parallèles aux chapitres 4 et 5.

Le rôle du deuxième chapitre est de choisir les outils informatiques de manière à délimiter les travaux au niveau pratique. Tout d'abord, un logiciel d'optimisation générique pouvant résoudre le modèle unifié est décrit. La bibliothèque de procédures GENCOL possède cette architecture générique et fera donc office de version séquentielle de référence. Ensuite, la problématique de l'évaluation de la puissance de calcul d'un parc de machines est présentée, ce qui amène au choix du banc d'essai. Finalement, deux outils informatiques supplémentaires qui auront leur utilité dans le cadre des implantations informatiques sont présentés.

Le troisième chapitre est consacré à l'étude du potentiel parallèle d'un optimiseur séquentiel traitant le modèle unifié afin d'en identifier les modules pouvant le plus bénéficier du parallélisme. Les modules analysés correspondent aux composantes du logiciel qui devraient normalement se retrouver dans n'importe quelle implantation informatique. Cette analyse se veut donc générale, malgré qu'elle soit

basée sur l'implantation spécifique qu'est GENCOL. Les modules choisis feront chacun l'objet d'implantations parallèles qui seront décrites dans les deux chapitres suivants.

Le quatrième chapitre décrit une implantation informatique permettant de résoudre en parallèle, par génération de colonnes, la relaxation linéaire d'un problème provenant de la décomposition de Dantzig-Wolfe. La méthode consiste à résoudre le problème-maître et les sous-problèmes en parallèle de façon asynchrone. Une revue de la littérature sur la génération de colonnes en parallèle débute le chapitre. La méthode ainsi que ses spécifications sont ensuite décrites sans entrer dans les détails informatiques. Des résultats numériques ne sont présentés que sur un petit ensemble de problèmes. Par contre, plusieurs études sur diverses classes de problèmes sont analysées afin de juger si le comportement observé sur les problèmes disponibles se généralise. Certaines stratégies d'accélération sont proposées et évaluées. Le chapitre conclut sur l'efficacité de la méthode en pratique en révisant son potentiel parallèle à la baisse.

Le cinquième et dernier chapitre décrit une implantation informatique d'un algorithme de plus court chemin avec contraintes de ressources où les nœuds des réseaux sont traités en parallèle. La formulation du problème de plus court chemin avec contraintes de ressources débute le chapitre. Par la suite, une revue de la littérature fait un survol rapide des travaux antérieurs sur le parallélisme dans le cadre du problème de plus court chemin classique, le traitement parallèle du problème avec contraintes de ressources étant inexistant. La section suivante décrit la méthode proposée de façon plus formelle ainsi que les spécifications à respecter. Des résultats numériques sur deux classes de problèmes sont présentés et analysés.

Le chapitre conclut sur l'efficacité de la méthode en pratique en confirmant son potentiel parallèle.

La conclusion souligne les contributions du mémoire à l'avancement des connaissances dans le domaine de la génération de colonnes tout en mentionnant de nouvelles voies de recherche.

CHAPITRE 1
Portée des travaux

Le but du premier chapitre est de délimiter clairement la portée des travaux au niveau théorique. Le deuxième chapitre traitera du niveau pratique en choisissant le logiciel d'optimisation séquentiel de référence, le banc d'essai et les outils parallèles. Plus précisément, la première section de ce chapitre définit le cadre théorique du projet, soit le modèle unifié décrit par Desaulniers *et al.* (1994) en mettant l'accent sur le caractère général du modèle. La deuxième section met en évidence la pertinence de développer des méthodes parallèles pour les problèmes de grande taille et justifie le choix des classes de problèmes retenues pour valider les implantations informatiques. La troisième section présente quelques concepts théoriques du domaine du parallélisme afin de munir le lecteur des outils nécessaires à la compréhension des travaux. La quatrième section définit les critères d'évaluation qui serviront à mesurer quantitativement les performances des implantations parallèles aux chapitres 4 et 5. Finalement, la cinquième section définit le potentiel parallèle, concept qui sera utilisé ultérieurement au chapitre 3 pour analyser un logiciel d'optimisation résolvant le modèle unifié.

1.1 Cadre théorique

La plupart des problèmes de tournées de véhicule et d'horaires d'équipage présentement étudiés dans la littérature peuvent être formulés au moyen d'un modèle unifié (Desaulniers *et al.*, 1994; Desrosiers *et al.*, 1995). Le modèle sur lequel sont

basées les implantations informatiques dont traite ce mémoire est une formulation non linéaire d'un problème de flot multi-commodités en variables entières, problème auquel s'ajoutent des contraintes de ressources. Ce modèle constitue une extension d'une formulation du problème de voyageurs de commerce multiples avec contraintes de ressources, dénoté m-TSPR (*multiple-Traveling Salesman Problem with Resource constraints*). D'une certaine façon, ce problème peut être visualisé comme celui de visiter un ensemble de clients au moyen d'une flotte hétérogène de véhicules. Les chemins parcourus par ces véhicules sont toutefois soumis à des restrictions sur des ressources comme la capacité des véhicules, la durée de chaque itinéraire, l'heure de service de chaque client, etc.

Le problème général est défini sur un réseau et consiste à visiter un ensemble de clients associés aux nœuds ou aux arcs de ce réseau, au moyen de véhicules ou d'équipages parcourant des chemins à coût minimum. D'une part, l'utilisation de ces véhicules ou équipages, plus généralement appelés *commodités*, est limitée *localement* par la structure du réseau et par diverses contraintes de ressources et peut également être limitée par des relations de préséance et de couplage entre les clients à visiter. D'autre part, cette utilisation des commodités est limitée *globalement* par des contraintes de coordination entre celles-ci.

De manière plus formelle, K représente l'ensemble des commodités. À chaque commodité $k \in K$ est associé un graphe orienté $G^k(\mathcal{N}^k, \mathcal{A}^k)$ où $\mathcal{A}^k$ est l'ensemble des arcs et $\mathcal{N}^k$ l'ensemble des nœuds, incluant un nœud origine $o(k)$ et un nœud destination $d(k)$. Un ensemble de ressources R^k est aussi associé à chaque commodité. Chacun des nœuds $i \in \mathcal{N}^k$ comporte une fenêtre de réalisabilité $[a_i^{kr}, b_i^{kr}]$ pour chaque ressource $r \in R^k$. De plus, une consommation de ressources t_{ij}^{kr} est associée à chaque arc $(i,j) \in \mathcal{A}^k$.

Les variables binaires X_{ij}^k représentent le flot de la commodité $k \in K$ qui traverse l'arc $(i,j) \in \mathcal{A}^k$. Les variables continues T_i^{kr} représentent la valeur de la ressource $r \in R^k$ au nœud $i \in \mathcal{N}^k$. Finalement, un ensemble S de variables additionnelles Y_s ayant pour bornes $[l_s, u_s]$ complètent la structure du problème.

Un coût c_{ij}^k est associé à chaque arc $(i,j) \in \mathcal{A}^k$ ainsi qu'un coût c_s à chaque variable Y_s. À l'aide de cette notation, on définit les vecteurs suivants : $\boldsymbol{X}^k = (X_{ij}^k | (i,j) \in \mathcal{A}^k)$ le vecteur des variables de flots. $\boldsymbol{T}^k = (T^{kr} | r \in R^k)$ le vecteur des variables de ressources et $\boldsymbol{Y} = (Y_s | s \in S)$ le vecteur des variables additionnelles. Le problème de flot multi-commodités en variables entières avec contraintes de ressources consiste alors à trouver la solution en terme des vecteurs $\boldsymbol{X}^k$, $\boldsymbol{T}^k$ et $\boldsymbol{Y}$, $\forall k \in K$ qui minimise le coût selon les contraintes définies précédemment.

Pour plus d'information, le lecteur est invité à consulter Desaulniers *et al.* (1994). La figure 1.1 donne une idée générale de la structure du problème traité. Les vecteurs $\boldsymbol{C}_X^k$ et $\boldsymbol{C}_T^k$ correspondent aux coûts associés aux vecteurs $\boldsymbol{X}^k$ et $\boldsymbol{T}^k$. Le vecteur $\boldsymbol{b}^0$ et les matrices $\boldsymbol{V}_X^k$ et $\boldsymbol{V}_T^k$ représentent les contraintes globales. De même, les vecteurs $\boldsymbol{b}^k$ et les matrices $\boldsymbol{Z}_X^k$ et $\boldsymbol{Z}_T^k$ représentent les contraintes séparables par commodités. Finalement, la matrice $\boldsymbol{V}_s$ complète la structure du problème.

La difficulté du problème traité requiert l'exploitation de la structure particulière du modèle afin de résoudre des problèmes de taille utile en pratique. Or, ce problème possède une structure diagonale avec contraintes liantes qu'il est possible d'exploiter à l'aide de la méthode de décomposition de Dantzig-Wolfe (Dantzig et Wolfe, 1960). Les *sous-problèmes* sont constitués par les contraintes séparables par commodités et sont souvent résolus au moyen d'algorithmes spécialisés. Les contraintes de coordination entre ces sous-problèmes sont prises en charge par un *problème-maître*.

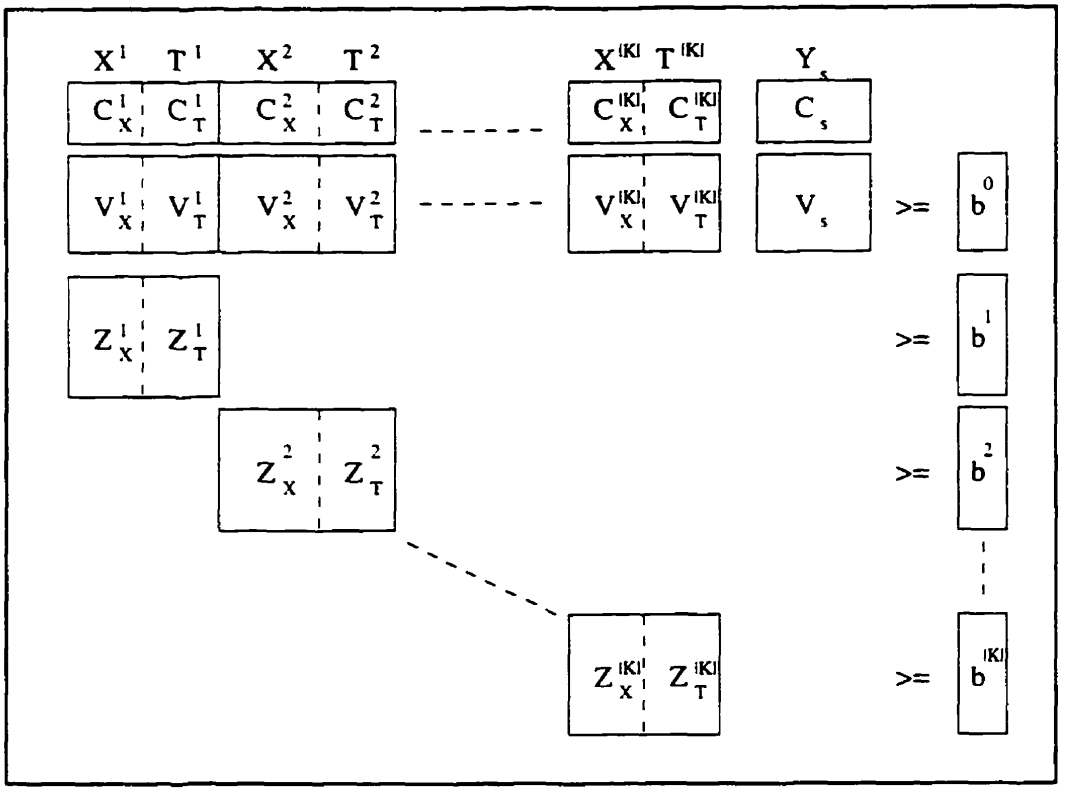


Figure 1.1 Structure du problème traité

Le principe de décomposition de Dantzig-Wolfe consiste à décrire l'ensemble des solutions de chaque sous-problème au moyen d'une combinaison convexe de ses points extrêmes. Toutefois, le grand nombre de points extrêmes rend impossible en pratique leur énumération explicite. Par contre, les méthodes de *génération de colonnes* permettent d'explorer l'espace des solutions de manière implicite.

De manière plus formelle, on associe un sous-problème à chaque commodité $k \in K$. Les points extrêmes qui décrivent ces solutions sont des chemins dans G^k et sont regroupés par commodité dans des ensembles P^k. À chaque chemin $p \in P^k$ est associée une variable continue θ_p^k correspondant à une solution $(\boldsymbol{x}_p^k, \boldsymbol{t}_p^k)$ du sous-problème $k \in K$. Sous forme condensée, on définit pour un sous-problème $k \in K$ donné les vecteurs suivants : le vecteur des variables de chemin $\boldsymbol{\Theta}_P^k = (\theta_p^k \mid p \in$

P^k), le vecteur de coût associé $C_P^k = (C_X^k x_p^k + C_T^k t_p^k \mid p \in P^k)$ et la matrice des contraintes correspondante $V_P^k = (V_X^k x_p^k + V_T^k t_p^k \mid p \in P^k)$. Après relaxation des contraintes d'intégrité sur les variables X_{ij}^k, la substitution des points extrêmes dans la formulation originale donne lieu à un problème-maître qui ne comporte que des contraintes linéaires. Les contraintes non linéaires sont prises en charge par les sous-problèmes. L'algorithme utilisé pour résoudre les sous-problèmes impose qu'un flot d'une unité traverse le réseau, ce qui a pour conséquence de borner la solution et d'empêcher la génération de rayons extrêmes.

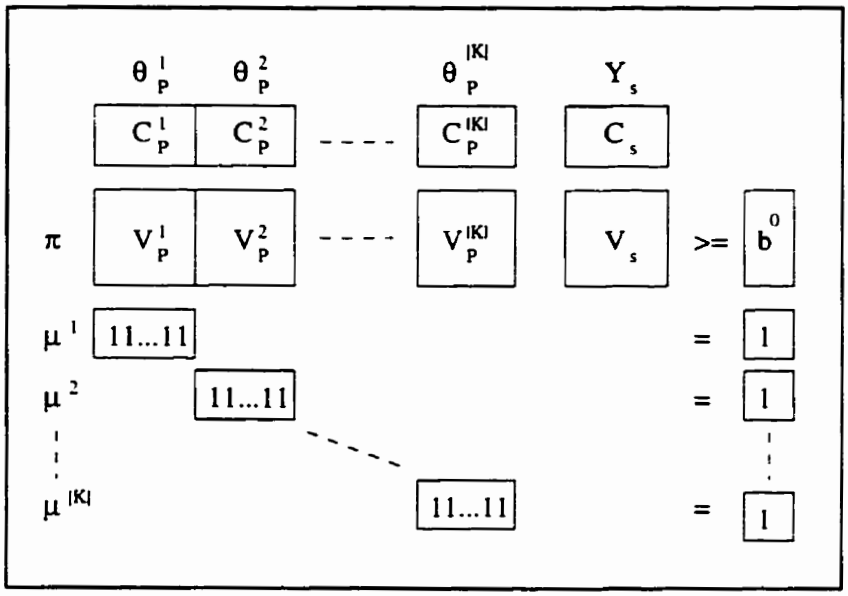


Figure 1.2 Le problème-maître

Les figures 1.2 et 1.3 illustrent la formulation décomposée. Les variables duales π sont associées aux contraintes globales et les variables duales μ^k, $\forall k \in K$, aux contraintes séparables par commodité.

La présente section a mis en évidence le caractère général du modèle. La section suivante restreint le choix des problèmes cibles de manière à pouvoir évaluer la performance des implantations informatiques dans des délais raisonnables.

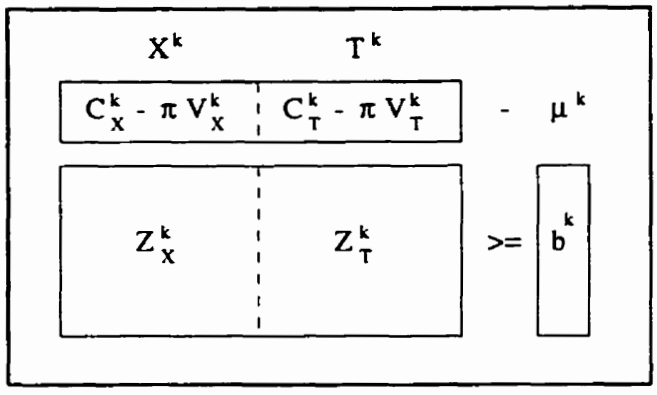


Figure 1.3 Un sous-problème k

1.2 Problèmes cibles

La sélection des problèmes cibles est cruciale tant au niveau de l'évaluation des méthodes proposées que de la portée des travaux. Le sujet de recherche restreint d'office l'éventail des problèmes à ceux de grande taille, souvent rencontrés dans l'industrie. Ces problèmes, de par leur difficulté, offrent aussi un attrait particulier pour les chercheurs. Dans ce contexte, le choix des problèmes de grande taille est tout à fait justifié.

S'il est entendu que les méthodes parallèles proposées doivent performer indépendamment du domaine d'application, il n'en demeure pas moins que leur évaluation doit s'effectuer dans les délais prescrits pour ce projet de recherche. Il s'avère donc nécessaire de ne considérer qu'un sous-ensemble représentatif des classes de problèmes disponibles. Le domaine de la planification des opérations aériennes apparaît comme le candidat idéal.

Les problèmes d'optimisation dans le domaine de la planification aérienne se traitent généralement en trois parties distinctes. C'est dans cette optique qu'un système de prise de décision *modulaire* traitant les trois étapes de la planification des

opérations aériennes a été développé. Les trois problèmes de planification possèdent des structures mathématiques semblables aux problèmes de tournées de véhicule qui consistent à visiter un ensemble de clients au moyen d'une flotte hétérogène de véhicules. En effet, ces trois problèmes consistent à visiter un ensemble de clients avec des chemins respectant certaines contraintes. Lors de l'*affectation des avions* (*fleet assignment*), les clients sont les vols et les chemins sont les rotations d'avion. Les flottes d'avions peuvent être hétérogènes et le problème peut aussi comporter des contraintes d'entretien des avions. Pour la *construction des rotations d'équipage* (*air crew pairing*), les clients sont les segments de vol et les chemins sont les rotations d'équipage. Les contraintes qui régissent la construction des rotations d'équipage varient d'une compagnie aérienne à l'autre. Finalement, lors de la *distribution des rotations au personnel* (*air crew rostering*), les clients sont les rotations et les chemins sont les horaires mensuels pour chaque membre du personnel naviguant. D'autres contraintes comme le nombre minimum de congés ainsi que des périodes de formation obligatoire peuvent compliquer le problème. Dans certains cas, le principe de la séniorité stricte doit être respecté : les préférences des employés seniors doivent être satisfaites avant celles des employés juniors. Ces problèmes sont suffisamment importants pour être différenciés des problèmes de distribution des rotations au personnel et constituent la classe des problèmes de *distribution des rotations au personnel avec séniorité stricte* (*preferential bidding*). Ces trois (ou quatre) classes de problèmes sont toutefois très différentes de par la nature des contraintes que doivent respecter les chemins considérés. D'ailleurs, du point de vue informatique, ces problèmes ne présentent pas le même profil d'exécution.

Problèmes d'affectation des avions : Les problèmes d'affectation des avions (Ioachim, 1994; Desaulniers *et al.*, 1997b) se divisent en trois catégories. Ceux de la première catégorie ne considèrent pas les contraintes de maintenance des avions et

comportent des flottes d'avions homogènes. Ils sont résolus couramment en quelques secondes avec des heuristiques du type premier arrivé premier servi (*fifo*) ou dernier arrivé premier servi (*lifo*). Les problèmes de la deuxième catégorie s'apparentent à ceux de la première mais comportent des flottes d'avions hétérogènes. La plupart d'entre eux se modélisent au moyen d'un programme linéaire en nombres entiers et une solution optimale peut être obtenue en moins de 30 minutes avec un logiciel comme CPLEX (CPLEX, 1994). La troisième catégorie traite le cas général des flottes d'avions hétérogènes et valide les contraintes de maintenance des avions. Ces problèmes sont de grande taille et se résolvent par génération de colonnes. Toutefois, il existe présentement peu de demande pour ce type d'application. Par conséquent, cette classe de problèmes n'est pas retenue.

Problèmes de construction des rotations d'équipage : Les problèmes mensuels de construction des rotations d'équipage (Desaulniers *et al.*, 1997a) de grande taille comptent plusieurs dizaines de milliers de vols. La construction d'un tel horaire mensuel exige au delà d'une semaine de travail en temps d'ordinateur. Afin d'y parvenir, il est d'usage de résoudre de façon heuristique trois problèmes de taille croissante : un horaire journalier avec les vols réguliers, un horaire hebdomadaire avec la semaine la plus représentative et l'horaire mensuel lui-même. De plus, la génération des services de vols, [1] qui sont l'unité de base dans la construction des réseaux, implique une forte combinatoire. Si des techniques récentes permettent de générer tous les services de vols, les limitations technologiques empêchent de construire le réseau associé. Il est donc nécessaire d'utiliser des heuristiques basées sur la composition des services de vols pour épurer la liste jusqu'à l'obtention d'un sous-ensemble de taille utilisable en pratique.

[1] Un service de vols est une séquence de vols commençant et se terminant par un repos.

Les sous-problèmes consistent à résoudre un problème de plus court chemin avec contraintes de ressources (Desrochers, 1986) sur les réseaux construits à partir des services de vols sélectionnés. Ces sous-problèmes requièrent un temps de calcul considérable et exigent l'emploi d'heuristiques supplémentaires pour réduire l'espace d'états : échantillonage des arcs, échantillonage des états à chaque nœud, etc. Quant au nombre de colonnes générées, il existe un seuil (autour de 100 colonnes) au-dessus duquel le nombre de pivots (algorithme du type simplexe) nécessaires à la réoptimisation du problème-maître devient proportionnel à la taille de ce dernier. Le ratio du temps de calcul entre les sous-problèmes et le problème-maître varie de 0.5 à 5. Ces résultats proviennent de l'expérience [2] personnelle de l'auteur, expérience acquise en tant que professionnel de recherche et ne faisant pas partie des présents travaux.

Pour obtenir une solution entière dans les délais respectant les exigences des clients, il est nécessaire de contraindre le problème de façon agressive à chaque nœud de branchement. Les décisions imposées retranchent une partie du problème de manière irréversible, la partie éliminée n'étant plus considérée. Ce type de décisions complique parfois la réoptimisation du problème-maître qui suit immédiatement leur application. Même si ces méthodes ne créent qu'un fils par nœud, la résolution d'un problème type exige l'exploration de 100 à 200 nœuds de branchement. La difficulté de ces problèmes justifie leur sélection.

Problèmes de distribution des rotations au personnel : Les problèmes de distribution des rotations au personnel (Gamache *et al.*, 1994; Gamache, 1995) et, plus

[2] L'auteur a participé activement en 1996 à deux tests (*benchmarks*) de construction des rotations d'équipage et un de distribution des rotations au personnel chez une compagnie spécialisée dans les problèmes du domaine aérien.

spécifiquement, les problèmes de distribution des rotations au personnel avec séniorité stricte (Gamache *et al.*, 1997), possèdent une dynamique différente et requièrent environ une journée de calcul. Les problèmes de distribution des rotations au personnel avec séniorité stricte comportent souvent plusieurs centaines de sous-problèmes alors que les problèmes de construction des rotations d'équipage en nécessitent rarement plus de 20. La structure du problème est telle que l'obtention d'une solution continue réalisable constitue parfois l'essentiel du travail. Les décisions de branchement sont généralement fines, ce qui conduit à des nœuds de branchement où l'optimisation des sous-problèmes est rarement nécessaire. Le ratio du temps de calcul entre les sous-problèmes et le problème-maître est souvent inférieur à 1. Les problèmes de distribution des rotations au personnel avec séniorité stricte sont choisis car des jeux de données sont disponibles. [3]

L'emploi de méthodes parallèles augmenterait la puissance de calcul disponible, ce qui permettrait de *réduire l'usage des heuristiques* et ainsi d'*obtenir des solutions de meilleure qualité*. C'est dans ce contexte que le choix des classes de problèmes de construction des rotations d'équipage et de distribution des rotations au personnel avec séniorité stricte est justifié.

Compte tenu du sujet traité, le reste du mémoire doit faire appel à des notions de parallélisme plus ou moins spécialisées pour décrire la problématique. C'est pour cette raison que la section suivante offre un bref survol du domaine du parallélisme.

[3] Au moment de la décision, le logiciel pour traiter les problèmes de distribution des rotations au personnel n'avait pas encore été mis à jour pour la version de GENCOL utilisée dans ce mémoire.

1.3 Concepts théoriques

Cette section constitue une brève introduction au domaine du parallélisme et a pour but de n'en présenter que les éléments essentiels à la compréhension de ce projet de recherche. Après une courte définition du parallélisme, les concepts sont décrits dans l'ordre suivant : les modèles de machines parallèles, les types de synchronisation, les modèles de gestion des tâches parallèles et la mesure du temps d'exécution. Le lecteur intéressé à un traitement purement théorique du sujet peut consulter Jájá (1992). Pour une approche plus pratique, Bauer (1990) est une excellente alternative. Finalement, Foster (1995) propose un ouvrage sur la conception et l'implantation d'algorithmes parallèles qui comporte une bibliographie exhaustive (plus de 300 références).

1.3.1 Parallélisme

Pour les fins de ce mémoire, un *programme informatique* consiste en un ou plusieurs algorithmes traitant un ensemble de données dans le but de produire des résultats corrects. Chaque algorithme couplé à ses données peut être divisé en une ou plusieurs *tâches* susceptibles d'être effectuées concurremment. Le but du *parallélisme* est de concevoir les algorithmes d'un programme de façon à favoriser le traitement du plus grand nombre possible de tâches en parallèle afin de réduire le temps d'attente de l'usager.

Ces définitions ne reflètent pas toute la complexité du parallélisme. Il existe des définitions plus formelles (Foster, 1995) qui exigent la connaissance de concepts avancés dans le domaine du parallélisme. Toutefois, ces concepts ne sont pas

nécessaires dans le cadre de ce mémoire.

1.3.2 Modèles de machines parallèles

Une *machine parallèle* (Bauer, 1990; Foster, 1995) est un ensemble de processeurs capables de travailler ensemble pour résoudre un problème. Cette définition est suffisamment générale pour inclure les super-ordinateurs, les stations de travail, les ordinateurs personnels et les ordinateurs multiprocesseurs. On peut diviser les ordinateurs multiprocesseurs en deux catégories : les ordinateurs massivement parallèles et les ordinateurs faiblement parallèles. La première catégorie est constituée en grande partie d'ordinateurs dédiés : un système d'exploitation fait sur mesure, très peu de mémoire à accès rapide par processeur et une très faible puissance de calcul par processeur. Évidemment, tous ces défauts sont compensés par le très grand nombre de processeurs, soit de 100 à plus de 10 000. Les deux catégories de machines multiprocesseurs étant si différentes l'une de l'autre, il est courant que les algorithmes parallèles développés pour une des deux catégories performent significativement moins bien pour l'autre. Il ne sera question dans ce mémoire que de la seconde catégorie, soit les ordinateurs ayant entre 2 et 30 processeurs et dont le système d'exploitation est une variante d'UNIX (Leffler *et al.*, 1989). Ce choix est justifié par le caractère sur-spécialisé des machines massivement parallèles et les problèmes d'implantation qui en découlent. De plus, les ordinateurs multiprocesseurs de la deuxième catégorie constituent un marché en pleine expansion.

Les récentes percées technologiques dans le domaine des communications ont permis d'accroître significativement la largeur de bande des réseaux locaux d'ordinateurs. Il n'est pas rare de rencontrer des réseaux opérant à 100 Mbits/s et plus, ce qui est impressionnant quand on considère que les ordinateurs de la dernière décennie

opéraient sensiblement à la même vitesse pour accéder à leur mémoire centrale! Dans ce contexte, de nouvelles possibilités jusqu'alors insoupçonnées s'offrent aux utilisateurs dont, entre autres, la perspective d'utiliser un réseau de vieilles machines en parallèle pour créer une *machine virtuelle* (Geist *et al.*, 1994; Foster, 1995) d'une puissance de calcul dépassant même celle du modèle mono-processeur dernier cri.

La définition de la machine parallèle présentée précédemment est trop floue pour pouvoir décrire une architecture parallèle avec précision. Il est donc nécessaire de définir quelques modèles plus en détails. Toutefois, il faudrait un chapitre entier pour identifier et répertorier les différentes architectures parallèles et c'est pour cette raison que seuls les modèles qui ont un intérêt pour le présent projet de recherche seront explicités.

Un premier modèle de machines parallèles regroupe les ordinateurs à *mémoire distribuée*. Ces ordinateurs sont composés d'un certain nombre de *nœuds* constitués chacun d'un processeur et reliés entre eux par un *réseau*. Chaque nœud exécute son propre programme, a accès à une mémoire locale et peut transmettre des messages aux autres nœuds. Le coût d'envoi d'un message dépend de la position des nœuds les uns par rapport aux autres ainsi que de la quantité de traffic sur le réseau. La caractéristique principale de ce modèle est qu'un accès à la mémoire locale prend significativement moins de temps que la transmission d'un message. Ce modèle favorise ainsi les algorithmes qui maximisent la *localité* des données.

Un second modèle regroupe les ordinateurs à *mémoire partagée*. Dans cette famille, la mémoire est centralisée et est accessible à travers une hiérarchie de bus de données ou de niveaux de *mémoire cache* (mémoire à accès rapide). Cette famille peut à son tour être représentée par un modèle idéal nommé PRAM (*Parallel Random Access Machine*). Ce modèle est très populaire dans la littérature (Jájá, 1992) car

sa simplicité permet des preuves faciles. Évidemment, d'autres modèles beaucoup plus complexes ont par la suite été développés pour mieux représenter la réalité mais leur étude ne fait pas partie de ce projet de recherche.

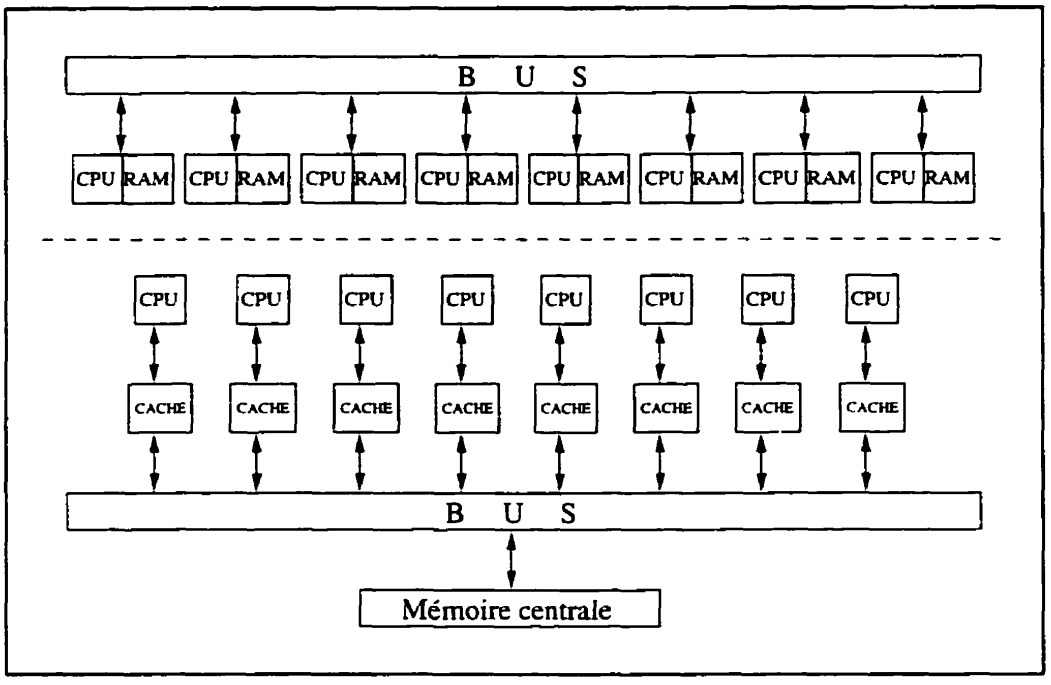


Figure 1.4 Types de machines parallèles

La figure 1.4 montre en exemple deux machines parallèles : une à mémoire distribuée (diagramme du haut) et l'autre à mémoire partagée (diagramme du bas). Dans le cas de la machine à mémoire distribuée, CPU désigne un processeur, RAM un bloc de mémoire, BUS un réseau quelconque d'interconnexions et les flèches symbolisent des canaux de communication bi-directionnels. La paire CPU-RAM forme une unité d'exécution indépendante qui peut communiquer avec ses pairs à l'aide du bus d'interconnexions. Quant à la machine à mémoire partagée, CACHE représente un bloc de mémoire rapide réservé à un processeur. Cette fois-ci c'est la paire CPU-CACHE qui forme l'unité d'exécution indépendante. Ces unités d'exécution communiquent entre elles en s'échangeant des messages au moyen de la mémoire centrale.

Ceci dit, peu importe l'architecture et la façon de classifier les machines parallèles, l'essentiel est de pouvoir utiliser efficacement la puissance de calcul disponible. Pour ce faire, il faut coordonner l'effort des différents processeurs à l'aide de mécanismes de synchronisation qui sont décrits à la section suivante.

1.3.3 Synchronisation

Une part importante dans le développement d'algorithmes parallèles est consacrée à la coordination du travail effectué par les processeurs. Cette coordination peut être plus ou moins stricte et donne lieu aux trois *types de synchronisation* que voici.

- *Synchronisation globale* : les processeurs travaillent sur un ensemble de tâches parallèles qui doivent être complétées avant de passer à l'étape suivante. Ce type de synchronisation est aussi connu sous le nom de *barrière*. Les algorithmes basés sur cette méthode sont très peu susceptibles d'être affectés par des problèmes d'inter-blocage [4] (*deadlocks*) et ils ne requièrent généralement que peu de communications inter-processeurs.

- *Synchronisation locale* : un processeur doit attendre après un ou plusieurs autres processeurs pour de l'information ou une ressource commune avant de poursuivre son travail. Ce type de synchronisation peut être plus efficace que le précédent mais il est aussi plus difficile à implanter car il nécessite une quantité importante de communications inter-processeurs.

- *Asynchrone* : chaque processeur travaille seul et possiblement avec des données périmées. L'absence complète de synchronisation permet d'éviter les temps

[4] Un inter-blocage est un état non prévu par l'algorithme qui peut causer un gel d'un ou de plusieurs processeurs.

d'attente mais peut aussi augmenter la quantité de travail inutile. La définition et l'implantation des conditions d'arrêt sont les principales difficultés de cette méthode.

Naturellement, une implantation parallèle peut être amenée à utiliser une combinaison de ces trois types de base.

Les programmes parallèles récents utilisent le modèle des *fils* [5] d'exécution (*threads*). On peut voir ces fils comme autant d'unités d'exécution indépendantes partageant les données d'un même programme. Une coordination de l'effort de travail des fils nécessite généralement l'emploi de *mécanismes de synchronisation* présentés ici en ordre croissant de complexité.

- *Verrou d'exclusion mutuelle* (*mutex*) : entité qui ne peut appartenir qu'à un seul fil à la fois. Pour s'approprier un mutex, un fil doit tenter de le verrouiller. Si le mutex est déjà verrouillé, le fil bloque jusqu'à ce que son propriétaire le déverrouille. La principale utilité des mutex est de permettre un accès exclusif à des portions de code ou de données.
- *Variable de condition* : lorsqu'employée avec un mutex, elle permet de mettre un fil en attente jusqu'à ce qu'une condition soit satisfaite. À ce moment précis, un signal est envoyé et un des fils en attente est réveillé et obtient automatiquement le mutex associé.
- *Sémaphore* : variable entière qui peut être diminuée ou augmentée de façon exclusive par un fil.
- *Verrou lectures/écriture* : version plus élaborée du mutex qui permet l'accès concurrentiel à des données en mode lecture par plusieurs fils. Un verrouillage

[5] La définition technique des fils d'exécution est donnée à la section 2.3.2.

en mode écriture garantit un accès exclusif.

Ces mécanismes de synchronisation font partie des outils qui serviront plus tard à développer des implantations informatiques capables de traiter le problème en parallèle. Ils peuvent être utilisés dans différents contextes de gestion des tâches parallèles, qui sont présentés à la prochaine section.

1.3.4 Gestion des tâches parallèles

Il est courant de diviser les tâches parallèles en deux grandes catégories. Les tâches *homo-parallèles* requièrent un traitement algorithmique semblable et donc un seul type de processus. Les tâches *hétéro-parallèles* requièrent un processus spécialisé pour chaque tâche ou classe de tâches. La façon d'organiser ces processus et de leur distribuer les tâches constitue une *méthode de gestion*. Ces méthodes sont qualifiées selon le type de tâches traitées.

- *Pipeline* : méthode hétéro-parallèle dans laquelle chacune des tâches doit être traitée séquentiellement par divers processus spécialisés.
- *Serviteurs* : méthode généralement hétéro-parallèle où chaque tâche est traitée par un processus dédié, i.e., un processus par tâche.
- *Maître-esclaves* : méthode globalement hétéro-parallèle où la gestion est centralisée dans un seul processus, le processus maître, et où l'essentiel du travail est exécuté par des processus esclaves. Le type des processus esclaves détermine si la méthode est localement hétéro-parallèle ou homo-parallèle.
- *Pool de travail* : méthode homo-parallèle constituée de processus identiques capables de traiter l'ensemble des tâches.

Il n'y a pas de type de gestion qui prévaut sur les autres. Dans la plupart des cas, un seul s'établit comme le choix tout indiqué. Étant donné ce choix, il faut ensuite définir un ensemble de métriques pour évaluer la qualité des implantations parallèles. Les métriques utilisées dans ce projet de recherche sont toutes basées sur la notion de temps d'exécution, qui est formalisée à la section suivante.

1.3.5 Mesure du temps d'exécution

Le *temps d'exécution* d'un programme sur des données de taille N dans un environnement informatique multi-usagers peut être défini de plusieurs façons. Pour un programme séquentiel donné, la quantité $\hat{T}(N)$ représente le temps de calcul (CPU *time*) et la quantité $T(N)$, le temps réel écoulé (*wall clock time*). Ces deux mesures (temps utilisé par la machine versus temps perçu par l'usager) sont interchangeables quand on peut négliger l'influence des autres usagers.

Dans le cas d'un programme parallèle exécuté sur p processeurs, les deux mesures précédentes ne sont plus équivalentes (Foster, 1995). Le temps de calcul $\hat{T}(N,p)$ doit tenir compte du travail de tous les processeurs tandis que le temps réel écoulé $T(N,p)$ conserve la même signification que $T(N)$. Le cas d'un parc de machines hétérogènes n'étant pas considéré dans ce mémoire (voir la section 2.2 pour la justification de cette décision), le nombre de processeurs p est ici un entier dont la valeur est connue avec précision. Typiquement, l'utilisation du parallélisme pour réduire le temps réel écoulé amène aussi une augmentation du temps de calcul des p processeurs. Pour un logiciel parallèle donné, $T(N,1)$ reste semblable à $\hat{T}(N,1)$ et $T(N,p)$ devient normalement inférieur à $\hat{T}(N,p)$ pour $p > 1$. Une grande partie de l'effort de développement des algorithmes parallèles est d'obtenir que $T(N,p)$ tende de façon asymptotique vers $\hat{T}(N,p)/p$ pour $p > 1$.

Les critères d'évaluation définis à la prochaine section découlent directement des deux mesures du temps d'exécution, soit le temps de calcul et le temps réel écoulé.

1.4 Critères d'évaluation

Les *critères d'évaluation* permettent de quantifier et de comparer certains aspects des algorithmes proposés aux chapitres 4 et 5. Les deux métriques suivantes seront utilisées dans ce mémoire comme critères d'évaluation car elles sont simples à interpréter et très populaires dans la littérature.

En premier lieu, l'*accélération* (Dhall et Lakshmivarahan, 1990) permet d'évaluer la vitesse relative d'un algorithme parallèle par rapport à un algorithme séquentiel de référence. Afin que les travaux de recherche aient une quelconque valeur, il est essentiel de choisir une référence qui peut être considérée comme l'état de l'art dans le domaine. Ce choix sera fait dans le prochain chapitre. De façon plus formelle, l'accélération est définie comme

$$A(N,p) = \frac{T(N)}{T(N,p)}. \tag{1.1}$$

où $T(N)$ correspond au temps requis par le meilleur algorithme séquentiel connu. Rappelons que p représente le nombre de processeurs, N la taille du problème et $T(\cdot)$ le temps réel écoulé. Généralement, $T(N)$ est inférieur à $T(N,1)$ parce que les mécanismes parallèles impliquent une gestion supplémentaire qui est superflue avec un seul processeur. De plus, lorsque $p > 1$, on s'attend à ce que $A(N,p) \geq 1$. Il existe une variante de cette mesure (*scaled speedup*) qui tient compte du fait que la machine multiprocesseurs pourrait traiter des problèmes de plus grande taille que la machine mono-processeur (Gustafson, 1988; Cosnard, Robert et Tourancheau,

1989). Cette mesure soulève toutefois des objections (Wilson, 1995) et donne lieu à d'autres métriques (*isoefficiency* de Kumar *et al.*, 1994).

En deuxième lieu, l'*efficacité* définie comme

$$E(N,p) = \frac{A(N,p)}{p}, \tag{1.2}$$

permet de quantifier le travail utile soutiré à la machine parallèle. En général, $E(N,p) < 1$ et $E(N,p)$ tend à décroître avec p croissant.

Il existe deux variantes de ces métriques. La variante locale où seule la portion parallèle de l'algorithme est mesurée et la variante globale où toute l'exécution est prise en compte. La variante locale permet d'évaluer la force brute de l'algorithme proposé et de comparer avec d'autres algorithmes si la portion parallèle est la même. Quant à la variante globale, elle donne une idée directe de l'amélioration (ou de la dégradation) de vitesse perçue par l'usager.

Ces critères, dans leurs deux variantes, serviront à évaluer quantitativement les implantations informatiques présentées ultérieurement. Il est aussi pertinent de définir deux autres métriques afin de mieux analyser les résultats.

Le ratio suivant

$$W(N,p) = \frac{\hat{T}(N,p)}{\hat{T}(N)}, \tag{1.3}$$

où $\hat{T}(\cdot)$ représente le temps de calcul, permet d'apprécier les variations de la quantité de *travail* total. Lorsque $p > 1$, on observe en général que $W(N,p) \geq 1$. Cet accroissement de la quantité de travail est dû bien souvent à l'inefficacité de l'algorithme parallèle.

Finalement, il faut mentionner que l'accélération maximale $A_{\max}(\cdot)$ est limitée

par la proportion séquentielle s [6] de l'algorithme (Amdhal, 1967) :

$$A_{\max}(N, p) = \frac{1}{s}. \tag{1.4}$$

De plus, elle est généralement atteinte à une valeur inférieure à $p_{\max}$, le nombre maximal de processeurs disponibles.

Ces critères d'évaluation serviront à mesurer la performance des implantations informatiques. Il est nécessaire, toutefois, d'évaluer *a priori* le potentiel parallèle des algorithmes candidats afin de n'implanter que les plus prometteurs.

1.5 Potentiel parallèle

Le *potentiel parallèle* d'un algorithme constitue une mesure qualitative de son espérance de gain relativement au temps réel d'exécution. Plus spécifiquement, le potentiel parallèle est défini aux fins de ce mémoire au moyen des cinq caractéristiques suivantes.

- *Granularité* : quantité de tâches parallèles. En général, il est préférable d'avoir un nombre élevé de tâches pouvant être exécutées en parallèle. Toutefois, le temps de traitement individuel des tâches tend à diminuer au fur et à mesure que leur nombre augmente, ce qui peut accroître l'importance des coûts de communication et exiger une architecture spécialisée.
- *Type de tâches parallèles* : dépendantes ou indépendantes. Des tâches indépendantes simplifient grandement leur ordonnancement car elles peuvent être exécutées dans n'importe quel ordre. À l'inverse, des tâches dépendantes sont plus difficiles à gérer et diminuent le degré de parallélisme.

[6] La définition technique de la proportion séquentielle est donnée à la section suivante.

- *Volume de communication* : un fort volume a tendance à augmenter les délais et à dégrader les performances. Inversement, un faible volume peut aider à diminuer les délais de communication.

- *Proportion séquentielle* : proportion du temps total d'exécution d'un algorithme parallèle pendant laquelle il n'est possible de traiter qu'une tâche à la fois. L'accélération maximale est inversement proportionnelle à cette quantité (voir section 1.4).

- *Type de comportement* : déterministe ou non. Le déterminisme est la qualité d'un algorithme permettant d'obtenir des résultats (solutions, temps d'exécution, statistiques, etc.) identiques lors d'exécutions différentes dans les mêmes conditions. Un algorithme déterministe efficace est plus difficile à concevoir qu'un algorithme non-déterministe car il exige généralement beaucoup plus de points de synchronisation. Par contre, son caractère déterministe lui confère une robustesse supérieure en permettant la reproduction de résultats antérieurs avec sensiblement le même temps d'exécution.

Ces caractéristiques qui composent le potentiel parallèle vont permettre une analyse approfondie, au chapitre 3, du logiciel d'optimisation séquentiel de référence qui sera choisi au chapitre 2.

1.6 Conclusion

Ce chapitre a permis de délimiter la portée théorique des travaux décrits dans ce mémoire au moyen du modèle unifié. Le choix des classes de problèmes retenues, soit les problèmes de construction des rotations d'équipage et de distribution des

rotations au personnel avec séniorité stricte en planification aérienne, a été justifié par leur temps de calcul élevé ainsi que par l'utilisation massive d'heuristiques nécessaires à leur résolution. Des concepts du domaine du parallélisme ont été présentés afin d'assurer un niveau de connaissance suffisant de la part du lecteur. Par la suite, les critères d'évaluation ont été définis, soit l'accélération et l'efficacité. Finalement, le potentiel parallèle a été caractérisé en cinq points, soit la granularité, le type de tâches parallèles, le volume de communication, la proportion séquentielle et le type de comportement.

Le chapitre suivant a pour tâche de délimiter la portée des travaux au niveau pratique en choisissant le logiciel d'optimisation séquentiel de référence, le banc d'essai et les outils parallèles.

CHAPITRE 2

Choix des outils informatiques

Le chapitre précédent a délimité la portée des travaux au niveau théorique et en a défini les principaux concepts. S'il a su mettre en évidence le caractère général des travaux, il a tout de même fallu circonscrire les problèmes cibles pour des raisons d'ordre pratique. Il en va de même quant au choix des outils informatiques. La première section décrit un logiciel d'optimisation générique pouvant résoudre le modèle unifié et choisit une implantation séquentielle considérée comme l'état de l'art dans le domaine. La deuxième section présente la problématique de l'évaluation de la puissance de calcul d'un parc de machines et détermine le banc d'essai. La dernière section définit deux outils informatiques supplémentaires qui seront utilisés aux chapitres 4 et 5.

2.1 Choix du logiciel d'optimisation séquentiel

L'environnement informatique du projet de recherche est composé de deux parties distinctes. D'une part, un environnement de développement qui englobe un langage de programmation ainsi qu'un ensemble d'outils d'aide au développement. D'autre part, une bibliothèque de procédures du domaine de la recherche opérationnelle permettant la création d'applications scientifiques se conformant au modèle théorique défini au chapitre précédent. Dans le cadre de ce projet de recherche, l'environnement informatique est choisi d'office et se compose de l'environnement de développement du groupe de recherche GENCOL et du logiciel du même nom. Il est important

de mentionner que le choix du logiciel GENCOL ne diminue en rien l'aspect général des travaux qui portent sur l'étude du parallélisme dans le cadre du modèle unifié.

L'interface fonctionnelle d'une bibliothèque de procédures comme GENCOL définit les services disponibles et les interactions possibles entre les modules des applications (utilisateurs de la bibliothèque) et la bibliothèque elle-même. Les implantations informatiques dont traite ce mémoire doivent s'intégrer à l'interface fonctionnelle actuelle de GENCOL ou, sinon, exiger le moins de modifications possibles à celle-ci. S'il est vrai que les implantations informatiques issues des présents travaux de recherche sont fortement couplées au logiciel GENCOL, rien n'empêche de généraliser les conclusions à d'autres logiciels traitant les mêmes problèmes. Il sera mentionné explicitement dans le texte lorsque la généralisation ne pourra être appliquée.

Cette section a un double but. D'une part, décrire les modules de la bibliothèque GENCOL ainsi que la technique de génération de colonnes. D'autre part, démontrer que GENCOL est un logiciel représentatif de ceux capables de résoudre les problèmes pouvant être décrits par le modèle unifié.

La méthode de génération de colonnes suppose l'existence d'un problème-maître et d'un ou de plusieurs sous-problèmes. Dans GENCOL, le problème-maître est un programme linéaire qui est résolu avec CPLEX CALLABLE LIBRARY (CPLEX, 1994), dénoté CPLEX par la suite. D'autres implantations pourraient utiliser des optimiseurs de programmes linéaires provenant d'autres compagnies que CPLEX, basées ou non sur la même technologie. Quant aux sous-problèmes, ce sont des problèmes de plus court chemin avec contraintes de ressources sur des réseaux. Ces sous-problèmes sont résolus dans GENCOL à l'aide d'un algorithme de programmation dynamique (Desrochers, 1986). La méthode consiste alors à résoudre par

génération de colonnes la *relaxation linéaire* du problème en traitant alternativement le problème-maître et les sous-problèmes de la façon suivante. Tout d'abord, la résolution du problème-maître détermine la solution optimale en nombres réels respectant les contraintes liantes avec les colonnes disponibles. Après quoi, les sous-problèmes récupèrent l'information duale du problème-maître afin de recalculer les coûts réduits appropriés. Ils génèrent ensuite de nouvelles colonnes de coût réduit minimal. Si aucune colonne de coût réduit négatif n'est obtenue, la solution courante est une solution optimale du problème relaxé. Sinon, ces colonnes sont ajoutées au problème-maître et le processus se poursuit. Par la suite, la solution optimale entière du problème est obtenue par un processus de séparation et d'évaluation progressive (*Branch and Bound*). La boîte inférieure de la figure 2.1 illustre la méthode de génération de colonnes. Dans la figure, MP représente le module du problème-maître et SP le module des sous-problèmes.

Pour traiter les problèmes provenant d'applications commerciales, s'ajoutent à l'optimiseur GENCOL deux étages supplémentaires propres à chaque application. Au niveau intermédiaire, l'application reformule le problème en fonction du modèle unifié. Au niveau supérieur, on retrouve les interfaces usagers permettant d'analyser et de modifier les données et les solutions. Ces deux étages ne font pas l'objet du présent mémoire et ne sont mentionnés que pour brosser un tableau plus complet du logiciel. La boîte supérieure de la figure 2.1 illustre les deux étages supplémentaires.

Le logiciel GENCOL, qui en est à sa quatrième génération, est un outil d'optimisation qui utilise l'approche de la génération de colonnes. Les problèmes traités sont généralement de grande taille et proviennent en majorité d'applications commerciales. GENCOL permet la résolution optimale des problèmes pouvant être décrits de façon exacte par le modèle unifié. Toutefois, la résolution de problèmes de taille utile en pratique nécessite souvent un compromis entre le temps d'exécution et

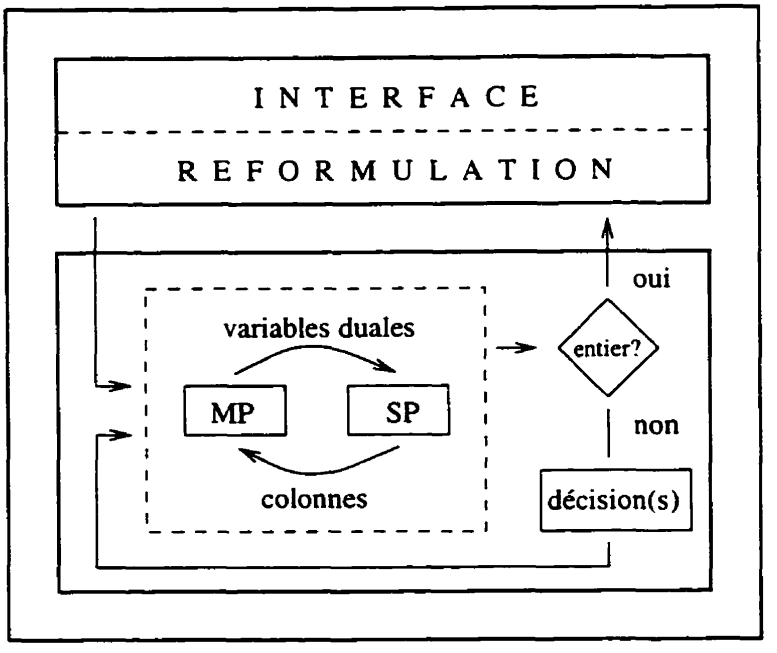


Figure 2.1 Une application basée sur GENCOL

l'optimalité de la solution. Ce compromis est réalisé par un *langage de données* qui prend la forme d'un fichier de paramètres. Ce langage facilite les tests lors du développement de nouvelles applications en permettant un ajustement externe, fin ou grossier, du processus de résolution pour des problèmes particuliers.

Contrairement à ses prédécesseurs, GENCOL n'est pas un *exécutable* (programme autonome) monolithique mais plutôt une bibliothèque de modules permettant de concevoir des logiciels d'optimisation très complexes. Les interfaces fonctionnelles sont les roues d'engrenages qui permettent de tenir en place les modules de la bibliothèque. La qualité de l'implantation dépend donc directement des interfaces fonctionnelles reliant les modules entre eux et avec le module de l'application.

Il est suffisant de n'expliciter que les relations logiques entre les différents modules à l'aide de la figure 2.2. Les flèches symbolisent le type de connaissance que le module à l'origine d'une flèche a du module à la pointe de cette flèche. Une flèche

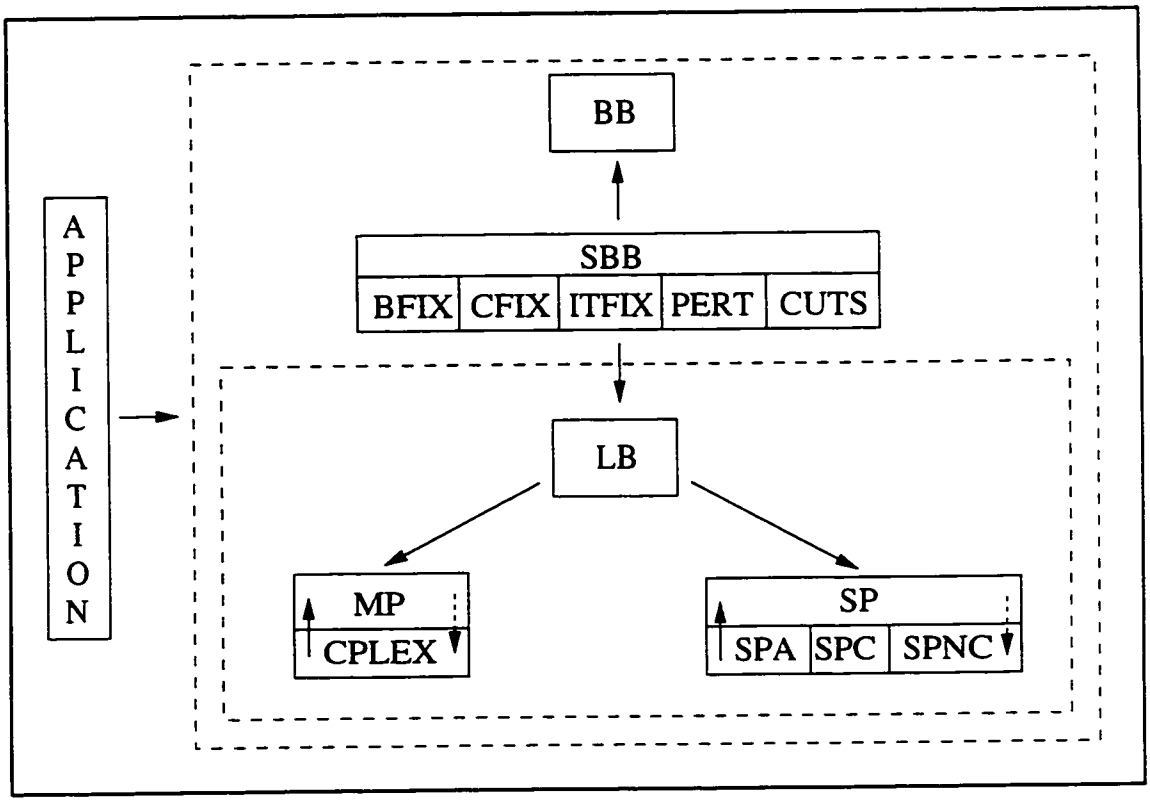


Figure 2.2 GENCOL : une bibliothèque de modules

pleine entre deux modules A et B signifie que le module A connaît l'interface fonctionnelle et les structures de données principales du module B. Une flèche pointillée signifie que le module A ne connaît que l'interface fonctionnelle du module B.

APPLICATION représente un logiciel d'application qui utilise GENCOL pour résoudre une classe de problèmes donnée. Il existe entre autres une application appelée GENCOL, souvent confondue avec la bibliothèque, qui n'est simplement qu'un pilote (*driver*) pour celle-ci et qui permet aux usagers de traiter les problèmes les plus simples sans avoir à connaître les détails de l'implantation.

LB est le module de recherche d'une borne inférieure. Dans la version courante, le module LB est appelé SPP et la borne inférieure implantée est la relaxation linéaire du problème. SP est un module générique de résolution de sous-problèmes. SPA, SPC et SPNC constituent trois implantations du module SP résolvant des problèmes de plus court chemin avec contraintes de ressources sur des graphes acycliques, cycliques et acycliques avec coûts linéaires sur les nœuds respectivement. MP est un module générique de résolution du problème-maître. CPLEX est une instance d'un logiciel de résolution de programmes linéaires. BB est un module générique de séparation et d'évaluation progressive qui peut être utilisé dans d'autres projets. SBB regroupe les stratégies de branchement et de coupes spécifiques au problème de flot multi-commodités. De plus, SBB peut aussi procéder à un pré-traitement capable d'éliminer des contraintes redondantes dans le problème. ITFIX, CFIX, etc. sont différentes méthodes de prise de décisions ou de coupes. Dans le contexte de ce mémoire, le seul détail pertinent sur ces méthodes est que certaines d'entre elles peuvent introduire des contraintes supplémentaires au niveau du sous-problème, ce qui en ralentit la résolution.

Le lecteur conviendra que l'architecture modulaire présentée à la figure 2.2

aurait pu provenir de n'importe quel autre logiciel résolvant le modèle unifié. Si l'utilisation de CPLEX ainsi que les noms des modules sont spécifiques à GENCOL, il n'en demeure pas moins qu'un optimiseur pour le modèle unifié doit posséder un module pour traiter le problème-maître, un autre pour traiter les sous-problèmes, etc. Rappelons qu'il sera mentionné explicitement dans le texte lorsque les conclusions sont spécifiques au logiciel GENCOL.

Ceci dit, un autre élément important de l'environnement informatique est le choix du banc d'essai. La section suivante porte sur ce choix et justifie une hypothèse sur le paramètre p (le nombre de processeurs) faite à la section 1.3.5. Cette justification est nécessaire car ce choix a des répercussions directes sur la validité des critères d'évaluation définis à la section 1.4.

2.2 Choix du banc d'essai

Les critères d'évaluation définis à la section 1.4 requièrent la connaissance de p, le nombre de processeurs. Si le banc d'essai est composé de machines hétérogènes, il est nécessaire de calculer la puissance relative des machines avant d'en faire la somme. Il s'est développé une pléthore de tests de performance (*benchmarks*) pour comparer la puissance relative des ordinateurs. S'il y a une seule chose qu'il faut retenir de tous ces tests, c'est qu'il en existe autant qu'il y a d'usagers. En effet, un ordinateur est un ensemble de composantes (unité centrale de traitement, unité de calcul arithmétique en point flottant, mémoire, entrées/sorties, etc.) dont la qualité des interactions importe autant sinon plus que leur performance individuelle. Ceci a pour conséquence d'amoindrir la portée des tests les plus fiables, soit ceux qui évaluent une composante spécifique de la machine. Quant aux tests composites,

ils n'ont de valeur que dans la mesure où l'utilisation des composantes respecte les proportions des programmes dont on veut faire usage, ces proportions n'étant généralement pas connues en pratique.

Dans le cadre de ce mémoire, le banc d'essai servira à résoudre en parallèle des problèmes découlant du modèle unifié. Une façon d'évaluer p pour le parc de machines disponibles sans connaître l'utilisation exacte des composantes des machines serait d'utiliser comme test de performance l'algorithme séquentiel lui-même. Ce choix suppose cependant que les algorithmes parallèles utilisent toutes les machines du parc selon les mêmes proportions que l'algorithme séquentiel. Or, la plupart des algorithmes parallèles sont constitués de tâches hétérogènes (voir section 1.3.4) qui, une fois distribuées aux machines, biaisent localement les proportions. Ainsi, une mesure fiable de la puissance relative des machines devrait tenir compte des tâches qui leur sont assignées, ce qui fait intervenir l'algorithme parallèle lui-même dans le calcul de p. Ce dernier argument implique que la valeur de p fluctue selon l'algorithme parallèle utilisé. Cette valeur pourrait même changer significativement entre deux exécutions du même algorithme parallèle sur le même parc de machines si les tâches sont assignées à des machines différentes. En conclusion, pour déterminer correctement la valeur de p d'un banc d'essai composé de machines hétérogènes, il faudrait développer des méthodes d'évaluation qui dépassent le cadre de ce projet de recherche.

Dans le but d'éviter les problèmes mentionnés, le parc de machines est choisi homogène. Le banc d'essai ne comporte donc qu'une seule machine parallèle, soit un SPARCSERVER 1000 à huit processeurs opérant avec le système d'exploitation Solaris 2.3 (et plus récemment 2.4 et 2.5). C'est une machine multiprocesseurs à mémoire partagée fournie par le centre de recherche GERAD de Montréal.

2.3 Choix des outils informatiques spécialisés

Les outils informatiques utilisés lors du projet de recherche sont décrits dans cette section. D'abord, la bibliothèque de transport de messages IPC qui servira à simuler une machine virtuelle (voir section 1.3.2) sur un réseau. Ensuite, la bibliothèque THREADS de SUN qui fournira les mécanismes de synchronisation (voir section 1.3.3).

2.3.1 La bibliothèque IPC

La bibliothèque IPC (Pires, 1995) est un ensemble de procédures écrites en langage C offrant des services de transport de messages. Son but principal est de faciliter la création d'une machine virtuelle (voir section 1.3.2) sur un réseau. La bibliothèque utilise un processus secondaire, appelé démon, associé à chaque processus principal, pour assurer la communication entre ces derniers. Les démons communiquent entre eux avec le protocole TCP/IP (ISO, 1986; Comer et Stevens, 1993) de la bibliothèque BSD SOCKETS (Leffler *et al.*, 1989). Les processus principaux doivent honorer le modèle maître-esclaves (voir section 1.3.4) et être reliés entre eux par une topologie en étoile comme le montre la figure 2.3. Sur celle-ci, M représente le processus maître et E un des processus esclaves. On peut aussi observer dans l'agrandissement que chacun de ces processus se compose en fait de deux processus distincts : le processus principal (maître ou esclave) et le processus secondaire (démon).

Le modèle de transport de message implanté utilise le concept de *function hooks* [7] qui est un compromis entre le transport de messages conventionnel et l'appel de procédures à distance. Que ce soit avec l'un ou l'autre de ces modèles,

[7] Il n'existe pas de terme français approprié pour *function hooks*.

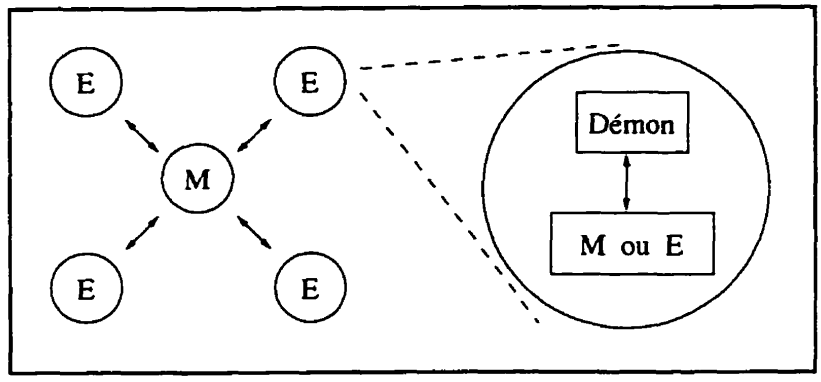


Figure 2.3 Topologie en étoile avec démons

le protocole de communication sous-jacent doit ultimement transmettre un message. La différence entre les *function hooks* et le transport de messages conventionnel réside dans le choix du module qui exécute le routage final du message, soit l'application ou la bibliothèque. Le transport de messages conventionnel laisse le travail à l'application alors que les *function hooks* permettent à celle-ci d'enregistrer une procédure pour chaque classe de message définie par l'application. À ce moment, le comportement de l'application n'est plus séquentiel mais bien orienté événement. La bibliothèque se compare alors à un gestionnaire de fenêtres. Quant à la différence entre les *function hooks* et l'appel de procédures à distance conventionnel, elle réside dans le fait qu'un appel à une procédure à distance ne bloquera pas l'appelant dans le cas des *function hooks*. En somme, le concept des *function hooks* est une extension aux interfaces fonctionnelles existantes afin de faciliter l'échange de services entre deux modules situés sur des machines ou des processus différents.

Si cette bibliothèque est utile pour simuler une machine virtuelle, elle ne peut toutefois pas profiter des architectures parallèles à mémoire partagée (voir section 1.3.2). Une bibliothèque supplémentaire, THREADS de SUN, est donc nécessaire.

2.3.2 La bibliothèque THREADS de SUN

Traditionnellement, les programmes parallèles étaient composés de plusieurs processus indépendants communicant ensemble par le biais d'un fichier spécial. Les positions dans ce fichier spécial correspondaient à des adresses dans la mémoire virtuelle de chacun des processus et simulaient en quelque sorte une mémoire partagée. Cette méthode avait un désavantage significatif : la mémoire partagée se retrouvait dans un fichier et était donc beaucoup plus lente à accéder. Des versions ultérieures ont par la suite éliminé la nécessité de passer par un fichier en utilisant directement la mémoire centrale. Toutefois, une difficulté est demeurée : une adresse dans la mémoire partagée doit correspondre à plusieurs adresses virtuelles, soit une par processus, ce qui en ralentit l'accès. La solution au problème est de n'avoir qu'un seul processus ayant plusieurs *fils* d'exécution (*threads*). Ces fils peuvent travailler indépendamment les uns des autres mais partagent la mémoire virtuelle du processus, ce qui élimine toute traduction d'adresses. Le couplage des fils aux processeurs ainsi que l'ajustement du niveau de parallélisme font intervenir des règles qui dépendent de la bibliothèque utilisée.

En attendant que la bibliothèque PTHREADS de POSIX (IEEE, 1995) ne devienne un véritable standard, il est préférable de se servir d'une bibliothèque qui a déjà acquis une certaine maturité, comme celle de la compagnie SUN (Sunsoft, 1994). Pour bien comprendre le modèle de parallélisme de SUN, il est nécessaire de définir un autre concept en plus des fils d'exécution définis auparavant : celui de processus léger. [8] Un *processus léger* est une entité intermédiaire entre les fils d'exécution et le processus UNIX. Ce dernier décide du nombre de processus légers ainsi que du nombre de fils d'exécution associés à chacun d'eux. Jusque là, le

[8] Processus léger est une traduction littérale de *lightweight process*.

système d'exploitation n'est pas entré en jeu, ce qui permet de faire des changements de contexte (passer d'un fil à un autre à l'intérieur d'un processus léger) très rapidement. Par la suite, ces processus légers sont eux-mêmes couplés aux processeurs par le système d'exploitation. Le nombre de processus légers permet donc de fixer le niveau maximal de parallélisme d'un processus. Cette architecture permet de concevoir efficacement des versions parallèles de deux types d'applications très différentes.

- Des applications du type *serveur de données* où des centaines de fils d'exécution travaillent sporadiquement sur un sous-ensemble des processeurs disponibles, i.e., le nombre de processus légers limite l'utilisation des ressources de la machine.
- Des *applications scientifiques* où chaque fil d'exécution est associé à un processeur par le biais d'un processus léger afin de maximiser l'utilisation de la machine.

La figure 2.4 présente le modèle de parallélisme discuté en illustrant les relations entre les fils d'exécution, les processus légers, les processus et les processeurs à différents niveaux : l'usager, le noyau (*kernel*) et le matériel.

2.4 Conclusion

Ce chapitre avait pour but de choisir les outils informatiques pour la suite des travaux. Tout d'abord, un logiciel d'optimisation générique pouvant résoudre le modèle unifié a été caractérisé. Ensuite, le choix de GENCOL comme implantation

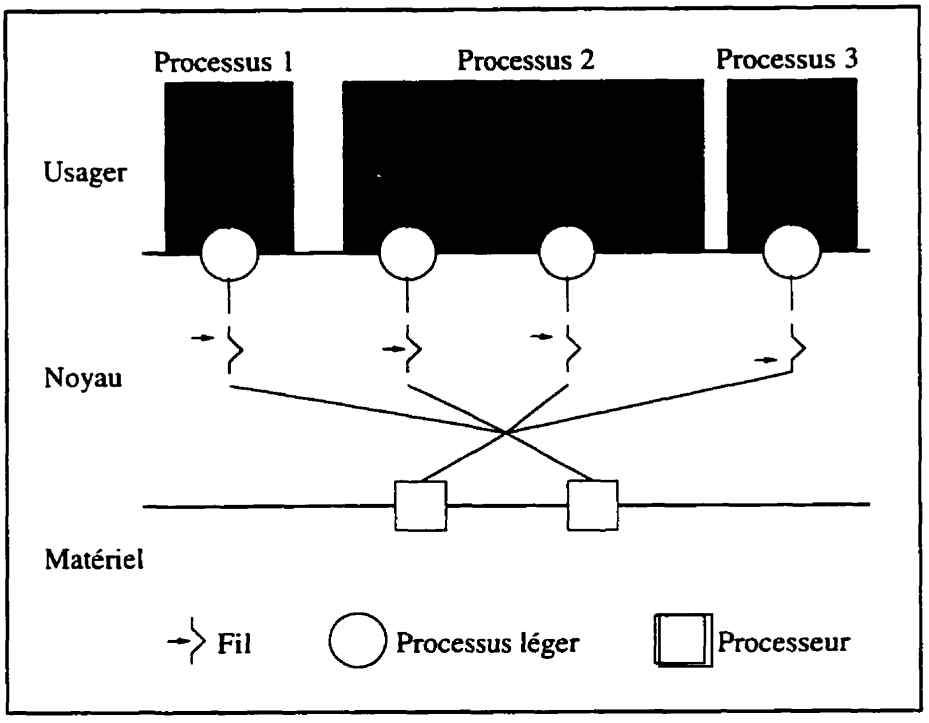


Figure 2.4 Modèle de parallélisme de la bibliothèque THREADS de SUN

séquentielle de référence a été justifié, GENCOL pouvant être considéré comme l'état de l'art dans le domaine tout en possédant une architecture modulaire qui correspond au logiciel générique. La problématique de l'évaluation de la puissance de calcul d'un parc de machines a ensuite été abordée. Cette discussion a conduit au choix du banc d'essai, soit une machine parallèle multiprocesseur à mémoire partagée. Finalement, la dernière section a défini deux outils informatiques supplémentaires : la bibliothèque de transport de messages IPC et la bibliothèque de mécanismes de synchronisation THREADS de SUN.

Maintenant que les contextes théoriques et pratiques ont été établis, il est possible de procéder à l'analyse du potentiel parallèle de l'optimiseur. Le chapitre suivant passe en revue les différents modules de l'optimiseur pour identifier ceux qui sont les plus aptes à profiter du parallélisme. Les choix et conclusions de cette

analyse aboutiront aux implantations informatiques des deux derniers chapitres.

CHAPITRE 3

Analyse du potentiel parallèle

Ce chapitre procède à l'analyse d'un optimiseur séquentiel pour le modèle unifié afin d'en identifier les modules pouvant bénéficier le plus du parallélisme. Cette analyse se veut générale, ce qui n'empêche pas de discuter de l'implantation spécifique qu'est GENCOL. Les modules choisis feront chacun l'objet d'implantations parallèles qui seront décrites dans les chapitres suivants.

Une *analyse préliminaire* de la bibliothèque GENCOL a été effectuée au préalable afin de limiter les recherches aux composantes les plus prometteuses. Cette analyse préliminaire est essentiellement constituée de communications privées (Lingaya, 1996) et de tests exécutés avec le logiciel QUANTIFY (PURE SOFTWARE, 1996). Le logiciel QUANTIFY permet de compiler des statistiques sur le profil d'exécution de programmes et d'en faire une présentation graphique facile à interpréter.

La figure 3.1 regroupe, par module, les composantes de la bibliothèque retenues pour être analysées dans ce chapitre. L'analyse est présentée dans l'ordre suivant : l'*arbre de branchement*, la *relaxation linéaire*, le *problème-maître* et le *sous-problème* correspondant respectivement aux modules SBB/BB, LB, MP et SP. La possibilité d'implantations parallèles au niveau des applications est évoquée dans la section 3.5.

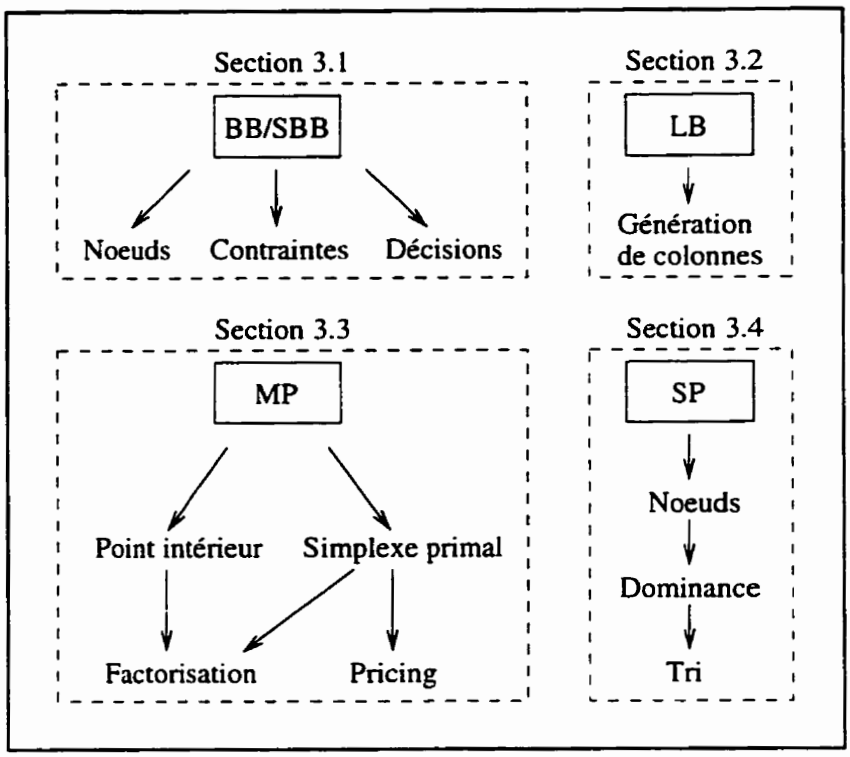


Figure 3.1 Composantes étudiées

3.1 Arbre de branchement

Le principe général d'un algorithme de branchement est de résoudre un problème relaxé, de vérifier que la solution de ce problème relaxé satisfait ou non l'ensemble des contraintes (y compris celles qui sont relaxées) et d'ajouter certaines des contraintes violées s'il y a lieu. Ce processus crée de nouveaux problèmes sur lesquels les étapes précédentes sont répétées. L'action centrale d'un algorithme de branchement consiste en l'*ajout de contraintes*, ce qui donne une direction à l'évolution du processus de résolution et permet d'éliminer sans les résoudre certains des problèmes créés en analysant les bornes inférieures fournies par la résolution d'autres problèmes. Afin de limiter l'espace mémoire requis pour la description des nouveaux problèmes non encore résolus, GENCOL conserve seulement les différences entre un nouveau problème

et son problème parent. L'algorithme de branchement comprend trois opérations (voir figure 3.1) pouvant potentiellement bénéficier du parallélisme : évaluation des décisions, élimination des contraintes redondantes et résolution des nœuds de branchement.

3.1.1 Évaluation des décisions

SBB correspond à la partie non générique du branchement, [9] conçue spécifiquement pour résoudre les problèmes de flot multi-commodités avec contraintes de ressource. C'est dans ce module que sont implantées les différentes méthodes de séparation et de coupes, avec les algorithmes associés qui manipulent directement les structures de données de SP et MP lors de l'ajout ou du retrait de contraintes. Il arrive parfois que certaines méthodes demandent un temps de calcul non négligeable, par exemple lors de la recherche de cycles dans un graphe. Toutefois, ces méthodes coûteuses sont spécialisées pour des classes de problèmes particulières, ce qui est en contradiction avec les objectifs du projet de recherche. Cette composante du logiciel n'est donc pas considérée.

3.1.2 Élimination des contraintes redondantes

Il existe des méthodes de branchement qui ajoutent des contraintes au niveau des sous-problèmes, ce qui ralentit leur résolution. Cependant, ces contraintes permettent parfois d'éliminer des nœuds et des arcs et de simplifier les réseaux. Cette simplification peut éventuellement rendre ces mêmes contraintes redondantes par

[9] Le module BB correspond à la partie générique et peut être utilisé en dehors du cadre du modèle unifié.

rapport à la structure des réseaux simplifiés. L'élimination des contraintes redondantes nécessite pour certaines classes de problèmes un temps de calcul pouvant même dépasser celui de la résolution du nœud de branchement. Toutefois, le temps de calcul demeurant faible dans le cas général, cette partie du logiciel ne sera pas retenue non plus.

3.1.3 Résolution des nœuds de branchement

Dans le cas de méthodes de séparation, la validité des contraintes ajoutées ne peut être prouvée et plusieurs ensembles de contraintes doivent être explorés, créant autant de nouveaux problèmes qui *peuvent être traités en parallèle.* Dans le cas de méthodes de coupes, les contraintes ajoutées sont valides et un seul nouveau problème doit être résolu.

Une structure arborescente s'exploite facilement en parallèle et, de ce fait, l'arbre de branchement pourrait s'avérer un candidat idéal pour une implantation parallèle sans difficultés majeures. Cependant, le choix des stratégies utilisées est grandement influencé par le type de machine parallèle ainsi que par la structure du problème traité, ce qui rend l'implantation parallèle moins robuste. L'article de Crainic et Gendron (1994) présente une revue de la littérature sur les algorithmes de branchement en parallèle.

Le traitement de l'arbre de branchement en parallèle n'élimine pas les difficultés rencontrées lors de la résolution séquentielle; au contraire, il peut même les amplifier. Ainsi, les deux aspects suivants prennent encore plus d'importance : choix d'un critère d'arrêt et de l'ordre d'exploration des nœuds de l'arbre. De plus, le parallélisme requiert des stratégies supplémentaires pour venir à bout de nouvelles

difficultés comme :

- la gestion de la liste des nœuds inexplorés;
- le degré de synchronisation de l'information;
- le volume de communication;
- la génération des premiers nœuds;
- la répartition de la charge de travail entre les processeurs.

Il faut aussi être en mesure d'évaluer la qualité et la performance d'une implantation parallèle. Plusieurs critères (voir section 1.4) ayant chacun leurs défauts sont disponibles :

- qualité (coût) de la solution;
- nombre de nœuds explorés;
- temps de calcul total;
- accélération par rapport à une référence;
- efficacité d'utilisation des processeurs ou de sélection des nœuds.

Les mesures d'efficacité demeurent les plus problématiques à cause de certains cas pathologiques (Corrêa et Ferreira, 1995) où l'utilisation intensive des processeurs peut donner lieu à des mesures d'efficacité élevée alors que le travail effectué est totalement inutile. Les mesures d'accélération suivent de près avec l'apparition d'anomalies (Li et Wah, 1986) : accélération inférieure à $1/p$ ou plus grande que p, où p est le nombre de processeurs. En somme, une implantation de qualité se doit

de *maximiser l'utilisation des processeurs* tout en *minimisant le nombre de nœuds explorés.*

Dans le cas des problèmes de grande taille dont traite ce mémoire (voir section 1.2), il est courant de faire appel à des méthodes de coupes heuristiques. Ces méthodes de coupes n'engendrent qu'un seul fils par nœud et empêchent ainsi tout parallélisme. Cette composante est donc aussi rejetée.

3.1.4 Conclusion

L'élimination des contraintes redondantes ainsi que l'évaluation des décisions de branchement possèdent toutes deux un certain potentiel parallèle qui est toutefois limité à des classes de problèmes particulières. La résolution des nœuds de branchement en parallèle requiert une structure arborescente qui n'est souvent pas présente dans les problèmes visés, empêchant ainsi tout parallélisme. Pour ces deux raisons, aucune des composantes du module de branchement n'est retenue.

L'effort de calcul pour résoudre un nœud de branchement consiste essentiellement à évaluer la borne inférieure de la relaxation linéaire du problème au nœud courant. La prochain section analyse le potentiel parallèle de la résolution de cette relaxation linéaire par l'approche de génération de colonnes.

3.2 Relaxation linéaire

Le module LB est responsable du calcul de la borne inférieure d'un problème relaxé provenant d'un branchement. LB résout ce problème relaxé par génération de colonnes (voir section 1.1). D'un point de vue macroscopique, le module LB décide dans quel ordre et combien de sous-problèmes sont résolus avant chaque réoptimisation du problème-maître. L'algorithme de génération de colonnes est la seule composante étudiée de LB (voir figure 3.1).

3.2.1 Génération de colonnes

LB comporte un algorithme de choix des sous-problèmes efficace mais simple qui n'impose pas de dépendance entre les sous-problèmes. LB permet donc la résolution parallèle des sous-problèmes, souvent nombreux, sans avoir à modifier la structure globale de l'algorithme.

L'algorithme parallèle proposé consiste à résoudre le problème-maître et les sous-problèmes de façon asynchrone. Le problème-maître rend disponibles aux sous-problèmes les nouvelles variables duales après chaque réoptimisation et les sous-problèmes font de même avec les colonnes générées pour le problème-maître. Cette approche est intuitive et convient parfaitement dans le cas général puisqu'elle ne dépend pas du branchement, du problème-maître ni de la structure des sous-problèmes.

L'analyse *a priori* du potentiel parallèle (voir section 1.5) de cette méthode est présentée au tableau 3.1. La granularité des tâches parallèles est déterminée par les sous-problèmes (et le problème-maître). Tel que mentionné précédemment,

Tableau 3.1 Potentiel parallèle de la génération de colonnes

Caractéristiques	
Granularité	sous-problèmes
Type de tâches parallèles	indépendantes
Volume de communication	faible
Proportion séquentielle	faible à moyenne
Type de comportement	non déterministe

l'algorithme de génération de colonnes n'impose pas d'ordre pour la résolution des sous-problèmes, ceux-ci sont donc considérés indépendants. L'information échangée entre les sous-problèmes et le problème-maître est composée de colonnes et de variables duales, ce qui devrait représenter un faible volume de communication. Quant à la proportion séquentielle, [10] elle varie selon le ratio du temps dans les sous-problèmes sur le temps dans le problème-maître. Pour les problèmes de grande taille traités dans ce mémoire, ce ratio est au-dessus de 1 mais décroit avec la taille des problèmes. Finalement, la nature asynchrone de l'algorithme parallèle proposé fait en sorte que son comportement n'est pas déterministe. Les temps de résolution peuvent fluctuer d'une exécution à l'autre ainsi que la qualité de la solution dans le cas de résolutions heuristiques.

3.2.2 Conclusion

Le module LB qui implante un algorithme de génération de colonnes semble être un bon candidat pour le parallélisme. Le chapitre 4 présente une implantation de cette méthode, nommée DG, et en évalue le potentiel réel. Une revue de la littérature sur le sujet débutera le chapitre.

[10] Le chapitre 4 fournira des explications plus détaillées sur la proportion séquentielle et fera le lien avec l'accélération maximale.

De la même façon que la relaxation linéaire représente l'essentiel du travail pour résoudre un nœud de branchement, l'effort de calcul consommé par un algorithme de génération de colonnes est concentré dans la résolution du problème-maître et des sous-problèmes. Les deux prochaines sections portent sur l'analyse du potentiel parallèle de ces deux algorithmes.

3.3 Problème-maître

MP constitue une *interface* entre un logiciel de résolution de programmes linéaires produisant des solutions de base (comme CPLEX) et les autres modules de GENCOL. Cette interface est nécessaire puisque CPLEX considère les variables du problème-maître comme des ensembles de coefficients, alors que les autres modules de GENCOL traitent ces variables comme des chemins possédant une structure particulière. En tant qu'interface, MP doit permettre aux autres modules d'effectuer toutes les opérations de construction et de modification des programmes linéaires nécessaires et maintenir la synchronisation avec les structures de CPLEX en conséquence. Toutefois, dans ce mémoire, il ne sera question que de l'optimiseur de programmes linéaires car c'est dans celui-ci qu'est consommé la presque totalité du temps de calcul du module MP.

Il y a deux principales catégories de logiciels de résolution de programmes linéaires : ceux basés sur les méthodes de point intérieur et ceux basés sur la méthode du simplexe. Les deux types d'algorithmes sont de nature très différente et exigent donc des analyses séparées.

3.3.1 Point intérieur

Le but du mémoire n'est pas de décrire ni de passer en revue toutes les méthodes de point intérieur. Il est suffisant de mentionner que les meilleures implantations actuelles sont du type prédicteur-correcteur et que la plupart des variantes ont toutes la même structure algorithmique, donc la même distribution du temps de calcul entre les différentes parties de l'algorithme. L'article de Lustig et Rothberg (1996) présente les développements les plus récents ainsi qu'une implantation considérée comme l'état de l'art dans ce domaine. Cette implantation a comme difficulté supplémentaire, inhérente à son aspect commercial, de traiter le cas général, celui-ci n'ayant pas, par définition, de structures spéciales à exploiter.

Une part importante de la recherche dans les méthodes de point intérieur est consacrée à la factorisation de Cholesky (Golub et Van Loan, 1989) d'un produit de matrices impliquant la matrice des contraintes du programme linéaire. Même si cette factorisation n'est effectuée qu'une seule fois par itération, elle sert souvent tout au long de celle-ci. Il est donc important de minimiser la densité de cette factorisation et de la conserver dans des structures de données qui facilitent un traitement rapide. Des algorithmes de factorisation symbolique minimisant le remplissage (*minimum degree ordering*) ainsi que des algorithmes d'agrégation (*supernode partition*) sont utilisés comme prétraitement. Une factorisation de Cholesky tirant profit des agrégations afin de minimiser le partage des données entre les processeurs est ensuite appliquée.

La factorisation de Cholesky étant l'opération la plus coûteuse d'une itération de point intérieur, il n'est pas étonnant que la plupart des auteurs se contentent

de paralléliser celle-ci. Cependant, Lustig et Rothberg sont plus agressifs et identifient d'autres points d'intérêt comme le test du ratio pour déterminer le pas du prédicteur et du correcteur. Cette partie de l'algorithme n'avait pas fait l'objet de recherches auparavant car le grain de parallélisme [11] y était et y est encore trop faible. Toutefois, des améliorations récentes dans le domaine des compilateurs permettent maintenant de générer du code machine capable de répartir dynamiquement (durant l'exécution) la charge de travail sur chaque processeur. Cette technique est mise à profit par Lustig et Rothberg lors de la formation de certaines matrices nécessitant des produits. En effet, le code machine généré est à même de distribuer les variables aux processeurs selon la demande puisque chaque variable de la matrice peut être calculée de façon indépendante. À l'inverse, dans certains cas, comme dans la résolution de systèmes linéaires avec des matrices triangulaires (*forward and backward substitutions*), il y a peu de parallélisme à exploiter et la recherche d'une structure parallèle durant la résolution peut alors devenir plus coûteuse que le gain qu'elle apporte. Dans ces cas-là, une allocation statique est effectuée.

3.3.2 Simplexe primal

L'algorithme du simplexe peut se résumer pour les fins de ce mémoire et sans perte de généralité aux deux opérations suivantes.

Choix du pivot : Pour déterminer le pivot, il faut sélectionner tout d'abord, parmi les variables hors base, une variable susceptible d'améliorer la solution et déterminer par la suite la variable bloquante qui devra quitter la base. Si peu de recherches ont été faites sur la possibilité de paralléliser la dernière opération, il n'en est pas

[11] Le grain de parallélisme est proportionnel à la taille des tâches parallèles.

de même pour la première. En effet, le choix de la variable entrante implique une opération, le *pricing*, sur toutes les variables hors-base afin d'en sélectionner la meilleure. Ce *pricing* est indépendant de l'ordre dans lequel les variables sont traitées et il est donc possible de l'effectuer en parallèle. Cependant la recherche s'est poursuivie au niveau des méthodes de *pricing* donnant lieu à des heuristiques efficaces comme celle de Devex (Harris, 1973) ainsi qu'à des façons rapides de mettre à jour les coûts de la méthode de descente maximale [12] (Goldfarb et Reid, 1977) d'une itération à l'autre sans avoir à recalculer toutes les normes. De plus, il est courant de ne traiter qu'un petit sous-ensemble des variables hors-base à la fois (*partial pricing*, voir Bazaraa, Jarvis et Sherali, 1990). Ces dernières améliorations diminuent le caractère indépendant de l'algorithme de *pricing* au point d'en annuler le potentiel parallèle. Il n'est plus évident d'obtenir des résultats en parallèle aussi bons que ne le laissait présager la méthode au départ.

Changement de base : La plupart des implantations contemporaines (Bartels et Golub, 1969; Nazareth, 1987; Bazaraa, Jarvis et Sherali, 1990; Suhl et Suhl 1990) conservent la base courante sous forme d'une décomposition LU (Golub et Van Loan, 1989). Cette méthode a l'avantage de garder une représentation moins dense de la base que ne le ferait son inverse ou encore la méthode PFI (*product form of the inverse*). Ainsi, une implantation parallèle à ce niveau revient à concevoir un algorithme parallèle de factorisation LU. Plusieurs articles présentent des méthodes parallèles pour effectuer cette factorisation dans le cas creux ou dense. Les classes de problèmes visées dans ce mémoire donnent lieu à des matrices creuses. Si les temps de calcul sont satisfaisants pour les matrices denses (Deprez, Dongarra et Tourancheau, 1995), il n'en est pas de même pour les matrices creuses. En effet, une telle implantation parallèle suppose qu'on garde un contrôle sur la stabilité

[12] La méthode de descente maximale est souvent la plus efficace pour les problèmes de grande taille.

numérique et l'ajout de coefficients (*fill-in*) tout en conservant un grain de parallélisme adéquat. De plus, des méthodes de mise à jour permettent d'utiliser la même factorisation durant plusieurs itérations. Ces méthodes sont très rapides et offrent donc peu de potentiel parallèle.

3.3.3 Conclusion

Les méthodes de point intérieur présentent un bon potentiel parallèle et des implantations récentes le prouvent. Toutefois, ces méthodes ne tirent pas profit d'une solution existante afin d'accélérer la réoptimisation du problème après l'ajout de colonnes. Quant à l'algorithme du simplexe primal classique, il offre un certain potentiel parallèle, possiblement plus faible que les méthodes de point intérieur, et qui est amoindri en pratique par l'utilisation de techniques heuristiques. À l'heure actuelle, l'implantation efficace d'un algorithme de simplexe primal parallèle demeure une question ouverte.

Même si ce n'est pas un gage de succès pour l'implantation de la version parallèle, il est avantageux d'avoir sous la main une implantation séquentielle de bonne qualité. Comme une version séquentielle (ex. le code de CPLEX) n'est pas disponible et que sa conception exigerait un temps considérable, cette partie du logiciel ne sera pas retenue pour la phase de développement. C'est d'autant plus dommage que la tendance actuelle favorise un accroissement plus rapide du temps de calcul du problème-maître par rapport au temps de calcul des sous-problèmes quand la taille des problèmes augmente.

Le problème-maître ne constitue toutefois que la moitié de l'algorithme de génération de colonnes. La section suivante analyse l'autre moitié de l'algorithme, soit

le sous-problème.

3.4 Sous-problème

LB est conçu pour gérer le processus de recherche d'une borne inférieure sans savoir quels sont les algorithmes utilisés pour résoudre les problèmes de plus court chemin découlant des sous-problèmes. Dans GENCOL, ces algorithmes constituent des modules séparés partageant une interface commune, SP. Ceci permet d'implanter, dans d'autres projets, des algorithmes spécialisés pour des types particuliers de sous-problèmes. L'algorithme de plus court chemin avec contraintes de ressources le plus utilisé est SPA, spécialisé pour les graphes acycliques. D'autres algorithmes peuvent aussi être implantés, comme par exemple le module SPNC (Ioachim, 1994) traitant des sous-problèmes avec coûts dépendant linéairement de la valeur d'une ressource, et le module SPC (Lingaya, 1996) traitant les mêmes types de sous-problèmes que SPA en admettant les graphes avec cycles. De manière générale, les divers algorithmes de plus court chemin avec contraintes de ressources comprennent trois opérations pouvant potentiellement bénéficier du parallélisme : le traitement des nœuds du réseau, la dominance entre étiquettes et le tri des étiquettes.

3.4.1 Traitement des nœuds du réseau

Même si les problèmes résolus sont plus complexes, les divers algorithmes de plus court chemin avec contraintes de ressources utilisés ont des points en commun avec l'algorithme de Dijkstra (1959) qui résout le problème de plus court chemin classique. Par conséquent, il est pertinent de profiter des travaux publiés précédemment

sur la résolution en parallèle du plus court chemin classique. De ces travaux est tirée l'idée générale suivante : il faut traiter plusieurs nœuds du réseau en parallèle. Le traitement d'un nœud est principalement constitué de deux étapes, soit la prolongation des étiquettes (fonctions d'extension) et la dominance entre étiquettes. L'architecture modulaire du logiciel et le niveau d'abstraction suffisamment bas de la composante visée permettent un isolement des mécanismes parallèles et donnent toutes libertés pour développer des structures de données adaptées.

Tableau 3.2 Potentiel parallèle du traitement des nœuds

Caractéristiques	
Granularité	noeuds du réseau
Type de tâches parallèles	dépendantes
Volume de communication	faible
Proportion séquentielle	faible à moyenne
Type de comportement	déterministe

L'analyse *a priori* du potentiel parallèle (voir section 1.5) de cette méthode est présentée au tableau 3.2. La granularité des tâches parallèles est déterminée par les nœuds du réseau. Les nœuds étant reliés entre eux par des arcs que doivent traverser les étiquettes, ils constituent donc des tâches dépendantes. Il n'y a pas de véritable communication entre les processeurs chargés de traiter les nœuds. Par contre, leur coordination nécessite l'usage de mécanismes de synchronisation qui peuvent être perçus comme une forme de communication entre processeurs. Celle-ci devrait toutefois être faible. La proportion séquentielle est reliée au ratio du temps dans les sous-problèmes sur le temps dans le problème-maître. Pour les problèmes de grande taille traités dans ce mémoire, ce ratio est au-dessus de 1 mais décroit avec la taille des problèmes. La nature déterministe de l'algorithme sera démontrée au chapitre 5.

Le chapitre 5 présente une implantation de cette méthode, nommée STA, et en

évalue le potentiel réel. Une revue de la littérature sur le sujet et une définition plus formelle du problème débutera le chapitre.

Les deux prochaines sections abordent les détails de l'algorithme de plus court chemin avec contraintes de ressources implanté dans GENCOL. Vue la nature de ces deux sections et leur relation directe avec GENCOL, les conclusions qui en sont tirées sont spécifiques à GENCOL et ne se généralisent probablement pas.

3.4.2 Dominance entre étiquettes

Les algorithmes de dominance entre les étiquettes, utilisés dans toutes les versions de GENCOL, sont la principale source du succès empirique de la résolution du modèle de flot multi-commodités avec contraintes de ressource. La proportion du temps total d'exécution écoulé dans les fonctions de dominance dépasse couramment 25% et peut même atteindre 50%. Ces considérations semblent suffisantes pour justifier le développement de nouveaux algorithmes de dominance parallèles. Toutefois, le module SPA [13] comporte des fonctions de dominance spécialisées, dépendant du nombre de ressources considérées, le coût étant ici traité comme une autre ressource. Certaines sur-spécialisations améliorent encore la performance de ces fonctions en tenant compte du patron des ressources choisies. Par contre, dans le cas où l'application manipule des classes d'étiquettes non comparables, la dominance doit être effectuée non plus sur la liste complète des étiquettes mais plutôt sur des sous-ensembles indépendants de ces mêmes étiquettes, ce qui pourrait alors se faire en parallèle. Cependant, cette situation est un cas particulier et ne peut justifier à elle seule une approche parallèle. Cette partie du module est donc rejetée.

[13] Le module SPA est un optimiseur pour des problèmes de plus court chemin avec contraintes de ressources sur des graphes acycliques.

3.4.3 Tri des étiquettes

Exception faite des vecteurs à une dimension, tous les autres algorithmes de dominance commencent par trier les étiquettes en ordre lexicographique. Ce tri consomme la majeure partie du temps de calcul dans les algorithmes de dominance traitant des vecteurs de dimension 2 ou 3 et semble donc un meilleur choix que la dominance elle-même. L'algorithme de tri de la version séquentielle est spécialisé en fonction des structures de données utilisées et tire profit de la distribution non uniforme des étiquettes à un nœud donné, ce qui n'est pas évident à exploiter en parallèle. Si la littérature foissonne d'articles sur les algorithmes de tri en parallèle, ceux-ci requièrent souvent des architectures spécialisées, ne traitent que les cas théoriques et sont appliqués sur des structures de données très simples, comme les tableaux (Jájá, 1992; Kumar *et al.*, 1994; Foster, 1995). Les difficultés de conception d'un algorithme parallèle efficace de tri sur des listes chaînées, possiblement petites et nombreuses, couplées à une forte composante séquentielle (la prolongation des étiquettes) font en sorte que cette partie du module est aussi rejeté.

3.4.4 Conclusion

Le traitement des nœuds dans l'algorithme de plus court chemin avec contraintes de ressources semble démontrer un bon protentiel parallèle. Le niveau d'abstraction de l'algorithme permet un isolement des mécanismes parallèles et facilite son implantation informatique. Les opérations de dominance entre les étiquettes et le tri des étiquettes possèdent un faible potentiel parallèle, étant déjà sur-spécialisées en fonction du patron des ressources choisies et de la distribution non uniforme des

étiquettes. Le maintien de ces optimisations dans une éventuelle implantation parallèle est primordial afin d'assurer un minimum de performance, le potentiel parallèle étant déjà faible. Toutefois, leur conception pose des difficultés et fait en sorte que ces deux composantes ne sont pas retenues.

De la même manière que GENCOL se sert de CPLEX, il existe des applications qui utilisent GENCOL comme une bibliothèque de procédures. Une étude complète se doit donc d'analyser aussi le potentiel parallèle de ces modules externes. Évidemment, l'aspect spécialisé des applications fait en sorte qu'aucune des idées proposées ne sera retenue. Ces idées peuvent toutefois faire l'objet de travaux subséquents.

3.5 Application

S'il n'a été question jusqu'à présent que des modules internes de GENCOL, cela ne signifie pas que toutes les options ont été explorées. Des données supplémentaires provenant de l'application bâtie à partir de GENCOL, celle-ci étant vue ici comme un module externe, peuvent être mises à contribution. En effet, pour une application donnée, il est possible de tirer profit d'informations spécifiques à la classe de problèmes traités et d'ouvrir ainsi la voie à de nouvelles possibilités de parallélisme. Les trois stratégies présentées ci-après sont spécifiques à certaines classes de problèmes et ne seront donc pas retenues pour fins d'implantation.

3.5.1 Problèmes indépendants

À moins de traiter des cas vraiment très particuliers, il est fréquent en pratique d'avoir à résoudre un ensemble de problèmes. De plus, l'utilisateur ou le client possède souvent plus d'une machine. Il suffit alors d'un langage de programmation simple, genre *script*, pour automatiser le processus en répartissant la résolution des problèmes sur différentes machines. Un ratio suffisant de problèmes par machine et une distribution le moindrement uniforme du temps de résolution des problèmes permet d'obtenir des résultats plus que concurrentiels par rapport à des logiciels intrinsèquement parallèles.

3.5.2 Résolution par fenêtre glissante

Certains problèmes ne peuvent être résolus globalement et il faut alors les traiter avec une méthode dite de fenêtre glissante. Dans les problèmes de construction des rotations d'équipage, un cas typique est celui de traiter une période d'un mois avec des fenêtres de sept jours et un pas de quatre jours, pour assurer un minimum de 3 jours de recouvrement d'une tranche à l'autre. De plus, il n'est pas rare d'améliorer la qualité de la solution en couvrant la période d'un mois deux ou trois fois. Puisque cette méthode est heuristique au départ, rien n'empêche de résoudre plus d'une tranche à la fois en autant qu'elles soient toutes disjointes entre elles. Il est donc possible, comme dans le cas précédent, de concevoir une version parallèle efficace avec un minimum d'effort.

3.5.3 Nouvelles modélisations

La dernière proposition concerne le niveau d'abstraction le plus élevé possible : la modélisation elle-même. Il est tout à fait concevable qu'on puisse modéliser une classe de problèmes de plusieurs façons mathématiquement équivalentes et que certaines se parallélisent plus facilement que d'autres. Il est important, à ce moment-ci, de rappeler que les travaux des quinze dernières années portent sur des modèles ainsi que sur des implantations informatiques à caractère principalement séquentiel. Il apparaît de plus en plus évident que l'implantation parallèle des méthodes de résolution nécessite en général des changements radicaux à ces méthodes. L'auteur fait la conjecture que cet argument s'étend aussi à la conception des modèles.

3.6 Conclusion

Ce chapitre a présenté l'analyse d'un optimiseur pour le modèle unifié afin d'en identifier les modules pouvant bénéficier le plus du parallélisme. Le module du branchement a été rejeté car la structure arborescente n'est souvent pas présente dans les problèmes visés. Le module de la relaxation linéaire a été retenu et fera l'objet d'une implantation parallèle, nommée DG. L'algorithme parallèle proposé consiste à résoudre le problème-maître et les sous-problèmes de façon asynchrone. Le module du problème-maître a été rejeté pour des raisons d'ordre pratique, une version séquentielle de qualité n'étant pas disponible. Finalement, le module du sous-problème a été retenu et fera l'objet d'une implantation parallèle, nommée STA. La méthode proposée considère le traitement de plusieurs nœuds du réseau en parallèle.

Rappelons au lecteur que l'évaluation du potentiel parallèle des deux méthodes proposées constitue une prédiction qui sera réévaluée à la suite de leur implantation informatique respective. Le chapitre suivant procède à la description de l'implantation de DG.

CHAPITRE 4

Résolution de la relaxation linéaire en parallèle

La relaxation linéaire d'un problème provenant de la décomposition de Dantzig-Wolfe peut être résolue par génération de colonnes. Dans le processus de génération de colonnes, le choix de l'ordre de résolution des sous-problèmes n'a pas d'incidence sur le résultat final. Cette caractéristique peut être exploitée avec le parallélisme en résolvant le problème-maître et les sous-problèmes de façon asynchrone. Ce chapitre décrit l'implantation informatique d'une telle méthode.

La première section passe en revue la littérature sur les méthodes de résolution parallèles semblables. La deuxième section décrit en détail la méthode proposée et ses spécifications sans toutefois entrer en profondeur dans les détails informatiques. La troisième section présente des résultats numériques, les généralise à d'autres classes de problèmes, propose des méthodes d'accélération et conclut sur l'efficacité de la méthode en pratique.

4.1 Revue de la littérature

La génération de colonnes en parallèle n'a pas suscité beaucoup de travaux de recherche jusqu'à maintenant. Les énergies sont plutôt concentrées sur de nouvelles variantes des méthodes de point intérieur ou sur des heuristiques spécialisées. Les auteurs offrent souvent en guise de conclusion des ébauches d'algorithmes parallèles dérivés de leurs travaux mais fournissent rarement des résultats numériques.

Certains auteurs (Arthur, Frendewey, Schumichrast, 1987; Beasley, 1987; Ho, Lee, Sundarraj, 1988) n'expérimentent que sur des machines parallèles spécialisées : ordinateurs vectoriels, *mainframe*, [14] etc. Le succès de leurs résultats est plus ou moins relié aux avantages qu'ils peuvent soutirer de l'architecture spécialisée de leur banc d'essai. De plus, ces résultats datent déjà d'une décennie, ce qui correspond à plusieurs générations d'ordinateurs. D'ailleurs, Ho, Lee et Sundarraj attribuent une partie de leur succès au fait que les ordinateurs de l'époque ne pouvaient contenir les problèmes étudiés dans leur mémoire centrale alors que la machine virtuelle le pouvait, les sous-problèmes étant répartis sur celle-ci. De nos jours, ce genre de gain est plutôt rare. Il faut dire aussi que seul l'article de Ho, Lee et Sundarraj porte véritablement sur la génération de colonnes en parallèle. Les deux autres sont plutôt axés sur la résolution en parallèle de programmes linéaires de grande taille. Des travaux plus récents et directement reliés à la génération de colonnes (Chraim, 1994; Chraim et Sansó, 1995) existent mais ils portent sur des classes de problèmes spécifiques et la méthodologie d'évaluation diffère de celle utilisée dans ce mémoire.

En somme, l'idée proposée ici n'est pas nouvelle, c'est plutôt le contexte dans lequel elle est traitée qui diffère. D'abord, la nature même des sous-problèmes découlant de la décomposition de Dantzig-Wolfe du modèle unifié est très différente de ceux provenant de programmes linéaires plus conventionnels. Les algorithmes séquentiels actuels peuvent tirer profit du fait que les sous-problèmes permettent d'envoyer plusieurs colonnes à la fois au problème-maître. La nécessité de résoudre un grand nombre de sous-problèmes n'est pas évidente puisqu'une information équivalente peut être obtenue de cette façon à moindre coût. D'ailleurs, ce ne sont pas toutes les classes de problèmes qui comportent un grand nombre de sous-problèmes, et ce, même pour les problèmes de grande taille. D'autre part,

[14] Les *mainframes* sont des ordinateurs puissants occupant une grande superficie et vendus à quelques exemplaires dans le monde.

la méthodologie (voir section 1.4) qui prévaut dans ce mémoire exige que les algorithmes proposés soient comparés à l'état de l'art dans le domaine. Ainsi, la version séquentielle ne sera pas handicapée de ses stratégies d'accélération ni des heuristiques pouvant provenir du niveau des applications. Cette approche rigoureuse permet d'évaluer le véritable potentiel de la méthode en la validant dans le cadre d'applications commerciales.

4.2 Méthode et spécifications

Cette section a pour but de décrire la méthode proposée ainsi que les spécifications à respecter sans pour autant entrer dans les détails de l'implantation informatique. Pour ce faire, la première sous-section présente la méthode ainsi que l'architecture informatique sous-jacente. La deuxième sous-section propose une extension à la méthode afin de prendre en compte les contraintes de branchement. L'extension proposée consiste à gérer l'arbre de branchement sous la forme d'une pile de décisions élémentaires. La troisième sous-section conçoit une stratégie d'allocation efficace des sous-problèmes dans le but de réduire les délais de communications. La dernière sous-section adapte les stratégies heuristiques de la version séquentielle au contexte parallèle.

4.2.1 Description de la méthode

L'idée de base de la méthode proposée consiste à *résoudre le problème-maître et les sous-problèmes de façon asynchrone*. Le problème-maître rend disponibles aux

sous-problèmes les nouvelles variables duales après chaque réoptimisation et les sous-problèmes font de même pour le problème-maître avec les colonnes générées. C'est une stratégie de force brute destinée à tirer avantage d'un parc de machines.

Le modèle de gestion des tâches parallèles qui convient à la méthode proposée est le modèle maître-esclaves (voir section 1.3.4). Le processus maître est responsable de la résolution du problème-maître alors que les esclaves s'occupent de la résolution des sous-problèmes. Cette façon de faire nécessite une bibliothèque de procédures de communication pour assurer le lien entre le problème-maître et les sous-problèmes. La bibliothèque de procédures IPC (voir section 2.3.1) a été conçue spécifiquement pour assumer cette tâche. En effet, la topologie en étoile de la bibliothèque IPC est adaptée à ce cas particulier.

L'ordre de résolution des sous-problèmes n'étant pas connu *a priori*, la méthode proposée n'est pas déterministe. Par conséquent, les temps d'exécution ainsi que la qualité des solutions (dans le cas de résolutions heuristiques) peuvent fluctuer, ce qui complique les méthodes d'évaluation. Toutefois, ni la convergence ni l'exactitude d'un algorithme de génération de colonnes ne dépendent de l'ordre de résolution des sous-problèmes. La version parallèle demeure donc une méthode exacte au même titre que la version séquentielle et sa convergence est aussi garantie.

Il faut cependant apporter quelques précisions sur cette conclusion. En effet, il est possible dans la version parallèle de ne générer aucune colonne de coût réduit négatif pour chacun des sous-problèmes à une itération donnée sans pour autant atteindre l'optimalité. Pour s'assurer de l'optimalité, il faut aussi que les sous-problèmes aient tous été résolus avec l'ensemble des variables duales les plus récentes. Soulignons que dans le cadre de ce chapitre, une itération de génération de colonnes commence à la fin d'une optimisation du problème-maître et se termine à la fin de

l'optimisation suivante.

Dans le but d'accroître la polyvalence de la méthode proposée, l'architecture matérielle tout indiquée est la machine virtuelle (voir section 1.3.2). Il va sans dire que la taille du parc de machines ainsi que la vitesse et le temps de réponse du réseau les reliant influencent directement la qualité des résultats. Il n'y a pas de restrictions sur le type de chacune des machines.

Ayant défini la méthode tant au niveau logiciel que matériel, il est maintenant possible de quantifier l'accélération maximale $A_{\max}(\cdot)$, qu'elle peut apporter à la résolution d'un problème donné. Rappelons que le paramètre N représente la taille du problème et le paramètre p, le nombre de processeurs utilisés. En théorie, chaque exécution de l'algorithme parallèle peut donner lieu au parcours d'un chemin différent dans l'espace des solutions admissibles. Toutefois, le temps T_{mp} passé à résoudre le problème-maître est incompressible. La résolution du problème-maître peut donc être considéré comme une composante *séquentielle* même si elle est exécutée en parallèle avec la résolution des sous-problèmes. De plus, le temps consommé par les autres composantes du logiciel, comme le branchement, est négligeable la plupart du temps. Ceci permet une application directe de la loi d'Amdhal (voir section 1.4) et donne lieu à la borne suivante :

$$A_{\max}(N,p) = \frac{T_{\text{sp}}(N) + T_{\text{mp}}(N)}{T_{\text{mp}}(N)}. \tag{4.1}$$

Cette borne est une approximation, acceptable en pratique, de l'accélération maximale globale. Dans l'équation, T_{sp} mesure le temps passé dans les sous-problèmes.

Il n'y a pas de métrique permettant d'évaluer directement l'accélération maximale locale mais les trois métriques suivantes pourraient s'avérer de bons indicateurs : nombre total de sous-problèmes résolus (# SP), nombre total de colonnes

générées et pourcentage d'utilisation des processeurs. Rappelons au lecteur que le parc de machines est homogène (voir section 2.2).

Même s'il est difficile d'évaluer de façon théorique les accélérations locale et globale maximales d'un problème donné, il est quand même possible d'énumérer les caractéristiques d'un problème qui devraient avoir un impact positif sur l'accélération maximale :

- nombre élevé de sous-problèmes;
- sous-problèmes comportant un temps de calcul élevé;
- problème-maître comportant un temps de calcul faible.

En d'autres mots, le potentiel parallèle des classes de problèmes est déterminé par le ratio $T_{\mathrm{sp}}(N)/T_{\mathrm{mp}}(N)$, où un ratio élevé implique un potentiel élevé. De plus, les sous-problèmes étant exécutés en même temps, leur nombre devra être de $p-1$ au minimum.

4.2.2 Gestion des contraintes de branchement

Le problème traité est une relaxation du problème original en nombres entiers découlant du modèle unifié. Un processus de séparation et d'évaluation progressive est nécessaire à l'obtention d'une solution entière. Toutefois, la méthode définie jusqu'à présent ne tient pas compte de ce processus. Une extension à la méthode qui soit indépendante des classes de problèmes et des méthodes de branchement est donc nécessaire.

Tout d'abord, la dépendance envers les méthodes de branchement peut être éliminée à l'aide du raisonnement suivant. Chaque méthode de branchement doit produire un ensemble de décisions compatibles avec la structure des sous-problèmes et du problème-maître. L'application de ces décisions consiste à modifier le problème en manipulant des *structures élémentaires* : nœuds, arcs, etc. À ce niveau d'abstraction, il n'est plus nécessaire de savoir d'où proviennent les décisions, i.e., de connaître la méthode de branchement.

Le processus maître est responsable de la gestion du problème initial, il lui est donc nécessaire de maintenir une version détaillée de l'arbre de branchement. En contrepartie, les esclaves ont une vision plus locale du problème. Comme l'illustre la figure 4.1(a), il faut, pour aller du nœud 3 au nœud 4, défaire les décisions des nœuds 3 et 1 et appliquer celles des nœuds 2 et 4. En général, il faut monter dans l'arbre d'un nombre de nœuds possiblement nul et y descendre d'au moins un nœud. Il est ainsi possible de faire abstraction de la structure en arbre et de représenter les déplacements chez les esclaves sous la forme d'une pile comme le montre la figure 4.1(b). On peut simplifier les transactions maître-esclaves en mémorisant chez les esclaves les décisions intermédiaires de sorte qu'une ascension dans l'arbre puisse être décrite par une simple valeur entière représentant le nombre de niveaux à remonter.

En conclusion, la résolution en nombres entiers est possible en simulant un arbre de branchement sous la forme d'une pile sur chacun des esclaves. Seules les décisions élémentaires sont transmises et les transactions peuvent être simplifiées en mémorisant les décisions intermédiaires.

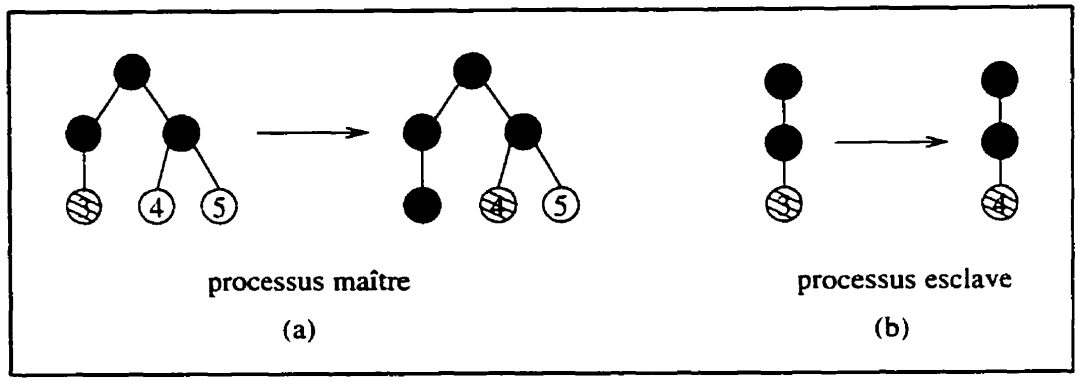


Figure 4.1 Gestion de l'arbre de branchement

4.2.3 Réduction des délais de communication

Comme le montre la figure 4.2(a), les requêtes de résolution de sous-problèmes (lignes pleines [15]) ainsi que la réception des colonnes générées (lignes pointillées) entraînent des délais de communication, même avec une approche asynchrone. Ces délais de communication peuvent entraîner une sous-utilisation de la puissance de calcul disponible. Les périodes d'inactivité des machines sont représentées par des zones grises alors que la résolution du problème-maître et des sous-problèmes sont représentées par MP et SP respectivement. Il faut admettre qu'il est physiquement impossible de transmettre des ordres aux esclaves de manière instantanée. Toutefois, le mécanisme de sélection des sous-problèmes possède une certaine marge de manœuvre qu'il est possible d'exploiter.

Le mécanisme de sélection des sous-problèmes de la version séquentielle est biaisé de façon à favoriser la couverture complète des sous-problèmes lors d'une séquence d'itérations de génération de colonnes. Ceci correspond dans la version parallèle à sélectionner les sous-problèmes ayant été résolus avec les ensembles de variables duales les moins récentes. En contrepartie, il n'est plus possible de procéder

[15] Les lignes pleines incluent aussi, dans certains cas, l'envoi des variables duales mais cette action n'a pas d'impact sur la discussion en cours.

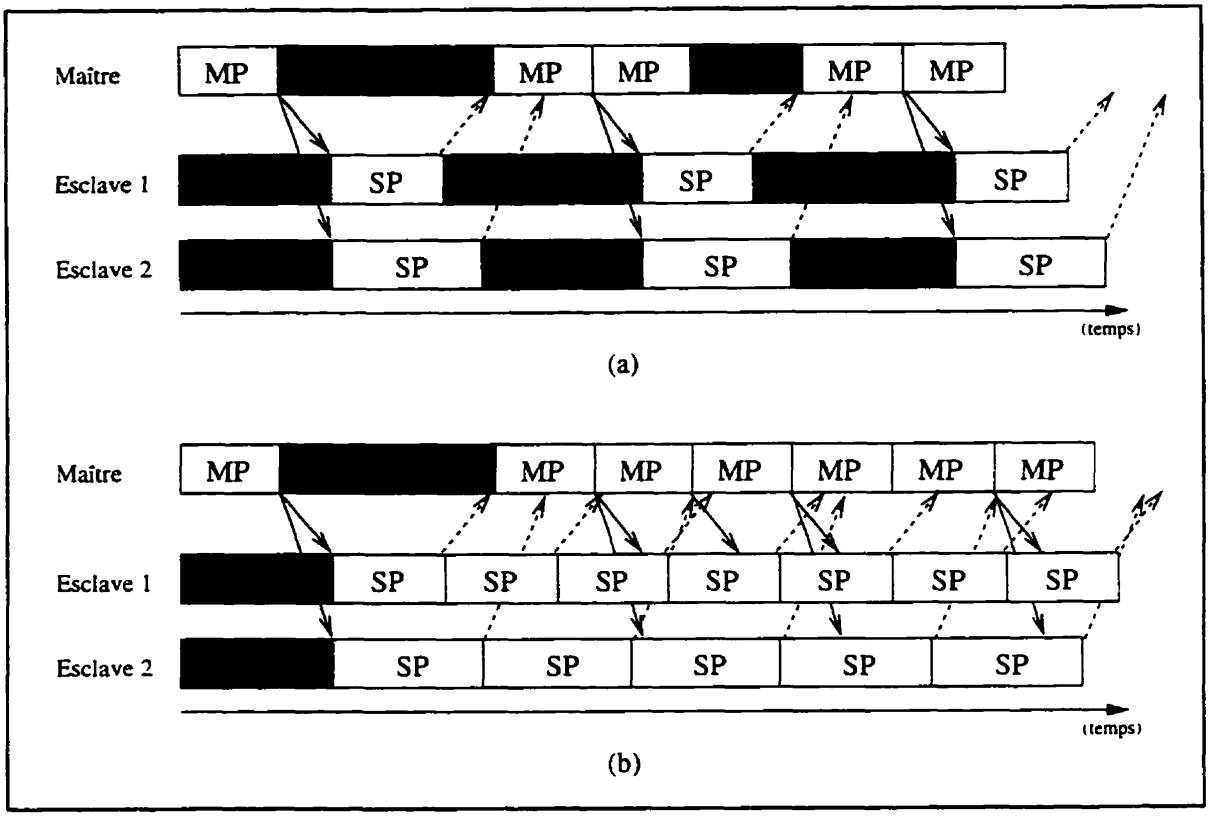


Figure 4.2 Gestion des délais

à la distribution des sous-problèmes avec une stratégie de type *pulling* où les esclaves décident eux-mêmes quels sous-problèmes traiter, stratégie qui tend à minimiser les délais de communication. Il faut alors se contenter d'un algorithme de type *pushing* où le maître décide de façon unilatérale de la distribution des sous-problèmes. Mais tout compte fait, cette stratégie cadre mieux avec le modèle maître-esclaves.

Malgré cette centralisation de la distribution des sous-problèmes, il demeure possible de diminuer l'impact des délais, voire même de les éliminer, en maintenant une pile de sous-problèmes à résoudre sur chacun des esclaves. Lorsqu'un esclave a résolu un sous-problème, il envoie les colonnes générées et commence immédiatement la résolution du prochain sous-problème. De même, le maître s'assure d'envoyer de nouveaux sous-problèmes aux esclaves dont la pile est presque vide (figure 4.2(b)).

4.2.4 Adaptation des heuristiques

Il existe, dans la version séquentielle, une heuristique qui permet de générer des colonnes complémentaires lors d'une itération. Cette heuristique est couramment employée lors la recherche d'une solution réalisable. Son effet ultime est d'éliminer un sous-ensemble de nœuds et d'arcs du réseau d'un sous-problème donné en fonction de la solution des sous-problèmes précédents à la même itération. Son utilisation crée une dépendance entre les sous-problèmes, impliquant $A_{\max}(N, p) \leq 1$ pour *toute* valeur de p. Une façon de circonvenir cette difficulté est d'effectuer l'heuristique localement sur chacun des esclaves. Évidemment, implantée de cette façon, la version parallèle de l'heuristique sera moins efficace car elle perdra sa vision globale du processus de résolution.

4.3 Résultats expérimentaux

Contrairement à ce que laissait présager l'analyse, l'implantation informatique de la méthode proposée, nommée DG, entraîne des modifications substantielles dans l'utilisation de GENCOL par les applications. DG complique et fige le protocole d'initialisation de GENCOL, créant des *conflits* avec les applications existantes. Ce type de complications augmente aussi le niveau de couplage [16] (Schach, 1993) entre GENCOL et les applications.

Le mécanisme de synchronisation de l'information requis par DG restreint les manipulations directes des structures de données de GENCOL par les applications. Ces manipulations sont *cruciales* dans le cas des problèmes de distribution des rotations au personnel avec séniorité stricte pour pouvoir résoudre les pilotes séquentiellement sans devoir redémarrer l'exécutable.

Il est cependant possible de prendre une application donnée, en l'occurrence l'exécutable GENCOL, et d'en faire une implantation parallèle. Cette dernière sera en mesure de traiter la plupart des problèmes de recherche, dont les problèmes de tournées de véhicule (Desrosiers *et al.*, 1995; Desaulniers, Lavigne et Soumis, 1996). Ces problèmes ont été largement étudiés, ce qui permet d'avoir accès à des stratégies de résolution (paramètres) efficaces. De plus, ces problèmes présentent un profil d'exécution qui s'apparente aux problèmes de construction des rotations d'équipage. Les prochaines sous-sections résument l'essentiel des résultats numériques, proposent des stratégies d'accélération ainsi qu'une analyse des résultats obtenus.

[16] La force des liens qui unissent deux modules constitue le niveau de couplage. Un niveau de couplage élevé signifie une forte dépendance entre les modules et implique une augmentation des coûts de maintenance.

4.3.1 Problèmes de tournées de véhicule

Le tableau 4.1 présente les résultats obtenus sur un problème de tournées de véhicule avec fenêtres de temps comportant 400 clients et 5 dépôts. Les métriques inscrites au tableau ont été définies préalablement aux sections 1.4 et 4.2.1. Rappelons qu'un problème de tournées de véhicule consiste à construire des horaires pour des flottes de véhicules de manière à visiter un ensemble de clients à coût minimum (voir section 1.1). Ce problème a été résolu en nombres réels de façon heuristique de manière à réduire la durée des expérimentations. Le critère d'arrêt du dernier modèle de paramètres exigeait une amélioration de la fonction économique d'une unité en 10 itérations. Les écarts entre les valeurs des fonctions économiques pour les différents essais sont inférieurs à 0.01%.

Tableau 4.1 Résultats sur le problème à 5 dépôts et 400 clients

	Nombre d'esclaves $(p-1)$			
	Séq	2	4	5
# SP	1032	1244	1516	1366
$T_{\text{sp}}(N,p)$ (s)	3016	3520	4423	3992
$W(N,p)$	1	1.17	1.47	1.32
$T_{\text{mp}}(N,p)$ (s)	1336	1945	2202	1835
$T_{\text{total}}(N,p)$ (s)	4374	2306	2395	2043
$A(N,p)$	1	1.90	1.83	2.14
$E(N,p)$	1	0.63	0.37	0.36

De ces résultats sont tirées les observations suivantes :

- la quantité de travail $W(N,p)$ ainsi que le nombre de sous-problèmes résolus (# SP) augmentent avec le nombre d'esclaves;
- l'accélération semble indépendante du nombre d'esclaves : $A(N,p) \approx 2$;
- l'efficacité est très faible et diminue avec le nombre d'esclaves;

- le temps de résolution du problème-maître est relativement constant et significativement plus élevé (50%) qu'avec la version séquentielle;
- l'accélération maximale obtenue (2.14) est significativement inférieure à l'accélération maximale théorique (3.27).

Le problème ne comportant que 5 dépôts et donc 5 sous-problèmes, il ne peut tirer profit de plus de 6 processeurs, soit 5 esclaves et un maître. Toutefois, l'efficacité avec $p \geq 6$ processeurs peut être dérivée de celle à 6 processeurs : $E(N,p) = A(N,6)/p = 2.14/p$. Ce faible résultat met en évidence la pauvreté du potentiel parallèle de ces problèmes et, indirectement, celle de la méthode.

De façon à pouvoir utiliser plus efficacement tous les processeurs, le problème suivant comporte 400 clients et 10 dépôts. Ce problème a été résolu en nombres réels de façon heuristique de manière à réduire la durée des expérimentations. Le critère d'arrêt du dernier modèle de paramètres exigeait une amélioration de la fonction économique d'une unité en 10 itérations. Les écarts entre les valeurs des fonctions économiques pour les différents essais sont inférieurs à 0.01%. Les résultats obtenus sont présentés au tableau 4.2 et viennent corroborer ceux obtenus avec le premier problème. L'accélération maximale mesurée (1.94) est significativement inférieure à l'accélération maximale théorique (2.76).

La forte corrélation entre les résultats des deux problèmes laisse présager que des expérimentations additionnelles n'apporteraient pas d'informations supplémentaires significatives. Ces résultats expérimentaux feront l'objet d'une analyse détaillée à la section 4.3.4. Les problèmes de plus grande taille ont été écartés à cause de la faible valeur de $A_{\max}(N,p)$, i.e., $T_{\mathrm{mp}}(N) \geq T_{\mathrm{sp}}(N)$. Cette affirmation

Tableau 4.2 Résultats sur le problème à 10 dépôts et 400 clients

	Nombre d'esclaves $(p-1)$			
	Séq	2	4	7
# SP	1040	1388	2314	2923
$\hat{T}_{\text{sp}}(N,p)$ (s)	2784	3915	6594	8512
$W(N,p)$	1	1.40	2.37	3.06
$T_{\text{mp}}(N,p)$ (s)	1605	2314	2085	2286
$T_{\text{total}}(N,p)$ (s)	4424	2634	2283	2481
$A(N,p)$	1	1.68	1.94	1.78
$E(N,p)$	1	0.56	0.39	0.22

peut être démontrée par l'entremise de publications sur le sujet. D'ailleurs, la section suivante se propose de généraliser les résultats obtenus à plusieurs autres classes de problèmes.

4.3.2 Généralisation des résultats

Les résultats obtenus précédemment pour les problèmes de tournées de véhicule montrent que la méthode obtient une efficacité passable, tout au plus. Cette piètre performance s'explique en partie par le faible ratio du temps T_{sp} de résolution du sous-problème sur le temps T_{mp} de résolution du problème-maître. Rappelons que l'accélération maximale $A_{\text{max}}(\cdot)$ est directement calculée à partir de ce ratio (voir section 4.2). Il est possible de généraliser les résultats précédents en calculant l'accélération maximale pour des problèmes étudiés dans la littérature et pour lesquels il existe des résultats numériques.

4.3.2.1 Problèmes de tournées de véhicule

Les problèmes de tournées de véhicule (Desaulniers, Lavigne et Soumis, 1996) consistent à construire des horaires pour des flottes de véhicules de manière à visiter un ensemble de clients à coût minimum (voir section 1.1). Le tableau 4.3 présente les résultats obtenus sur des problèmes de tournées de véhicule avec fenêtres de temps et temps d'attente minimal ayant de 400 à 600 clients et 2 ou 5 sous-problèmes. Ces problèmes ont tous été résolus en nombres entiers de façon heuristique (gaps d'intégrité inférieurs à 0.9%). Les deux dernières colonnes du tableau contiennent les informations pertinentes à ce mémoire. Elles ont été calculées à partir de résultats qui ne figurent pas dans le tableau afin d'alléger la présentation. Pour plus de détails, le lecteur intéressé peut consulter le tableau 3 de la publication ci-haut mentionnée.

Tableau 4.3 Résultats de Desaulniers, Lavigne et Soumis sur des problèmes de tournées de véhicule

Problèmes	# clients	$\lvert K \rvert$	$T_{\text{tot}}(N)$ (s)	$\frac{T_{\text{sp}}(N)}{T_{\text{mp}}(N)}$	$A_{\max}(N)$
1	400	2	549	0.42	1.44
2	400	5	497	0.75	1.78
3	500	2	1162	0.35	1.37
4	500	5	980	0.58	1.60
5	600	2	1843	0.31	1.32
6	600	5	1940	0.50	1.51

Selon ces résultats, l'accélération maximale pour un nombre de sous-problèmes donné décroit avec une augmentation du nombre de clients. De plus, ces résultats concordent avec les expérimentations numériques effectuées par l'auteur dans le cadre de ce mémoire. Le lecteur notera que le nombre très faible de sous-problèmes est une caractéristique de cette classe. Ces problèmes offrent donc peu d'attrait pour la méthode proposée.

4.3.2.2 Problèmes de construction des rotations d'équipage

Les problèmes de construction des rotations d'équipage (Lasry, 1996; Desaulniers *et al.*, 1997a) consistent à construire des horaires (rotations) pour les équipages aériens à coût minimum (voir section 1.2). Le tableau 4.4 présente les résultats obtenus par Desaulniers *et al.* sur des problèmes de construction des rotations d'équipage hebdomadaires de moyen courrier ayant de 477 à 1157 vols et comportant chacun 14 sous-problèmes. Ces problèmes ont tous été résolus en nombres entiers de façon heuristique (gaps d'intégrité inférieurs à 0.45%). Comme précédemment, les informations pertinentes à ce mémoire ont été calculées à partir des résultats recueillis et figurent dans les deux dernières colonnes du tableau. Les trois problèmes retenus sont parmis les plus gros, les autres sont omis afin d'abréger le texte.

Tableau 4.4 Résultats de Desaulniers *et al.* sur des problèmes de construction des rotations d'équipage

Problèmes	$\|N\|$	$\|A\|$	$\|K\|$	# vols	$T_{\text{tot}}(N)$ (s)	$\frac{T_{\text{sp}}(N)}{T_{\text{mp}}(N)}$	$A_{\max}(N)$
9209 PP B737	2051	16 148	14	570	2 185	4.19	1.58
9004 FAP B737	3829	41 238	14	477	3 589	19.42	3.56
9209 FAP B737	8829	95 504	14	1157	13 983	6.10	4.83

Le ratio du temps de résolution du sous-problème sur le temps de résolution du problème-maître semble intéressant à première vue. Pourtant, les accélérations maximales calculées ne reflètent pas cette tendance. Ce phénomène s'explique par le temps de calcul non-négligeable consommé par le branchement ainsi que les mécanismes d'agrégation de contraintes. Rappelons que la loi d'Amdhal (voir section 1.4) se calcule avec la proportion séquentielle du logiciel. Dans ce cas précis, l'équation 4.1 qui calcule l'accélération maximale ne s'applique pas car elle néglige le branchement et l'agrégation.

Ceci dit, les accélérations demeurent faibles, quoique plus élevées que celles des problèmes de tournées de véhicule. Le nombre de sous-problèmes (14) permettrait en théorie d'atteindre l'accélération maximale. Par contre, les problèmes journaliers équivalents n'auraient que deux sous-problèmes [17] chacun, ce qui réduirait l'accélération maximale à 2, peu importe le ratio.

Le tableau 4.5 présente les résultats obtenus par Lasry sur un problème de construction des rotations d'équipage hebdomadaire de court courrier ayant 986 vols et comportant 21 sous-problèmes. Ce problème a été résolu en nombres entiers de façon optimale en utilisant une modélisation où les vols sont sur les nœuds et une autre où les vols sont sur les arcs. Il est important de mentionner que le problème traité par Lasry provient de flottes d'avions court courrier et que la combinatoire impliquée est beaucoup plus grande que pour des flottes moyen courrier comme celles traitées par Desaulniers *et al.* De plus, la modélisation utilisée par Lasry considère *tous* les services de vols (séquences de vols séparés par des repos) de façon implicite dans les réseaux. Celle de Desaulniers *et al.* ne considère qu'un sous-ensemble des services de vols qui sont représentés de façon explicite dans les réseaux.

Tableau 4.5 Résultats de Lasry sur un problème de construction des rotations d'équipage

Problèmes	$\|N\|$	$\|A\|$	$\|K\|$	# vols	$T_{\text{tot}}(N)$ (s)	$\frac{T_{\text{sp}}(N)}{T_{\text{mp}}(N)}$	$A_{\max}(N)$
arcs	2014	4905	21	986	29 799	0.16	1.16
nœuds	49 698	1028	21	986	13 346	0.32	1.32

Les accélérations maximales sont encore plus faibles que celles de Desaulniers *et al.* C'etait prévisible puisque les problèmes de court courrier produisent des solution comportant des rotations plus longues impliquant des colonnes plus denses dans le problème-maître, ce qui en ralentit l'exécution.

[17] Les problèmes comportant deux bases, il y aura 14 sous-problèmes pour le problème hebdomadaire et 2 pour le problème journalier.

En conclusion, la classe des problèmes de construction des rotations d'équipage ne semble pas donner beaucoup d'espoirs à la méthode proposée. D'ailleurs, les problèmes de grande taille actuellement étudiés comportent environ 1000 vols par jour et requièrent un temps considérable dans le problème-maître, ce qui tend à diminuer encore plus l'accélération maximale.

4.3.2.3 Problèmes d'affectation des avions

Les problèmes d'affectation des avions (Desaulniers *et al.*, 1997b) consistent à construire des horaires pour des flottes d'avions de manière à couvrir un ensemble de vols à coût minimum (voir section 1.2). Le tableau 4.6 présente les résultats obtenus par Desaulniers *et al.* sur le problème Nord-Américain avec des fenêtres de temps de 20 et 30 minutes. Ce problème a été résolu en nombres entiers de façon heuristique (gaps d'intégrité inférieurs à 0.5%). Les résultats pertinents à ce mémoire se retrouvent dans les deux dernières colonnes du tableau et ont été calculés à partir des informations recueillies dans la publication.

Tableau 4.6 Résultats de Desaulniers *et al.* sur des problèmes d'affectation des avions

Problèmes	$\|N\|$	$\|A\|$	$\|K\|$	# vols	$T_{\text{tot}}(N)$ (s)	$\frac{T_{\text{sp}}(N)}{T_{\text{mp}}(N)}$	$A_{\max}(N)$
Amérique (20 min)	2617	5500	9	252	1469	0.33	1.33
Amérique (30 min)	2765	6498	9	252	3508	0.37	1.37

Les accélérations maximales sont comparables à toutes celles calculées jusqu'à présent et demeurent très faibles. Cette classe de problèmes ne semble pas non plus pouvoir profiter de la méthode de résolution proposée.

4.3.2.4 Problèmes de distribution des rotations au personnel

Les problèmes de distribution des rotations au personnel (Gamache *et al.*, 1994) consistent à distribuer des rotations aux équipages aériens de manière à construire des horaires mensuels à coût minimum (voir section 1.2). Le tableau 4.7 présente les résultats obtenus par Gamache *et al.* sur des problèmes de distribution des rotations au personnel ayant de 55 à 239 sous-problèmes. Ces problèmes ont tous été résolus en nombres réels de façon optimale. Il est à noter que les deux premiers problèmes figurent deux fois dans le tableau. Le commentaire entre parenthèses indique le numéro de la méthode utilisée pour résoudre le problème. La méthode M1 correspond à une résolution dépourvue d'heuristiques et de stratégies d'accélération alors que la méthode M4 en tire profit.

Tableau 4.7 Résultats de Gamache *et al.* sur des problèmes de distribution des rotations au personnel

Problèmes	$\|N\|$	$\|A\|$	$\|K\|$	$T_{\text{tot}}(N)$ (s)	$\frac{T_{\text{sp}}(N)}{T_{\text{mp}}(N)}$	$A_{\max}(N)$
CC Orly Avril (M4)	805	33 539	55	290	0.73	1.73
CC Orly Juillet (M4)	814	29 755	56	571	0.70	1.70
HS Orly Avril (M4)	1281	147 775	239	1481	0.71	1.71
HS Orly Juillet (M4)	1286	147 486	237	2329	0.60	1.60
CC Orly Avril (M1)	805	33 539	55	8994	23.05	24.04
CC Orly Juillet (M1)	814	29 755	56	16814	23.12	24.12

Les accélérations maximales sont très faibles malgré le nombre impressionnant de sous-problèmes. La résolution en nombres entiers, qui n'est pas présentée ici afin d'alléger le texte, présente des accélérations maximales pouvant atteindre 2.12 unités, ce qui demeure encore assez faible comme résultat. Le lecteur notera que même si une méthode parallèle basée sur la stratégie M1 était développée et

que l'accélération maximale pouvait être atteinte, la version parallèle ne rattraperait même pas la version séquentielle utilisant la stratégie M4. La définition de l'accélération telle que présentée dans ce mémoire (voir section 1.4) prend ici toute sa signification, celle-ci faisant explicitement appel à une référence séquentielle considérée comme l'état de l'art.

4.3.2.5 Conclusion

De toutes les classes de problèmes étudiées, seuls les problèmes de construction des rotations d'équipage hebdomadaires de moyen courrier démontraient un quelconque potentiel parallèle à l'égard de la méthode proposée. Ces problèmes ne sont toutefois plus considérés de grande taille, les problèmes actuels étant 10 fois plus gros et requérant un temps de calcul considérable dans le problème-maître. Il existe peut-être des classes de problèmes pour lesquelles la méthode obtiendrait du succès. Toutefois, il semble de plus en plus évident que celles-ci seraient rares et ne pourraient justifier le caractère général de la méthode. La section suivante discute des stratégies d'accélération mises en oeuvre pour pallier la faible efficacité de la méthode.

4.3.3 Stratégies d'accélération

Les faibles performances de la méthode exigent l'ajout de stratégies d'accélération. D'après les observations précédentes, la difficulté réside dans l'augmentation de la quantité de travail tant au niveau des sous-problèmes que du problème-maître.

4.3.3.1 Réduction du nombre de sous-problèmes résolus

Il est courant en pratique de ne résoudre qu'un sous-ensemble des sous-problèmes à chaque itération de génération de colonnes, de façon à minimiser le temps total de résolution. L'algorithme parallèle permet de traiter plus de sous-problèmes lors d'une itération donnée et non de générer plus rapidement les mêmes colonnes que la version séquentielle. En effet, une augmentation du nombre d'esclaves entraîne une augmentation du nombre de sous-problèmes résolus sans toutefois impliquer une augmentation de la vitesse globale de convergence. Par conséquent, le nombre maximal d'esclaves utilisables efficacement pour un problème donné est plus proche du nombre de sous-problèmes résolus à chaque itération par la version séquentielle que du nombre total de sous-problèmes. La difficulté étant reliée à des caractéristiques intrinsèques du problème, il n'y a pas de solution possible à première vue autre que celle de diminuer le nombre d'esclaves, ce qui est à rejeter. Des stratégies plus complexes, comme la reformulation du problème dans le but de maximiser le nombre de sous-problèmes et de minimiser leur recouvrement dans l'espace des solutions, pourraient porter fruits. Ces stratégies sont toutefois à rejeter pour les applications commerciales de par leur complexité. De plus, l'algorithme séquentiel pourrait possiblement bénéficier lui aussi de tels stratégies, influençant négativement l'accélération maximale.

4.3.3.2 Réduction du temps de résolution du problème-maître

L'augmentation du temps de calcul dans le problème-maître est imputable au fait que la version parallèle traite plus de sous-problèmes que la version séquentielle lors d'une itération donnée. Une quantité accrue de colonnes générées implique

une augmentation plus que linéaire du nombre de pivots de simplexe. Tout comme précédemment, une réduction du nombre d'esclaves est à rejeter.

Une première solution serait de réduire le nombre de colonnes envoyées au problème-maître en améliorant le mécanisme de sélection plutôt que de générer moins de colonnes à la source. Toutefois, cette problématique est une question ouverte dans le domaine de la génération de colonnes. D'ailleurs, s'il existait une meilleure méthode de sélection (autre que celle basée sur les coûts réduits), la version séquentielle en profiterait aussi et l'écart entre les deux versions se maintiendrait probablement.

Une deuxième solution serait d'interrompre l'optimisation du problème-maître après un certain nombre de pivots fixé d'avance. Cette stratégie permettrait de récupérer et d'utiliser immédiatement les variables duales courantes (et non optimales) et d'ajouter les colonnes générées entre temps dans le problème-maître avant de poursuivre son optimisation. En pratique cette stratégie ne fonctionne pas. L'optimisation du problème-maître peut nécessiter, par CPLEX, l'ajout d'une perturbation ainsi qu'une relaxation de certaines contraintes pour les problèmes difficiles, i.e., de grande taille. L'arrêt prématuré de l'optimisation oblige CPLEX à ramener immédiatement le problème à sa formulation originale. Ceci peut entraîner des comportements pathologiques et, potentiellement, la non convergence de l'algorithme de génération de colonnes.

Une dernière solution consiste à réduire la taille du problème-maître afin de diminuer l'impact de l'augmentation du nombre de colonnes générées. Toutefois, cette stratégie exige de réoptimiser le problème-maître plus souvent et aboutit à une augmentation du temps total de calcul.

4.3.3.3 Conclusion

Aucune des stratégies d'accélération proposées ne donne des résultats probants. De plus, ces stratégies ont la caractéristique commune de vouloir minimiser le nombre d'esclaves, ce qui va à l'encontre de la méthode. La section suivante procède à l'analyse des résultats dans le but de déceler et d'expliquer les faiblesses de la méthode.

4.3.4 Analyse des résultats

Les résultats expérimentaux ont mis en évidence le peu de potentiel parallèle de la méthode sur les problèmes de tournées de véhicule. Il a même été possible de généraliser ces résultats à plusieurs autres classes de problèmes. Toutefois, c'est l'application des stratégies d'accélération qui a permis en grande partie d'exhiber les défauts de la méthode.

- Forte limitation sur le nombre d'esclaves : nombre de sous-problèmes par itération. L'ajout de machines supplémentaires jusqu'à concurrence du nombre total de sous-problèmes diminue l'efficacité.
- L'augmentation du nombre de colonnes générées entraîne un ralentissement du problème-maître.

D'autres difficultés ne découlant pas directement des résultats expérimentaux présentés dans ce mémoire sont aussi à souligner.

- La quantité de mémoire utilisée par la machine virtuelle est linéairement proportionnelle au nombre d'esclaves. En effet, chaque esclave doit posséder une copie des données du problème (les réseaux) afin de pouvoir résoudre les sous-problèmes. Ceci a pour conséquence de limiter la taille des problèmes selon la capacité de traitement de la machine la moins puissante du parc. Une distribution appropriée des sous-problèmes pourrait, dans certains cas, diminuer l'impact de cette limitation.

- Lors de la résolution en nombres entiers, deux scénarios sont possibles. Ou bien la décision imposée a peu d'impact sur le problème et le nœud de branchement n'exige pas la résolution de sous-problèmes (ex. distribution des rotations au personnel avec séniorité stricte). Ou bien la décision imposée a un impact majeur sur le problème et la première réoptimisation du problème-maître prend alors un temps significatif (ex. construction des rotations d'équipage). Les deux scénarios impliquent une diminution du potentiel parallèle de la résolution en nombres entiers versus la résolution de la relaxation linéaire.

À la lumière de ces observations, il est pertinent de revenir sur les caractéristiques recherchées dans les problèmes et de les compléter :

- nombre élevé de sous-problèmes les plus *indépendants* possibles;
- sous-problèmes comportant un temps de calcul élevé;
- problème-maître comportant un temps de calcul faible et *peu dépendant* du nombre de colonnes ajoutées.

Le potentiel parallèle mérite aussi une révision. Les stratégies heuristiques de la version séquentielle rendent les sous-problèmes dépendants. La granularité demeure

inchangée mais elle cache le fait que le nombre maximal d'esclaves est limité par le nombre optimal de sous-problèmes à résoudre à chaque itération plutôt que par le nombre total de sous-problèmes. La proportion parallèle, au moment d'écrire ces lignes, a diminué avec l'augmentation de la taille des problèmes. Le tableau 4.8 montre le potentiel parallèle révisé de DG.

Tableau 4.8 Potentiel parallèle révisé de DG

Caractéristiques	
Granularité	sous-problèmes
Type de tâches parallèles	dépendantes
Volume de communication	faible
Proportion séquentielle	moyenne à élevée
Type de comportement	non déterministe

Même si la méthode n'a été testée qu'avec des problèmes de recherche, les résultats ont permis d'exhiber les défauts de celle-ci, résultats qui ont pu être généralisés à plusieurs autres classes de problèmes. De plus, la méthode ne peut être implantée dans un module bien défini car ses répercussions sont globales à l'ensemble du logiciel. En effet, l'initialisation du processus d'optimisation ainsi que les manipulations de données propres à chaque application se généralisent difficilement. DG vient compliquer et figer l'interface fonctionnelle de GENCOL, créant ainsi des conflits avec les applications existantes. L'intégration de la méthode dans des applications exige donc des *développements substantiels* qui peuvent difficilement être justifiés par les résultats obtenus jusqu'à présent.

Hormis tous ses défauts, la méthode peut être utilisée dans des situations particulières où l'accélération brute prime sur l'efficacité. Avec une si faible accélération et un temps de développement non négligeable pour les applications commerciales, force est de conclure que la méthode n'a pas sa place dans une bibliothèque générale de génération de colonnes.

4.4 Conclusion

Ce chapitre a décrit l'implantation informatique d'une méthode parallèle de génération de colonnes où le problème-maître et les sous-problèmes sont résolus de façon asynchrone. La première section a effectué une revue de la littérature sur le sujet et a mis en évidence le contexte général dans lequel il était traité ainsi que la rigueur de l'approche. La deuxième section a décrit en détail la méthode proposée et ses spécifications. La troisième section a présenté des résultats numériques. les a généralisés à plusieurs classes de problèmes tout en proposant des stratégies d'accélération. À la lumière des résultats obtenus, on conclut que la méthode n'a que très peu d'applications pratiques : les performances sont pauvres et son intégration sans heurts à des applications commerciales est douteuse.

Le chapitre suivant procède à la description de l'implantation de STA, le module de plus court chemin avec contraintes de ressources en parallèle.

CHAPITRE 5

Plus court chemin avec contraintes de ressources en parallèle

L'implantation décrite au chapitre précédent n'a pas permis d'obtenir les résultats escomptés. En effet, DG comporte des lacunes graves : peu de potentiel parallèle, tâches dépendantes et comportement non déterministe. S'il est possible de circonvenir à la première lacune en réduisant le marché des applications visées, il n'en est pas de même pour les deux autres. C'est à la lumière de ces observations qu'est implanté l'algorithme de plus court chemin avec contraintes de ressources en parallèle.

Il n'est pas viable d'implanter un algorithme général pour le problème de plus court chemin avec contraintes de ressources. D'ailleurs, il existe dans la version séquentielle des algorithmes spécialisés pour des types particuliers de problèmes de plus court chemin. Par conséquent, ce mémoire ne comporte que l'implantation d'un seul algorithme de plus court chemin avec contraintes de ressources, nommé STA, spécialisé pour les graphes acycliques. Ce choix est justifié par le nombre important de problèmes provenant d'applications en transport, où l'utilisation de graphes acycliques comme modèle de sous-problèmes est courante.

Le présent chapitre décrit l'implantation du module STA. La première section définit le problème de plus court chemin avec contraintes de ressources. La deuxième section effectue une revue de la littérature sur les problèmes de plus court chemin en parallèle. La troisième section décrit en détail la méthode proposée. La quatrième section présente des résultats numériques, propose des stratégies d'accélération et

conclut sur l'efficacité de la méthode en pratique.

5.1 Plus court chemin avec contraintes de ressources

Le présent travail de recherche n'entend pas apporter d'améliorations ni à l'algorithme de plus court chemin avec contraintes de ressources ni aux algorithmes spécialisés de dominance qui en font partie. Toutefois, il est nécessaire de décrire le problème générique et de présenter un algorithme de résolution en précisant les points pertinents à ce mémoire. Pour plus de détails, le lecteur peut consulter la section 4.3 de Desrosiers *et al.* (1995).

Soit $G(\mathcal{N}, \mathcal{A})$ un graphe acyclique où $\mathcal{A}$ est l'ensemble des arcs et $\mathcal{N}$ l'ensemble des nœuds, incluant un nœud origine o et un nœud destination d. Un chemin sur le graphe G est défini comme une séquence de nœuds $(i_0, i_1, ..., i_K, i_{K+1})$ telle que chaque arc (i_k, i_{k+1}) appartient à l'ensemble $\mathcal{A}$. De plus, tout chemin doit avoir pour origine le nœud o et pour fin le nœud d. Un coût c_{ij} est associé à chaque arc $(i, j) \in \mathcal{A}$ et le coût d'un chemin est la somme du coût des arcs qui le composent. Un chemin doit respecter les contraintes de ressources à chaque nœud traversé. Au nœud $i \in \mathcal{N}$, une fenêtre de réalisabilité $[a_i^r, b_i^r]$ est associée à chaque ressource $r \in R$. Une valeur d_{ij}^r représente la consommation de ressource associée à chaque arc $(i, j) \in \mathcal{A}$. Il est possible de voir ces contraintes de ressources comme une généralisation à $|R|$ dimensions des contraintes de fenêtres de temps. Le problème de plus court chemin avec contraintes de ressources se définit alors comme celui de trouver sur le graphe G le chemin de coût minimum respectant les contraintes sur la consommation des ressources à chacun des nœuds du chemin. Desrochers (1986) a proposé un algorithme de programmation dynamique de type

pulling pour résoudre cette classe de problèmes. L'algorithme de *pulling* consiste à créer des étiquettes à un nœud j à partir de toutes les étiquettes situées sur le nœud terminal i des chemins pouvant arriver au nœud j dans un état réalisable avec l'ajout d'un arc (i, j). Un algorithme polynomial (Desrochers, 1986), communément appelé dominance, est ensuite appliqué au nœud j pour réduire l'ensemble des étiquettes aux seules étiquettes efficaces. Ce processus n'étant appliqué qu'une seule fois par nœud, les étiquettes créées sont donc permanentes.

5.2 Revue de la littérature

Il n'y a pas, au moment de la rédaction de ce mémoire, d'articles portant sur le problème de plus court chemin avec contraintes de ressources en parallèle malgré l'abondante littérature [18] sur le problème de plus court chemin classique en parallèle. Il n'est pas possible d'utiliser directement les résultats de ces travaux pour trois raisons majeures.

- Les problèmes étudiés comportent une structure particulière comme, par exemple, celle de posséder plusieurs nœuds puits. Pire encore, le problème est parfois celui de trouver le plus court chemin entre toutes les paires de nœuds (*all pairs shortest path problem*), ce qui ne correspond pas du tout au cas que traite ce mémoire.

- Les auteurs se contentent souvent de ne donner que les complexités de leurs algorithmes sans fournir de résultats expérimentaux. Il est donc difficile d'évaluer de façon objective le potentiel réel de leurs méthodes.

[18] Il existe au delà de 150 articles sur la résolution en parallèle des problèmes de plus court chemin, et ce, en excluant les heuristiques basées sur des méthodes tabous.

- Les auteurs qui fournissent des résultats expérimentaux les ont obtenus sur des machines massivement parallèles. Ces algorithmes ne sont toutefois pas applicables sur le type de machines disponibles au GERAD.

Il existe aussi des problèmes, comme le sac de campeur (*knapsack*), dont la structure mathématique est un cas particulier du problème de plus court chemin avec contraintes de ressources. La simplicité de la structure mathématique permet des optimisations plus poussées et donne lieu à des algorithmes plus efficaces. Il est possible, par exemple, de considérer le réseau du problème de sac de campeur de façon implicite ce qui n'est pas le cas pour les problèmes dont traite ce mémoire. En somme, il est fort probable que les optimisations utilisées pour traiter les cas particuliers ne se généralisent pas ou perdraient une grande part de leur potentiel.

Même s'il n'est pas possible pour les raisons mentionnées précédemment d'utiliser directement l'information recueillie lors de la recherche bibliographique, il est tout de même pertinent de faire une synthèse des idées sous-jacentes à ces nouveaux algorithmes. Voici les deux idées les plus remarquables.

- Puisque le problème de trouver le plus court chemin du nœud source au nœud puits est équivalent, dans le cas classique, à trouver le plus court chemin en sens inverse, il est tout naturel de concevoir un algorithme hybride partant des deux côtés en même temps (Bertsekas et Polymenakos, 1994). De plus, rien n'empêche que l'algorithme bénéficiant de cette stratégie soit lui-même un algorithme parallèle. Cette technique pourrait donc en théorie doubler le potentiel parallèle de cet algorithme. Cependant, dans le cas du plus court chemin avec contraintes de ressources, la fonction d'extension des étiquettes est non injective et ne permet pas la construction de chemins en sens inverse.

- Certains algorithmes, dont ceux dits à enchères (*auction algorithms*), construisent plusieurs chemins en même temps. La principale caractéristique de ces algorithmes est qu'ils permettent une certaine coopération entre les chemins, les coûts des nœuds intermédiaires d'un chemin pouvant être utilisés par d'autres chemins. Bertsekas et Polymenakos (1994) proposent une implantation totalement asynchrone où les unités d'exécution peuvent être amenées à travailler avec des coûts périmés.

L'aboutissement de cette synthèse est clair : la seule idée compatible avec le problème traité est celle de construire plusieurs chemins en même temps, ce qui correspond à l'idée générale, proposée au chapitre 2, de traiter les nœuds en parallèle.

5.3 Méthode et spécifications

Cette section décrit la méthode proposée, son implantation dans le module STA ainsi que les spécifications à respecter. La première sous-section présente les détails de la méthode ainsi que l'architecture informatique sous-jacente tout en faisant état des structures particulières rencontrées en pratique dans les réseaux. La seconde sous-section calcule une borne supérieure sur l'accélération et en dérive les caractéristiques des problèmes favorables à la méthode.

5.3.1 Description de la méthode

L'idée de base de la méthode proposée est de traiter plusieurs nœuds d'un réseau en même temps dans le but de terminer le processus de résolution plus rapidement.

C'est une stratégie de force brute destinée à tirer avantage des nouvelles technologies parallèles. La figure 5.1 donne une idée générale de la méthode ainsi que du genre de réseau rencontré en pratique.

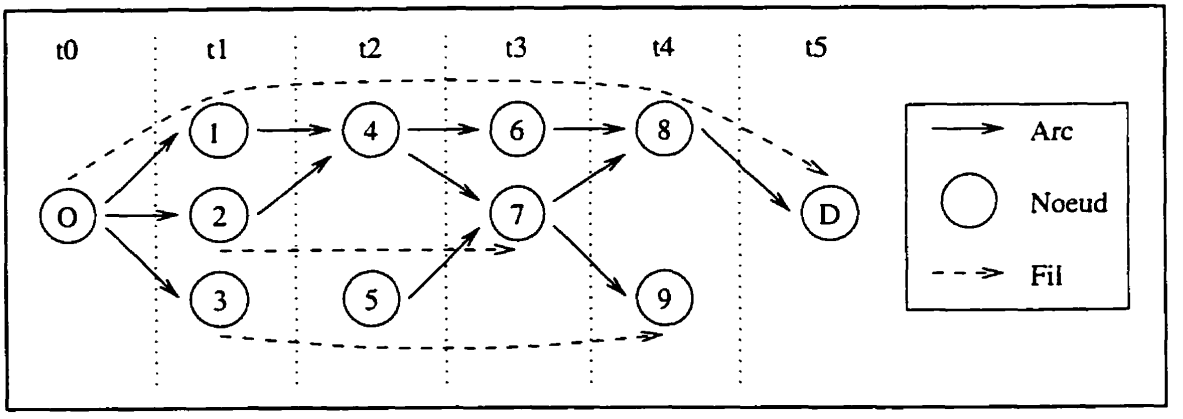


Figure 5.1 Exemple d'exécution de STA avec trois fils d'exécution

L'idée de traiter les nœuds en parallèle est venue lors de l'analyse de l'algorithme de plus court chemin avec contraintes de ressources de la version séquentielle (voir algorithme 5.1). En effet, l'algorithme doit passer au travers de tous les nœuds du réseau dans un *ordre topologique*. Il est connu que pour un réseau donné, il peut exister plusieurs ordonnancements des nœuds qui satisfont cet ordre. Il suffirait de remplacer la pile par une liste et l'algorithme 5.2, qui est celui implanté dans GENCOL, produirait un autre ordonnancement tout aussi valide. La multiplicité des ordonnancements révèle le caractère parallèle du processus de résolution camouflé dans l'algorithme séquentiel.

Pour traiter les tâches en parallèle, une tâche consistant à traiter un nœud du réseau, le modèle choisi est celui du pool de travail. [19] Cette décision est déterminante dans le choix des outils matériels et logiciels.

[19] Un pool de travail est composé de processus identiques capables de traiter l'ensemble des tâches (voir section 1.3.4).

```
POUR chaque nœud j du réseau en ordre topologique FAIRE
    Initialiser la pile p du nœud j
    POUR tous les prédécesseurs i du nœud j FAIRE
        POUR toutes les étiquettes e de i FAIRE
            Appliquer la fonction d'extension de l'arc (i,j) sur e pour produire e'
            SI e' est valide ALORS
                Empiler e' sur p
    Appliquer la fonction de dominance sur les étiquettes de p
```

Algorithme 5.1 Plus court chemin séquentiel

- Une implantation informatique efficace exige l'emploi de machines multiprocesseurs à mémoire partagée (voir section 1.3.2). En effet, le grain de travail (les nœuds du réseau) de STA est trop fin pour pouvoir exploiter efficacement un réseau de machines. La faible localité des données résulterait en un volume élevé d'information à échanger et exigerait un trop grand niveau de synchronisation entre les unités de traitement.

- L'implantation du module STA nécessite des mécanismes de synchronisation ainsi que des fils d'exécution (voir sections 1.3.3 et 2.3.2). La bibliothèque THREADS de SUN est choisie à cause de sa maturité. De plus, la seule machine parallèle disponible au GERAD provient du même manufacturier. Toutefois, dans un soucis de compatibilité future, l'implantation n'utilise qu'un sous-ensemble restreint des fonctionnalités disponibles de manière à pouvoir être récrite facilement avec la bibliothèque POSIX advenant une prise du marché par cette dernière.

En pratique, les réseaux traités comportent des structures particulières qui peuvent compliquer l'implantation informatique. Cette dernière doit tenir compte des trois caractéristiques suivantes.

```
POUR chaque nœud i du réseau FAIRE
    | Compter le nombre d'arcs entrant au nœud i
Initialiser une pile p
POUR chaque nœud i du réseau FAIRE
    | SI i n'a pas d'arcs entrant ALORS
    |     | Mettre i sur la pile p
Initialiser une liste l
TANT que p n'est pas vide FAIRE
    | Dépiler un nœud i de p
    | Ajouter i à la fin de l
    | POUR tous les arcs (i, j) FAIRE
    |     | Diminuer le nombre d'arcs entrant en j
    |     | SI le nombre d'arcs entrant en j est nul ALORS
    |     |     | Empiler j sur p
Réordonner les nœuds du réseau selon l
```

Algorithme 5.2 Tri topologique

- Comme en fait foi la figure 5.1, il peut arriver que le réseau traité ne soit pas connexe. En fait, l'existence d'un chemin entre le nœud source et le nœud puits n'est même pas garantie. Les réseaux non connexes proviennent généralement de l'application d'heuristiques pour réduire la taille des réseaux.

- De plus, le réseau peut aussi posséder plusieurs nœuds puits, ce qui revient à résoudre plusieurs problèmes de plus court chemins *dépendants*.

- Finalement, certaines classes de problèmes comportent des réseaux périodiques dans le but de compresser les données. Il est possible, par exemple, de ne conserver explicitement que les arcs et les nœuds d'une journée pour un problème hebdomadaire. Puisqu'ils sont compressés, ces réseaux nécessitent un traitement spécial.

L'analyse partielle de la méthode effectuée jusqu'à présent conduit à l'algorithme 5.3 qui permet de traiter le problème de plus court chemin avec contraintes

de ressources en parallèle. L'algorithme s'apparente fortement à celui de la version séquentielle (voir algorithme 5.1). À l'exception de la phase d'initialisation qui n'est effectuée qu'une seule fois, cet algorithme est exécuté par chacun des fils. Ceci implique que la fonction de dominance doit obligatoirement pouvoir être exécutée concurremment (*reentrant function*). Cette situation ne cause pas de problèmes puisque la fonction de dominance fait partie de STA. Cette obligation est aussi nécessaire pour la fonction d'extension qui est définie au niveau de l'application. En pratique, quelques changements mineurs sont nécessaires pour s'assurer que la fonction d'extension possède cette propriété. STA comporte aussi des extensions supplémentaires afin d'émuler correctement la version séquentielle de référence SPA : groupes d'étiquettes ne pouvant se dominer, vérification des contraintes de branchement et fonctions de dominance spécialisées pour certains patrons de ressources. Ces extensions ne sont pas présentées afin d'alléger le texte.

```
Initialisation
TANT que le réseau n'est pas complètement traité FAIRE
    Sélection du nœud j
    Initialiser la pile p du nœud j
    POUR tous les prédécesseurs i du nœud j FAIRE
        POUR toutes les étiquettes e de i FAIRE
            Appliquer la fonction d'extension de l'arc (i, j) sur e pour produire e'
            SI e' est valide ALORS
                Empiler e' sur p
    Appliquer la fonction de dominance sur les étiquettes de p
```

Algorithme 5.3 Plus court chemin parallèle

L'implantation de STA nécessite la conception de trois procédures : sélection du nœud suivant, critère d'arrêt et initialisation. Soulignons que le nombre de fils d'exécution est déterminé par l'usager et ne fait pas partie de l'algorithme.

Sélection du nœud suivant : Tel que mentionné précédemment, il existe plusieurs ordres de traitement des nœuds du réseau. La règle déterminant le prochain nœud à traiter étant donné un ensemble de nœuds déjà considérés consiste à choisir un nœud si et seulement si tous ses prédécesseurs ont été traités et s'il ne l'a pas déjà été lui-même. En effet, les contraintes de préséance entre les nœuds sont déterminées par la structure même du réseau, i.e. les arcs. Si plusieurs nœuds satisfont cette règle, on choisit le nœud le plus près du nœud source. [20] Cette stratégie heuristique tend à explorer le réseau en largeur d'abord. Le support des réseaux périodiques exige une extension à cette procédure qui n'est pas présentée ici afin d'abréger le texte.

Critère d'arrêt : Il existe plusieurs alternatives possibles présentées en ordre croissant de complexité d'implantation.

- Le nombre de nœuds traités est égal au nombre de nœuds dans le réseau.
- Le nombre de nœuds puits traités est égal au nombre de nœuds puits dans le réseau. Cette solution a le mérite de ne pas forcer le traitement de tous les nœuds du réseau : ex. nœud 5 (voir figure 5.1).
- Aucun nœud ne satisfait la règle de sélection et il n'y a pas de nœuds en cours de traitement. Cette dernière solution permet d'économiser la gestion d'un compteur (ressource globale) sur les nœuds. Elle suppose toutefois la connaissance de l'état des autres fils d'exécution.

Il serait souhaitable d'éviter d'avoir à traiter des parties non productives du réseau comme les nœuds 5 et 9 (voir figure 5.1). Malheureusement, il n'y a pas de solution

[20] En cas d'égalité, le nœud est choisi arbitrairement.

efficace à ce problème. De toute façon, un réseau connexe par ses arcs pourrait très bien être non connexe à cause des ressources. Dans ce contexte, il est préférable de retenir la solution la plus facile à implanter, soit celle de traiter tous les nœuds.

Initialisation : La fonction d'initialisation est responsable de la mise en place du réseau en vue du traitement parallèle. Celle-ci doit être effectuée à chaque résolution du plus court chemin. L'algorithme consiste à calculer le nombre d'arcs incidents à chacun des nœuds du réseau (voir début de l'algorithme 5.2).

La méthode ayant été décrite en détail, il est maintenant possible de conclure sur son aspect déterministe. L'algorithme est déterministe d'un point de vue externe (bibliothèque GENCOL) mais ne l'est pas d'un point de vue interne à STA puisque les nœuds ne sont pas toujours traités dans le même ordre. En effet, l'ordre de traitement des prédécesseurs i d'un nœud j peut varier mais leurs étiquettes seront toujours traitées dans le même ordre au nœud j parce que la boucle sur les prédécesseurs (voir algorithme 5.1) ne dépend que de l'ordre dans lequel les arcs ont été définis dans les données.

5.3.2 Caractérisation des problèmes favorables à la méthode

Le calcul d'une borne supérieure sur l'accélération donne déjà une bonne idée des caractéristiques recherchées.

L'accélération globale maximale est une application directe de la loi d'Amdhal :

$$A_{\max}(N,p) = \frac{T_{\text{sp}}(N) + T_{\text{mp}}(N)}{T_{\text{mp}}(N) + (T_{\text{sp}}(N) - T_{\text{GenLs}}(N))} \tag{5.1}$$

L'équation tient compte du fait que le problème de plus court chemin ne constitue

qu'une partie du traitement d'un sous-problème (voir algorithme 5.4). Rappelons que $T_{\mathrm{sp}}(N)$ et $T_{\mathrm{mp}}(N)$ représentent le temps passé dans le sous-problème et dans le problème-maître respectivement. De même, N et p représentent la taille du problème et le nombre de processeurs. Quant à $T_{\mathrm{GenLs}}(N)$, il ne tient compte que du temps de résolution du problème de plus court chemin.

SetNet	: mise en place du réseau
MaskNet	: réduction heuristique
PriceNet	: mise à jour des coûts des arcs
GenLs	: génération des étiquettes (plus court chemin)
ChooseLs	: génération des colonnes
CleanLs	: recyclage des étiquettes
UnsetNet	: ménage

Algorithme 5.4 Résolution d'un sous-problème

L'accélération locale maximale se calcule en transformant le problème original en un problème de plus long chemin où les coûts sur les arcs correspondent au temps de traitement de chacun des nœuds du problème original. Toutefois, ces temps de traitement ne sont pas connus *a priori* parce qu'ils dépendent du nombre d'étiquettes à prolonger et à dominer ainsi que de la complexité variable des fonctions d'extension. Pour illustrer ce propos, la figure 5.2 montre deux réseaux de topologie identique mais où les fonctions d'extension prenant plus de temps de calcul sont identifiées en pointillé. Le réseau (a) obtiendra ainsi une courbe de l'accélération en fonction du nombre de processeurs plus favorable que le réseau (b). Par conséquent, l'accélération locale maximale ne peut être estimée que pour les problèmes dont la structure du réseau et la complexité relative des fonctions d'extension sont bien définies.

À la lumière de ces observations, il est quand même possible d'énumérer les caractéristiques des problèmes favorables à la méthode proposée :

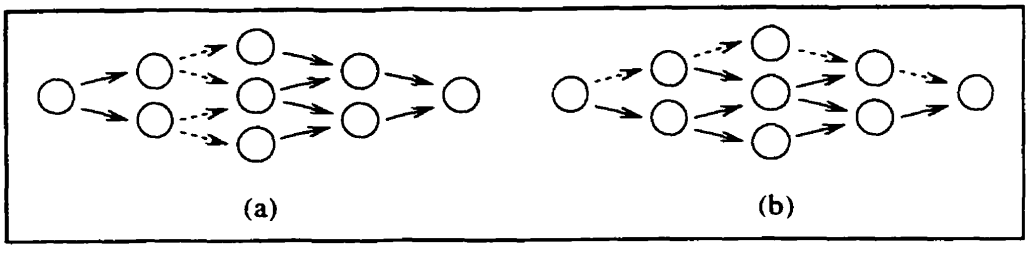


Figure 5.2 Réseaux avec différents types d'arcs

- nombre de nœuds par niveau élevé, ce niveau étant déterminé par la distance en nombre d'arcs à partir du nœud source;
- nombre d'arcs élevé;
- fonctions d'extension complexes;
- problème-maître comportant un faible temps de calcul.

Il faut cependant nuancer ces caractéristiques, comme le démontre la figure 5.3. Les deux réseaux ont les mêmes caractéristiques mais le premier est nettement désavantagé pour un traitement parallèle par la présence d'un goulot d'étranglement.

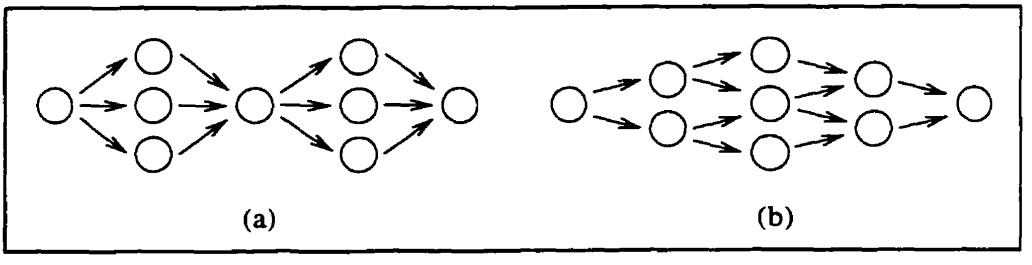


Figure 5.3 Goulot d'étranglement versus réseau uniforme

5.4 Résultats expérimentaux

Les spécifications de l'interface fonctionnelle de STA étant calquées sur celles de SPA, les deux modules sont interchangeables. Le caractère modulaire de STA rend

possible un traitement parallèle des sous-problèmes pour toutes les applications utilisant SPA comme module de résolution du problème de plus court chemin. Cette caractéristique permet de traiter les problèmes de distribution des rotations au personnel avec séniorité stricte (Gamache *et al.*, 1997) et de construction des rotations d'équipage (Desaulniers *et al.*, 1997a) dont les résultats expérimentaux sont résumés dans les deux premières sous-sections. Les sous-sections suivantes proposent des stratégies d'accélération et une analyse des résultats obtenus.

5.4.1 Problèmes de distribution des rotations au personnel avec séniorité stricte

Les problèmes de distribution des rotations au personnel avec séniorité stricte (Gamache *et al.*, 1997) consistent à distribuer des rotations aux équipages aériens de manière à construire des horaires mensuels à coût minimum (voir section 1.2). Le tableau 5.1 décrit quantitativement les 4 problèmes de distribution des rotations au personnel avec séniorité stricte utilisés pour évaluer la méthode. Seule la première phase de chacun des problèmes a été résolu. Cette phase consiste à déterminer le nombre minimal d'employés.

Tableau 5.1 Caractéristiques des problèmes de distribution des rotations au personnel avec séniorité stricte

No	# employés	$\|K\|$	# contraintes	$\|N\|$	$\|A\|$	$\|R\|$
1	26	26	212	986	1026	4
2	34	34	313	1394	1593	4
3	37	37	347	1528	1683	4
4	117	117	1016	4238	6881	4

Le tableau 5.2 présente les résultats obtenus sur le premier problème. Les mesures ayant STA comme indice sont locales (plus court chemin) alors que celles

ayant GLOB comme indice sont globales (voir sections 1.3.5 et 1.4). Ce problème est considéré comme étant de petite taille. L'idée sous-jacente à résoudre ce problème est d'observer les performances de la méthode pour des problèmes de taille inférieure à ceux visés par ce mémoire afin d'en mesurer la robustesse.

Tableau 5.2 Résultats sur le problème # 1 : 26 employés (212 contraintes)

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\hat{T}_{\text{sta}}(N,p)$ (s)	497	516	581	627	697	758	820	868	908
$W_{\text{sta}}(N,p)$	1	1.04	1.17	1.26	1.40	1.52	1.65	1.75	1.83
$T_{\text{sta}}(N,p)$ (s)	497	517	295	217	192	186	191	197	207
$A_{\text{sta}}(N,p)$	1	0.96	1.68	2.29	2.58	2.68	2.61	2.52	2.40
$E_{\text{sta}}(N,p)$	1	0.96	0.84	0.76	0.65	0.54	0.43	0.36	0.30
$T_{\text{glob}}(N,p)$ (s)	721	746	526	440	427	418	419	419	441
$A_{\text{glob}}(N,p)$	1	0.97	1.37	1.64	1.69	1.72	1.72	1.72	1.64

De ces résultats sont tirées les observations suivantes :

- l'accélération locale $A_{\text{sta}}(N,p)$ est limitée à 2.68 pour 5 processeurs et décroît sensiblement par la suite;
- la quantité de travail $W(N,p)$ augmente avec le nombre de processeurs;
- l'efficacité est très faible et diminue avec le nombre de processeurs;
- l'accélération globale maximale (1.72) est significativement inférieure à l'accélération maximale théorique (3.22).

Les tableaux 5.3 et 5.4 présentent les résultats obtenus sur les problèmes # 2 et # 3. Ces deux problèmes sont considérés comme étant de moyenne taille et sont aussi destinés à mesurer la robustesse de la méthode.

Tableau 5.3 Résultats sur le problème # 2 : 34 employés (313 contraintes)

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\hat{T}_{\text{sta}}(N,p)$ (s)	1233	1283	1424	1519	1622	1715	1824	1940	1987
$W_{\text{sta}}(N,p)$	1	1.04	1.15	1.23	1.32	1.39	1.48	1.57	1.61
$T_{\text{sta}}(N,p)$ (s)	1233	1283	719	519	434	394	384	394	397
$A_{\text{sta}}(N,p)$	1	0.96	1.72	2.37	2.84	3.13	3.21	3.13	3.11
$E_{\text{sta}}(N,p)$	1	0.96	0.86	0.79	0.71	0.63	0.53	0.45	0.39
$T_{\text{glob}}(N,p)$ (s)	1680	1724	1167	981	908	839	834	840	828
$A_{\text{glob}}(N,p)$	1	0.97	1.44	1.71	1.85	2.00	2.01	2.00	2.03

Tableau 5.4 Résultats sur le problème # 3 : 37 employés (347 contraintes)

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\hat{T}_{\text{sta}}(N,p)$ (s)	4838	5008	5466	5874	6313	6584	6956	7192	7369
$W_{\text{sta}}(N,p)$	1	1.04	1.13	1.21	1.30	1.36	1.44	1.49	1.52
$T_{\text{sta}}(N,p)$ (s)	4838	5009	2761	2027	1730	1608	1613	1529	1554
$A_{\text{sta}}(N,p)$	1	0.97	1.75	2.39	2.80	3.01	3.00	3.16	3.11
$E_{\text{sta}}(N,p)$	1	0.97	0.88	0.80	0.70	0.60	0.50	0.45	0.39
$T_{\text{glob}}(N,p)$ (s)	12014	12131	10041	9239	8974	8916	9227	9004	8601
$A_{\text{glob}}(N,p)$	1	0.99	1.20	1.30	1.34	1.35	1.30	1.33	1.40

On observe un comportement similaire à celui du petit problème. Toutefois, les accélérations locales sont significativement supérieures à celle du problème de petite taille et elles décroissent moins rapidement après avoir atteint leur valeur maximale.

Le tableau 5.5 présente les résultats obtenus sur le dernier problème. Ce problème est considéré comme étant de grande taille.

Tableau 5.5 Résultats sur le problème # 4 : 117 employés (1016 contraintes)

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$T_{sta}(N,p)$ (s)	6262	6785	7138	7505	7849	8229	9136	9009	9317
$W_{sta}(N,p)$	1	1.08	1.14	1.20	1.25	1.31	1.46	1.44	1.49
$T_{sta}(N,p)$ (s)	6262	6788	3582	2536	2033	1763	1775	1553	1536
$A_{sta}(N,p)$	1	0.92	1.75	2.47	3.08	3.55	3.53	4.03	4.08
$E_{sta}(N,p)$	1	0.92	0.87	0.82	0.77	0.71	0.59	0.58	0.51
$T_{glob}(N,p)$ (s)	14387	14942	11952	10673	10225	9895	10009	9721	9806
$A_{glob}(N,p)$	1	0.96	1.20	1.35	1.41	1.45	1.44	1.48	1.47

Les remarques qui s'appliquaient entre le problème de petite taille et les deux problèmes de moyenne taille s'appliquent aussi entre ces derniers et le problème de grande taille. L'accélération maximale locale dépasse les 4 unités mais son impact est amoindri globalement par l'augmentation en temps de calcul du problème-maître. Cette augmentation est liée aux caractéristiques intrinsèques du problème et ne découle pas de l'utilisation de la méthode.

5.4.2 Problèmes de construction des rotations d'équipage

Les problèmes de construction des rotations d'équipage (Desaulniers *et al.*, 1997a) consistent à construire des horaires (rotations) pour les équipages aériens à coût

minimum (voir section 1.2). Les résultats suivants proviennent d'un problème journalier de court courrier de 499 vols comprenant 3 bases : $|R| = 20$, $|A| = 517\,575$, $|N| = 15\,625$ et $|K| = 3$. Les résultats obtenus sont inscrits aux tableaux 5.6 à 5.8. Le problème a été résolu avec trois modèles de complexité croissante durant un nombre fixe d'itérations résultant en 60 sous-problèmes pour le modèle A, 60 sous-problèmes pour le modèle B et 20 sous-problèmes pour le modèle C. Cette approche permet de réduire le temps total de résolution mais invalide les statistiques globales. Les deux premiers modèles sont utilisés en production alors que le troisième modèle ne sert (pour l'instant) que durant les compétitions (*benchmarks*) entre fournisseurs car il exige au delà de 2 minutes de temps de calcul par sous-problème.

Tableau 5.6 Résultats sur le problème journalier : modèle A

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\hat{T}_{\text{sta}}(N,p)$ (s)	199	273	277	297	289	307	314	317	330
$W_{\text{sta}}(N,p)$	1	1.37	1.39	1.49	1.45	1.54	1.58	1.59	1.66
$T_{\text{sta}}(N,p)$ (s)	199	273	146	105	82	70	65	57	59
$A_{\text{sta}}(N,p)$	1	0.73	1.37	1.89	2.44	2.83	3.06	3.51	3.36
$E_{\text{sta}}(N,p)$	1	0.73	0.68	0.63	0.61	0.57	0.51	0.50	0.42

Tableau 5.7 Résultats sur le problème journalier : modèle B

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\hat{T}_{\text{sta}}(N,p)$ (s)	1926	2506	2559	2466	2492	2378	2400	2431	2443
$W_{\text{sta}}(N,p)$	1	1.30	1.33	1.28	1.29	1.23	1.25	1.26	1.27
$T_{\text{sta}}(N,p)$ (s)	1928	2510	1295	836	646	494	430	369	356
$A_{\text{sta}}(N,p)$	1	0.77	1.49	2.31	2.98	3.90	4.49	5.23	5.41
$E_{\text{sta}}(N,p)$	1	0.77	0.74	0.77	0.75	0.78	0.75	0.75	0.68

De ces résultats sont tirées les observations suivantes :

- l'accélération locale croît avec le nombre de processeurs à une exception près;

Tableau 5.8 Résultats sur le problème journalier : modèle C

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\hat{T}_{sta}(N,p)$ (s)	2883	3811	3833	3684	3814	3674	3653	3748	3934
$W_{sta}(N,p)$	1	1.32	1.33	1.28	1.32	1.27	1.27	1.30	1.36
$T_{sta}(N,p)$ (s)	2883	3813	1924	1239	967	750	625	553	510
$A_{sta}(N,p)$	1	0.76	1.50	2.33	2.98	3.85	4.62	5.22	5.66
$E_{sta}(N,p)$	1	0.76	0.75	0.78	0.75	0.77	0.77	0.75	0.71

- la quantité de travail $W(N,p)$ augmente avec le nombre de processeurs pour le modèle A mais demeure relativement constante pour les modèles B et C;
- l'efficacité est très faible et diminue avec le nombre de processeurs pour le modèle A. Les modèles B et C présentent des efficacités acceptables et constantes;
- l'accélération globale maximale est atteinte pour 8 processeurs avec les modèles B et C, ce qui laisse présager des gains additionnels pour des processeurs supplémentaires.

Afin de mesurer l'efficacité de la méthode pour les cas hebdomadaire et mensuel, il est nécessaire de résoudre des problèmes supplémentaires. N'ayant qu'un problème journalier, il devra être modifié pour émuler un problème mensuel ou hebdomadaire. En effet, le problème journalier comporte une contrainte supplémentaire au niveau des sous-problèmes. Cette contrainte, qui a pour but d'éliminer les cycles dans les chemins, est inutile pour les problèmes hebdomadaires et mensuels. En désactivant cette contrainte, le problème, tel que résolu, comporte la même dynamique au niveau des sous-problèmes qu'un véritable problème mensuel ou hebdomadaire. Les résultats obtenus sont inscrits aux tableaux 5.9 à 5.11. Le problème a été résolu avec trois modèles de complexité croissante durant un nombre fixe

d'itérations comme précédemment.

Tableau 5.9 Résultats sur le problème hebdomadaire/mensuel : modèle A

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\tilde{T}_{sta}(N,p)$ (s)	135	141	162	169	182	189	215	210	223
$W_{sta}(N,p)$	1	1.05	1.20	1.25	1.35	1.40	1.59	1.56	1.65
$T_{sta}(N,p)$ (s)	135	141	86	63	53	47	46	43	43
$A_{sta}(N,p)$	1	0.96	1.58	2.16	2.53	2.86	2.91	3.16	3.17
$E_{sta}(N,p)$	1	0.96	0.79	0.72	0.63	0.57	0.49	0.45	0.40

Tableau 5.10 Résultats sur le problème hebdomadaire/mensuel : modèle B

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\tilde{T}_{sta}(N,p)$ (s)	1112	1174	1218	1098	1344	1199	1204	1149	1126
$W_{sta}(N,p)$	1	1.06	1.10	0.99	1.21	1.08	1.08	1.03	1.01
$T_{sta}(N,p)$ (s)	1112	1174	618	378	350	256	218	183	162
$A_{sta}(N,p)$	1	0.95	1.80	2.94	3.18	4.34	5.09	6.07	6.88
$E_{sta}(N,p)$	1	0.95	0.90	0.98	0.79	0.87	0.85	0.87	0.86

Tableau 5.11 Résultats sur le problème hebdomadaire/mensuel : modèle C

	Nombre de processeurs (p)								
	Séq	1	2	3	4	5	6	7	8
$\tilde{T}_{sta}(N,p)$ (s)	1801	1879	1853	1926	1770	1861	1729	1733	1753
$W_{sta}(N,p)$	1	1.04	1.03	1.07	0.98	1.03	0.96	0.96	0.97
$T_{sta}(N,p)$ (s)	1801	1879	934	656	455	390	303	263	236
$A_{sta}(N,p)$	1	0.96	1.93	2.75	3.96	4.62	5.94	6.85	7.65
$E_{sta}(N,p)$	1	0.96	0.96	0.92	0.99	0.92	0.99	0.98	0.96

Le modèle A présente sensiblement la même dynamique que pour le problème journalier alors que les modèles B et C montrent des résultats plus prometteurs : accélération croissante avec p et efficacité supérieure à 80% pour le modèle B et à 90% pour le modèle C. Cette amélioration de la performance s'explique par l'absence

de la contrainte d'élimination des cycles dans les problèmes mensuels et hebdomadaires. Cette contrainte étant non réentrante, [21] elle nécessite des mécanismes supplémentaires de synchronisation afin d'assurer sa validité dans un contexte parallèle.

5.4.3 Stratégies d'accélération

Bien que la méthode montre de bonnes performances, il est pertinent de présenter quelques stratégies d'accélération qui pourront être expérimentées ultérieurement.

Allocation statique des noeuds aux processeurs : La procédure de sélection du nœud suivant est située à l'intérieur d'une zone critique qui ne peut être visitée que par un seul processeur. Ce genre de mécanisme augmente le niveau de synchronisation entre les unités d'exécution et diminue le parallélisme. Une façon de circonvenir cette difficulté est d'allouer les nœuds aux processeurs *a priori*. Cette allocation statique exige une analyse de la topologie du réseau similaire à celle requise pour déterminer l'accélération locale maximale. Cette analyse peut être plus ou moins coûteuse en fonction du niveau d'équilibrage de charge désiré entre les processeurs. Afin d'être efficace, l'analyse devrait être effectuée le moins souvent possible, ce qui n'est pas évident vu le caractère dynamique des réseaux. [22]

Gestion dynamique du nombre de processeurs : Tel qu'observé lors des expérimentations numériques, l'accélération locale maximale n'est pas toujours obtenue avec $p_{\max}$. Il existe donc une taille de problème minimale pour chaque valeur de p en

[21] La validation d'une contrainte non réentrante par plus d'un processeur à la fois peut donner des résultats erronés.

[22] Les heuristiques d'élagage et les décisions de branchement modifient constamment les réseaux.

dessous de laquelle $p-1$ processeurs obtiendraient un meilleur temps de résolution. De plus, les heuristiques d'élagage des réseaux font en sorte que la taille de ceux-ci, vue par l'algorithme de plus court chemin avec contraintes de ressources, s'en trouve diminuée. Une gestion dynamique du nombre de processeurs permettrait de diminuer l'impact de ce phénomène. Les critères de décision potentiels sont variés : nombre de nœuds dans le réseau, ratio du nombre d'arcs sur le nombre de nœuds, temps de traitement du réseau précédent, etc. Un calcul préliminaire effectué au moyen des résultats expérimentaux des problèmes de distribution des rotations au personnel avec séniorité stricte montre qu'en prenant le meilleur temps pour résoudre chacun des sous-problèmes (critère parfait), on obtient une amélioration de l'accélération d'à peine 5%. Cette remarque laisse présager que l'idée n'a pas beaucoup de potentiel.

Diminution de la granularité au niveau des arcs : La topologie des réseaux des problèmes de construction des rotations d'équipage les rend plus aptes au parallélisme que ceux des problèmes de distribution des rotations au personnel avec séniorité stricte. Les résultats expérimentaux ont confirmé cette affirmation. Toutefois, dans un cas comme dans l'autre, l'auteur fait la conjecture que l'accélération locale ne devrait pas augmenter significativement avec $p > 10$. Une façon d'accroître le potentiel parallèle est de considérer non pas les nœuds mais plutôt chaque arc entrant dans un nœud comme une tâche à traiter en parallèle. Cette diminution de la granularité n'est pas possible pour l'instant car la technologie des machines parallèles disponibles au GERAD ne le permet pas.

5.4.4 Analyse des résultats

L'implantation informatique de la méthode a passé les tests avec succès. STA a obtenu des efficacités appréciables variant de 50% à 75% pour les problèmes de distribution des rotations au personnel avec séniorité stricte et de 50% à 95% pour les problèmes de construction des rotations d'équipage. Les résultats obtenus laissent présager que la méthode bénéficierait de processeurs supplémentaires pour les problèmes de construction des rotations d'équipage. Par contre, ce n'est pas aussi clair pour les problèmes de distribution des rotations au personnel avec séniorité stricte : la topologie des réseaux limite le niveau de parallélisme.

Plus spécifiquement, la méthode et son implantation informatique présentent deux qualités recherchées dans un logiciel d'optimisation : la robustesse et la modularité.

La méthode est *robuste* pour les raisons suivantes :

- les résultats sont reproductibles et il y a peu de fluctuations dans les temps de calcul;
- l'accélération est toujours supérieure à 1 (pour $p > 1$) même pour les problèmes de petite taille;
- il est possible d'utiliser sans crainte tous les processeurs disponibles puisque l'accélération locale est pratiquement toujours croissante en fonction du nombre de processeurs;
- la méthode est indépendante du problème-maître et n'a aucune influence sur son temps d'exécution;

- l'augmentation de la quantité de mémoire utilisée est à peine notable, soit de 1% à 2% pour les problèmes de distribution des rotations au personnel avec séniorité stricte et de 2% à 5% pour les problèmes de construction des rotations d'équipage.

L'implantation est *modulaire* car elle utilise les spécifications de l'interface fonctionnelle de SPA, rendant les deux modules interchangeables. Cette propriété permet à la méthode de traiter des problèmes provenant de diverses applications. De plus, l'idée générale derrière la méthode, qui est de traiter les nœuds du réseau en parallèle, peut se généraliser et être appliquée aux autres modules de plus court chemin avec contraintes de ressources.

En toute objectivité, la méthode possède aussi certains défauts qu'il est important de mentionnner. Le niveau de parallélisme semble être limité à une dizaine de processeurs, ce qui est assez faible. Toutefois, des développements technologiques du côté matériel permettraient de diminuer éventuellement le grain de parallélisme au niveau des arcs, ce qui accroîtrait le potentiel parallèle. La méthode requiert une machine multiprocesseurs, ce qui en limite l'usage. Par contre, ces dernières font une entrée cette année dans le milieu des ordinateurs personnels. Cette situation ne peut que favoriser le développement du genre de méthodes proposées dans ce chapitre.

En conclusion, la méthode possède des qualités indéniables qui lui confèrent une place dans une bibliothèque générale de génération de colonnes.

5.5 Conclusion

Ce chapitre a décrit l'implantation informatique d'un algorithme parallèle de plus court chemin avec contraintes de ressources où les nœuds des réseaux sont traités en parallèle. La première section a défini le problème de plus court chemin avec contraintes de ressources. La deuxième section a effectué une revue de la littérature sur les problèmes de plus court chemin en parallèle afin d'en retenir les idées exploitables dans le contexte du problème de plus court chemin avec contraintes de ressources. La troisième section a décrit en détail la méthode proposée et ses spécifications. La quatrième section a présenté des résultats numériques et a proposé des méthodes d'accélération. Les résultats obtenus sont encourageants et justifient l'utilisation de la méthode. La robustesse de la méthode et l'aspect modulaire de son implantation informatique font en sorte que STA fait maintenant partie intégrante de la bibliothèque GENCOL.

CONCLUSION

Ce mémoire a décrit les travaux de recherches effectués dans le cadre d'un projet de maîtrise portant sur la résolution en parallèle de problèmes de grande taille pouvant être représentés par le modèle unifié. Il comporte trois contributions à l'avancement des connaissances dans le domaine de la recherche opérationelle et plus précisément en génération de colonnes.

En premier lieu, une *analyse* du potentiel parallèle d'un logiciel générique de génération de colonnes résolvant le modèle unifié, GENCOL. Cette analyse a cerné les composantes du logiciel pouvant le plus bénéficier du parallélisme. La composante responsable de l'arbre de branchement a été rejetée car la structure arborescente n'est souvent pas présente dans les problèmes visés. La conception d'un algorithme parallèle au niveau du problème-maître est laissée à des travaux futurs, une version séquentielle de qualité n'étant pas disponible. La relaxation linéaire ainsi que le sous-problème ont été retenus et ont donné lieu à deux implantations informatiques.

En deuxième lieu, une implantation informatique parallèle de *la résolution de la relaxation linéaire provenant de la décomposition de Dantzig-Wolfe* en résolvant le problème-maître et les sous-problèmes de façon asynchrone. Les travaux antérieurs portant sur certains problèmes biens précis laissaient présager des résultats plus qu'intéressants. Or, le cas général présente un profil d'exécution très différent et offre un faible potentiel parallèle. De plus, l'algorithme parallèle a tendance à augmenter la quantité de travail tant au niveau du problème-maître que des sous-problèmes. Cette tendance couplée à la grande variance dans les temps d'exécution diminuent la robustesse de la méthode. Finalement, la difficulté d'intégration de la méthode dans des applications commerciales est un handicap sérieux pour une méthode qualifiée

de générale. À la suite de ces observations, il a été décidé que la méthode ne ferait pas partie de la bibliothèque GENCOL. La possibilité d'utiliser la méthode dans des contextes particuliers est laissée au niveau des applications.

En troisième lieu, une implantation informatique parallèle de *la résolution du problème de plus court chemin avec contraintes de ressources* où les nœuds des réseaux sont traités en parallèle. Les résultats obtenus sont encourageants et justifient l'utilisation de la méthode. La robustesse de la méthode et le caractère modulaire de son implantation informatique font en sorte que STA fait maintenant partie intégrante de la bibliothèque GENCOL. Toutefois, la nécessité d'une machine parallèle multiprocesseurs est un frein (pour l'instant) à son utilisation. Les améliorations possibles à la méthode au niveau du grain de parallélisme (arcs versus nœuds) et de la sélection des nœuds sont laissées à des travaux futurs.

En conclusion, les travaux de recherche ont montré que le domaine de la génération de colonnes pouvait bénéficier des nouvelles technologies parallèles. Cependant, ces bénéfices requièrent des efforts supplémentaires de développement et varient selon les applications. Le parallélisme doit donc être perçu comme un outil de plus dans une stratégie globale de résolution de problèmes et non comme une solution miracle à ces derniers.

RÉFÉRENCES BIBLIOGRAPHIQUES

AMDAHL, G. (1967). Validity of the single processor approach to achieving large-scale computer capabilities. *AFIPS Conference Proceedings*, **30**.

ARTHUR, J.L., FRENDEWEY, J.O., SCHUMICHRAST, R.T. (1987). *Experience with vectorizing a linear programming code.* Oregon State University, Corvallis, OR. USA.

BARTELS, R.H., GOLUB, G.H. (1969). The simplex method of linear programming using LU-decomposition. *Communications of the ACM*, **12**, 266–268.

BAUER, B.E. (1991). *Practical parallel programming.* Academic Press Limited. London, England.

BAZARAA, M.S., JARVIS, J.S., SHERALI, H.D. (1990). *Linear programming and network flows.* John Wiley & Sons, Inc., New York, USA.

BEASLEY, J.E. (1987). *Linear programming on CRAY supercomputers.* Department of management science, Imperial College, London, England.

BERTSEKAS, D.P., POLYMENAKOS, L.C. (1994). Parallel shortest path auction algorithms. *Parallel Computing*, **20**, 1221–1247.

CHRAIM, J. (1994). *Conception et implantation d'algorithmes parallèles pour la fabrication d'horaires d'équipages aériens.* Mémoire de maitrîse. École Polytechnique de Montréal, Canada.

CHRAIM, J., SANSÓ, B. (1995). *Solving very large airline planning problems : a parallel column generation approach.* Cahiers du GERAD G–95–52, École des Hautes Études Commerciales, Montréal, Canada.

COMER, D.E., STEVENS, D.L., (1993). *Internetworking with TCP/IP (volume III).* Prentice-Hall, New Jersey, USA.

CORRÊA, R., FERREIRA, A. (1995). On the effectiveness of synchronous parallel branch-and-bound algorithms. *Parallel Processing Letters*, **5**, No 3, 375–386.

COSNARD, M., ROBERT, Y., TOURANCHEAU, B. (1989). Evaluating speedups on distributed memory architectures. *Parallel Computing*, **10**, 247–253.

CPLEX Optimization, Inc. (1994). *Using the CPLEX callable Library.* Incline Village, NV, USA.

CRAINIC, T.G., GENDRON, B. (1994). Parallel branch-and-bound algorithms : survey and synthesis. *Operations Research*, **42**, No 6, 1042–1066.

DANTZIG, G.B., WOLFE, P. (1960). Decomposition principle for linear programs. *Operations Research*, **8**, 101–111.

DEPREZ, F., DONGARRA, J.J., TOURANCHEAU, B. (1995). Performance study of LU factorization with low communication overhead on multiprocessors. *Parallel Processing Letters*, **5**, No 2, 157–169.

DESROCHERS, M. (1986). *La fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes.* Thèse de doctorat. Université de Montréal, Centre de Recherche sur les Transports, Publication #470. Montréal, Canada.

DESROSIERS, J., DUMAS, Y., SOLOMON, M., SOUMIS, F. (1995). Time Constrained Routing and Scheduling. *Handbooks in Operations Research and Management Science*, **8** (Volume on Networks), North Holland, Amsterdam, 35–139.

DESAULNIERS, G., DESROSIERS, J., IOACHIM, I., SOLOMON, M.M., SOUMIS, F. (1994). *A unified framework for deterministic time constrained vehicle routing and crew scheduling problems.* Cahiers du GERAD G–94–46, École des Hautes Études Commerciales, Montréal, Canada.

DESAULNIERS, G., DESROSIERS, J., DUMAS, Y., MARC, S., RIOUX, B., SOLOMON, M.M., SOUMIS, F. (1997a). Crew Pairing at Air France. *European Journal of Operational Research 97*, 245–259.

DESAULNIERS, G., DESROSIERS, J., SOLOMON, M.M., SOUMIS, F. (1997b). Daily aircraft routing and scheduling. *Management science*, **43**, No 6, 841–855.

DESAULNIERS, G., LAVIGNE, J., SOUMIS, F. (1996). *Multi-depot vehicle scheduling problems with time windows and waiting costs.* Cahiers du GERAD G–96–33. École des Hautes Études Commerciales, Montréal, Canada.

DHALL, S.K., LAKSHMIVARAHAN, S. (1990). *Analysis and design of parallel algorithms : arithmetic and matrix problems.* McGraw-Hill, New York, USA.

DIJKSTRA, E.W. (1959). A note on two problems in connection with graphs. *Numerische Mathematik*, **1**, 269–271.

FOSTER, I. (1995). *Designing and building parallel programs.* Addison-Wesley Publishing Company, New York, USA.

GAMACHE, M. (1995). *Fabrication d'horaires mensuels pour les membres d'équipages en transport aérien.* Thèse de doctorat. École Polytechnique de Montréal, Canada.

GAMACHE, M., SOUMIS, F., MARQUIS, G., DESROSIERS, J. (1994). *A column generation approach for large scale aircrew rostering problems.* Cahiers du GERAD G–94–20, École des Hautes Études Commerciales, Montréal, Canada.

GAMACHE, M., SOUMIS, F., VILLENEUVE, D., DESROSIERS, J., GÉLINAS, É. (1997). *The preferential bidding system at Air Canada.* Cahiers du GERAD G–97–12, École des Hautes Études Commerciales, Montréal, Canada.

GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHEK, R., SUNDERAM, V. (1994). *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing.* MIT, Cambridge, MA, USA.

GOLDFARB, D., REID, J.K. (1977). A practicle steepest-edge simplex algorithm. *Mathematical Programming*, **12**, 361–371.

GOLUB, G.H., VAN LOAN, C.F. (1989). *Matrix computations.* The John Hopkins University Press, Baltimore, USA.

GUSTAFSON, J.L. (1988). Reevaluating Amdahl's Law. *Communications of the ACM*, **31**, 532–533.

HARRIS, P.M.J. (1973). Pivot selection methods of the Devex LP code. *Mathematical Programming*, **5**, 1–28.

HO, J.K., LEE, T.C., SUNDARRAJ, R.P. (1988). Decomposition of linear programs using parallel computation. *Mathematical programming*, **42**, 391–405.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. (1986). *Connection oriented transport protocol specification.* International Standard 8072, ISO, Suisse.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS, Inc. (1995). *Standard for information technology : threads extension.* Standard 1003.1c–1995, USA.

IOACHIM, I. (1994). Planification des itinéraires d'une flotte d'avions avec contraintes de synchronisation d'horaires. Thèse de doctorat. École Polytechnique de Montréal, Canada.

IOACHIM, I., GÉLINAS, S., SOUMIS, F., DESROSIERS, J. (1994). *A Dynamic Programming Algorithm for the Shortest Path Problem with Time Windows and Linear Node Costs.* Les Cahiers du GERAD G-94-24, École des Hautes Études Commerciales, Montréal, Canada.

JÁJÁ, J. (1992). *An introduction to parallel algorithms.* Addison-Wesley Publishing Company, USA.

KUMAR, V., GRAMA, A., GUPTA, A., KARYPIS, G. (1994). *Introduction to parallel computing : design and analysis of algorithms.* Benjamin-Cummings, USA.

LASRY, A. (1996). *Rotations d'équipage pour un transporteur aérien en milieu régional.* Mémoire de maîtrise. École des Hautes Études Commerciales, Montréal, Canada.

LEFFLER, S., MCKUSICK, M., KARELS, M., QUATERMAN, J., (1989). *The design and implementation of the 4.3BSD UNIX operating system.* Addison-Wesley, Reading, MA, USA.

LI, G., WAH, B. (1986). Coping with anomalies in parallel branch-and-bound algorithms. *IEEE Transactions on Computers*, **35**, No 6, 568–573.

LINGAYA, N.C. (1996). Communication privée.

LUSTIG, I.J., ROTHBERG, E. (1996). Gigaflops in linear programming. *Operation Research Letters*, **18**, 157–165.

NAZARETH, J.L. (1987). *Computer Solution of Linear Programs*. Oxford University Press, New York, New York, USA.

PIRES, J.M. (1995). *C library for concurrent processing in the Unix environment*. Projet de fin d'études, École Polytechnique de Montréal, Canada.

PURE SOFTWARE, Inc. (1996). *Quantify user's guide*. Sunnyvale, CA, USA.

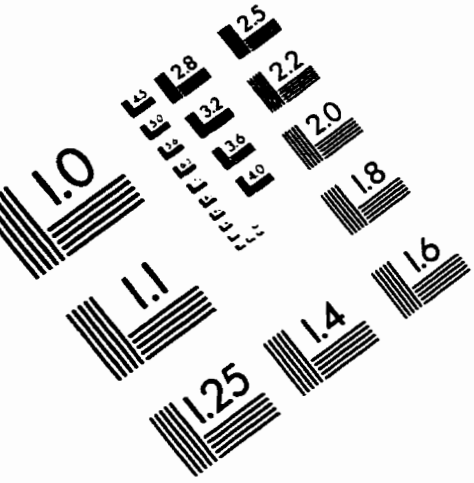
SCHACH, S.R. (1993). *Software engineering*. Irwin, Inc., and Aksen Associates, Inc., Boston, MA, USA.

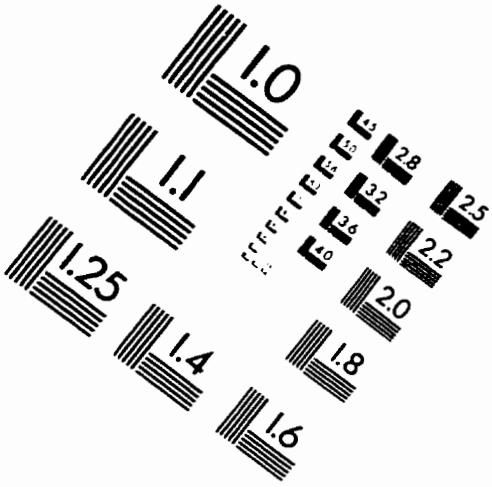
SUHL, U.H., SUHL, L.M. (1990). Computing sparse LU factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, **2**, No 4.

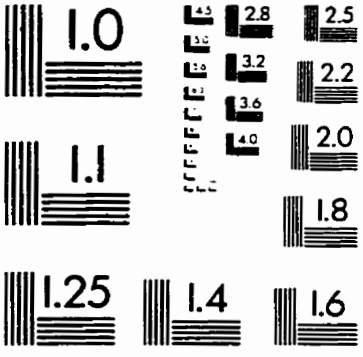
SUNSOFT, Inc. (1994). *Multithreaded programming guide*. Sunsoft Inc., USA.

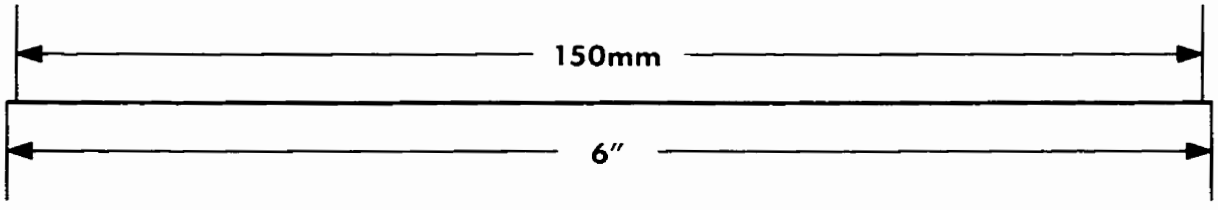
WILSON, G.V. (1995). *Practical parallel programming*. MIT, Cambridge, MA, USA.

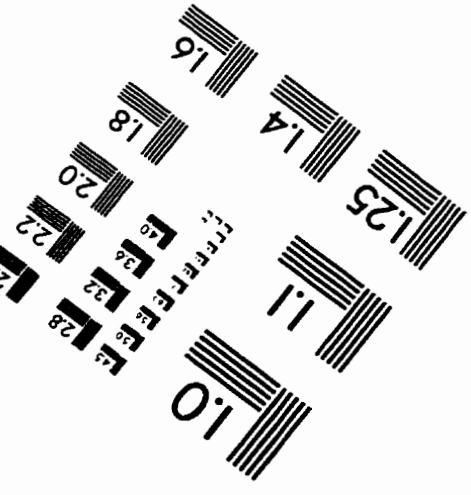
IMAGE EVALUATION
TEST TARGET (QA-3)

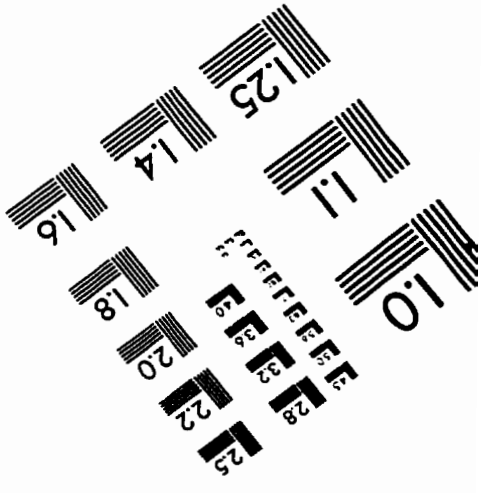












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989