Titre: Title:	Tournées de véhicules et satisfaisabilité logique
Auteur: Author:	Mihnea Stan
Date:	1996
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Stan, M. (1996). Tournées de véhicules et satisfaisabilité logique [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8938/

UNIVERSITÉ DE MONTRÉAL

TOURNÉES DE VÉHICULES ET SATISFAISABILITÉ LOGIQUE

MIHNEA STAN
DÉPARTEMENT DE MATHÉMATIQUES
ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIAE DOCTOR (Ph.D.)
(MATHÉMATIQUES DE L'INGÉNIEUR)
DÉCEMBRE 1996

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

TOURNÉES DE VÉHICULES ET SATISFAISABILITÉ LOGIQUE

présentée par: STAN Mihnea

en vue de l'obtention du diplôme de: Philosophiae Doctor

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

M. SMITH Benjamin T., Ph.D., président

Mme JAUMARD Brigitte, T. Doctorat, T.Hab., directrice de recherche

M. LAPORTE Gilbert, Ph.D., codirecteur de recherche

M. POTVIN Jean-Yves, Ph.D., membre

Mme ROUCAIROL Catherine, Thèse d'État, membre externe

À Irina

REMERCIEMENTS

J'aimerais exprimer ma gratitude à mes directeurs de recherche, Madame Brigitte Jaumard, professeure agrégée à l'École Polytechnique, et Monsieur Gilbert Laporte, professeur titulaire à l'École des Hautes Études Commerciales, pour m'avoir guidé tout au long de cette thèse. Je les remercie tout particulièrement de la confiance, la patience et le soutien moral et financier dont ils ont fait preuve à mon égard.

Aussi, je tiens à remercier mon co-directeur de recherche, Monsieur Michel Gendreau, professeur agrégé à l'Université de Montréal, pour m'avoir fait connaître et aimer les problèmes de transport.

J'aimerais remercier ceux qui m'ont permis de travailler durant cette thèse: Monsieur Benjamin Smith, professeur titulaire à l'École Polytechnique, Monsieur Gilles Savard, professeur agrégé à l'École Polytechnique, Monsieur Yvan Corriveau, directeur du groupe de recherche opérationnelle chez Air Canada, Monsieur Michel Sauvé, analyste principal chez Air Canada.

J'exprime ma gratitude au personnel du GERAD, et particulièrement à Monsieur Pierre Girard, analyste en informatique, et à Madame Nicole Paradis, responsable de l'édition, pour m'avoir aidé tout au long de mes études.

Finalement, je remercie Irina pour la patience dont elle a fait preuve.

RÉSUMÉ

Cette thèse propose de nouveaux algorithmes pour deux problèmes classiques de l'optimisation combinatoire: le transport et la satisfaisabilité logique. Plus précisement, nous sommes intéressés à la confection d'itinéraires de véhicules régis par des contraintes temporelles et au problème de satisfaisabilité.

Les problèmes de confection d'itinéraires de véhicules régis par des contraintes temporelles comportent la construction de routes optimales à partir d'un ou plusieurs dépôts vers un ensemble de clients, sous des contraintes temporelles. Ces problèmes appartiennent à la classe des problèmes NP-complets. Deux types de méthodes sont utilisés pour les résoudre: des méthodes exactes et des heuristiques. Les méthodes exactes ne sont pas pour le moment capables de résoudre des instances de tailles réelles.

Une première contribution de cette thèse est le développement de deux heuristiques efficaces, une pour le problème du voyageur de commerce avec fenêtres de temps et une autre pour le problème de fabrication de tournées de véhicules avec fenêtres de temps.

Les deux heuristiques ont été utilisées pour résoudre des problèmes tests et ont été comparées à d'autres heuristiques et méthodes exactes pour évaluer leurs performances. Pour le problème du voyageur de commerce avec fenêtres de temps, notre heuristique trouve la solution optimale pour 160 instances parmi les 194 résolues par deux méthodes exactes considérées. De plus, les résultats numériques montrent que notre heuristique est la seule méthode connue qui permet de résoudre dans un temps

raisonnable des instances caractérisées par de larges fenêtres de temps. L'approche de type recherche tabou que nous avons adopté pour le problème de fabrication de tournées de véhicules nous a permis d'améliorer la meilleure solution connue pour 20% des instances d'un ensemble de 56 problèmes tests fréquemment utilisés dans la littérature spécialisée.

La seconde partie de la thèse porte sur le problème de satisfaisabilité, un problème central de la théorie de la complexité et de l'intelligence artificielle. Le problème de satisfaisabilité consiste à déterminer l'existence d'une solution pour une équation booléenne donnée.

Une deuxième contribution que nous apportons est la résolution du problème de satisfaisabilité par un algorithme de type recherche tabou. Nous présentons aussi une méthode exacte pour le résoudre. La méthode exacte est basée sur le schéma de Davis-Putnam et utilise la technique de séparation et évaluation. Les facteurs qui influencent le comportement et la robustesse des deux approches ont été intensivement étudiés à l'aide de nombreux problèmes tests. Nous présentons des expériences comparatives avec quelques uns des meilleurs algorithmes de la littérature.

ABSTRACT

This dissertation addresses problems in transportation and logical satisfiability, two fields in combinatorial optimization. In particular, we are interested in time constrained vehicle routing problems and the satisfiability problem.

Time constrained vehicle routing problems can be described as the problems of designing optimal routes from one or several depots to a set of clients, subject to time constraints. These problems belong to the NP-complete class. Exact and approximate algorithms have been developed to solve them. Today, one can not rely on exact methods to solve to optimality real life instances.

A first contribution of this dissertation is the development of two efficient heuristics, one for the traveling salesman problem with time windows and another for the vehicle routing problem with time windows.

Both heuristics have been used to solve test problems from the literature. Their performances have been evaluated in comparison with other heuristics and exact methods. The heuristic we propose for the traveling salesman problem with time windows obtains better results than those generated by the best known heuristics. It also finds the optimal solution for 160 instances out of the 194 solved by two exact methods that we considered. Moreover, the computational results show that our heuristic is the only known method able to solve wide time windows instances in a reasonable time. For the vehicle routing problem with time windows we developed a tabu search algorithm. It allowed us to improve the best known solution in 20% of the instances from a 56 test problems set frequently used as a benchmark in the

literature.

The second part of the dissertation deals with the satisfiability problem, a central problem in complexity theory and artificial intelligence. The satisfiability problem is the first problem proved to belong to the NP-complete class. It consists in determining the existence of a solution for a given boolean equation.

A second contribution of this dissertation is the development of a tabu search heuristic for the satisfiability problem. We also present an exact method for its solution. The exact method is based on the Davis-Putnam scheme and uses the branch and bound technique. Extensive computational tests have been performed to study the factors that influence the performance and the robustness of the two approaches. We report computational comparisons with some of the most efficient algorithms of the literature.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

INTRODUCTION

L'objet de cette thèse est l'étude de deux problèmes de l'optimisation combinatoire: la fabrication d'itinéraires de véhicules régis par des contraintes temporelles et le problème de satisfaisabilité. Les deux problèmes appartiennent à la classe NP-complet. Dernièrement, beaucoup d'efforts ont été consacrés à la mise au point de méthodes pour les résoudre. Ces efforts s'expliquent par leur importance théorique et pratique. En effet, la fabrication d'itinéraires des véhicules a de nombreuses applications: le service de courier (bancaire, postal), la fabrication d'horaires pour les autobus scolaires, la livraison juste à temps des usines de fabrication. De plus, puisque les itinéraires de véhicules sont utilisés répétitivement, l'importance économique d'une bonne solution est évidente. Le problème de satisfaisabilité occupe une place centrale dans la logique mathématique et la théorie de la complexité. Il a de nombreuses applications, comme la détection des incohérences dans les circuits logiques.

Dans cette thèse, nous utilisons une méthode ayant depuis quelques années permis de résoudre des problèmes complexes de l'optimisation combinatoire: la recherche tabou. Cette méthode a été indépendamment développée par Glover [28] et Hansen [32]. Un avantage important de la recherche tabou réside dans son caractère universel. Ceci permet son application à un très large spectre de problèmes. Parmi les problèmes où elle a été utilisée avec succès on note: le problème du voyageur de commerce (Knox et Glover [47]), la coloration d'un graphe (Hertz et de Werra [35]), la satisfaisabilité maximale (Hansen et Jaumard [33]). L'utilisation de la recherche tabou pour une variété de problèmes a mené à des solutions optimales ou presque optimales et un temps de calcul inférieur à d'autres heuristiques de la littérature. L'utilisation d'une méthode de recherche tabou demande la définition du voisinage d'une solution, de même que la définition d'un mécanisme permettant

de sortir des points de minima locaux. Les Chapitres 2 et 3 présentent comment le caractère universel de la recherche tabou a été adapté pour tenir compte des caractéristiques propres au problème de fabrication des tournées de véhicules avec fenêtres de temps et au problème de satisfaisabilité.

Dans le premier chapitre nous avons développé un algorithme pour le problème du voyageur de commerce avec fenêtres de temps. Ce problème comprend la construction d'un chemin à coût minimum pour un véhicule qui visite un ensemble de clients. Chaque client est visité une seule fois et le service à un client peut débuter seulement à l'intérieur de sa fenêtre de temps. Des méthodes de résolution basées sur la programmation dynamique ont été proposées par Christofides, Mingozzi et Toth [8] et Psaraftis [58], [59]. Baker [4] a développé un algorithme de séparation et évaluation où des bornes inférieures sont calculées à partir du problème dual de la relaxation linéaire du modèle proposé. Ces méthodes ont produit de bons résultats sur des instances à 50 clients dont la largeur des fenêtres de temps reste relativement réduite. La complexité du problème augmente considérablement avec la largeur des fenêtres de temps. Lorqu'on veut résoudre des instances caractérisées par de larges fenêtres de temps, les méthodes exactes sont moins intéressantes car souvent elles rencontrent le phénomène de dégénérescence. À partir du chemin vide, l'heuristique que nous proposons insère les clients les uns après les autres dans le chemin partiel. À chaque insertion une optimisation locale du chemin partiel est effectuée. Si le processus d'insertion prend fin avec l'obtention d'un chemin réalisable, alors nous l'améliorons par des optimisations locales répétées. Les expériences numériques comparatives aux méthodes exactes montrent que l'algorithme proposé atteint la solution optimale ou une solution presque optimale pour la majorité des instances considérées. Elle permet aussi la résolution des instances caractérisées par de larges fenêtres de temps.

Le deuxième chapitre traite le problème de fabrication des tournées de véhicules avec fenêtres de temps. Le problème comporte deux objectifs. Le premier objectif est de trouver le nombre minimum de véhicules nécessaires pour desservir un ensemble de clients. Chaque client doit être visité à l'intérieur d'une fenêtre de temps par un seul véhicule. Un deuxième objectif est de minimiser la distance totale parcourue. Chaque client a un temps de service et une demande. Les véhicules sont homogènes et de capacité limitée. Des résultats antérieurs indiquent que les meilleures méthodes exactes sont seulement capables de résoudre des instances de taille réduite. Dans ce contexte, des heuristiques qui trouvent de bonnes solutions réalisables dans un temps raisonnable offrent une alternative attrayante. Nous nous sommes arrêtés sur un algorithme de type recherche tabou. De manière générale, à une itération tabou on essaie de diminuer la fonction objectif en déplacant un client d'une route à une autre. Si le changement de route a été accepté, alors on empêche le client choisi de retourner dans sa route d'origine pour un certain nombre d'itérations.

La définition du voisinage, lorsqu'une insertion ou retrait est considéré, représente une constante des algorithmes développés aux Chapitres 1 et 2. Les éléments du voisinage sont choisis par rapport au temps et à la distance euclidienne.

Au troisième chapitre nous proposons une approche de type recherche tabou pour le problème de satisfaisabilité. La fonction objectif est définie par le nombre de clauses non satisfaites. Une itération tabou choisit la variable qui, complementée, fait diminuer le plus l'objectif. Cette variable ne peut plus être complementée pour un certain nombre d'itérations. Pour éviter que cette méthode de descente reste dans un point de minimum local, nous avons développé des techniques de remontée. Les critères de diversification, fréquence et aspiration ont été considérés.

Nous présentons aussi une implantation efficace d'une méthode exacte basée sur le schéma de Davis-Putnam. La méthode exacte utilise la technique de séparation et évaluation et le problème de satisfaisabilité quadratique comme relaxation. Plusieurs critères de branchement ont été implantés et évalués.

CHAPITRE 1

Le problème du voyageur de commerce avec fenêtres de temps

Le problème du voyageur de commerce avec fenêtres de temps (PVCFT) consiste à trouver un chemin de coût minimum parcouru par un véhicule pour desservir un ensemble de clients. Le chemin doit commencer et finir au dépôt et chaque client doit être visité une seule fois à l'intérieur de sa fenêtre de temps. Ce type de problème a de nombreuses applications dans l'industrie. On note les services de courrier (bancaire, postal), la fabrication d'horaires pour les autobus scolaires, la fabrication de trajectoires pour les robots industriels. Le PVCFT est un problème NP-complet et nous présentons dans ce chapitre une heuristique d'insertion généralisée pour le résoudre.

1.1 Le problème

Le PVCFT est défini sur un graphe $G = (V, A)$ où $V = N \cup \{v_0, v_{n+1}\}$ est l'ensemble des noeuds, $N = \{v_1, v_2 \ldots, v_n\}$, et $A = \{(v_i, v_j) : v_i \in N \cup \{v_0\}, v_j \in N \cup \{v_{n+1}\}, i \neq j\}$ l'ensemble des arcs. Un noeud $v_i \in N$ indique un client. Les noeuds v_0 et v_{n+1} représentent, respectivement, la sortie du dépôt et l'entrée au dépôt. Chaque noeud v_i doit être visité durant une fenêtre de temps $[a_i, b_i]$ et a un temps de service. À chaque arc (v_i, v_j) on associe une distance c_{ij} et un temps de parcours t_{ij} non négatifs. On fait l'hypothèse, courante pour ce type de problème, que les temps de parcours t_{ij} sont égaux aux distances c_{ij}. Pour la simplicité, on suppose que le temps de parcours, ou la distance, c_{ij} de v_i à v_j inclut le service à v_i. Le PVCFT consiste à trouver le chemin orienté de coût minimum qui commence à v_0, finit à

v_{n+1}, et qui passe par chaque noeud de l'ensemble $V \setminus \{v_0, v_{n+1}\}$ exactement une seule fois; sur ce chemin, le noeud v_i doit être visité au plus tard au temps b_i et ne peut pas être quitté avant le temps a_i. On remarque que l'attente est permise et donc le noeud v_i peut être atteint avant le temps a_i. Selon la fonction objectif, deux versions du problème sont définies. La première version a comme objectif de minimiser le temps d'arrivée au noeud v_{n+1}. L'objectif de la deuxième version du problème est de minimiser la distance le long du chemin. Dans ce chapitre on considère la deuxième version du problème.

La formulation mathématique comporte deux types de variables: l'ensemble de variables binaires $X = \{x_{ij} : (v_i, v_j) \in A\}$ et l'ensemble de variables temporelles $T = \{t_i : i \in V\}$. La variable x_{ij} est égale à 1 si l'arc (v_i, v_j) est utilisé dans une solution et 0 sinon. La variable t_i indique le temps de début du service au noeud $v_i, v_i \in N \cup \{v_0\}$, alors que t_{n+1} représente le temps d'arrivée au dépôt.

Le chemin de coût minimum qui débute au dépôt, visite une seule fois tous les noeuds de N à l'intérieur de leur fenêtres de temps et retourne au dépôt avant b_{n+1} peut être formulé comme suit:

$$\min_{x_{ij}} \sum_{(v_i,v_j)\in A} c_{ij}x_{ij}$$

$$s.l.c. \quad \sum_{v_j\in N\cup\{v_{n+1}\}} x_{ij} = 1 \qquad \forall v_i \in N \qquad (1.1)$$

$$\sum_{v_j\in N} x_{0j} = 1 \qquad (1.2)$$

$$\sum_{v_i\in N\cup\{v_0\}} x_{ij} - \sum_{v_i\in N\cup\{v_{n+1}\}} x_{ji} = 0 \quad \forall v_j \in N \qquad (1.3)$$

$$\sum_{v_i\in N} x_{in+1} = 1 \qquad (1.4)$$

$$x_{ij}(t_i + t_{ij} - t_j) \leq 0 \qquad \forall (v_i, v_j) \in A \qquad (1.5)$$

$$a_i \leq t_i \leq b_i \qquad \forall v_i \in N \cup \{v_{n+1}\} \qquad (1.6)$$

$$x_{ij} \geq 0 \qquad \forall (v_i, v_j) \in A \qquad (1.7)$$

$$x_{ij} \in \{0, 1\} \qquad \forall (v_i, v_j) \in A \qquad (1.8)$$

La fonction objectif minimise le coût total. La contrainte (1.1) assure qu'un client est visité une seule fois. Les contraintes (1.2)-(1.4) sont les contraintes de conservation de flot. Les contraintes (1.5) et (1.6) garantissent la réalisabilité de l'horaire.

Parfois, il est possible de simplifier le problème. Deux noeuds v_i et v_j sont *compatibles* si et seulement si

$$a_i + c_{ij} \leq b_j.$$

Si cette condition n'est pas respectée, l'arc (v_i, v_j) est enlevé de l'ensemble A. De même, par la vérification répétitive de quelques conditions simples, la fenêtre de temps de chaque noeud v_k peut être réduite. La réduction des fenêtres de temps est très importante pour les méthodes exactes basées sur la programmation dynamique, car elles réduisent l'espace d'états possibles. Les règles suivantes sont adaptées de Desrochers, Desrosiers et Solomon [13]:

1. Temps d'arrivée au plus tôt des prédécesseurs:

$$a_k = \max\{a_k, \min_{(v_i,v_k)\in A}\{a_i + c_{ik}\}\};$$

2. Temps d'arrivée au plus tôt aux successeurs:

$$a_k = \max\{a_k, \min\{b_k, \min_{(v_k,v_j)\in A}\{a_j - c_{kj}\}\}\};$$

3. Temps de départ au plus tard des prédécesseurs:

$$b_k = \min\{b_k, \max\{a_k, \max_{(v_i,v_k)\in A}\{b_i + c_{ik}\}\}\};$$

4. Temps de départ au plus tard aux successeurs:

$$b_k = \min\{b_k, \max_{(v_k,v_j)\in A}\{b_j - c_{kj}\}\};$$

En appliquant ces tests, on pourrait identifier de nouveaux noeuds incompatibles et enlever de l'ensemble A les arcs correspondants. De même, on pourrait découvrir durant l'application de ces conditions que le problème n'est pas réalisable.

Étant donné l'importance du PVCFT et les limitations naturelles des algorithmes exacts pour ce problème, le besoin des heuristiques efficaces subsiste. À notre avis, il y a quatre grands domaines d'applications pour ces méthodes:

- l'obtention de bonnes solutions pour les problèmes d'ordonnancement sur une machine;
- la post-optimisation des tournées pour les problèmes de fabrication des tournées de véhicules avec fenêtres de temps (PFTVFT);
- l'obtention d'une solution initiale pour les algorithmes génétiques et ceux basés sur la recherche locale;

- l'obtention d'une borne supérieure initiale dans un algorithme de séparation et évaluation.

Notre contribution est de fournir une telle heuristique. Elle est développée autour de GENIUS, un algorithme approché proposé par Gendreau, Hertz et Laporte [23] pour le problème du voyageur de commerce (PVC). La revue de la littérature est présentée à la section suivante. L'heuristique est décrite à la section 2, suivie par des résultats numériques à la section 3, et les conclusions à la section 4.

1.2 Revue de la littérature

Le PVCFT a des applications dans l'ordonnancement de la production, plus précisément lorsque n tâches aux temps de début a_i, échéances b_i et temps de production c_{ij} asymétriques doivent être ordonnancées sur une machine. L'objectif est de déterminer une séquence réalisable des tâches qui minimise le temps au plus tard. Desrosiers et al. [15] décrivent une application du PVCFT à la fabrication des tournées pour les véhicules automatiques guidés. Le PVCFT peut-être interprété comme un sous-problème du problème de fabrication des tournées de véhicules avec fenêtres de temps (PFTVFT) où chaque noeud doit être affecté à une des routes qui commencent et finissent au dépôt. Une fois que les noeuds ont été affectés aux routes, on peut déterminer les séquences individuelles pour les routes en résolvant un PVCFT pour chaque route.

Le PVCFT est un problème difficile d'optimisation combinatoire. Savelsbergh [62] a montré que même déterminer si une solution réalisable existe est un problème NP-complet. Dans la littérature spécialisée, le PVCFT a reçu une attention limitée.

Dumas et al. [18] décrivent une méthode exacte basée sur la programmation

dynamique. Soient $N' = N\backslash\{v_{n+1}\}$ et $F(S, v_i, t)$ le chemin de coût minimum qui commence au noeud v_0, passe par tous les noeuds de $S \subseteq N'$ une seule fois, finit au noeud $v_i \in S$, et prêt à commencer le service au noeud v_i au plus tôt au temps t. Un processus d'étiquetage à deux dimensions est nécessaire, car le coût $F(S, v_i, t)$ et le temps t doivent être considérés séparement. La fonction $F(S, v_i, t)$ est calculée à l'aide de la formule de récurrence:

$$F(S, v_j, t) = \min_{(v_i, v_j) \in A}\{F(S\backslash\{v_j\}, v_i, t') + c_{ij} : t \geq t' + c_{ij}, a_i \leq t' \leq b_i\}$$

pour tout $S \subseteq N', v_j \in S$ et $a_j \leq t \leq b_j$. La formule de récurrence est initialisée par:

$$F(\{v_0, v_j\}, v_j, t) = \begin{cases} c_{0j}, & \text{si } (v_0, v_j) \in A, \\ \infty, & \text{sinon,} \end{cases}$$

où $a_j \leq t \leq b_j, t = \max\{a_0 + c_{0j}, a_j\}$.

La solution optimale du PVCPF est donnée par:

$$\min_{(v_i, v_{n+1}) \in A} \min_{a_i \leq t \leq b_i} \{F(N', v_i, t) + c_{in+1} : t + c_{in+1} \leq b_{n+1}\}.$$

Les formules ci-dessus définissent un algorithme de plus court chemin sur un graphe dont les noeuds sont identifiés aux états (S, v_i, t) et les arcs représentent des transitions d'un état à l'autre. À chaque étape $s, s = 1, \ldots, n$ de l'algorithme un chemin de dimension s est construit. Comme il y a plusieurs chemins qui finissent au noeud v_i en passant par tous les noeuds de S, les auteurs gardent seulement les chemins Pareto optimaux. Pour améliorer l'efficacité de l'algorithme, Dumas et al. [18] introduisent des tests d'élimination, basés sur les fenêtres de temps, qui réduisent d'une manière significative l'ensemble des états et le nombre de transitions.

Langevin et al. [49] formulent le PVCFT comme un problème de flot à deux commodités. Dans la formulation, une quantité de la première commodité est livrée

à chaque client et une quantité de la deuxième commodité est récupérée de chaque client. Si on considère une ressource, alors on peut identifier la première commodité avec la quantité qui reste à consommer et la deuxième commodité avec la valeur cumulée consommée de la ressource. On note que la somme des deux flots est constante à chaque noeud et égale à la valeur de la ressource au début. Pour le PVCFT le temps représente la ressource, et les contraintes de fenêtres sont facilement formulées à l'aide du flot qui doit être récupéré de chaque noeud.

Les auteurs ajoutent à cette formulation des contraintes d'élimination de sous-tours et appliquent l'algorithme du simplexe à la relaxation linéaire du problème. Une solution entière est obtenue en utilisant une méthode de séparation et évaluation. Le branchement est effectué sur les arcs du réseau et les noeuds de l'arbre de branchement sont traités selon le critère le meilleur d'abord.

Des heuristiques fondées sur les techniques d'échange ont été développées par Savelsbergh ([62], [63]) pour deux fonctions objectifs différentes.

La plupart des efforts de recherche se sont concentrés sur le PFTVFT. Des heuristiques ont été proposées par Solomon [65], Desrochers et al. [14] et Solomon, Baker et Shaffer [66]. Plus récemment, des algorithmes plus performants ont été développés. Ils sont basés sur la recherche tabou (Potvin et al. [57]) et la recherche génétique (Thangiah [69], Potvin et Bengio [56]). Des algorithmes exacts pour le PFTVFT ont été présentés par Kolen, Rinnooy Kan et Trienekens [48] et par Desrochers, Desrosiers et Solomon [13]. La dernière référence décrit un algorithme basé sur la méthode de génération de colonnes testé sur des instances ayant jusqu'à 100 noeuds. Les résultats sont spécialement bons pour les problèmes dont les contraintes sont très serrées car le nombre de colonnes réalisables est réduit. Le

PFTVFT sera présenté plus en détail au chapitre suivant.

1.3 L'algorithme

Avant de commencer la description de l'heuristique qu'on propose pour le PVCFT, on rappele les principales caractéristiques de l'algorithme GENIUS pour le PVC symétrique. Pour détails et illustrations, consulter Gendreau, Hertz et Laporte [23]. GENIUS consiste d'une phase de construction de routes (GENI, pour *generalized insertion*), suivie d'une phase de post-optimisation (US, pour *unstringing* et *stringing*). GENI débute avec une route qui comporte trois noeuds. À chaque itération, elle insère un noeud v dans la route partiellement construite exécutant en même temps une réoptimisation locale de la route. Pour chaque noeud $v \in V$, soit $N_p(v)$ son voisinage. Il est défini comme l'ensemble des p noeuds déjà dans la route qui sont les plus proches de v (p représente un paramètre d'entrée). Si la route contient moins de p noeuds, alors ces noeuds définissent $N_p(v)$. Le noeud v est inséré entre les noeuds v_i et v_j appartenant à $N_p(v)$. Pour une orientation donnée de la route, soit v_{i-1} et v_{i+1} le prédécesseur et le successeur du noeud v_i. Il y a deux types d'insertion GENI. Chacun est essayé pour les deux orientations de la route et pour des différents choix des v_i, v_j, v_k et v_l. La meilleure insertion est acceptée.

- *Type 1*: Soit $v_i, v_j \in N_p(v)(i \neq j), v_k \in N_p(v_{i+1})(k \neq i, j)$. Enlever les arcs $(v_i, v_{i+1}), (v_j, v_{j+1})$ et (v_k, v_{k+1}); insérer les arcs $(v_i, v), (v, v_j), (v_{i+1}, v_k)$ et (v_{j+1}, v_{k+1}). Ces modifications inversent les chemins $(v_{i+1}, \ldots, v_j)$ et $(v_{j+1}, \ldots, v_k)$.

- *Type 2*: Soit $v_i, v_j \in N_p(v)(i \neq j), v_k \in N_p(v_{i+1})(k \neq j, j+1), v_l \in N_p(v_{j+1})(l \neq i, i+1)$. Enlever les arcs $(v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1})$ et (v_{k-1}, v_k); insérer les arcs $(v_i, v), (v, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1})$ et (v_{i+1}, v_k). Ces modifications inversent les chemins $(v_{i+1}, \ldots, v_{l-1})$ et $(v_l, \ldots, v_j)$.

Lorsque tous les noeuds font partie de la route, la phase de post-optimisation, US, est appliquée. Dans US chaque noeud est enlevé à son tour du chemin en utilisant l'inverse de GENI, et inséré de nouveau dans le chemin en utilisant GENI. L'algorithme prend fin lorsqu'après avoir appliqué ces mouvements à tous les noeuds de la route on n'obtient pas une meilleure solution. Les résultats obtenus par Gendreau, Hertz et Laporte [23] et par Renaud, Boctor et Laporte [60] indiquent que GENIUS produit des solutions très proches de l'optimalité.

On peut étendre jusqu'à un certain point les principes qui se trouvent derrière GENIUS pour le PVCFT, car il faut tenir compte de la structure spécifique du problème. D'abord, la solution représente un chemin de v_0 à v_{n+1} et non une route comme pour le PVC. Deuxièmement, même si les temps de transport sont symétriques, ce chemin est orienté. Si un chemin orienté $(v_i, \ldots, v_j)$ est réalisable, le chemin inverse $(v_j, \ldots, v_i)$ peut ne pas respecter les contraintes de fenêtres de temps. Pour tenir compte de la nature asymétrique du PVCFT, on remplace le voisinage $N_p(v)$ par les voisinages orientés $N_p^+(v)$ et $N_p^-(v)$ contenant, respectivement, les plus proches p successeurs et prédécesseurs du noeud v. Ces voisinages sont définis ci-dessous. L'algorithme que nous avons développé garde en tout temps la réalisabilité des solutions par rapport aux fenêtres de temps. Cette approche est différente de certains algorithmes conçus pour les problèmes de confection de tournées de véhicules qui permettent des solution non-réalisables comme, par exemple, Gendreau, Hertz et Laporte [24]. La raison de ce choix, confirmée par des tests préliminaires, est que généralement il est très difficile de retrouver la réalisabilité à partir d'une solution qui ne respecte pas les contraintes de fenêtres de temps. En conséquence, le nombre d'insertions réalisables est plus restreint que dans le cas du PVC, surtout en tenant compte du fait que l'insertion d'un noeud dans un chemin

partiellement construit a un impact sur tous ses successeurs. À une étape quelconque de l'algorithme on pourrait être dans l'impossibilité d'insérer un nouveau noeud dans le chemin. Une procédure de retour en arrière est alors appliquée.

Malgré tout, la réalisabilité finale n'est pas garantie et l'algorithme peut terminer en laissant quelques noeuds non insérés. Finalement, les voisinages basés sur la distance ne sont pas très efficaces pour le PVCFT car deux noeuds proches en distances peuvent être incompatibles ou avoir des fenêtres de temps très éloignées. En conséquence, dans la définition du voisinage on tient compte des deux dimensions, spatiale et temporelle. On explique maintenant avec plus de détails comment GENIUS a été adapté pour le PVCFT.

Le voisinage doit tenir compte des distances et fenêtres de temps. Étant donné deux noeuds v_i et v_j, on définit la proximité de leurs fenêtres de temps par:

$$r_{ij} = \mid \min\{b_j, b_i + c_{ij}\} - \max\{a_j, a_i + c_{ij}\} \mid .$$

Cette proximité indique la flexibilité de visiter v_j après v_i. Une valeur élevée de r_{ij} montre que v_j n'est pas le meilleur candidat comme successeur immédiat de v_i dans un chemin. On note aussi que si $b_i + c_{ij} \geq a_j$, alors r_{ij} calcule la difference entre le temps au plus tard et le temps au plus tôt de début du service au noeud v_j. Au cas contraire, r_{ij} indique le temps minimum d'attente au noeud v_j.

Une manière naturelle de définir une *pseudo-distance* d_{ij} entre v_i et v_j est d'utiliser une combinaison convexe entre c_{ij} et r_{ij}:

$$d_{ij} = \alpha c_{ij} + (1 - \alpha) r_{ij}, \alpha \in [0, 1].$$

Des tests préliminaires ont montré que cette règle mène à des résultats instables: fixer α est problématique car les distances sont plus influentes pour certaines instances, alors que pour d'autres, les fenêtres de temps sont le facteur critique. Une façon plus satisfaisante de définir les voisinages est d'inclure dans $N_p^+(v)$ les p_1 plus proches successeurs de v déjà dans le chemin par rapport à c_{ij}, et les p_2 plus proches successeurs de v déjà dans le chemin par rapport à r_{ij}, avec la condition $p_1 + p_2 = p$. D'une manière similaire on définit $N_p^-(v)$ en termes de prédécesseurs. De plus, v_0 est inclus dans $N_p^-(v)$ et v_{n+1} est inclus dans $N_p^+(v)$.

Lors de l'insertion d'un noeud v dans le chemin courant il faut maintenir la réalisabilité. À cette fin, on calcule pour chaque noeud v_i *le temps critique de départ* z_i. Ce temps critique est mis à jour au long de l'algorithme. Ces coefficients mesurent le temps au plus tard des départs des noeuds pour garder le chemin réalisable. Les coefficients z_i sont calculés à rebours, à partir du noeud v_{n+1}:

$$\begin{aligned} z_{n+1} &= b_{n+1} \\ z_i &= \min\{b_i, z_j - c_{ij}\} \end{aligned}$$

si v_i est le prédécesseur de v_j sur le chemin courant.

Chaque fois qu'un noeud v est inséré dans le chemin courant en utilisant la procédure GENI, l'ordre de quelques noeuds peut être modifié. Considérons le chemin $(v_0, \ldots, v_{i_\alpha}, \ldots, v_{i_\beta}, \ldots, v_{n+1})$ obtenu après une insertion et $(v_{i_\alpha}, \ldots, v_{i_\beta})$ la portion du chemin modifiée par l'insertion. Une condition nécessaire pour qu'une insertion potentielle soit réalisable est que les temps d'arrivée aux noeuds, t_i, satisfassent:

$$t_i \leq b_i, i = v_{i_\alpha}, \ldots, v_{i_\beta}$$

En plus, l'insertion doit être réalisable pour tous les noeuds suivant v_{i_β}:

$$t_{i_{\beta+1}} = t_{i_\beta} + c_{i_\beta i_{\beta+1}} \leq z_{i_{\beta+1}}$$

Contrairement à GENI, les noeuds ne sont pas insérés dans le chemin aléatoirement, car cette règle rend l'insertion des derniers noeuds problématique. À la place, les noeuds sont d'abord classés selon un critère qui mesure la difficulté de leur insertion et ensuite insérés en ordre non croissant de difficulté. Plusieurs critères peuvent être utilisés pour ordonner les noeuds:

- ordre non décroissant des largeurs des fenêtres de temps, $b_i - a_i$;
- ordre non décroissant des débuts des fenêtres de temps, a_i;
- ordre non décroissant des fins des fenêtres de temps, b_i.

Des résultats préliminaires ont montré que la première règle est la plus efficace.

Pour faire une insertion, les noeuds non encore insérés sont considérés un à la fois, à partir de la liste ordonnée. Le premier noeud pour lequel il existe une insertion réalisable est inséré dans le chemin dans sa meilleure position. Lorsqu'il n'y a plus de noeuds qui peuvent être insérés d'une manière réalisable dans le chemin, la procédure suivante de retour en arrière est appliquée. Soit S la liste ordonnée des noeuds non insérés. À partir du successeur de v_0, enlever chaque noeud v_i un à la fois, le placer à la fin de la liste S, et essayer d'insérer dans le chemin chaque noeud de S, sauf ceux qui viennent d'être enlevés du chemin, en préservant l'ordre de la liste:

- si une insertion est réalisable, on l'implémente et la procédure standard d'insertion s'applique;

- s'il n'y a pas d'insertion réalisable, le même processus est répété avec le successeur de v_i, jusqu'à ce que v_{n+1} soit atteint.

Éventuellement, une insertion sera réalisable car au pire des cas tous les noeuds sauf v_0 et v_{n+1} peuvent être enlevés. Pour éviter le cyclage, l'algorithme compte le nombre de fois qu'un noeud v_i est remis dans la liste S. Soit θ_i cette valeur. Le processus se termine avec une solution non réalisable lorsque la valeur θ_i est égale à θ (un paramètre d'entrée) pour un indice i.

Si l'algorithme réussit à déterminer une solution réalisable, une procédure de post-optimisation, basée sur US, est appliquée. Chaque noeud v_i est enlevé à son tour du chemin auquel on applique une optimisation locale par recours à l'inverse de GENI (voir Gendreau, Hertz et Laporte [23]). Ensuite v_i est de nouveau inséré dans le chemin en utilisant GENI. Ce processus est employé d'une manière itérative tant qu'aucun des noeuds v_i n'apporte d'amélioration à la solution courante. Une modification locale du chemin est considérée seulement si elle maintient la réalisabilité des contraintes de fenêtres de temps pour tous les noeuds.

On décrit maintenant l'algorithme étape par étape:

- Étape 1: *initialisation.* Insérer tous les noeuds de $V \setminus \{v_0, v_{n+1}\}$ en ordre non décroissant de leur largeur des fenêtres de temps dans une liste S. Construire un premier chemin (v_0, v_{n+1}).
- Étape 2: *vérification de la réalisabilité.* Si $S = \emptyset$, alors on a trouvé une solution réalisable. Aller à l'Étape 5. Sinon, soit $k = 1$ et $s = |S|$.
- Étape 3: *tentative d'insertion.* Essayer d'insérer le $k^{ième}$ noeud de S dans le chemin, entre deux de ses p plus proches voisins en utilisant GENI. Si on

ne peut pas trouver d'insertion réalisable, alors on incrémente k d'une unité, $k = k + 1$; si $k = s + 1$, aller à l'Étape 4; si $k \leq s$, répéter l'Étape 3. Si on identifie au moins une insertion réalisable, implémenter la meilleure, enlever de S le noeud justement inséré dans le chemin, mettre à jour le chemin et aller à l'Étape 2.

- Étape 4: *retour en arrière.* On n'a pas identifié d'insertion réalisable à l'Étape 3. Enlever du chemin le noeud v_i, le successeur immédiat de v_0 et le mettre en dernière position dans la liste S. Si v_i a été enlevé du chemin de cette manière pour la θ_i fois ($\theta_i = \theta_i$), l'algorithme s'arrête sans avoir trouvé de solution réalisable. Sinon, remettre k à 1 et aller à l'Étape 3.

- Étape 5: *post-optimisation.* Appliquer au circuit une procédure de post-optimisation similaire à US: à partir du successeur immédiat de v_0, enlever du chemin chaque noeud à son tour en utilisant l'inverse de GENI et l'insérer de nouveau dans le chemin en utilisant GENI. Si cette solution est meilleure que la solution courante, elle devient la solution courante. Sinon, elle est rejetée. Si on n'a pas réussi à améliorer la solution courante après avoir enlevé et inséré à nouveau chaque noeud, l'algorithme s'arrête. Sinon, répéter cette étape à partir du successeur de v_0.

Dans notre implémentation, un chemin est représenté par une liste circulaire doublement chaînée. En plus, on garde pour chaque noeud un pointeur vers sa position dans le chemin courant. Ceci permet de changer les voisins immédiats (le successeur et le prédécesseur) d'un noeud en temps constant, $O(1)$. Ceci implique que pour insérer ou enlever un noeud d'un chemin, une fois que le type d'insertion ou de retrait a été déterminé, la complexité au pire cas est de l'ordre $O(n)$.

À l'Étape 1 on utilise l'algorithme de "quicksort" pour trier les noeuds. La complexité de cet algorithme est, au pire cas, de l'ordre $O(n^2)$, mais malgré ce résultat théorique, l'algorithme de quicksort a en pratique une complexité d'ordre $O(nlogn)$. La complexité de l'Étape 3 est influencée par le nombre d'insertions possibles. Ce nombre est déterminé à partir de la définition du voisinage et il est de l'ordre $O(p^4)$, car il faut considérer tous les choix possibles pour v_i, v_j, v_k et v_l. Donc, la complexité d'une insertion est de l'ordre $O(n+p^4)$ et alors la complexité de l'Étape 3 est de l'ordre $O(n^2 + np^4)$. L'Étape 4 a la même complexité que l'Étape 3, car il s'agit d'enlever un noeud, mécanisme inverse de l'insertion. L'Étape 5 représente un processus itératif de retraits et d'insertions. Sa complexité est toujours de l'ordre $O(n^2+np^4)$. On conclut que la complexité de l'algorithme est de l'ordre $O(n^2+np^4)$.

1.4 Résultats numériques

L'algorithme qu'on vient de décrire a été programmé en Pascal et tourné sur une machine SUN SS20/61 (SPECint 92 98.2, SPECfp 92 107.2). Des tests préliminaires ont été effectués pour déterminer les valeurs des paramètres p_1, p_2 et θ_i. Les résultats ont montré qu'en fixant $p_1 = 8, p_2 = 7$ et $\theta_i = 4$ pour tout i, l'algorithme est très efficace. Ensuite, l'algorithme a été exécuté sur 370 problèmes tests, groupés en trois catégories:

- Les instances de type PVCFT résolues d'une manière optimale par Langevin et al. [49].
- Les instances de type PVCFT résolues d'une manière optimale par Dumas et al. [18].
- Une extension des problèmes de Dumas et al. [18].

Notre algorithme trouve des solutions optimales ou très proches de l'optimum pour la plupart des problèmes proposés par Langevin et al. [49]. Sur les 82 problèmes résolus à l'optimalité, notre algorithme trouve la même solution pour 77 instances. Parmi les 90 problèmes tests considérés, l'algorithme proposé par Langevin et al. [49] n'a pas trouvé de solution réalisable dans huit cas, tandis que notre algorithme n'a pas trouvé une telle solution dans un seul cas. La table 1.1 présente ces résultats. La quatrième colonne contient les temps de calcul de l'algorithme exact, alors que la dernière colonne indique les temps de calcul de l'heuristique. Chaque ligne indique des moyennes sur le nombre de problèmes résolus parmi 10 possibles.

En ce qui concerne les problèmes tests résolus par Dumas et al. [18], notre algorithme a identifié des solutions réalisables pour 118 des 120 instances et 83 solutions optimales sur les 112 fournies. La table 1.2 résume ces observations. La quatrième colonne contient les temps de calcul de l'algorithme exact, alors que la dernière colonne indique les temps de calcul de l'heuristique. Chaque ligne indique des moyennes sur le nombre de problèmes résolus parmi 5 possibles.

On remarque aussi que l'écart moyen par rapport à la solution optimale est de moins de 0.5%. Pour les instances à 80 clients et une largeur moyenne des fenêtres de temps de 100, la méthode exacte n'a pas trouvé de solution. Notons que les données concernant les temps de transport utilisés par Dumas et al. [18] sont des valeurs entières obtenues par arrondi, $\bar{c}_{ij} = [c_{ij}]$, et en prenant ensuite $\bar{c}_{ij} = \bar{c}_{ik} + \bar{c}_{kj}$ lorsque $\bar{c}_{ij} > \bar{c}_{ik} + \bar{c}_{kj}$.

Il est difficile de comparer les temps d'exécution car les languages de programmation, ainsi que les ordinateurs utilisés sont différents. Toutefois, en considérant tous les facteurs, il semblerait qu'il soit de deux à trois fois plus rapide

d'utiliser Pascal sur un SUN SS20/61 que d'utiliser Fortran 77 sur un SUN SPARC2 (Langevin et al. [49]). De même, il serait trois fois plus rapide d'utiliser C sur un HP9000/735 (Dumas et al. [18]) que d'utiliser Pascal sur un SUN SS20/61. En faisant ces approximations grossières, notre algorithme semble être moins rapide pour les problèmes ayant des fenêtres de temps très étroites que les deux méthodes exactes. Ce phénomène est expliqué par le fait que les problèmes ayant des fenêtres des temps étroites sont facilement résolus par programmation dynamique puisque le nombre d'étiquettes à gérer est très petit dans ce cas. Notons qu'aucun algorithme exact connu n'est capable de résoudre des problèmes avec de larges fenêtres de temps. Dumas et al. [18] mentionnent que pour des problèmes de même taille, le temps d'éxecution augmente de manière exponentielle avec la largeur des fenêtres de temps.

Pour tester la performance de notre algorithme sur des problèmes avec fenêtres de temps larges, nous avons généré des problèmes tests à partir des problèmes utilisés par Dumas et al. [18] Nous avons gardé les mêmes clients et distances et nous avons élargi les fenêtres de temps. Pour un nombre de clients variant de 20 à 100, nous avons considéré les instances avec la largeur moyenne la plus grande et nous avons construit des fenêtres de temps d'une largeur d'au plus 200 minutes. Les résultats correspondant à ces tests sont présentés à la table 1.3. Ces résultats montrent clairement la robustesse de notre algorithme. Il réussit à identifier une solution réalisable dans tous les cas et les temps d'exécution obtenus ne varient pas avec la taille du problème traité. Il faut noter que notre algorithme dépense une grande partie de son temps d'exécution à trouver une solution initiale. Ceci est vrai surtout pour les instances à fenêtres de temps étroites.

1.5 Conclusions

L'algorithme que nous avons développé répond aux objectifs de départ. Pour des problèmes avec des fenêtres de temps étroites, notre heuristique n'est pas aussi rapide que les deux méthodes exactes proposées dans la littérature. Toutefois, lorsque la taille des fenêtres augmente, notre algorithme est la seule méthode connue qui permet de produire des solutions réalisables dans un temps raisonnable. Le succès de notre méthode est en partie dû aux mécanismes suivants:

- Utilisation de GENIUS pour produire des insertions et aussi dans la phase de post optimisation,
- Voisinages basés sur les distances spaciale et temporelle,
- La procédure de retour en arrière lorsqu'il n'y a pas d'insertion réalisable.

Tableau 1.1 Les problèmes de Langevin.

Clients	Largeur moyenne des fenêtres de temps	Moyenne optimale (pbs. résolus)	Temps (sec.)	Moyenne heuristique	Solutions réalisables (optimales)	Moyenne de l'écart (%)	Temps (sec.)
20	20	739.7	0.4	739.7	10 (10)	0.0	1.68
	30	741.7	0.4	741.7	10 (10)	0.0	1.73
	40	743.7	0.7	743.7	10 (10)	0.0	1.71
40	20	1005.8	1.7	1005.8	10 (10)	0.0	11.55
	30	1007.0	4.4	1007.04	10 (9)	0.0	13.86
	40	1008.3	7.3	1008.3	10 (10)	0.0	13.93
60	20	1234.3 (7)	9.6	1221.8	9 (5)	0.0	29.77
	30	1251.8 (8)	32.1	1183.4	10 (8)	0.0	28.23
	40	1250.3 (7)	43.4	1162.5	10 (5)	0.0	30.63

Tableau 1.2 Les problèmes de Dumas.

Clients	Largeur moyenne des fenêtres de temps	Moyenne optimale (pbs. résolus)	Temps (sec.)	Moyenne heuristique	Solutions réalisables (optimales)	Moyenne de l'écart (%)	Temps (sec.)
20	20	361.2	0.02	361.2	5 (5)	0.0	1.70
	40	316.0	0.05	316.0	5 (5)	0.0	2.10
	60	309.8	0.14	310.2	5 (4)	0.1	2.34
	80	311.0	0.23	314.2	5 (4)	1.0	3.15
	100	275.2	1.27	275.2	5 (5)	0.0	3.38
40	20	486.6	0.08	486.6	5 (5)	0.0	12.52
	40	461.0	0.24	461.0	5 (5)	0.0	14.16
	60	416.4	4.37	417.0	5 (4)	0.1	16.39
	80	399.8	7.52	399.8	5 (5)	0.0	13.95
	100	377.0	31.40	378.8	5 (3)	0.5	14.21
60	20	581.6	0.15	581.6	5 (5)	0.0	36.54
	40	590.2	0.89	590.2	5 (5)	0.0	36.79
	60	560.0	6.84	567.0	5 (3)	1.2	44.29
	80	508.0	46.62	517.2	5 (2)	1.8	32.93
	100	514.8	199.84	524.0	5 (1)	1.8	49.75
80	20	676.6	0.35	676.6	5 (5)	0.0	74.03
	40	630.0	2.68	630.4	5 (3)	0.0	66.32
	60	606.4	55.32	616.8	5 (1)	1.7	89.13
	80	593.8	220.29	601.4	5 (2)	1.3	75.75
	100	s/o	s/o	584.2	5	s/o	99.54
100	20	757.6	0.62	757.6	5 (5)	0.0	174.98
	40	701.8	7.40	715.0	4 (3)	0.0	117.97
	60	696.6	107.95	690.0	4 (2)	0.8	97.87
	80	669.0 (2)	626.03	675.6	5 (1)	2.5	118.14

Tableau 1.3 Extension des problèmes de Dumas.

Clients	Largeur moyenne des fenêtres de temps	Moyenne heuristique	Solutions réalisables	Temps (sec.)
20	120	269.2	5	4.08
	140	263.8	5	4.37
	160	261.2	5	4.77
	180	259.8	5	5.98
	200	245.2	5	6.31
40	120	372.8	5	18.38
	140	356.2	5	18.86
	160	348.0	5	20.00
	180	328.2	5	17.02
	200	326.2	5	22.80
60	120	492.0	5	51.59
	140	454.8	5	49.47
	160	451.6	5	47.53
	180	439.2	5	52.26
	200	439.6	5	43.50
80	120	581.8	5	121.02
	140	555.2	5	94.17
	160	524.8	5	85.69
	180	511.0	5	99.01
	200	508.6	5	112.34
100	100	671.2	5	129.54
	120	624.6	5	204.17
	140	634.6	5	207.67
	160	585.2	5	215.57
	180	585.2	5	225.12
	200	588.6	5	168.24

CHAPITRE 2

Le problème de fabrication des tournées de véhicules avec fenêtres de temps

Une instance du problème de fabrication des tournées de véhicules avec fenêtres de temps (PFTVFT) est définie par un dépôt, une flotte de véhicules et un ensemble de villes ou clients avec des demandes connues. Le problème d'optimisation associé consiste à déterminer un ensemble de routes à coût minimum, débutant et finissant au dépôt, qui dessert tous les clients. L'ensemble de routes doit être défini de façon à ce que chaque ville soit visitée une seule fois par un seul véhicule. Le service d'une ville doit commencer à l'intérieur de la fenêtre de temps définie par le temps au plus tôt et le temps au plus tard où le service peut débuter. Le PFTVFT représente un élément très important des systèmes de distribution et transport. Un exemple est la fabrication d'horaires de travail dans le contexte urbain ou aérien. Les villes sont identifiées à des tâches de travail aux différents moments dans le temps, alors que les routes correspondent à des périodes de travail.

Le PFTVFT est un problème NP-complet. Dans ce chapitre on présente un nouvel algorithme de type recherche tabou pour sa résolution.

2.1 Le problème

Soit $N = \{v_1, \ldots, v_n\}$ un ensemble de noeuds. On définit aussi deux autres noeuds, v_0 et v_{n+1}. Les noeuds $v_i, 1 \leq i \leq n$, sont associés aux clients et les noeuds v_0 et v_{n+1} représentent, respectivement, la sortie du dépôt et l'entrée au dépôt. Soit $V = N \cup \{v_0, v_{n+1}\}$. Le PFTVFT est défini sur un graphe $G = (V, A)$, déterminé

par l'ensemble des noeuds V et l'ensemble des arcs A, défini ci-dessus. À chaque arc $(v_i, v_j), v_i, v_j \in V, i \neq j$, on associe deux valeurs non négatives: une distance c_{ij} et un temps de parcours t_{ij}. Des fenêtres de temps $[a_i, b_i]$ et un temps de service s_i sont attachées à chaque noeud $v_i \in N$. De plus, à chaque noeud $v_i, v_i \in N$ est associée une demande d_i. Le temps auquel un véhicule visite le noeud v_i est noté par t_i. On remarque que $[a_0, b_0] = [a_{n+1}, b_{n+1}]$. L'ensemble des arcs A est défini par $A = \{(v_i, v_j) : a_i + t_{ij} \leq b_j, v_i \in N \cup \{v_0\}, v_j \in N \cup \{v_{n+1}\}, i \neq j\}$.

Pour simplifier, on fait l'hypothèse que les temps de parcours t_{ij} sont égaux aux distances c_{ij}. On suppose aussi que la distance c_{ij} de v_i à v_j inclut le service à v_i. Le PFTVFT consiste à détérminer un ensemble de chemins orientés qui commencent à v_0, finissent à v_{n+1}, et tels que chaque noeud $v_i, v_i \in N$ est visité une seule fois par un seul chemin. Chaque noeud $v_i, v_i \in N$ doit être visité au plus tard au temps b_i et ne peut pas être servi avant le temps a_i. On remarque que le noeud v_i peut être atteint avant a_i et dans ce cas il y a une attente à ce noeud. On remarque que la durée totale d'un chemin ne peut pas dépasser b_{n+1}. Si on associe un véhicule à chaque chemin, alors il faut respecter aussi la contrainte de capacité suivante. La somme des demandes des clients desservis par un véhicule ne peut pas dépasser la capacité du véhicule. On fait l'hypothèse que la demande à un noeud $v_i, v_i \in N$, est inférieure à la capacité des véhicules. Nous avons adopté comme objectif de déterminer un ensemble de chemins à coût minimum par rapport aux distances, par contraste à la durée totale. Cet objectif représente la formulation classique du problème et il est le plus étudié.

Sans la contrainte sur la capacité des véhicules, le problème est connu dans la littérature de spécialité comme le m-PVCFT.

Dans ce chapitre on considère une flotte homogène de véhicules et on fait l'hypothèse que la taille de cette flotte est une variable de décision. Savelsbergh [62] a montré que même trouver une solution réalisable pour le PFTVFT dont le nombre de véhicules est fixé, est un problème NP-complet. Dans notre approche on considère que les fenêtres de temps représentent des contraintes fortes: un véhicule ne peut pas arriver à une ville après la fin de la fenêtre de temps. Cette approche est différente de celle adoptée par ceux qui traitent les fenêtres de temps comme des contraintes molles, en les violant moyennant une pénalité.

Soit K l'ensemble des indices des véhicules et Q la capacité d'un véhicule. La formulation mathématique comprend trois types de variables: variables de flot $x_{ij}^k, (v_i, v_j) \in A, k \in K$, variables de temps $t_i^k, i \in V, k \in K$ et variables de chargement $l_i^k, i \in V, k \in K$. La variable x_{ij}^k est égale à 1 si l'arc (v_i, v_j) est utilisé par le véhicule k et 0 sinon. La variable t_i^k indique le début du service au noeud v_i par le véhicule k, alors que l_i^k correspond au chargement du véhicule k immédiatement après le service au client v_i.

Le problème de déterminer un ensemble de routes à coût minimum qui satisfait les contraintes du PFTVFT peut être formulé comme suit:

$$\min_{x_{ij}^k} \quad \sum_{k \in K} \sum_{(v_i, v_j) \in A} c_{ij} x_{ij}^k$$

$$s.l.c. \quad \sum_{k \in K} \sum_{j \in N \cup \{v_{n+1}\}} x_{ij}^k = 1 \qquad \forall i \in N \qquad (2.1)$$

$$\sum_{v_j \in N \cup \{v_{n+1}\}} x_{0j}^k = 1 \qquad \forall k \in K \qquad (2.2)$$

$$\sum_{v_i \in N \cup \{v_0\}} x_{ij}^k - \sum_{v_i \in N \cup \{v_{n+1}\}} x_{ji}^k = 0 \quad \forall k \in K \forall v_j \in N \qquad (2.3)$$

$$\sum_{v_i \in N \cup \{v_0\}} x_{in+1}^k = 1 \qquad \forall k \in K \qquad (2.4)$$

$$x_{ij}^k (t_i^k + t_{ij} - t_j^k) \leq 0 \qquad \forall k \in K, \forall (v_i, v_j) \in A \qquad (2.5)$$

$$a_i \leq t_i^k \leq b_i \qquad \forall k \in K, \forall i \in N \cup \{v_{n+1}\} \qquad (2.6)$$

$$x_{ij}^k (l_i^k + d_j - l_j^k) \leq 0 \qquad \forall k \in K, \forall (v_i, v_j) \in A \qquad (2.7)$$

$$l_i^k \leq Q \qquad \forall k \in K, \forall i \in N \cup \{v_0\} \qquad (2.8)$$

$$x_{ij}^k \geq 0 \qquad \forall k \in K, \forall (v_i, v_j) \in A \qquad (2.9)$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall k \in K, \forall (v_i, v_j) \in A \qquad (2.10)$$

La fonction objectif minimise le coût total. La contrainte (2.1) assure que chaque client est desservi par un seul véhicule. Les contraintes (2.2)-(2.4) décrivent la conservation de flot pour le véhicule k. Les contraintes (2.5)-(2.6) imposent la réalisabilité de l'horaire, alors que les contraintes (2.7)-(2.8) garantissent la réalisabilité des charges.

Le PFTVFT a été étudié d'une manière intensive dans la littérature. Étant donné sa forte nature combinatoire, les méthodes exactes ont eu un succès limité sur le PFTVFT. Le plus large problème résolu à l'optimalité dans la littérature de spécialité comprend 18 véhicules et 100 clients (Desrochers, Desrosiers et Solomon [13]). Une autre direction de recherche a été le développement d'heuristiques capables de résoudre des problèmes réels dans un temps raisonnable.

La contribution de ce chapitre est le développement d'un nouvel algorithme de type recherche tabou pour le PFTVFT. Il se compare de façon favorable avec les plus récentes heuristiques, sur un ensemble de 56 problèmes tests de la littérature. Le reste du chapitre est organisé comme suit. La revue de la littérature est présentée dans la section suivante. L'heuristique qu'on propose est définie à la section 3. Des résultats de calculs sont rapportés de manière comparative à la section 4. Finalement, on dérive les conclusions à la section 5.

2.2 Revue de la littérature

Kolen, Rinnooy Kan et Trienekens [48] ont été les premiers à proposer un algorithme exact et à présenter des résultats de calcul pour le PFTVFT. Ils ont développé une méthode de séparation et évaluation, où chaque noeud de l'arbre d'exploration correspond à une solution partielle. Une solution partielle est définie par un ensemble de routes fixées, une route partielle qui commence au dépôt et un ensemble de clients qui ne peuvent pas prolonger la route partielle comme le prochain client. Dans l'étape de séparation, un client qui n'appartient pas aux routes fixées, à la route partielle, ou à l'ensemble de clients qui ne peuvent pas prolonger la route partielle, est choisi à l'aide d'une heuristique. Lorsqu'il n'y a pas de route partielle, la décision est d'initier ou de ne pas initier une nouvelle route avec le client choisi. Associés à la décision binaire de prolonger la route partielle par le client choisi à l'étape de séparation, deux noeuds sont créés. Le premier noeud correspond à la décision de prolonger la route partielle par le client choisi. Le deuxième noeud est associé à l'addition du client choisi à la liste des clients qui ne peuvent pas prolonger la route partielle. À chaque noeud de l'arbre d'exploration la méthode calcule une borne inférieure pour toutes les extensions réalisables de la solution partielle. De plus,

l'approche comporte une relaxation de la contrainte qui oblige qu'un client, pas encore dans une route, soit desservi une seule fois. L'extension au moindre coût de la solution partielle courante est calculée tel que la somme des demandes des routes soit égale à la somme des demandes des clients et que chaque route finisse par un autre client.

Desrochers, Desrosiers et Solomon [13] ont développé un algorithme exact basé sur la technique de génération de colonnes. Ils ont formulé le PFTVFT comme un problème de recouvrement où chaque colonne représente une route réalisable. Le sous-problème est défini comme un problème de plus court chemin avec contraintes d'horaires. Une solution du sous-problème est obtenue par une méthode primale-duale qui résout les équations de récurrence du problème. Cette méthode a été dévéloppée par Desrochers [12]. Elle demande la création de deux ensembles d'étiquettes à chaque noeud. Le premier ensemble contient les étiquettes associées à des chemins réalisables et il définit des solutions primales. Ces solutions déterminent, à chaque noeud, une borne supérieure sur le coût des solutions efficaces. Le deuxième ensemble contient les étiquettes associées à des bornes inférieures sur le coût d'un chemin finissant au noeud v_i dans un état donné. L'algorithme modifie les ensembles d'étiquettes jusqu'à ce qu'une solution optimale soit atteinte. Pour obtenir une solution entière, les auteurs ont utilisé une stratégie de séparation et évaluation. Dans une première étape le branchement est effectué sur le nombre de véhicules utilisés. Une fois cette variable fixée, le branchement est dirigé à partir des valeurs des variables de flot. L'algorithme a été testé avec de très bons résultats sur des problèmes avec des fenêtres de temps très serrées, car le nombre de solutions, et donc de colonnes est limité. Des problèmes qui comportent jusqu'à 100 clients ont été résolus. Comme la combinatoire augmente d'une manière dramatique avec la largeur des fenêtres de temps, l'approche est moins efficace sur les instances à larges

fenêtres de temps.

Deux types d'heuristiques ont produit de très bons résultats pour le PFTVFT. Leur succès est dû à deux facteurs importants: la qualité générale de l'heuristique et l'incorporation dans leur mécanismes des caractéristiques propres au PFTVFT. Un premier type d'heuristique est la recherche tabou. Sa philosophie générale est améliorée en rajoutant des définitions spécifiques pour le voisinage et la stratégie de recherche. Le deuxième type d'heuristique est représenté par les algorithmes génétiques. La création des descendants et le mécanisme de mutation sont des leviers de décision significatifs pour les algorithmes génétiques. On restreint la revue des méthodes heuristiques à ces deux types.

Rochat et Taillard [61] présentent un algorithme de type recherche tabou qui comporte deux phases et qui utilise une technique probabiliste pour diversifier, intensifier et paralléliser une recherche locale adaptée à la résolution des problèmes de tournées de véhicules. Dans la première phase, l'aspect diversification est exploité à fond pour obtenir un ensemble de solutions initiales en appliquant une recherche locale à des solutions générées d'une manière aléatoire. À chaque route on associe une étiquette avec la valeur de la solution à laquelle elle appartient. La deuxième phase correspond à un processus itératif qui choisit d'une manière probabiliste un nouvel ensemble de routes selon les valeurs des étiquettes des routes courantes. Cette stratégie est résumée par Glover [27] comme suit:

- choisir une ou plusieurs variables dont l'importance est plus marquée et fixer ces variables à leurs valeurs préférées;
- calculer la nouvelle importance des variables pas encore fixées en tenant compte des restrictions ci-dessus;

- repéter le processus jusqu'à ce que toutes les variables aient été fixées.

Potvin et al. [57] décrivent un algorithme de type recherche tabou. Ils définissent le voisinage à l'aide de deux opérations qui représentent deux types d'échange: mouvements Or − opt et 2 − opt*. Un échange de type Or − opt [54] consiste à déplacer un, deux ou trois clients consécutifs dans une partie différente de la même route. Un échange de type 2 − opt* est défini de la manière suivante. Considérons deux routes qu'on brise en deux parties (en enlevant un arc de chacune d'entre elles). Deux nouvelles routes sont obtenues en combinant le début de la première route avec la fin de la deuxième route et le début de la deuxième route avec la fin de la première route. Pour restreindre le nombre de tels échanges, les auteurs définissent pour chaque client l'ensemble de ses plus proches voisins. Un mouvement de type Or − opt est considéré seulement si le client après lequel la séquence de clients sera insérée fait partie des p plus proches voisins du premier client appartenant à la séquence. D'une manière similaire, dans un échange de type 2 − opt*, le client qui marque le début de la deuxième partie de la deuxième route doit faire partie des q plus proches voisins du dernier client de la première partie de la première route (où p et q représentent des paramètres d'entrée). Les auteurs considèrent seulement des mouvements qui gardent la réalisabilité des routes. Pour réduire le nombre de routes utilisées, une procédure qui essaye d'éliminer les routes à moins de 3 clients est utilisée.

GIDEON (Thangiah, Nygard et Juell [68]) est un algorithme génétique basé sur la stratégie groupe d'abord, route ensuite. Cette stratégie comporte deux étapes. La première étape est dédiée à rassembler, selon des critères spécifiques, des éléments donnés dans des groupes. La deuxième étape consiste à améliorer l'objectif en apportant des modifications à chaque groupe. L'algorithme génétique est appliqué seulement pour identifier les groupes. Dès que cette opération est finie, des heuristiques

de recherche locale sont utilisées pour améliorer la solution obtenue par l'algorithme génétique. L'algorithme génétique est basé seulement sur des considérations de type géographique. Comme résultat, il n'y a pas de garantie que la solution finale sera réalisable par rapport aux fenêtres de temps ou aux contraintes de capacité. On tient compte des violations des contraintes en introduisant des pénalités dans la fonction objectif. Une heuristique de recherche locale déplace un client ou une séquence de deux clients d'une route à une autre. Une deuxième heuristique échange un client ou une séquence de deux clients entre deux routes.

Un inconvénient de l'utilisation des algorithmes génétiques pour le PFTVFT est la difficulté de coder plusieurs routes dans un chromosome et de définir des opérations qui fusionnent deux solutions en une seule pour générer des descendants réalisables. Potvin et Bengio [56] ont développé un algorithme, GENEROUS, qui évite ce problème en appliquant les opérateurs de mutation et reproduction aux solutions. Pour obtenir des descendants, les auteurs ont développé la méthode suivante. Des solutions parentes sont choisies à partir de la population courante selon une règle de "fitness". Une nouvelle solution est obtenue en utilisant l'heuristique $2-\text{opt}^*$. Pendant ce processus, certains clients se trouvent dans plusieurs routes, alors que d'autres ne font partie d'aucune. Un opérateur réparateur est appliqué pour résoudre cette difficulté. Finalement, des opérateurs de type mutation sont utilisés pour éliminer des routes contenant un nombre réduit de clients.

Pour un survol récent des problèmes de fabrication de tournées de véhicules et ordonnancement, voir Desrosiers et al. [15].

2.3 L'algorithme

L'algorithme que nous proposons est composé de deux phases. La première phase, l'initialisation, a comme but de déterminer une solution initiale. La deuxième phase représente un processus d'amélioration de la solution initiale. Elle comporte les caractéristiques d'une méthode de recherche tabou, combinées avec des procédures spécifiques aux PFTVFT.

Tout au long de l'algorithme on utilise les principes de GENIUS, qu'on a adapté au PFTVFT. On a vu au chapitre précédent les modifications apportées à GENIUS pour pouvoir traiter le PVCFT. En grande partie ces modifications s'appliquent aussi au PFTVFT. On les reprend ici pour l'homogénéité du texte.

Dans notre algorithme pour le PFTVFT, GENI est utilisé pour insérer un noeud v dans une route partiellement construite, en exécutant en même temps une optimisation locale de la route. Soit p un paramètre d'entrée. Pour chaque noeud $v \in V$ on définit son voisinage $N_p^s(v)$ dans la route s comme l'ensemble de p noeuds déjà dans la route qui sont les plus proches voisins de v.

Si la route contient moins de p noeuds, alors ces noeuds définissent $N_p^s(v)$. Là où le contexte le permet sans créer de confusion, on remplace la notation $N_p^s(v)$ par $N_p(v)$. Le noeud v est inséré entre les noeuds v_i et v_j appartenant à $N_p(v)$. Pour une route et une orientation données, soient v_{i-1} et v_{i+1} le prédécesseur et le successeur du noeud v_i. Il y a deux types d'insertion GENI. Chacune est essayée pour les deux orientations de la route et pour différents choix des v_i, v_j, v_k et v_l. La meilleure insertion est acceptée. Les deux types d'insertion GENI sont décrits au premier chapitre.

On peut utiliser l'inverse de GENI pour enlever un noeud de la route et procéder en même temps à une optimisation locale. Pour une route fixée, le processus itératif défini par l'inverse de GENI et par GENI conduit à une optimisation locale de la route. Ce processus itératif est noté US (pour "unstriging" et "stringing") par Gendreau, Hertz et Laporte [23].

Comme chaque route du PFTVFT représente un chemin orienté de v_0 à v_{n+1} et comme le chemin inverse d'un chemin réalisable peut ne pas l'être, la nature asymétrique du PFTVFT est évidente. Pour en tenir compte, on remplace les voisinage $N_p(v)$ par les voisinages orientés $N_p^+(v)$ et $N_p^-(v)$ contenant, respectivement, les plus proches p successeurs et prédécesseurs du noeud v.

Le voisinage doit tenir compte des distances et fenêtres de temps. Étant donné deux noeuds v_i et v_j, on définit la proximité de leurs fenêtres de temps par:

$$r_{ij} = \mid \min\{b_j, b_i + c_{ij}\} - \max\{a_j, a_i + c_{ij}\} \mid .$$

Une manière naturelle de définir une *pseudo-distance* d_{ij} entre v_i et v_j est d'utiliser une combinaison convexe entre c_{ij} et r_{ij}:

$$d_{ij} = \alpha c_{ij} + (1 - \alpha) r_{ij}, \alpha \in [0, 1].$$

Des tests préliminaires ont montré que cette règle mène à des résultats instables: fixer α est problématique car les distances sont plus influentes pour certaines instances, alors que pour d'autres les fenêtres de temps sont le facteur critique. Une façon plus satisfaisante de définir les voisinages est d'inclure dans $N_p^+(v)$ les p_1 plus proches succcesseurs de v déjà dans le chemin par rapport à c_{ij}, et les p_2 plus proches successeurs de v déjà dans le chemin par rapport à r_{ij}, avec la condition $p_1 + p_2 = p$. D'une manière similaire on définit $N_p^-(v)$ en termes de prédécesseurs. De plus, v_0

est inclus dans $N_p^-(v)$ et v_{n+1} est inclus dans $N_p^+(v)$.

L'algorithme que nous avons développé préserve en tout temps la réalisabilité des solutions. Cette approche est différente des méthodes qui permettent des solutions non réalisables et qui utilisent des pénalités pour retrouver la réalisabilité. Accepter des solutions qui ne respectent pas les contraintes a comme but d'augmenter le nombre d'insertions possible et donc d'explorer l'espace des solutions plus rapidement. L'inconvénient de cette approche est la difficulté de retomber sur une solution réalisable. En restant toujours dans l'espace réalisable, le nombre d'insertions est plus restreint, ce qui diminue la distance (définie par le nombre de noeuds qui ont changé de position) entre deux solutions.

Lors de l'insertion d'un noeud v dans le chemin courant s, il faut maintenir sa réalisabilité. À cette fin, on calcule pour chaque noeud v_i *le temps critique de départ* z_i. Ce temps critique est mis à jour au long de l'algorithme. Ces coefficients mesurent le temps au plus tard des départs des noeuds pour garder le chemin réalisable. Les coefficients z_i sont calculés à rebours, à partir du noeud v_{n+1}:

$$\begin{aligned} z_{n+1} &= b_{n+1}, \\ z_i &= \min\{b_i, z_j - c_{ij}\}, \end{aligned}$$

si v_i est le prédécesseur de v_j sur le chemin courant.

Chaque fois qu'un noeud v est inséré dans le chemin courant en utilisant la procédure GENI, l'ordre de quelques noeuds peut être modifié. Considérons le chemin $(v_0, \ldots, v_{i_\alpha}, \ldots, v_{i_\beta}, \ldots, v_{n+1})$ obtenu après une insertion et $(v_{i_\alpha}, \ldots, v_{i_\beta})$ la portion du chemin modifiée par l'insertion. Une condition nécessaire pour qu'une

insertion potentielle soit réalisable est que les temps d'arrivée aux noeuds t_i satisfassent:

$$t_i \leq b_i, i = v_{i_\alpha}, \ldots, v_{i_\beta}.$$

En plus, l'insertion doit être réalisable pour tous les noeuds suivant v_{i_β}:

$$t_{i_{\beta+1}} = t_{i_\beta} + t_{i_\beta i_{\beta+1}} \leq z_{i_{\beta+1}}.$$

Cette dernière condition permet d'éliminer possibles insertions non réalisables.

Lorqu'un noeud est enlevé de la route r et inséré dans la route s à l'itération t, on interdit de l'introduire de nouveau dans la route r jusqu'à l'itération $t + \theta$, où θ est un entier choisi d'une manière aléatoire de l'intervalle $[\underline{\theta}, \overline{\theta}]$.

On décrit maintenant l'algorithme étape par étape:

- Étape 1: *initialisation.* Obtenir une solution réalisable à l'aide de la procédure d'insertion I1 de Solomon [65]. Soit m_{i1} le nombre de véhicules utilisés par cette procedure. Pour $m = 1, \ldots, m_{i1}$, calculer une nouvelle solution en utilisant l'insertion parallèle. Il faut noter que pour chacune de ces solutions on définit les germes comme les m noeuds les plus éloignés du dépôt. La solution initiale m_1 est la meilleure parmi les m_{i1} générées. Soit $k = 0$.

- Étape 2: *critère d'arrêt.* Si $k < nrep$, alors $k = k + 1$ et aller à l'étape 3. Sinon l'algorithme s'arrête en retournant la meilleure solution trouvée.

- Étape 3: *optimisation*: Appliquer à chaque route de la solution courante la procédure d'optimisation US.

- Étape 4: *changement de route*: Enlever un client v_i de la route r et l'insérer dans la route s, $s \neq r$, ou initialiser une nouvelle route avec v_i. Pour limiter le

nombre de mouvements possibles, la route s doit contenir un des plus proches voisins de v_i. Considérer chaque noeud et implanter le mouvement qui fait décroître le plus la fonction objectif.

- Étape 5: *échange de type* $2-opt^*$. Couper en deux deux routes de la solution courante aux endroits qui permettent d'obtenir deux nouvelles routes réalisables par un échange de type $2-\text{opt}^*$. Considérer les routes de la solution courante deux à deux et implanter l'échange qui réduit le plus l'objectif. Cette étape est utilisée sans tenir compte des restrictions tabou.

- Étape 6: *réduction du nombre de véhicules.* Essayer de réduire le nombre de véhicules en éliminant les routes à moins de n/m_1 clients. Considérer chaque client v_i appartenant à une telle route et tenter de l'insérer dans une autre route. Aller à l'Étape 2.

À l'Étape 6 de l'algorithme, lors du processus de réduction du nombre de véhicules, les routes où l'on tente d'insérer un client doivent avoir déjà au moins n/m_1 clients. Si une route s respecte cette condition, alors on évalue si la valeur de la fonction objectif diminue. Des tests préliminaires ont montré que pour une partie des instances il vaut mieux accepter l'échange, même s'il n'y a pas de réduction de l'objectif. Pour une autre partie des instances il était impératif que la fonction objectif diminue. Nous avons décidé d'accepter l'échange sans une amélioration de l'objectif pour les premières n_1 itérations tabou et d'imposer la réduction de l'objectif après ce seuil.

On a étendu la structure de données utilisée pour le PVCFT au PFTVFT. Un chemin est représenté par une liste circulaire doublement chaînée et pour chaque noeud on garde un pointeur vers sa position dans le chemin où il se trouve. Ceci nous permet d'insérer ou d'enlever un noeud, une fois que le type d'insertion ou

d'enlèvement a été déterminé, avec une complexité de l'ordre $O(n)$. On a vu aussi au chapitre précédent que le nombre d'opérations nécessaires pour une insertion (enlèvement) est de l'ordre $O(n + p^4)$. On peut maintenant facilement déterminer la complexité de l'algorithme. Le nombre d'opérations nécessaires à l'étape d'initialisation est de l'ordre $O(n^2 + np^4)$. Une itération tabou a une complexité de l'ordre $O(n^2 + np^4)$, car chacune des Étapes 3, 4, 5 et 6 a cette complexité. Comme *nrep*, le nombre d'itérations tabou, n'est pas connu on ne peut pas conclure sur la complexité totale de l'algorithme.

2.4 Résultats numériques

L'algorithme présenté dans la section précedente a été implanté en Pascal et testé sur une station SUN 10 (135.5 MIPS, 27.3 MFLOPS, 128M MÉMOIRE). Des résultats préliminaires ont montré qu'en posant $p_1 = 5$ et $p_2 = 5$, on obtient de bonnes solutions. Les autres paramètres sont fixés comme suit: $[\underline{\theta}, \overline{\theta}] = [5, 10]$, $n_1 = 3 \times n$ et $nrep = 5 \times n$.

On a testé la performance de notre algorithme sur l'ensemble de problèmes tests défini par Solomon [65]. Cet ensemble comprend 56 instances à 100 clients. Il a été spécialement construit pour le PFTVFT et est l'ensemble de problèmes tests le plus utilisé dans la littérature consacrée aux problèmes de fabrication d'horaires de véhicules. Pour ces problèmes, le temps de transport correspond aux distances euclidiennes, calculées à un décimal. Selon leur caractéristiques, ces problèmes ont été divisés en trois catégories: de type R (les clients sont dispersés dans un carré $[0, 100] \times [0, 100]$ selon une loi uniforme), de type C (les clients forment des groupes dans le carré $[0, 100] \times [0, 100]$) et de type RC, une combinaison des types R et C. De plus, selon la taille de la flotte de véhicules, la capacité d'un véhicule, les largeurs

des fenêtres de temps et distances entre les clients, pour chacun des types définis ci-dessous, on a construit deux catégories de problèmes. Les ensembles de problèmes R1, C1, et RC1 ont un horizon étroit, défini par des véhicules avec une capacité réduite et des courtes distances entre les clients. Ceci implique que seulement un nombre limité de clients peuvent être desservis par un véhicule. Par contraste, les ensembles R2, C2 et RC2 se caractérisent par des véhicules avec une grande capacité et de grandes distances entre les clients. Pour ces ensembles, une plus petite flotte de véhicules sera suffisante. Les caractéristiques des problèmes tests sugèrent que des approches différentes sont recommandées pour différents types de problèmes. Par exemple, une méthode exacte est indiquée pour des instances dont les fenêtres de temps sont étroites. De plus, une heuristique peut avoir un comportement différent selon le type de problème. Une manière efficace d'obtenir de bon résultats est de changer ses paramètres pour tenir compte des caractéristiques propres à chaque type de problème.

On a comparé notre algorithme avec les approches les plus performantes (Gendreau, Laporte et Potvin [25]): GIDEON (Thangiah, Nygard et Juell [68]; Thangiah [69]; Thangiah and Gubbi [70]), GENEROUS (Potvin and Bengio [56]) et Rochat et Taillard [61].

Les résultats numériques sont rapportés dans les tableaux 2.1 à 2.12. Le tableau 2.1 présente une comparaison des solutions moyennes pour les 56 instances. Dans la première colonne on indique entre parenthèses le nombre d'instances pour chaque type de problème. Pour chaque type de problème on indique dans la colonne correspondant à chaque algorithme les moyennes suivantes: nombre de véhicules, distance parcourue, et temps d'exécution. Tous les temps sont donnés en secondes. On remarque que les temps d'exécution ne sont pas disponibles pour GIDEON.

GENEROUS a été exécuté sur une station Silicon Graphics, alors que l'algorithme de Rochat et Taillard [61] a été exécuté sur un Silicon Graphics Indigo (100 Mhz). On peut conclure que notre algorithme est très compétitif, car ses performances moyennes se trouvent dans la même classe que les approches les plus performantes.

Les tableaux 2.2 et 2.3 sont une mise à jour du tableau 6 de Rochat et Taillard [61] contenant les meilleures solutions publiées pour les instances de Solomon. La dernière colonne, notée *Heuristique*, présente les résultats obtenus par notre algorithme. Il faut noter que les solutions que Rochat et Taillard [61] rapportent sont les meilleures solutions obtenues au cours de nombreux tests, en utilisant des paramètres différents. La solution d'une instance est donnée par deux nombres. Le premier indique le nombre de véhicules utilisés. Le deuxième nombre représente la distance totale parcourue par les véhicules. On constate qu'on améliore (caractères gras) la valeur de la meilleure solution connue dans 10 instances sur 56 et on atteint (italiques) la solution optimale dans 4 autres instances.

Les tableaux 2.4 à 2.12 décrivent en détail les améliorations que notre algorithme a apporté aux meilleures solutions connues pour les instances de Solomon.

2.5 Conclusions

Nous avons présenté un algorithme de type recherche tabou pour le PFTVFT. On a testé l'algorithme sur les problèmes de Solomon [65], problèmes tests de référence dans la littérature. On a comparé nos solutions avec celles obtenues par les meilleurs algorithmes. Les résultats numériques montrent que notre méthode se compare favorablement avec les procedures les plus efficaces dévéloppées pour le PFTVFT.

Nous avons amélioré la meilleure solution connue pour 20% des instances. On attribue les résultats de notre implantation aux facteurs suivants:

- L'utilisation de GENI et son inverse comme mécanismes pour insérer et enlever des noeuds,
- La définition du voisinage,
- La procédure de réduction du nombre de véhicules,
- L'utilisation des échanges de type $2 - opt^*$.

Tableau 2.1 Moyennes comparatives des résultats.

Type de problème	GIDEON	GENEROUS	Rochat & Taillard	Heuristique
R1 (12)	12.8 1298.8 s/o	12.6 1296.8 823	12.58 1197.42 2700	12.58 1199.53 2567
C1 (9)	10.0 892.1 s/o	10.0 838.0 870	10.0 828.45 3200	10.11 834.92 305
RC1(8)	12.5 1472.2 s/o	12.1 1446.2 837	12.38 1369.48 2600	12.38 1390.75 2200
R2 (11)	3.2 1124.7 s/o	3.0 1117.7 3579	3.09 954.36 9800	3.00 1010.45 14810
C2 (8)	3.0 749.1 s/o	3.0 589.9 3429	3.00 590.32 7200	3.00 589.65 1800
RC2 (8)	3.3 1433.0 s/o	3.4 1360.6 3641	3.62 1139.79 7800	3.50 1186.14 9700

Tableau 2.2 Les meilleures solutions: instances R1, C1, RC1.

Instance	Référence	La meilleure solution	Heuristique
R101	Desrochers et al. (1992)	18/1608	19/1654.8
R102	Desrochers et al. (1992)	17/1434	17/1491.3
R103	Thangiah et al. (1994)	13/1207	14/1221.5
R104	Rochat & Taillard (1995)	10/984.69	10/990.4
R105	Rochat & Taillard (1995)	14/1377.09	14/1391.4
R106	Rochat & Taillard (1995)	12/1251.98	12/1269.9
R107	Rochat & Taillard (1995)	10/1159.86	11/1093.5
R108	Rochat & Taillard (1995)	9/997.33	10/961.5
R109	Rochat & Taillard (1995)	11/1245.65	12/1169.8
R110	Rochat & Taillard (1995)	11/1082.05	11/1099.0
R111	Thangiah et al. (1994)	10/1151	11/1083.7
R112	Rochat & Taillard (1995)	10/953.56	10/968.5
C101	Desrochers et al. (1992)	10/827	*10/827.3*
C102	Desrochers et al. (1992)	10/827	*10/827.3*
C103	Rochat & Taillard (1995)	10/828.14	10/835.1
C104	Rochat & Taillard (1995)	10/824.84	10/854.8
C105	Thompson & Psaraftis (1993)	10/829	**10/827.3**
C106	Desrochers et al. (1992)	10/827	11/860.6
C107	Desrochers et al. (1992)	10/827	*10/827.3*
C108	Desrochers et al. (1992)	10/827	*10/827.3*
C109	Potvin et al. (1993)	10/829	**10/827.3**
RC101	Thangiah et al. (1994)	14/1669	15/1651.0
RC102	Rochat & Taillard (1995)	13/1477.56	14/1495.0
RC103	Thangiah et al. (1994)	11/1110	12/1325.5
RC104	Rochat & Taillard (1995)	10/1135.84	10/1242.0
RC105	Rochat & Taillard (1995)	14/1540.24	15/1552.7
RC106	Rochat & Taillard (1995)	12/1384.95	12/1405.1
RC107	Rochat & Taillard (1995)	11/1230.98	11/1235.9
RC108	Rochat & Taillard (1995)	10/1170.72	10/1218.8

Tableau 2.3 Les meilleures solutions: instances R2, C2, RC2.

Instance	Référence	La meilleure solution	Heuristique
R201	Rochat & Taillard (1995)	4/1281.55	**4/1278.5**
R202	Rochat & Taillard (1995)	4/1088.07	4/1114.8
R203	Rochat & Taillard (1995)	3/948.70	3/1060.5
R204	Rochat & Taillard (1995)	2/869.24	**2/841.3**
R205	Rochat & Taillard (1995)	3/1063.24	3/1146.7
R206	Thangiah et al. (1994)	3/833.8	3/1020.8
R207	Rochat & Taillard (1995)	3/814.78	3/944.9
R208	Rochat & Taillard (1995)	2/738.55	2/773.2
R209	Thangiah et al. (1994)	2/855	3/1037.1
R210	Rochat & Taillard (1995)	3/967.45	3/968.2
R211	Rochat & Taillard (1995)	2/949.47	3/929.0
C201	Potvin et al. (1993)	3/590	**3/589.1**
C202	Potvin et al. (1993)	3/591	**3/589.1**
C203	Rochat & Taillard (1995)	3/591.2	3/598.0
C204	Potvin & Bengio (1994)	3/591	3/597.0
C205	Potvin et al. (1993)	3/589	**3/586.4**
C206	Potvin et al. (1993)	3/588	**3/586.0**
C207	Potvin et al. (1993)	3/588	**3/585.8**
C208	Potvin et al. (1993)	3/588	**3/585.8**
RC201	Thangiah et al. (1994)	4/1294	4/1503.3
RC202	Rochat & Taillard (1995)	4/1165.56	4/1332.0
RC203	Rochat & Taillard (1995)	3/1079.58	3/1145.5
RC204	Rochat & Taillard (1995)	3/806.75	3/897.8
RC205	Rochat & Taillard (1995)	4/1333.71	4/1409.8
RC206	Rochat & Taillard (1995)	3/1212.68	3/1275.8
RC207	Rochat & Taillard (1995)	3/1085.63	4/1033.4
RC208	Rochat & Taillard (1995)	3/833.95	3/891.5

Tableau 2.4 Instance C105. Distance totale: 827.3.

Nombre de clients	Routes	Distance	Capacité
12	5, 3, 7, 8, 10, 11, 9, 6, 4, 2, 1, 75	59.4	180
13	43, 42, 41, 40, 44, 46, 45, 48, 51, 50, 52, 49, 47	64.6	160
10	90, 87, 86, 83, 82, 84, 85, 88, 89, 91	75.9	170
8	13, 17, 18, 19, 15, 16, 14, 12	95.8	190
9	32, 33, 31, 35, 37, 38, 39, 36, 34	97.0	200
8	57, 55, 54, 53, 56, 58, 60, 59	101.7	200
9	98, 96, 95, 94, 92, 93, 97, 100, 99	95.8	190
9	81, 78, 76, 71, 70, 73, 77, 79, 80	127.1	150
11	67, 65, 63, 62, 74, 72, 61, 64, 68, 66, 69	59.3	200
11	20, 24, 25, 27, 29, 30, 28, 26, 23, 22, 21	50.7	170

Tableau 2.5 Instance C109. Distance totale: 827.3.

Nombre de clients	Routes	Distance	Capacité
8	13, 17, 18, 19, 15, 16, 14, 12	95.8	190
12	5, 3, 7, 8, 10, 11, 9, 6, 4, 2, 1, 75	59.4	180
10	90, 87, 86, 83, 82, 84, 85, 88, 89, 91	75.9	170
9	32, 33, 31, 35, 37, 38, 39, 36, 34	97.0	200
11	67, 65, 63, 62, 74, 72, 61, 64, 68, 66, 69	59.3	200
8	57, 55, 54, 53, 56, 58, 60, 59	101.7	200
9	81, 78, 76, 71, 70, 73, 77, 79, 80	127.1	150
9	98, 96, 95, 94, 92, 93, 97, 100, 99	95.8	190
11	20, 24, 25, 27, 29, 30, 28, 26, 23, 22, 21	50.7	170
13	43, 42, 41, 40, 44, 45, 46, 48, 51, 50, 52, 49, 47	64.6	160

Tableau 2.6 Instance C201. Distance totale: 589.1.

Nombre de clients	Routes	Distance	Capacité
35	93, 5, 75, 2, 1, 99, 100, 97, 92, 94, 95, 98, 7, 3, 4, 89, 91, 88, 84, 86, 83, 82, 85, 76, 71, 70, 73, 80, 79, 81, 78, 77, 96, 87, 90	237.5	620
33	20, 22, 24, 27, 30, 29, 6, 32, 33, 31, 35, 37, 38, 39, 36, 34, 28, 26, 23, 18, 19, 16, 14, 12, 15, 17, 13, 25, 9, 11, 10, 8, 21	194.3	630
32	67, 63, 62, 74, 72, 61, 64, 66, 69, 68, 65, 49, 55, 54, 53, 56, 58, 60, 59, 57, 40, 44, 46, 45, 51, 50, 52, 47, 43, 42, 41, 48	157.3	560

Tableau 2.7 Instance C202. Distance totale: 589.1.

Nombre de clients	Routes	Distance	Capacité
33	20, 22, 24, 27, 30, 29, 6, 32, 33, 31, 35, 37, 38, 39, 36, 34, 28, 26, 23, 18, 19, 16, 14, 12, 15, 17, 13, 25, 9, 11, 10, 8, 21	194.3	630
32	67, 63, 62, 74, 72, 61, 64, 66, 69, 68, 65, 49, 55, 54, 53, 56, 58, 60, 59, 57, 40, 44, 46, 45, 51, 50, 52, 47, 43, 42, 41, 48	157.3	560
35	93, 5, 75, 2, 1, 99, 100, 97, 92, 94, 95, 98, 7, 3, 4, 89, 91, 88, 84, 86, 83, 82, 85, 76, 71, 70, 73, 80, 79, 81, 78, 77, 96, 87, 90	237.5	620

Tableau 2.8 Instance C205. Distance totale: 586.4.

Nombre de clients	Routes	Distance	Capacité
33	20, 22, 24, 27, 30, 29, 6, 32, 33, 31, 35, 37, 38, 39, 36, 34, 28, 26, 23, 18, 19, 16, 14, 12, 15, 17, 13, 25, 9, 11, 10, 8, 21	194.3	630
32	67, 63, 62, 74, 72, 61, 64, 66, 69, 68, 65, 49, 55, 54, 53, 56, 58, 60, 59, 57, 40, 44, 46, 45, 51, 50, 52, 47, 43, 42, 41, 48	157.3	560
35	93, 5, 75, 2, 1, 99, 100, 97, 92, 94, 95, 98, 7, 3, 4, 89, 91, 88, 86, 84, 83, 82, 85, 76, 71, 70, 73, 80, 79, 81, 78, 77, 96, 87, 90	234.8	620

Tableau 2.9 Instance C206. Distance totale: 586.0.

Nombre de clients	Routes	Distance	Capacité
33	20, 22, 24, 27, 30, 29, 6, 32, 33, 31, 35, 37, 38, 39, 36, 34, 28, 26, 23, 18, 19, 16, 14, 12, 15, 17, 13, 25, 9, 11, 10, 8, 21	194.3	630
32	67, 63, 62, 74, 72, 61, 64, 66, 69, 68, 65, 49, 55, 54, 53, 56, 58, 60, 59, 57, 40, 44, 46, 45, 51, 50, 52, 47, 42, 41, 43, 48	156.9	560
35	93, 5, 75, 2, 1, 99, 100, 97, 92, 94, 95, 98, 7, 3, 4, 89, 91, 88, 86, 84, 83, 82, 85, 76, 71, 70, 73, 80, 79, 81, 78, 77, 96, 87, 90	234.8	620

Tableau 2.10 Instances C207 & C208. Distance totale: 585.8.

Nombre de clients	Routes	Distance	Capacité
32	67, 63, 62, 74, 72, 61, 64, 66, 69, 68, 65, 49, 55, 54, 53, 56, 58, 60, 59, 57, 40, 44, 46, 45, 51, 50, 52, 47, 42, 41, 43, 48	156.9	560
33	20, 22, 24, 27, 30, 29, 6, 32, 33, 31, 35, 37, 38, 39, 36, 34, 28, 26, 23, 18, 17, 19, 16, 14, 12, 15, 13, 25, 9, 11, 10, 8, 21	194.1	630
35	93, 5, 75, 2, 1, 99, 100, 97, 92, 94, 95, 98, 7, 3, 4, 89, 91, 88, 86, 84, 83, 82, 85, 76, 71, 70, 73, 80, 79, 81, 78, 77, 96, 87, 90	234.8	620

Tableau 2.11 Instance R201. Distance totale: 1278.5.

Nombre de clients	Routes	Distance	Capacité
22	33, 65, 63, 64, 11, 19, 62, 7, 88, 90, 18, 84, 8, 49, 46, 48, 60, 17, 91, 100, 93, 89	350.3	270
25	27, 31, 69, 39, 67, 23, 75, 73, 40, 53, 87 57, 41, 22, 56, 26, 68, 54, 74, 4, 55, 25, 24, 80, 77	304.5	375
28	5, 83, 45, 36, 47, 82, 52, 28, 12, 29, 76, 30, 71, 9, 51, 81, 79, 78, 34, 3, 50, 10, 20, 66, 35, 32, 70, 1	270.5	417
25	95, 59, 92, 14, 42, 2, 72, 21, 15, 98, 61, 16, 44, 38, 86, 85, 99, 94, 6, 96, 97, 37, 43, 13, 58	253.2	396

Tableau 2.12 Instance R204. Distance totale: 841.3.

Nombre de clients	Routes	Distance	Capacité
52	89, 6, 96, 92, 98, 85, 61, 16, 14, 43, 15 41, 22, 75, 56, 23, 67, 39, 12, 76, 28, 53, 79, 33, 81, 9, 51, 20, 66, 65, 71, 35, 34, 78, 29, 24, 55, 4, 72, 74, 73, 21, 40, 13, 94, 95, 37, 100, 42, 57, 2, 58	410.0	750
48	27, 52, 18, 82, 48, 7, 88, 31, 69, 10, 62, 11, 19, 86, 38, 44, 91, 93, 99, 59, 97, 87, 60, 8, 83, 84, 5, 17, 45, 46, 47, 36, 49, 64, 63, 90, 32, 30, 70, 1, 50, 77, 3, 68, 80, 54, 25, 26	431.3	708

CHAPITRE 3

Le problème de satisfaisabilité

Une instance du problème de satisfaisabilité (SAT) est définie par un ensemble de variables booléennes et un ensemble de clauses formé à partir de ces variables. La question associée est de déterminer s'il existe une affectation de valeurs qui satisfait simultanément toutes les clauses. Le problème de satisfaisabilité appartient à la classe des problèmes NP-complets et dans ce chapitre on présente une heuristique de type recherche tabou et un algorithme exact pour le résoudre. Une des nombreuses applications du problème SAT est la détection d'erreurs dans des systèmes (de communication, d'ordinateurs, etc.) de grande taille (voir [50] pour un exemple).

3.1 Le problème

Soit $X = \{x_1, x_2, \ldots, x_n\}$ un ensemble de n variables booléennes, et $\overline{X} = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$ l'ensemble des compléments de ces variables. Un littéral y_j est défini soit comme une variable x_j (littéral positif) ou son complément $\overline{x}_j$ (littéral négatif). Une formule booléenne F en forme normale disjonctive (FND) est définie comme une disjonction de m clauses, où chaque clause C_i s'écrit comme une conjonction de littéraux:

$$F = \bigvee_{i=1}^{m} C_i$$

où

$$C_i = \bigwedge_{j \in N_i} y_j \qquad N_i \subseteq N = \{1, 2, \ldots, n\} \qquad i = 1, 2, \ldots, m.$$

Une *affectation de valeurs* pour X est une fonction $t : X \rightarrow \{Vrai, Faux\}$. Soit $x \in X$. Si $t(x) = Vrai$, on dit que x est vrai sous t; si $t(x) = Faux$, on dit que

x est faux sous t. La valeur d'un littéral (positif) x est *Vrai* sous t si et seulement si la valeur de la variable x est *Vrai* sous t; la valeur d'un littéral (négatif) $\bar{x}$ est *Vrai* si et seulement si la valeur de la variable x est *Faux*. Une clause appartenant à une formule booléenne est satisfaite par une affectation si et seulement si au moins la valeur de l'un de ses littéraux est fause. La formule booléenne est satisfaite si et seulement si il existe une affectation qui satisfait simultanément toutes ses clauses.

Le problème de satisfaisabilité est défini comme suit: étant donnée une formule booléenne F, déterminer s'il existe une affectation de valeurs pour l'ensemble X de variables qui satisfait $F = 0$ (on utilise la convention *Vrai* $\equiv 1$ et *Faux* $\equiv 0$).

S'il existe une affectation qui rend toutes les clauses fausses, alors le problème SAT est dit *satisfaisable*; on dit encore que l'ensemble C, $C = \{C_i, 1 \leq i \leq m\}$ est *satisfaisable*. S'il n'existe pas de telle affectation, alors le problème est dit *non-satisfaisable* (l'ensemble C est *non-satisfaisable*).

Cook [9] a montré que le problème de satisfaisabilité (SAT), dans le cas où chaque clause contient au moins trois littéraux, appartient à la classe des problèmes NP-complets.

Un problème SAT dont chaque clause contient au plus k littéraux est appelé problème k-SAT. Ainsi le problème 3-SAT contient au plus trois littéraux par clause. Cook [9] a prouvé l'existence d'une transformation polynômiale pour réduire n'importe quel problème SAT ($k > 3$) à un problème 3-SAT. Une conséquence directe est l'appartenance du problème 3-SAT à la classe des problèmes NP-complets.

Nous proposons un nouvel algorithme d'énumération implicite pour le problème

de satisfaisabilité d'une expression logique. On définit une relaxation ainsi qu'un schéma de décomposition à partir des instances polynômiales, soient les problèmes de 2-satisfaisabilité et de Horn-satisfaisabilité pour les deux plus connues. Une heuristique de type Recherche Tabou est incluse pour accélérer la recherche d'une solution (s'il en existe une). Nous présentons des résultats de calcul ainsi que des expériences comparatives avec quelques uns des meilleurs algorithmes de la littérature.

Le reste du chapitre est organisé comme suit. La revue de la littérature est présentée dans la section suivante. Des schémas généraux de décomposition et relaxation, utilisant des instances polynômiales sont définies à la Section 3. Toujours à la Section 3, on rappele les implications logiques qu'on peut déduire à partir des instances polynômiales, en particulier à partir du problème 2-SAT. À la Section 4 on décrit une heuristique spécialisée de type recherche tabou. Un algorithme exact pour SAT est décrit à la Section 5. Des résultats numériques sont rapportés à la Section 6. La dernière section contient les conclusions.

3.2 Revue de la littérature

Le problème de satisfaisabilité est un problème fondamental pour la théorie de la complexité et l'intelligence artificielle. Naturellement, il a reçu une attention considérable. Plus récemment un grand nombre d'articles ont été consacrés à l'étude de techniques efficaces de résolution concernant le côté algorithmique et l'expérimentation numérique. La plupart des algorithmes exacts sont basés sur une technique de séparation et évaluation et utilisent divers tests de pré-traitement à l'intérieur de différents schémas de relaxation ou de décomposition. Tous ces algorithmes incluent le schéma de Davis et Putman [10] dans la forme de Loveland [52]. Ce schéma est connu comme le schéma DPL; il fait appel récursivement aux règles de tautologie,

littéral pur et clause unaire pour simplifier les sous-problèmes associés à chaque noeud.

Schéma DPL (Davis-Putnam-Loveland)

- **Règle de tautologie.** Eliminer toutes les clauses qui contiennent à la fois une variable et son complément. Par la suite on fait l'hypothèse que ces clauses ont été enlevées.

- **Règle de la clause unaire.** S'il y a deux clauses unaires (clauses formées par un seul littéral) C et C' tel que $C = y$ et $C' = \overline{y}$, alors F est non-satisfaisable. S'il y a une clause unaire $C = y$, une affectation de valeur qui vérifie $F = 0$, doit satisfaire $y = 0$. Eliminez de F toutes les clauses contenant y et simplifiez les clauses qui contiennent $\overline{y}$.

- **Règle du littéral pur.** Si un littéral y apparaît dans F, mais pas son complément, alors enlever de F toutes les clauses contenant y: s'il existe une affectation de variables qui satisfait $F = 0$, alors il en existe une telle que $y = 0$.

- **Règle de sélection.** Lorsque ni la règle de la clause unaire ni celle du littéral pur ne permettent de simplifications de F, alors sélectioner un littéral y et écrire F de la manière suivante:

$$F = yF^1 \vee \overline{y}F^2 \vee F^3$$

où les formules booléennes F^1, F^2 et F^3 sont indépendantes des littéraux y et $\overline{y}$. La formule booléenne F est satisfaisable si et seulement si la formule $(F^1 \wedge F^2) \vee F^3$ est satisfaisable.

La règle de sélection est très souvent définie comme une règle de séparation:

- **Règle de séparation** Lorsque ni la règle de la clause unaire ni celle du littéral pur ne permettent de simplifications de F, alors sélectioner un littéral y sur lequel brancher. La formule booléenne F est satisfaisable si et seulement si une des formules $F_1 \vee F_3$ et $F_2 \vee F_3$ est satisfaisable.

La règle de sélection est attrayante car elle évite la séparation du problème initial en sous-problèmes (qui peuvent être nombreux). Elle nécessite cependant d'écrire la formule $(F_1 \wedge F_2) \vee F_3$ sous forme FND pour appliquer récursivement le schéma DPL. D'habitude ceci implique une augmentation significative du nombre de clauses. La plupart des auteurs préférent maintenant utiliser la règle de séparation à l'intérieur d'une technique de séparation et évaluation.

Les principes majeurs des plus récents algorithmes sont décrits brièvement par la suite.

À partir d'un schéma DPL, Jeroslow et Wang [42] ont obtenu une technique efficace de séparation et évaluation (en utilisant l'exploration en profondeur d'abord). Les auteurs définissent un critère de séparation et une heuristique basée sur le nombre d'affectations de valeurs éliminées par une clause. Le Bars [51], Billionnet et Sutter [5] proposent des algorithmes qui combinent le schéma DPL avec une génération partielle des consensus.

Plusieurs auteurs ont formulé le problème SAT comme un programme linéaire en variables 0-1. Pourtant, Blair, Jeroslow et Lowe [6] ont montré que la relaxation continue "is very likely a wasteful way" d'implanter la règle de la clause unaire, car une solution de la relaxation continue ne mène pas à plus de réductions que l'application de cette règle. De plus, un vecteur dont toutes les composantes sont égales à 0.5 est toujours une solution réalisable, lorsque les clauses unaires ont été

éliminées. Ces résultats amènent à la conclusion que la relaxation continue ne permet pas toujours d'obtenir une très bonne borne supérieure (l'objectif est de minimiser le nombre de clauses avec une affectation de valeurs égale à 1, si on considère la forme FND du problème). Hooker et Fedjki [36] prouvent que l'ajout des coupes de séparation à l'intérieur de la relaxation continue du programme linéaire en variables 0-1 correspondant au problème SAT possède un intérêt et peut représenter une meilleure méthode de résolution que les algorithmes basés sur le schéma DPL, surtout pour les instances difficiles. Kamath et al. [44] ont résolu de larges instances en utilisant l'algorithme de programmation linéaire en variables 0-1 de Karmarkar, Resende et Ramakrishan [45], dévelopé à partir d'une méthode de points intérieurs. Pourtant, leur algorithme n'est pas complet, car il ne garantit pas ni de déterminer une solution lorsqu'il en existe une, ni de conclure à la non existence d'une solution lorsqu'il n'en existe pas.

D'autres relaxations ont été obtenues en considérant des instances polynômiales du problème SAT. Gallo et Urbani [20] combinent une relaxation de Horn avec un schéma de branchement polychôtomique. Les instances polynômiales peuvent être aussi utilisées dans un schéma de décomposition. Dans un papier récent, Gallo et Pretolani [21] exploitent cette idée avec des instances de Horn et montrent que l'algorithme obtenu a un meilleur comportement que celui de Gallo et Urbani [20].

3.3 Schémas de relaxation et décomposition

Les instances polynômiales les plus connues du problème SAT sont les instances 2-satisfaisabilité (2-SAT) et Horn-satisfaisabilité (Horn-SAT).

Le problème 2-SAT considère qu'une clause ne contient pas plus de 2 littéraux.

Plusieurs algorithmes, dont la complexité est linéaire, ont été proposés pour sa résolution. La majorité de ces algorithmes ont été développés à partir des propriétés des graphes. Comme exemple, on note l'algorithme en temps linéaire de Aspvall, Plass et Tarjan [2], basé sur la recherche des composantes fortement connexes.

Le problème Horn-SAT est défini comme un problème SAT dont chaque clause contient au plus un seul littéral négatif. Comme pour le problème 2-SAT, plusieurs algorithmes dont la complexité est linéaire ont été développés pour sa résolution. Dowling et Gallier [16] présentent un algorithme dont la complexité est de l'ordre $O(l)$, où l indique le nombre total d'occurrences des littéraux. Il existe aussi plusieurs instances polynômiales qui sont des variantes du Horn-SAT (par exemple, Horn-SAT déguisé, une instance qui peut être réduite à une instance Horn-SAT en prenant le complément des variables, Aspvall [1]).

Dans un algorithme exact les instances polynômiales peuvent être utilisées de deux manières différentes. La première, qui est aussi la plus facile à implanter, consiste à partager l'ensemble des clauses en deux sous-ensembles. Le premier sous-ensemble contient toutes les clauses qui satisfont la propriété demandée (quadratique ou Horn). Le deuxième sous-ensemble est composé du restant des clauses de l'instance SAT.

Algorithme 1. Schéma de relaxation.

1. Appliquer les trois premières règles du schéma DPL.
2. Séparer la formule booléenne en deux parties:

$$F = F_P \vee F_{NP}$$

où F_P correspond à une instance polynômiale du problème SAT.

3. Résoudre $F_P = 0$. Si il n'y a pas de solution, F est non-satisfaisable. Sinon, vérifier si la solution satisfait F. Si c'est le cas, arrêter: F est satisfaisable.

4. Appliquer la règle de séparation de la procédure DPL.

La deuxième manière d'utiliser une instance polynômiale consiste à séparer chaque clause en deux sous-clauses: $C_j = C_j^1 \wedge C_j^2$ tel que $F^1 = \bigvee_j C_j^1$ correspond à une instance polynômiale du problème SAT.

Algorithme 2. Schéma de décomposition.

1. Appliquer les trois premières règles du schéma DPL.

2. Séparer chaque clause C_j en deux sous-clauses C_j^1 et C_j^2 tel que

$$F^1 = \bigvee_j C_j^1$$

correspond à une instance polynômiale du problème SAT.

3. Résoudre $F^1 = 0$. Si il y a une solution, arrêter: F est satisfaisable.

4. Appliquer la règle de séparation de la procédure DPL.

Le premier algorithme est a priori préférable si on s'attend à ne pas trouver de solution, alors que le deuxième est plus efficace lorsque on s'attend à en touver une. Pourtant, il y a plusieurs manières de définir une décomposition pour l'Algorithme 2 et la difficulté est de choisir celle qui a le plus de chances à déterminer une solution pour l'équation boolénne. Un bon compromis serait de combiner les deux algorithmes.

On remarque que plus F_P contient de clauses (dans le schéma de relaxation) et plus il y a de littéraux dans les clauses de F^1 (dans le schéma de décomposition), plus les deux schémas sont efficaces respectivement. Un moyen d'augmenter le

nombre de clauses de F_P et le nombre de littéraux dans les clauses de F^1 est de générer plus de clauses à l'aide des consensus. Par exemple, si $C_1 = x_1x_2\overline{x}_3$ et $C_2 = x_1\overline{x}_2\overline{x}_3$ appartiennent à F_{NP}, leur consensus $C_{1,2} = x_1\overline{x}_3$ appartient à F_P et, s'il est généré, il améliore la qualité de la relaxation. De même, dans le schéma de décomposition, si C_1 et C_2 sont définis par $C_1 = (C_1^1 = x_1x_2\overline{x}_3) \wedge (C_1^2 = 1)$ et $C_2 = (C_2^1 = x_1\overline{x}_2 \wedge C_2^2 = \overline{x}_3)$, ajoutant leur consensus $C_{1,2} = x_1\overline{x}_3$ à F^1 peut améliorer le schéma de décomposition.

Dans ce chapitre, on étudie en particulier le schéma de relaxation avec des problèmes 2-SAT comme instances polynômiales. On note une caractéristique intéressante des problèmes 2-SAT: il est facile d'identifier les littéraux qui ont une valeur fixée dans toutes les solutions, de même que le sous-ensemble maximal de littéraux qui ont la même valeur dans toutes les solutions (Hansen, Jaumard et Minoux [34]). Ces conclusions logiques sont exploitées dans l'Algorithme 1 pour simplifier le sous-problème courant à chaque itération, et dans l'Algorithme 2 pour définir un schéma de branchement efficace, de même qu'une meilleure décomposition à l'itération suivante.

On décrit brièvement comment déduire les conclusions logiques impliquées par l'équation booléenne $F_Q = 0$. On considère le graphe $G = (V, E)$ où chaque noeud $y \in V$ est associé à un littéral de $X \cup \overline{X}$. Pour chaque clause yz de l'équation booléenne on associe deux arcs dans l'ensemble E: $(y, \overline{z})$ et $(z, \overline{y})$. On déduit les propriétés suivantes:

- (F_Q) est consistante si et seulement si il n'y a pas de composante fortement connexe contenant à la fois un littéral et son complément,
- un littéral y_j prend la valeur 0 (respectivement 1) dans toutes les solution de

(F_Q) si et seulement si il y a un chemin de y_j à $\overline{y}_j$ (respectivement de $\overline{y}_j$ à y_j) dans le graphe G,

- deux littéraux y_j et y_k ont des valeurs idéntiques dans toutes les solutions de (F_Q) si et seulement si ils appartiennent à la même composante fortement connexe du graphe G.

3.4 Une heuristique de type recherche tabou

Pour accélerer la recherche d'une solution (lorsqu'elle existe) dans l'algorithme exact, on a conçu une heuristique de type recherche tabou, adaptée aux problèmes SAT (voir Glover [29] pour une référence générale). On rappele ci-dessous le schéma général d'un algorithme de type recherche tabou et ensuite on décrit les règles spécifiques que l'on a ajouté pour améliorer sa performance.

Étape 0 *Initialisation*

. Choisir aléatoirement un vecteur booléen initial x;

change $\leftarrow$ *vrai*; $x^* \leftarrow x$; $z^* \leftarrow z(x)$:= nombre de clauses non satisfaites;

$T(i)$:= statut tabou de la variable $x_i \leftarrow 0, i = 1, \ldots, n$;

Initialiser les valeurs des paramétres *nrep* et *tabou*

Étape 1 *Étape itérative*

tant que *change* **faire**

change $\leftarrow$ *faux*;

répéter *nrep* **fois**

déterminer j, la direction de changement ;

soit x_j, (x avec sa j^e composante modifiée), le nouveau point;

$x \leftarrow x_j$;

$T(j) \leftarrow tabou$;

si $z(x_j) < z^*$ **alors**

$x^* \leftarrow x_j$;

$change \leftarrow vrai$;

fin si

$T(i) \leftarrow T(i) - 1$, pour tous $i = 1, \ldots, n$ tel que $T(i) > 0$;

fin

fin

La variable logique *change* est utilisée pour vérifier si une meilleure solution (c'est à dire avec moins de clauses non satisfaites) est trouvée durant un cycle de *nrep* échanges locaux. Le vecteur x^* correspond à la solution courante. Les valeurs des éléments de T indiquent le nombre d'itérations durant lesquelles un échange dans une direction donnée est interdit.

On utilise les règles additionnelles suivantes:

(i) **Diversification de la recherche**: on permet de recommencer l'algorithme avec cette fois-ci comme solution initiale la meilleure solution obtenue durant sa dernière application.

(ii) **Perturbation**: si on ne peut pas trouver de variable qui améliore la fonction objectif (c'est-à-dire le nombre de clauses satisfaites) en complémentant sa valeur et, si durant deux itérations les mêmes clauses prennent la valeur *Vrai*, on complémente la valeur de la variable, parmi celles apparaissant sous une forme ou autre dans les clauses prennant la valeur vrai, avec un statut tabou égal à 0 et menant à la moindre détérioration de la fonction objectif.

(iii) **Fréquence**: si on ne peut pas améliorer la fonction objectif par une application de la règle de perturbation, et si pour les deux dernières itérations les mêmes clauses prennent la valeur *Vrai*, on choisit la variable qui a changé de valeur à l'itération la plus éloignée.

(iv) **Aspiration**: on enlève le statut tabou d'une variable si complémenter sa valeur mène à une amélioration de la valeur courante de la fonction objectif.

3.5 L'algorithme exact

L'algorithme exact que l'on a développé est fondé sur un schéma d'énumération de type profondeur d'abord. Plusieurs règles ont été ajoutées pour augmenter le nombre de sous-problèmes (noeuds) éliminés. D'abord, on utilise les règles de réduction du schéma DPL. Ensuite, on applique l'algorithme de recherche tabou présenté à la section précédente. Si on trouve une solution, l'algorithme s'arrête: le problème est satisfaisable. Dans le cas contraire, on essaye de réduire le problème courant en utilisant un schéma de relaxation où F_P est défini par la conjonction de tous les termes quadratiques. On dérive les conclusions logiques de $F_P = 0$ et le problème courant est réduit en conséquence. Si $F_P = 0$ n'est pas cohérent, on exécute un retour en arrière: le sous-problème courant n'admet pas de solution. Finalement, si le sous-problème n'a pas été résolu, on le sépare en deux nouveaux sous-problèmes.

3.6 Résultats numériques

Il est bien connu que pour un k fixé, la difficulté d'un problème k-SAT est donnée par le rapport entre m et n. Si le rapport entre m et n est plus petit qu'un seuil, alors l'instance contient beaucoup de solutions et donc elle est facile à résoudre. Autour du seuil une instance contient un nombre très réduit de solutions et c'est difficile de

prouver sa cohérence. Plus le rapport entre m et n excède le seuil, plus c'est facile de montrer que l'instance n'admet pas de solution. Si $k = 3$, alors le seuil prend la valeur 3.5. Pour évaluer les performances d'un algorithme, les problème-tests doivent considérer, pour un ensemble de paramètres m, n et k donné, des rapports entre m et n qui se trouvent d'un côté et de l'autre du seuil.

Les algorithmes présentés aux sections précedentes ont été codés dans le langage C et exécutés sur une station de travail SPARC10MOD30 (86 mips, 10.6 mflops, 32M mémoire). On a implanté une structure de données efficace pour permettre l'accès et la mise à jour faciles des sous-problèmes d'une itération à l'autre. De plus, on a gardé l'espace mémoire au minimum. D'ailleurs, à chaque itération la plupart des tests sont éxécutés en temps linéaire. On décrit la structure de données dans la sous-section suivante. Les problème-tests sont introduits à la sous-section 3.6.2. La pertinence de la relaxation 2-SAT est discutée à la sous-section 3.6.3. Plusieurs stratégies de branchement sont discutées à la sous-section 3.6.4 L'efficacité de l'heuristique de recherche tabou est investiguée à la sous-section 3.6.5. On présente des comparaisons avec des algorithmes de la littérature à la sous-section 3.6.6.

3.6.1 Structure de données

Chaque instance du problème SAT est représentée utilisant des listes circulaires doublement chaînées orthogonales. Verticalement, il y a $2n$ listes doublement chaînées: chaque liste correspond à la liste des clauses contenant un littéral donné. Horizontalement, il y a m listes circulaires doublement chaînées: chaque liste contient les littéraux d'une clause donnée. Une telle structure correspond à une double représentation d'une instance. Cette redondance est fortement balancée par la flexibilité des listes doublement chaînées pour se déplacer très rapidement entre les

noeuds de l'arbre de branchement. Des procédures efficaces sont utilisées pour fixer les valeurs des variables lors du branchement ou l'application des règles de réduction et pour les restitutions qui arrivent lors des étapes de retour en arrière. Construire les listes orthogonales prend un temps de l'ordre $O(l)$, où l indique le nombre total d'occurrences des littéraux. Ajouter et enlever un élément dans une liste circulaire doublement chaînée prend un temps constant. De plus, les règles du littéral pur et de la clause unaire prennent, respectivement, un temps de l'ordre $O(m)$ et $O(n)$.

3.6.2 Les problèmes tests

Pour obtenir des problèmes tests on a considéré deux modèles: le modèle de la densité fixe et le modèle de la probabilité fixe (Hooker et Fedjki [36]). Dans le modèle de la densité fixe, chaque clause est construite en choisissant aléatoirement k variables distinctes de l'ensemble $\{x_1, x_2, \ldots, x_n\}$ et en prenant le complément de chacune avec une probabilité de 0.5. On a implanté ce modèle en utilisant un algorithme de Nijenhuis et Wilf [53] qui permet la restauration et génère une clause dans un temps de l'ordre $O(k)$. Le modèle de probabilité fixe construit chaque clause en permettant l'apparition de chaque variable avec la probabilité p (p est tel que l'espérance du nombre de littéraux par clause est k) et prenant le complément de chaque variable avec la probabilité 0.5. Pour ne pas obtenir d'instance facile, on ne considère pas les clauses unaires ou vides. Pour chaque instance obtenue à partie d'un des modèles, m, n et k sont les paramètres d'entrée. Même si le modèle de la densité fixe peut être moins pertinent pour générer des problèmes réels (Hooker et Fedjki [36]), les instances obtenues sont plus difficiles à résoudre que celles générées par le modèle de la probabilité fixe (pour des valeurs identiques de m, n et k).

3.6.3 Relaxation 2-SAT

En dépit du fait que la dérivation des conclusions logiques peut être obtenue en temps polynômial $O(n^3)$ (et en temps espéré $O(n)$, dépendamment du problème test), il coûte trop cher (en unités de temps) de les calculer à chaque noeud de l'arbre de branchement. Utilisant les conclusions des résultats numériques de Hansen, Jaumard et Minoux [34], on a ajouté un test qui limite l'usage de la relaxation 2-SAT aux cas où le nombre de clauses de la relaxation quadratique est plus grand que 1.3 fois le nombre de variables qui y apparaissent. Les résultats sont présentés à la table 3.1. Les paramètres *noeuds*, *totalcpu* et *lcacpu* indiquent, respectivement, le nombre de noeuds de l'arbre de branchement, le temps total et le temps pour dériver les conclusions logiques. Les temps sont donnés en secondes. On a noté *pourcent* le pourcentage de problèmes satisfaisables. Chaque rangée de la table correspond à des moyennes obtenues sur des séries de 10 problèmes. Pour chaque rangée, la première ligne correspond à des résultats obtenus en utilisant la relaxation 2-SAT et la deuxième ligne correspond à des résultats pour le cas où la relaxation n'est pas employée.

Dans la première partie de la table 3.1, la règle de branchement de l'algorithme exact est la même que celle de Jeroslow et Wang [42], alors que dans la deuxième partie, on branche sur la variable qui mène au sous-problème contenant le plus grand nombre de clauses quadratiques. On observe que la première règle de branchement est beaucoup plus efficace que la deuxième. La relaxation quadratique est utile seulement avec la deuxième règle de branchement, menant à une grande réduction dans le nombre de noeuds et le temps d'exécution.

3.6.4 Règles de branchement

Nous avons observé, en étudiant les résultats numériques des articles consacrés au problème SAT et aussi en testant différentes règles de branchement avec notre algorithme exact, que la règle de branchement joue un rôle crucial pour l'efficacité d'un algorithme exact donné. Dans plusieurs cas, ce rôle est plus important que celui des règles de réduction. Nous avons comparé sept méthodes de branchement, qui sont décrites ci-dessous.

La règle de Jeroslow et Wang [42] comprend les considérations suivantes. Soit $\mathcal{C}$ un ensemble de clauses. On définit le *poids* de $\mathcal{C}$ comme suit:

$$w(\mathcal{C}) = \sum_{p} N_p 2^{-p}$$

où N_p est le nombre de clauses de longueur p de $\mathcal{C}$, c'est-à-dire le nombre de clauses qui ont p littéraux.

Pour une variable x_i et une affectation de valeurs t, soit C_i^t le sous-ensemble de $\mathcal{C}$ où x_i apparaît sous forme directe ($t(x_i) = 1$) ou sous forme complémentée ($t(x_i) = 0$). L'heuristique pour la variable de branchement est de choisir t^* et i^* qui maximisent $w(C_i^t)$, c'est-à-dire:

$$w(C_{i^*}^{t^*}) = \max_{t,i} w(C_i^t).$$

Alors, la variable de branchement sera x_{i^*}, fixant d'abord x_{i^*} à 1 si $t^* = 1$ ou à 0 si $t^* = 0$.

Pour chaque variable x_i et son affectation $t(x_i) = 1$ ou 0, $w(C_i^t)$ peut être considéré comme un indicateur du degré auquel l'ensemble des clauses de $\mathcal{C}$ qui restent est relaxé après avoir fixé x_i à la valeur t. Un poids important $w(C_j^i)$ implique

l'existence d'un plus grand nombre d'affectations dans le sous-ensemble de clauses de $\mathcal{C}$ qui reste après avoir fixé x_i à la valeur t. Alors on maximise la probabilité de trouver une affectation qui satisfait toutes les clauses qui restent, ce qui mène à déterminer une solution du problème près de la racine de l'arbre de recherche, si une telle solution existe.

Dubois et al. [17] considèrent aussi des poids associés aux clauses. Le poids d'une clause C de longueur p est défini par $w(C^p) = -ln(1 - 1/(2^p - 1)^2)$. On remarque que les poids sont décroissants avec la longueur de la clause. Les auteurs tiennent compte des occurences des variables dans toutes les clauses et définissent deux règles de branchement. La première est basée sur le choix d'un littéral, alors que la deuxième repose sur le choix d'une variable. Les définitions des fonctions de branchement s'écrivent, respectivement, de la manière suivante:

$$\max_x(f(x), f(\overline{x}))$$

et

$$\max_x(f(x) + f(\overline{x}) + \alpha\min(f(x), f(\overline{x})))$$

où $f(x) = \sum_p w(C^p)|x|_p$ et $|x|_p$ indique le nombre d'occurences de la variable x dans des clauses de longueur p. Dubois et al. [17] considèrent aussi des modifications mineures à ces règles de branchement pour tenir compte de l'importance des clauses quadratiques.

Les autres méthodes de branchement sont décrites ci-dessous:

BR1: choisir un littéral selon la règle de Jeroslow et Wang [42] et explorer d'abord la branche où il est égal à 0.

BR2: choisir un littéral selon la règle de Jeroslow et Wang [42] et explorer d'abord la branche où la variable correspondante (littéral positif) est égale à 0.

BR3: choisir un littéral selon la règle de Dubois et al. [17] et explorer d'abord la branche où il est égal à 0.

BR4: choisir une variable selon la règle de Dubois et al. [17] et explorer d'abord la branche où elle est égale à 0.

BR5: choisir le littéral qui apparaît le plus souvent dans les clauses de plus petite cardinalité de l'instance et explorer d'abord la branche où il est égal à 0.

BR6: choisir le littéral qui apparaît le plus souvent dans l'ensemble de clauses et explorer d'abord la branche où il est égal à 0.

BR7: choisir le littéral qui apparaît le plus souvent dans l'ensemble de clauses et explorer d'abord la branche où la variable correspondante (littéral positif) est égale à 0.

Les résultats obtenus sur des instances générées à partir du modèle de la probabilité fixe sont résumés dans les tables 3.2 et 3.2. On a considéré six séries de 100 instances: deux ayant des caractéristiques identiques à celles de Kamath et al. [44] (kamath1: $n = 1000$, $m = 8000$ et $k = 10$; kamath2: $n = 1000$, $m = 2000$ et $k = 3$), deux similaires à Hooker and Fedjki [36] (hooker1: $n = 100$ et $m = 900$; hooker2: $n = 100$ et $m = 850$), et deux autres provenant de Gallo et Pretolani [21] (gallo1: $n = 50$ et $m = 1100$; gallo2: $n = 50$ et $m = 1000$). Nous avons aussi séparé les instances satisfaisables (table 3.2) de celles non satisfaisables (table 3.3) pour mieux souligner les performances des règles de branchement.

La règle de branchement BR5 se détache des autres autant pour des instances satisfaisables que non-satisfaisables. Elle est dérivée de l'observation que les clauses courtes (en nombre de littéraux) réduisent l'espace des solutions plus vite que les

clauses longues. Une clause unaire fixe une variable à une valeur, une clause quadratique mène à la fixation d'une de ces variables si l'autre variable est déjà fixée. Donc, il est désirable de choisir comme variable de branchement une variable qui apparaît dans les clauses les plus courtes. De cette manière on essaye de réduire l'espace de solutions le plus vite possible (le plus près de la racine). Le même raisonnement conduit à choisir la variable avec le plus d'apparitions dans les clauses courtes.

3.6.5 Recherche tabou

Les paramètres *tabou* et *nrep* ont été fixés à des valeurs aléatoirement générés entre $0.1n$ et $0.2n$ et, respectivement, 100. Dans le but d'évaluer l'efficacité et la robustesse des règles spéciales qu'on a décrit à la section 4, nous avons considéré plusieurs séries de 10 problèmes tests sur lesquels nous avons appliqué l'heuristique de recherche tabou avec les règles (i), (ii) et (iii), seulement avec les règles (ii) et (iii), seulement avec les règles (i) et (ii) et seulement avec les règles (i) et (iii). On a toujours utilisé la règle (iv), le critère d'aspiration.

Chaque ligne de la table 3.4 correspond à des moyennes sur 10 problèmes, et chaque problème a été résolu jusqu'à 10 fois. Tous les problèmes tests de cette table sont satisfaisables. Pour chaque problème test nous avons indiqué l'itération où la recherche tabou réussit à déterminer une solution.

De plus, pour chaque ligne de la table nous avons indiqué les règles spécifiques qui ont été appliquées. Les résultats obtenus montrent que la combinaison des quatres règles spécifiques est performante, car dans la majorité des séries de problèmes tests, elle trouve une solution en moins de temps.

3.6.6 Résultats numériques comparatifs

On compare les performances de notre algorithme avec trois autres algorithmes de la littérature: ceux de Kamath et al. [44] Hooker et Fedjki [36] et Gallo et Pretolani [21]. Pour chaque comparaison, on utilise un générateur de problèmes tests similaire à celui des auteurs, avec les mêmes paramètres. Ainsi les problèmes tests, même s'ils ne sont pas les identiques à ceux considérés dans les articles cités, ont la même structure et les mêmes caractéristiques. Dans le but de donner plus de détails sur nos résultats, nous avons inclus l'écart type (σ) pour les temps de calcul.

Comme stratégie de branchement, la nouvelle variable à fixer est celle qui apparaît le plus souvent dans les clauses. De cette manière on réduit le plus vite possible la taille de l'équation booléenne.

Les tables 3.5 et 3.6 contiennent la comparaison des résultats numériques obtenus par notre algorithme et celui de Kamath et al. [44]. Pour vérifier l'efficacité de l'heuristique de recherche tabou, chaque problème de la table 3.5 a été résolu 100 fois à la table 3.6. Les paramètres sont: *noeuds* pour le nombre de noeuds dans l'arbre de branchement, *taboucpu* pour le temps dépensé dans l'heuristique de recherche tabou, *totalcpu* pour le temps total de notre algorithme, *kamath* pour le temps total de l'algorithme de Kamath et al. [44] sur une machine parallèle KORBX et σ pour l'écart type des temps de calcul. Tous les temps sont donnés en secondes. Une comparaison grossière entre la vitesse d'un SPARC10 (86 mips) et KORBX (32 mips) indique un ratio de 2.7 en faveur du premier. On conclu que notre algorithme est beaucoup plus rapide (un ordre de grandeur) que celui de Kamath et al. [44].

Les tables 3.7, 3.8, et 3.9, et 3.10 contiennent, respectivement, les comparaisons des résultats numériques obtenus par notre algorithme et ceux de Hooker et

Fedjki [36] et Gallo et Pretolani [21]. Chaque rangée indique des moyennes obtenues sur des séries de 10 ou 20 problèmes.

Dans les deux colonnes les plus à droite, nous avons rapporté les résultats des autres algorithmes sur des problèmes similaires. Les nombres entre parenthèses correspondent aux pourcentages des instances satisfaisables dans une série. Les paramètres sont identiques à ceux de la table 3.5. Pour chaque paire de rangées, la première correspond à des résultats obtenus en utilisant la relaxation 2-SAT, alors que dans la deuxième la relaxation n'est pas incluse dans l'algorithme. Nous avons aussi séparé les instances satisfaisables de celles non satisfaisables pour mieux souligner l'efficacité de l'heuristique de recherche tabou pour les instances satisfaisables (et le fait qu'elle est inutile pour les instances non satisfaisables). Même s'il est difficile de comparer les temps de calcul sur différentes machines, une évaluation grossière montre que le IBM PS/2 (5 mips) peut être considéré 17 fois moins rapide qu'un SPARC10 et qu'un VAX8530 est estimé être 2 fois moins rapide qu'un SPARC10. Il résulte que notre algorithme semble être plus rapide que ceux de Hooker et Fedjki [36] et Gallo et Pretolani [21].

3.7 Conclusions

Nous avons développé une heuristique et une méthode exacte pour le problème SAT. On a obtenu des résultats numériques pour les deux algorithmes en étudiant l'application des règles spécifiques au problème SAT à un algorithme de recherche tabou, et, respectivement, le comportement de l'algorithme exact par rapport à un large ensemble de méthodes de branchement. De plus, nous avons comparé les performances de l'algorithme exact avec trois autres algorithmes de la littérature. La contribution de ce chapitre repose sur l'adaptation de la méthode de recherche tabou

au problème SAT et sur le développement des procédures efficaces à l'intérieur de l'algorithme exact.

Notre algorithme a participé à la Deuxième Compétition DIMACS (11-13 octobre, 1993), organisée par l'Université Rutgers. Toutes les instances proposées par les organisateurs on été résolues par notre algorithme exact. Durant la période de la compétition nous avons comparé notre algorithme à ceux des autres participants. Les résultats, selon les problèmes tests, nous ont été plus ou moins favorables. Il faut noter que les performances des algorithmes n'ont pas été hiérarchisées.

Tableau 3.1 Relaxation 2-SAT (modèle de la probabilité fixe).

			Instances Satisfaisables			Instances Non-satisfaisables		
	m	pourcent	noeuds	lcacpu	totalcpu	noeuds	lcacpu	totalcpu
BRa	375	90	10.8	0.0	0.03	6.0	0.0	0.05
			10.8		0.03	6.0		0.05
	410	70	34.0	0.01	0.08	15.3	0.0	0.05
			35.1		0.07	15.3		0.05
	450	0				37.2	0.03	0.12
						42.4		0.10
BRb	375	90	561	0.98	1.57	2098	3.58	6.19
			3735		2.82	28662		18.10
	410	70	519	0.95	1.62	529	0.98	1.70
			2811		2.85	5962		6.65
	450	0				907	3.25	4.49
						16017		12.90

Tableau 3.2 Comparaison des règles de branchement (1).

			Instances satisfaisables			
	instance	pourcent	noeuds	σ (noeuds)	totalcpu	σ (totalcpu)
BR1	kamath1	100	355.6	5.8	52	0.47
	kamath2	100	44.8	8.6	1.43	0.21
	hooker1	32	55.1	51.8	0.60	0.43
	hooker2	66	66.5	98.6	0.59	0.57
	gallo1	25	13679.8	14960.3	87	98
	gallo2	91	13249.4	16489.0	73	94
BR2	kamath1	100	368.0	8.3	53	0.60
	kamath2	100	30.1	6.5	1.04	0.16
	hooker1	32	41.8	33.4	0.52	0.31
	hooker2	66	53.0	69.9	0.54	0.49
	gallo1	25	11854.4	9407.3	87	71
	gallo2	91	8989.0	9728.8	54	59
BR3	kamath1	100	318.5	9.8	202	5.45
	kamath2	100	30.8	5.5	4.04	0.64
	hooker1	32	83.1	65.1	2.51	1.77
	hooker2	66	73.8	67.5	2.08	1.61
	gallo1	25	20861.3	10298.0	443	213
	gallo2	91	22617.0	15860.6	399	274
BR4	kamath1	100	381.3	7.7	259	5.5
	kamath2	100	24.5	5.3	3.26	0.65
	hooker1	32	64.5	63.4	1.91	1.52
	hooker2	66	53.5	99.5	1.58	2.28
	gallo1	25	12876.2	9591.7	262	199
	gallo2	91	9111.0	10074.4	153	173
BR5	kamath1	100	409.5	7.8	6.60	0.10
	kamath2	100	56.0	11.0	0.51	0.04
	hooker1	32	55.9	56.3	0.36	0.26
	hooker2	66	60.0	61.5	0.36	0.29
	gallo1	25	13963.9	10308.0	46	35
	gallo2	91	15349.2	14517.4	42	40
BR6	kamath1	100	275.9	5.3	3.14	0.03
	kamath2	100	32.3	5.4	0.33	0.02
	hooker1	32	212.4	215.5	1.02	0.92
	hooker2	66	333.0	665.0	1.42	2.56
	gallo1	25	52965.5	45559.4	154	136
	gallo2	91	31508.3	45797.7	77	115
BR7	kamath1	100	284.2	5.7	3.10	0.04
	kamath2	100	26.6	5.3	0.33	0.02
	hooker1	32	259.5	313.9	1.41	1.63
	hooker2	66	270.3	543.5	1.15	2.24
	gallo1	25	57705.5	43128.9	179	134
	gallo2	91	48372.5	54826.2	127	145

Tableau 3.3 Comparaison des règles de branchement (2).

			Instances Non-satisfaisables			
	instance	pourcent	noeuds	σ (noeuds)	totalcpu	σ (totalcpu)
BR1	kamath1	100				
	kamath2	100				
	hooker1	32	104.7	137.3	0.59	0.43
	hooker2	66	107.1	99.8	1.05	0.76
	gallo1	25	58218.3	5893.4	396	40
	gallo2	91	81509.3	11871.6	470	68
BR2	kamath1	100				
	kamath2	100				
	hooker1	32	50.2	45.1	0.70	0.40
	hooker2	66	49.8	41.3	0.64	0.36
	gallo1	25	34108.6	3080.2	257	22
	gallo2	91	42906.0	3280.4	280	20
BR3	kamath1	100				
	kamath2	100				
	hooker1	32	99.7	133.1	3.21	3.54
	hooker2	66	108.2	134.0	2.98	3.02
	gallo1	25	37652.7	3490.3	787	69
	gallo2	91	54340.8	7117.5	948	119
BR4	kamath1	100				
	kamath2	100				
	hooker1	32	88.4	115.2	2.88	3.10
	hooker2	66	87.5	125.9	2.50	2.80
	gallo1	25	32714.4	2848.4	686	55
	gallo2	91	47505.4	3895.5	832	63
BR5	kamath1	100				
	kamath2	100				
	hooker1	32	85.3	87.7	0.69	0.47
	hooker2	66	90.8	97.8	0.66	0.48
	gallo1	25	44808.3	4155.9	154	13
	gallo2	91	64988.6	6336.8	183	18
BR6	kamath1	100				
	kamath2	100				
	hooker1	32	504.1	554.0	2.75	2.44
	hooker2	66	700.1	1042.5	3.35	4.25
	gallo1	25	191560.4	24335.6	584	71
	gallo2	91	249481.4	32918.9	642	81
BR7	kamath1	100				
	kamath2	100				
	hooker1	32	326.6	446.0	1.92	2.04
	hooker2	66	334.1	408.7	1.82	1.76
	gallo1	25	143257.4	16740.1	458	51
	gallo2	91	203989.0	22214.0	543	56

Tableau 3.4 Performance et robustesse de l'heuristique de recherche tabou.

Règles inclues	L'itération où on trouve une solution										Temps de calcul (secondes)	Itérations
	1	2	3	4	5	6	7	8	9	10		
	$n = 100$, $m = 700$, 5-sat (probabilité fixe)											
(1),(2),(3)	7	1	1	1	2	1	1	1	1	2	0.53	1.8
(2),(3)	5	1	1	1	5	1	1	1	1	> 10	1.19	1.9 (1)
(1),(2)	3	1	1	> 10	2	2	1	1	1	2	0.71	1.6 (1)
(1),(3)	1	2	1	> 10	> 10	1	1	1	1	2	0.82	1.2 (2)
	$n = 1000$, $m = 2000$, 3-sat (probabilité fixe)											
(1),(2),(3)	1	1	1	1	1	1	1	1	1	1	4.16	1.0
(2),(3)	1	1	1	1	1	1	1	1	1	1	4.16	1.0
(1),(2)	2	1	1	1	1	2	1	1	1	1	4.22	1.2
(1),(3)	> 10	> 10	> 10	> 10	> 10	> 10	> 10	> 10	> 10	> 10	14.15	10(10)
	$n = 100$, $m = 1400$, 5-sat (densité fixe)											
(1),(2),(3)	1	1	1	1	1	1	1	1	3	1	6.62	0.66
(2),(3)	1	1	1	1	1	1	1	1	5	1	8.29	0.82
(1),(2)	1	1	1	1	1	1	2	1	1	3	6.69	0.66 (1)
(1),(3)	1	2	1	1	1	1	1	1	> 10	2	9.97	0.99
	$n = 100$, $m = 350$, 3-sat (densité fixe)											
(1),(2),(3)	1	1	1	1	1	1	1	1	1	2	0.23	1.1
(2),(3)	1	1	1	1	1	1	1	1	1	6	0.34	1.5
(1),(2)	1	1	1	1	2	> 10	1	2	1	1	0.31	1.2 (1)
(1),(3)	1	1	1	1	1	1	1	1	1	1	0.19	1.0
	$n = 50$, $m = 900$, 5-sat (densité fixe)											
(1),(2),(3)	1	10	8	> 10	> 10	1	> 10	5	> 10	> 10	2.82	5.0 (5)
(2),(3)	1	> 10	5	5	4	1	7	2	> 10	> 10	3.99	3.6 (3)
(1),(2)	3	> 10	9	> 10	> 10	1	2	1	> 10	2	2.23	3.0 (4)
(1),(3)	> 10	> 10	> 10	> 10	> 10	> 10	> 10	> 10	> 10	> 10	3.53	> 10(10)

Tableau 3.5 Comparaison avec l'algorithme de Kamath et al. (1).

			Moyennes sur des séries de 100 problèmes Chaque problème résolu une fois				
n	m	k	noeuds	taboucpu	σ (taboucpu)	totalcpu	kamath
50	100	5	0	0.002	0.004	0.01	0.7
100	200	5	0	0.005	0.005	0.03	1.1
200	400	7	0	0.009	0.004	0.08	3.5
400	800	10	0	0.02	0.01	0.22	5.6
400	800	7	0	0.03	0.02	0.17	7.8
500	1000	10	0	0.02	0.004	0.27	7.4
1000	2000	10	0	0.08	0.03	0.60	18.5
1000	2000	7	0	0.17	0.08	0.54	21.5
1000	2000	3	0	1.16	0.3	1.35	50.4
1000	4000	10	0	0.17	0.05	1.23	25.1
1000	4000	4	52.2	5.79	1.67	6.41	1085.4
1000	8000	10	0	0.46	0.13	2.58	38.0
1000	16000	10	0	1.31	0.19	5.27	66.4
1000	32000	10	0	5.08	1.11	13.60	232.4

Tableau 3.6 Comparaison avec l'algorithme de Kamath et al. (2).

			Moyennes sur des séries de 100 problèmes Chaque problème résolu 100 fois				
n	m	k	noeuds	taboucpu	σ (taboucpu)	totalcpu	kamath
50	100	5	0	0.001	0.0003	0.01	0.7
100	200	5	0	0.004	0.0007	0.03	1.1
200	400	7	0	0.008	0.0007	0.08	3.5
400	800	10	0	0.02	0.0008	0.22	5.6
400	800	7	0	0.03	0.003	0.17	7.8
500	1000	10	0	0.02	0.0007	0.27	7.4
1000	2000	10	0	0.07	0.002	0.60	18.5
1000	2000	7	0	0.16	0.01	0.54	21.5
1000	2000	3	0.4	1.18	0.2	1.37	50.4
1000	4000	10	0	0.17	0.007	1.23	25.1
1000	4000	4	49	4.7	0.72	5.25	1085.4
1000	8000	10	0	0.45	0.02	2.57	38.0
1000	16000	10	0	1.30	0.03	5.55	66.4
1000	32000	10	0	4.89	0.17	13.42	232.4

Tableau 3.7 Comparaison avec l'algorithme de Hooker et Fedjki (1).

		Instances satisfaisables				Hooker et Fedjki	
m	pourcent	noeuds	taboucpu	totalcpu	σ(totalcpu)	noeuds	totalcpu (min.)
600	100	0 19	0.04	0.18 0.36	0.03 0.07		
650	100	0 14.8	0.06	0.22 0.35	0.05 0.05		
700	100	3 15.9	0.21	0.42 0.36	0.28 0.08	29 (100)	43
800*	70	33.7 71	0.33	1.01 1.08	1.39 1.13	34 (55)	77
850*	55	84 177	0.40	1.70 2.44	1.84 1.53	28 (20)	69
900	40	64 172	0.417	1.35 2.45	1.02 1.93	34 (10)	93
1000	0					7 (0)	28
1500	0					3.8 (0)	22
						IBM PS/2 MODEL 80 OS/2 VERSION 1.1 OPER.SYS.	

* moyennes sur 20 problèmes

Tableau 3.8 Comparaison avec l'algorithme de Hooker et Fedjki (2).

		Instances Non-satisfaisables				Hooker et Fedjki	
m	pourcent	noeuds	taboucpu	totalcpu	σ(totalcpu)	noeuds	totalcpu (min.)
600	100						
650	100						
700	100					29 (100)	43
800*	70	138.6 138.6	0.72	2.76 2.00	1.93 1.84	34 (55)	77
850*	55	106 106	0.74	2.45 1.73	0.87 0.88	28 (20)	69
900	40	226 226	0.752	3.99 3.31	3.15 3.20	34 (10)	93
1000	0	121 121	0.82	2.75 1.95	2.38 2.38	7 (0)	28
1500	0	3.4 3.4	1.22	1.68 0.46	0.17 0.04	3.8 (0)	22
						IBM PS/2 MODEL 80 OS/2 VERSION 1.1 OPER.SYS.	

* moyennes sur 20 problèmes

Tableau 3.9 Comparaison avec l'algorithme de Gallo et Pretolani (1).

		Instances satisfaisables				Gallo et Pretolani	
m	pourcent	noeuds	lcacpu	totalcpu	σ(totalcpu)	noeuds	totalcpu
900	100	4,761 6,185	2.25	8.46 7.58	5.48 4.91	1,618 (100)	20.40
1000	90	41,688 55,348	20.87	85.67 81.23	78.62 75.16	27,308 (80)	393.34
1100	20	47,767 64,208	25.64	111.59 108.77	91.98 90.24	46,840 (10)	767.68
1200	0					35,650 (10)	681.84
1300	0					26,714 (0)	561.44
						VAX 8530	

Tableau 3.10 Comparaison avec l'algorithme de Gallo et Pretolani (2).

		Instances Non-satisfaisables				**Gallo et Pretolani**	
m	pourcent	noeuds	lcacpu	totalcpu	σ(totalcpu)	noeuds	totalcpu
900	100					1,618 (100)	20.40
1000	90	139,960 18,4182	69.63	288.69 271.06	0 0	27,308 (80)	393.34
1100	20	98,896 131,309	51.70	233.02 227.56	25.64 26.67	46,840 (10)	767.68
1200	0	73,914 99,095	40.68	201.51 207.01	19.42 19.66	35,650 (10)	681.84
1300	0	63,027 85,582	36.02	194.77 206.56	21.37 22.58	26,714 (0)	561.44
						VAX 8530	

CONCLUSION

La contribution de cette thèse est le dévelopement de nouveaux algorithmes pour les suivants problèmes de l'optimisation combinatoire: le problème du voyageur de commerce avec fenêtres de temps, le problème de fabrication des tournées de véhicules avec fenêtres de temps et le problème de satisfaisabilité. Nous avons réalisé des implantations informatiques de ces algorithmes et nous avons prouvé leur efficacité par des expérimentations numériques présentées aux Chapitres 1, 2 et 3.

Le dénominateur commun des Chapitres 2 et 3 est l'utilisation de la méthode de recherche tabou. Elle est employée pour résoudre le problème de fabrication des tournées de véhicules avec fenêtres de temps et, respectivement, le problème de satisfaisabilité. Pour ces problèmes nous avons montré comment s'adapter à leurs caractéristiques en définissant des voisinages et règles permettant de sortir des points de minima locaux spécifiques. Le premier chapitre, consacré au problème du voyageur de commerce avec fenêtres de temps, a introduit une procédure qui permet l'obtention d'une solution initiale réalisable pour la grande majorité des instances considérées. La difficulté associée à l'obtention d'une solution initiale est plus prononcée pour les instances caractérisées par des fenêtres de temps étroites, car le nombre de solutions réalisables est réduit.

Au Chapitre 3 nous avons aussi implanté un algorithme exact pour le problème de satisfaisabilité. Bien que d'un point de vue théorique l'algorithme ne constitue pas une nouveuté, la structure de données utilisée pour son implantation le rend très rapide. L'algorithme nous a servi pour évaluer differentes stratégies de branchement sur un nombre élevé d'instances. La thèse a permis de confirmer la valeur de bonnes

heuristiques, la recherche tabou en particulier, pour résoudre de problèmes complexes de l'optimisation combinatoire.

BIBLIOGRAPHIE

[1] ASPVALL, B. (1980). Recognizing Disguised NR(1) Instances of the Satisfiability Problem. *Journal of Algorithms*, **1**, 97-103.

[2] ASPVALL, B., PLASS M.F., et TARJAN R.E. (1979). A Linear Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters* **8**, 121-123.

[3] AUSIELLO, G. et ITALIANO, G. F. (1991). On-line Algorithms for Polynomially Solvable Satisfiability Problems. *The Journal of Logic Programming* **10**, 69-90.

[4] BAKER, E. (1983). An Exact Algorithm for the Time Constrained Traveling Salesman Problem. *Operations Research*, **31**, 938-945.

[5] BILLIONNET, A. et SUTTER, A. (1992). An Efficient Algorithm for the 3-Satisfiability Problem. *Operations Research Letters* **12**, 29-36.

[6] BLAIR, C.E., JEROSLOW, R.G. et LOWE, J.K. (1986). Some Results and Experiments in Programming Techniques for Propositional Logic. *Computers and Operations Research* **13**, 633-645.

[7] CHANG, C. et LEE, R. (1973). *Symbolic Logic and Mechanical Theorem-Proving.* Academic Press, New York.

[8] CHRISTOFIDES, N., MINGOZZI, A. et TOTH, P. (1981). State Space Relaxation Procedures for the Computation of Bounds to Routing Problems. *Networks*, **11**, 145-164.

[9] COOK, S. (1971). The complexity of Theorem-Proving Procedures. *Proceedings of the 3rd ACM Symposium on Theory of Computing*, 151-158.

[10] DAVIS, M. et PUTNAM, H. (1960). A Computing Procedure for Quantification Theory. *Journal of the ACM* **7**, 201-215.

[11] DEMING, R.W. (1979). Independence Numbers of Graphs - an Extension of the Koenig-Egervary Theorem. *Discrete Mathematics* **27**, 23-33.

[12] DESROCHERS, M. (1988). An algorithm for the Shortest Path Problem with Resource Constraints. *Cahiers du GERAD G-88-27, École des H.E.C., Montréal, Canada, H3T 1V6.*

[13] DESROCHERS, M., DESROSIERS, J., et SOLOMON, M. (1992). A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, **40**, 342–354.

[14] DESROCHERS, J., LENSTRA, J.K., SAVELSBERGH M.W.P. et SOUMIS. F. (1993). Vehicle Routing with Time Windows: Optimization and Approximation. *Vehicle Routing: Methods and Studies*, B.L. Golden and A.A. Assad (eds.), North-Holland, Amsterdam, 65-84.

[15] DESROSIERS, J., DUMAS, Y., SOLOMON, M.M. et SOUMIS, F. (1995). Time Constrained Routing and Scheduling. *Handbooks in Operations Research and Management Science*, Volume 8 on Network Routing, M. Ball, T. Magnanti, C. Monma and G. Nemhauser (eds.), Elsevier Science Publisher B.V., 35-139.

[16] DOWLING, W. et GALLIER, J. (1984). Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming* **3**, 267-284.

[17] DUBOIS, O., ANDRE, P., BOUFKHAD, Y. et CARLIER, J. (1993). SAT versus UNSAT. *Second DIMACS Challenge, October 11-13.*

[18] DUMAS, Y., DESROSIERS, J., GÉLINAS, É. et SOLOMON, M.M. (1995). An Optimal Algorithm for the Travelin Salesman Problem with Time Windows. *Operations Research*, **43**, 367-371.

[19] EVEN, S., ITAI, A. et SHAMIR, A. (1976). On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal of Computing* **5**, 691-703.

[20] GALLO, G. et URBANI, G. (1989). Algorithms for Testing the Satisfiability of Propositional Formulae. *Journal of Logic Programming* **7**, 45-61.

[21] GALLO, G. et PRETOLANI, D. (1992). *A New Algorithm for the Propositional Satisfiability Problem. Research Report, Dipartimento di Informatica. University of Pisa, Italy.*

[22] GAVRIL, F. (1977). Testing for Equality Between Maximum Matching and Minimum Node Covering. *Information Processing Letters* **6**, 199-202.

[23] GENDREAU, M., HERTZ, A. et LAPORTE, G. (1992). New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research*, **40**, 1086-1094.

[24] GENDREAU, M., HERTZ, A. et LAPORTE, G. (1994). A Tabu Search Heuristic for the Vehicle Routing Problem. *Management Science*, **40**, 1276-1290.

[25] GENDREAU, M., LAPORTE G. et POTVIN J.-Y. (1994). Metaheuristics for the Vehicle Routing Problem. *Publication CRT-963, Centre de recherche sur les transports, Université de Montréal.*

[26] GENDREAU, M., HERTZ, A., LAPORTE, G. et STAN, M. (1995). A Generalized Insertion Heuristics for the Traveling Salesman Problem with Time Windows. *Publication CRT-95-07, Centre de recherche sur les transports, Université de Montréal.*

[27] GLOVER, F. (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, bf 8, 156-166.

[28] GLOVER, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, **13**, 533-549.

[29] GLOVER, F. (1989). Tabu Search - Part I. *ORSA Journal on Computing* **1**, 190-206.

[30] HANSEN, P. (1976). A Cascade Algorithm for the Logical Closure of a Set of Binary Relations. *Information Processing Letters* **5**, 50-55.

[31] HANSEN, P. et JAUMARD, B. (1985). Uniquely Solvable Quadratic Equations. *Discrete Applied Mathematics* **12**, 147-154.

[32] HANSEN, P. (1986). The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

[33] HANSEN, P. et JAUMARD, B. (1990). Algorithms for the Maximum Satisfiability Problem. *Computing*, **44**, 279-303.

[34] HANSEN, P., JAUMARD, B. et MINOUX, M. (1986). A Linear Expected-time Algorithm for Deriving All Logical Conclusions Implied by a Set of Boolean Inequalities. *Mathematical Programming* **34**, 223-231.

[35] HERTZ, A. et de WERRA, D. (1987). Using Tabu Search Techniques for Graph Coloring. *Computing*, **29**, 345-351.

[36] HOOKER, J.N. et FEDJKI, C. (1990). Branch-and-Cut Solution of Inference Problems in Propositional Logic. *Annals of Mathematics and Artificial Intelligence* **1**, 123-139.

[37] HOOKER, J. N. (1993). Solving the Incremental Satisfiability Problem. *The Journal of Logic Programming* **15**, 177-186.

[38] IBARAKI, T. et KATOH, N. (1983). On-line Computation of transitive closures of graphs. *Information Processing Letters* **16**, 95-97.

[39] ITALIANO, G. F. (1986). Amortized Efficiency of a Path Retrieval Data Structure. *Theoretical Computer Science* **48**, 273-281.

[40] JAUMARD, B., MARCHIORO, P., MORGANE, A., PETRESCHI, R. et SIMEONE, B. (1990). On-line 2-satisfiability. *Annals of Mathematics and Artificial Intelligence* **1**, 155-165.

[41] JAUMARD, B., STAN, M. et DESROSIERS, J. (1996). Tabu Search and a Quadratic Relaxation for the Satisfiability Problem. *American Mathematical Society - DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **26**, 457-477.

[42] JEROSLOW, G. et WANG, J. (1990). Solving Propositional Satisfiability Problems. *Annals of Mathematics and Artificial Intelligence* **1**, 167-187.

[43] JOHNSON, E.L. et PADBERG, M.W. (1982). Degree Two Inequalities, Cliques Facets, and Biperfect Graphs. *Annals of Discrete Mathematics* **16**, 169-187.

[44] KAMATH, A.P., KARMARKAR, N.K., RAMAKRISHNAN, K.G. et RESENDE, M.G.C. (1990). Computational Experience with an Interior Point Algorithm on the Satisfiability Problem. *Annals of Operations Research* **25**, 43-58.

[45] KARMARKAR, N., RESENDE, M.G.C. et RAMAKRISHNAN, K.G. (1988). An Interior-point Algorithm for Zero-one Integer Programming. 13^{th} *International Symposium on Mathematical Programming, Mathematical Programming Society, Tokyo.*

[46] KARP, R.M. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, New York, 85-103.

[47] KNOX, J. et GLOVER, F. (1989). Comparative Testing of Traveling Salesman Heuristics Derived from Tabu Search, Genetic Algorithms and Simulated Annealing. *Center for Applied Artificial Intelligence, University of Colorado.*

[48] KOLEN, A.W.J., RINNOOY KAN, A.H.G. et TRIENEKENS, H.W.J.M. (1987). Vehicle Routing with Time Windows. *Operations Research*, **35**, 266-273.

[49] LANGEVIN, A., DESROCHERS, M., DESROSIERS, J., GÉLINAS, S. et SOUMIS, F. (1993). A Two Commodity Flow Formulation for the Traveling Salesman and the Makespan Problems with Time Windows. *Networks*, **23**, 631-640.

[50] LARABEE, T. (1990). Efficient Generation of Test Patterns Using Boolean Satisfiability. *Ph. D. Dissertation, Stanford University, California.*

[51] LE BARS, L. (1987). Etude du problème de satisfaisabilité d'une expression logique. *Mémoire d'Ingénieur de l'Institut d'Informatique d'Entreprise, Evry, France.*

[52] LOVELAND, D.W. (1978) *Automated Theorem-Proving: a Logical Basis.* North Holland, Amsterdam.

[53] NIJENHUIS, A. et WILF, S. (1975). *Combinatorial Algorithms.* Academic Press.

[54] OR, I. (1976). Traveling Salesman-type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking. *Ph. D. Dissertation. Northwestern University, Evanston, IL.*

[55] PETRESCHI, R. et SIMEONE, B. (1980). A Switching Algorithm for the Solution of Quadratic Boolean Equations. *Information Processing Letters* **11**, 193-198.

[56] POTVIN, J.-Y. et BENGIO, S. (1993). A Genetic Approach to the Vehicle Routing Problem with Time Windows. *Publication CRT-953, Centre de recherche sur les transports, Université de Montréal.*

[57] POTVIN, J.-Y., KERVAHUT, T., GARCIA, B.L. et ROUSSEAU, J.-M. (1993). A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows. *Publication CRT-855, Centre de recherche sur les transports, Université de Montréal.*

[58] PSARAFTIS, H. (1980). A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. *Transportation Science*, **14**, 130-154.

[59] PSARAFTIS, H. (1983). An Exact Algorithm for the Single Vehicle Many-to-Many Dial-a-Ride Problem with Time Windows. *Transportation Science*, **17**, 351-358.

[60] RENAUD, J., BOCTOR, F.F. et LAPORTE, G. (1994). A Fast Composite Heuristic for the Symmetric Traveling Salesman Problem. *Publication CRT-981, Centre de recherche sur les transports, Université de Montréal.*

[61] ROCHAT, Y. et TAILLARD, E.D. (1995). Tabu Search for Vehicle Routing Problem with Time Windows. *Publication CRT-95-13, Centre de recherche sur les transports, Université de Montréal.*

[62] SAVELSBERGH, M.W.P. (1985). Local Search in Routing Problems with Time Windows. *Annals of Operations Research* **4**, 285-305.

[63] SAVELSBERGH, M.W.P. (1989). The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. *Working Paper, Center for Mathematics and Computer Science, Amsterdam.*

[64] SIMEONE, B. (1985). Consistency of Quadratic Boolean Equations and the Koenig-Egervary Property for Graphs. *Annals of Discrete Mathematics* **25**, 281-290.

[65] SOLOMON, M.M. (1987). Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research* **35**, 254-265.

[66] SOLOMON, M.M., BAKER, E. et SCHAFFER, J. (1988). Vehicle Routing and Scheduling with Time Window Constraints: Efficient Implementations of Solution Improvement Procedures. *Vehicle Routing: Methods and Studies*, B.L. Golden and A.A. Assad (eds.), North-Holland, Amsterdam, 85-106.

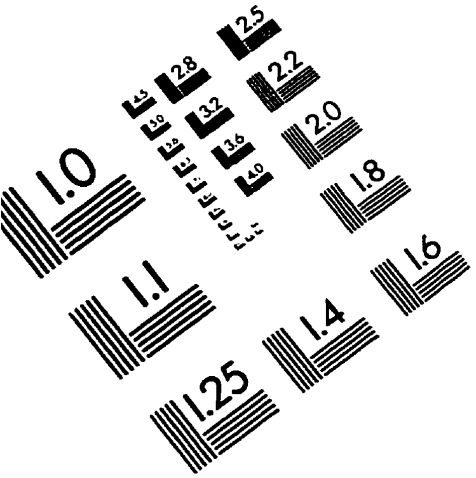
[67] TARJAN, R. E. (1972). Depth First Search and Linear Graph Algorithms. *SIAM Journal on Computing* **1**, 146-160.

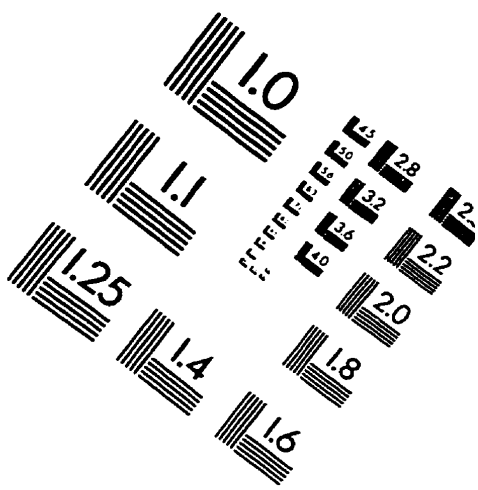
[68] THANGIAH, S. R., NYGARD,K. E. et JUELL, P. L. (1991). GIDEON: A Genetic Algorithm System for Vehicle Routing with Time Windows. *Proceedings of the Fifth International Conference on Genetic Algorithms, Urbana-Champaign, IL, 506-515.*

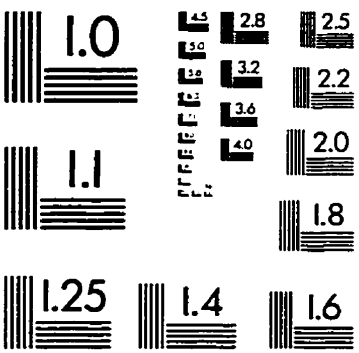
[69] THANGIAH, S.R. (1993). Vehicle Routing with Time Windows Using Genetic Algorithms. *Technical Report SRU-SpSc-TR-93-23, Slippery Rock University, Slippery Rock, PA.*

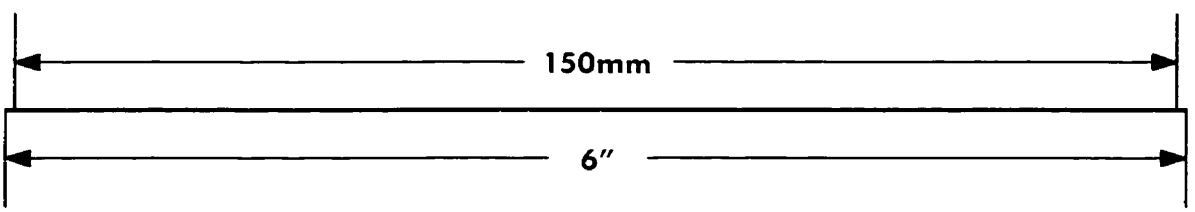
[70] THANGIAH, S. R. et GUBBI, A. V. (1993). Effect of Genetic Sectoring on Vehicle Routing Problems with Time Windows. *Proceedings on Managing Intelligent System Projects, Washington, DC, 146-153.*

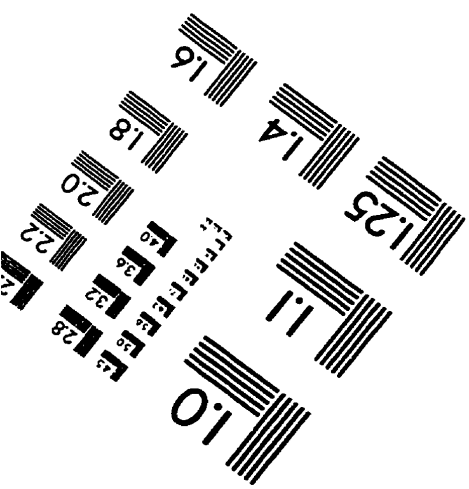
TEST TARGET (QA-3)

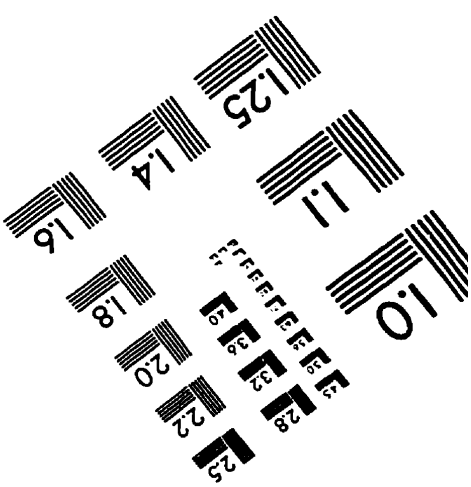












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989