**Université de Montréal**

# Problèmes de tournées multicritères dans des graphes

par

Jean-François Bérubé

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures

en vue de l'obtention du grade de

PhilosophiæDoctor (Ph.D.)

en informatique

Avril 2007

© Jean-François Bérubé , 2007

Faculté des études supérieures et postdoctorales
Grade conféré à compter du
10 JAN. 2008
Université de Montréal

QA
76
U54
2008
V.002

Université de Montréal

Direction des bibliothèques

**AVIS**

L'auteur a autorisé l'Université de Montréal à reproduire et diffuser, en totalité ou en partie, par quelque moyen que ce soit et sur quelque support que ce soit, et exclusivement à des fins non lucratives d'enseignement et de recherche, des copies de ce mémoire ou de cette thèse.

L'auteur et les coauteurs le cas échéant conservent la propriété du droit d'auteur et des droits moraux qui protègent ce document. Ni la thèse ou le mémoire, ni des extraits substantiels de ce document, ne doivent être imprimés ou autrement reproduits sans l'autorisation de l'auteur.

Afin de se conformer à la Loi canadienne sur la protection des renseignements personnels, quelques formulaires secondaires, coordonnées ou signatures intégrées au texte ont pu être enlevés de ce document. Bien que cela ait pu affecter la pagination, il n'y a aucun contenu manquant.

**NOTICE**

The author of this thesis or dissertation has granted a nonexclusive license allowing Université de Montréal to reproduce and publish the document, in part or in whole, and in any format, solely for noncommercial educational and research purposes.

The author and co-authors if applicable retain copyright ownership and moral rights in this document. Neither the whole thesis or dissertation, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms, contact information or signatures may have been removed from the document. While this may affect the document page count, it does not represent any loss of content from the document.

# Université de Montréal

Faculté des études supérieures

Cette thèse intitulée:

**Problèmes de tournées multicritères dans des graphes**

présentée par:

Jean-François Bérubé

a été évaluée par un jury composé des personnes suivantes:

Jacques Ferland

(président-rapporteur)

Jean-Yves Potvin

(directeur de recherche)

Michel Gendreau

(co-directeur)

Jean Vaucher

(co-directeur)

Gilbert Laporte

(membre du jury)

Frédéric Semet

(examinateur externe)

Thèse acceptée le:

13 novembre 2007

# Résumé

Cette thèse s'intéresse à divers problèmes de tournées dans des graphes où plusieurs objectifs doivent être optimisés. Contrairement à l'optimisation monocritère classique où il existe habituellement une solution de valeur optimale, les problèmes d'optimisation multicritère obligent le preneur de décision à faire un arbitrage entre des objectifs souvent contradictoires. Cet arbitrage se fait par l'expression de préférences, un concept central de la théorie de l'utilité adoptée par les économistes depuis plusieurs décennies. La manière dont ces préférences sont articulées a une influence directe sur la façon dont un problème peut être abordé. Lorsque les préférences sont connues avant le processus d'optimisation, il est possible de reformuler le problème multicritère original en un problème monocritère, par exemple, en agrégeant tous les objectifs en une fonction de coût généralisée. Ceci est évidemment impossible lorsque les préférences ne sont connues qu'après le processus de recherche. Le concept de solution optimale est alors remplacé par celui d'efficacité, qui permet de définir un ensemble de solutions également désirables, quelles que soient les préférences du preneur de décision. Ces solutions sont représentées dans l'espace des objectifs par la frontière de Pareto.

Nous abordons deux problèmes de tournées multicritères dans des graphes à l'aide d'algorithmes exacts adaptés à des situations où les préférences du preneur de décision sont connues avant ou après le processus d'optimisation. Le premier de ces problèmes est une application à l'industrie du tourisme visant à déterminer une planification de voyage optimale dans un contexte où les préférences sont connues *a priori*. Étant donné un itinéraire prédéterminé, le problème de planification de voyage consiste à proposer la meilleure combinaison de billets d'avion et de chambres d'hôtels, en fonction des préférences du voyageur. Nous modélisons ce problème à l'aide d'un graphe spatio-temporel représentant les transports par avion, les hô-

tels et les séjours dans chacune des villes de l'itinéraire. Notre modèle, qui inclut aussi des contraintes temporelles, est résolu en calculant des plus courts chemins dans ce graphe spatio-temporel. Malgré la taille importante de ce dernier, notre algorithme exact permet de résoudre des instances de taille réaliste dans un délai raisonnable.

Le second problème est celui du voyageur de commerce avec profits (PVCP) qui consiste à déterminer une tournée de coût minimal et de profit maximal dans un graphe complet où des coûts sont associés aux arêtes et des profits sont associés aux sommets. Nous considérons d'abord une variante du problème dans laquelle une quantité minimale de profit doit être amassée par une tournée de coût minimum. Comme pour le problème de planification de voyage, cette variante monocritère du PVCP repose sur l'hypothèse que les préférences sont connues avant le processus d'optimisation. Nous avons développé un algorithme de séparation et coupes basé sur des inégalités valides issues d'une autre variante du PVCP ainsi que de certains résultats polyhédraux théoriques existants. Des résultats numériques sur des instances comptant plus de 500 sommets sont présentés.

Finalement, nous abordons le PVCP dans un contexte où les préférences sont connues *a posteriori*. Pour ce faire, nous avons développé une méthode exacte destinée à résoudre des problèmes d'optimisation combinatoires bi-critères où la valeur de chacun des objectifs est entière. Cette approche, basée sur la méthode $\epsilon$-*constraint*, résout une série de sous-problèmes monocritères obtenus en formulant les objectifs du problème original sous forme de contraintes, à l'exception d'un seul. Nous montrons que cette méthode permet d'énumérer efficacement tous les points de la frontière de Pareto. De plus, nous présentons des heuristiques permettant d'accélérer significativement la résolution des sous-problèmes grâce aux informations recueillies lors de la résolution des sous-problèmes précédents. Nous rapportons les premiers résultats exacts pour le PVCP sur des instances générées à partir d'instances classiques du problème du voyageur de commerce. Nous expliquons aussi comment une modification de notre algorithme exact permet d'obtenir des solutions approchées d'excellente qualité, et ce très rapidement.

**Mots clés :** Optimisation multicritères, Chemins plus courts dépendant du temps, Problème du voyageur de commerce avec profits, Séparation et coupes, Frontière de Pareto, Problème $\epsilon$-contraint.

# Abstract

This thesis studies various multiple objective routing problems. As opposed to classical monocriterion optimization, where an optimal solution exists, multicriteria optimization problems usually involve a trade-off among conflicting objectives. The decision maker must therefore formulate his preferred objectives, a concept introduced several decades ago in utility theory developed in economics. The way those preferences are revealed directly impacts the way a problem is tackled. When the preferences are known before the optimization process, one may reformulate the original multicriteria problem into a monocriterion one. For example, the objectives might be aggregated into a generalized cost function. Clearly, this is impossible if the preferences are known only at the end of the optimization process. In this case, the optimal solution concept is replaced by an efficiency concept that allows one to identify a set of equally desirable solutions, whatever the preferences of the decision maker are. These solutions are represented by the Pareto front in the objective space.

Two multicriteria routing problems are addressed in this thesis and we consider situations where the decision maker's preferences are known before or after the optimization process. The first problem comes from the travel industry where an optimal travel plan must be found according to the traveler's preferences. Given a predetermined itinerary, the problem consists in finding the best travel plan, involving planes and hotels, based on the traveler's preferences. Our time-dependent framework models plane flights, hotels, stays in each city as well as global time constraints. The travel planning problem is solved by computing time-dependent shortest paths through a fixed sequence of nodes. Given the large size of time-dependent networks, an exact decomposition algorithm is devised to solve instances of realistic size in reasonable computation times.

Our second problem is the Traveling Salesman Problem with Profits (TSPP), a variant of the Traveling Salesman Problem in which a profit or prize value is associated with each vertex. The goal here is to visit a subset of vertices while addressing two conflicting objectives : maximize the collected prize and minimize the travel costs. We first consider a variant of the TSPP in which a minimum amount of profit must be collected by a minimum cost elementary cycle. As for our travel planning problem, this monocriterion variant of the TSPP assumes that the decision maker's preferences are known *a priori*. We have developed a branch-and-cut algorithm based on valid inequalities from cuts designed for the Orienteering Problem and from existing polyhedral results. Computational results on instances with more than 500 vertices are reported.

We also address the TSPP in a context where the decision maker's preferences remain unknown until the end of the optimization process. We describe an exact $\epsilon$-constraint method for bi-objective combinatorial optimization problems with integer objective values. This method tackles multicriteria optimization problems by solving a series of single objective subproblems, where all but one objectives are transformed into constraints. We show that the Pareto front of bi-objective problems can be efficiently generated with the $\epsilon$-constraint method. Furthermore, we describe heuristics based on information gathered from previous subproblems that significantly speed up the method. We report the first exact results for the TSPP on instances derived from classical vehicle routing and traveling salesman problem instances. Results on approximations of the Pareto front obtained from a variant of our exact algorithm are also reported.

**Key words :** Multicriteria optimization, Time-dependant shortest paths, Traveling Salesman Problem with Profits, Branch-and-cut, Pareto Front, $\epsilon$-constraint problem.

# Table des matières

# Liste des figures

# Liste des tableaux

# Remerciements

Cette thèse est le fruit de trois ans et demi de lecture, de réflexion, de programmation et de rédaction. S'il est mûr aujourd'hui, c'est en partie grâce aux deux professeurs qui m'ont accompagné dans mon travail. Merci, Monsieur Jean-Yves Potvin, pour votre disponibilité et votre souci du travail bien fait. Vous m'avez aidé à faire de mes travaux d'étudiant de véritables textes scientifiques dignes des meilleurs journaux en recherche opérationnelle. Merci, Monsieur Michel Gendreau, pour votre soutien constant. Votre impressionnante culture en recherche opérationnelle me fut très précieuse.

Il y a trois ans et demi, le fruit n'était qu'un désir de me lancer dans des études doctorales. Merci, Monsieur Jean Vaucher, d'avoir semé en moi le goût de la recherche. Merci aussi de m'avoir aidé à faire le pont entre la maîtrise et le doctorat en participant à la rédaction de mon premier article.

Merci au personnel du Centre de Recherche sur les Transports qui a su créer et entretenir un environnement propice à l'aboutissement de travaux comme celui-ci. Je remercie plus particulièrement Messieurs Serge Bisaillon, François Guertin, Daniel Charbonneau et Luc Rocheleau pour leur soutien technique.

Merci à mes amis Benoit Crevier, Nadia Lahrichi et Walter Rei pour ces longues et distrayantes discussions qui aident quelquefois à mettre de l'ordre dans nos idées. Je remercie aussi François Duranleau pour ses judicieux conseils en programmation informatique.

Merci finalement à Lise Raymond qui a gentiment accepté de faire une révision grammaticale et orthographique des sections françaises de ma thèse.

# Introduction

En raison de leurs nombreuses applications et parce qu'ils constituent pour la plupart d'importants défis pour les chercheurs, les problèmes de tournées dans des graphes suscitent toujours un intérêt indiscutable dans la communauté scientifique. Bien que traditionnellement monocritères, les problèmes de tournées sont de plus en plus abordés sous un angle multicritère, souvent plus proche des applications qu'on leur prête. Plutôt que d'avoir un objectif unique, le problème multicritère impose la prise en compte de ses nombreux aspects, souvent contradictoires, mais néanmoins incontournables dans la réalité.

Bien entendu, l'optimisation multicritère n'est pas limitée aux problèmes de tournées. Elle apparaît dans toute situation de prise de décision dont l'objectif ne peut être traduit correctement par un critère unique non ambigu. Les premiers à s'intéresser à des problèmes d'optimisation impliquant un nombre quelconque d'objectifs furent les économistes qui élaborèrent les théories de l'utilité et du bien-être. Stadler (1979) [94] présente un historique de l'optimisation multicritère retraçant les publications des fondateurs de la théorie de l'utilité, tels que Jevons, Menger et Walras. Depuis, outre les économistes, de nombreux spécialistes en recherche opérationnelle se sont intéressés à l'optimisation multicritère, qui demeure néanmoins assez méconnue de la majorité de leurs confrères. Pour cette raison, une partie importante du premier chapitre de cette thèse sera consacrée à initier le lecteur néophyte aux rudiments de l'optimisation multicritère.

Si on considère un problème de minimisation, la formulation générale d'un problème d'optimisation multicritère (POM) s'écrit :

$$\text{“min”}\ f(\vec{x}) = \vec{z} \qquad \text{sujet à :}\ \vec{x} \in \mathcal{X} \tag{1}$$

avec le vecteur des variables de décision $\vec{x} \in \mathbb{R}^n$, l'ensemble des solutions réalisables $\mathcal{X}$ et le vecteur des valeurs des objectifs $\vec{z} \in \mathbb{R}^k$. L'ensemble des valeurs que peuvent prendre les variables de décision définit l'*espace de décision* (noté $\Omega = \{\vec{x} \in \mathbb{R}^n\}$), tandis que l'*espace des objectifs* (noté $\Lambda = \{\vec{z} \in \mathbb{R}^k\}$) correspond aux vecteurs d'évaluation associés aux éléments de l'espace de décision. L'ensemble des solutions réalisables $\mathcal{X}$ constitue un sous-ensemble de $\Omega$ alors que l'ensemble des vecteurs d'évaluation réalisables $\mathcal{Z}$ est un sous-ensemble de $\Lambda$. Les fonctions d'évaluation $f_i(\vec{x}) : \Omega \rightarrow \mathbb{R}$ permettent d'obtenir les composantes du vecteur $\vec{z}$ correspondant à un vecteur de variables de décision $\vec{x}$.

Puisqu'il est souvent impossible de trouver une solution optimisant simultanément tous les objectifs, on cherchera habituellement un compromis raisonnable plutôt qu'une solution optimale. C'est la raison pour laquelle l'opérateur de minimisation est placé entre guillemets dans l'équation (1). Cette impossibilité à identifier une solution clairement optimale force l'introduction de concepts étrangers à l'optimisation monocritère, dont le plus important sert à établir une hiérarchie dans l'ensemble des solutions réalisables. Ainsi, les économistes font l'hypothèse qu'une relation dite de *préférence* permet de définir un pré-ordre complet des éléments de $\mathcal{X}$. Cette relation est donc, par définition, une relation complète, réflexive et transitive*. Elle permet au preneur de décision d'exprimer un arbitrage entre les critères afin qu'une solution puisse éventuellement être désignée comme étant la plus *désirable*, ou comme étant au moins aussi désirable que les plus désirables.

Bien que l'existence de préférences soit essentielle, encore faut-il que ces dernières soient connues pour qu'un quelconque processus d'optimisation puisse en tenir compte. Hwang et Masud (1979) [54] distinguent quatre *contextes de prise de décision* définis en fonction du moment où les préférences sont révélées et du type d'information véhiculée par ces préférences. Leurs modèles d'articulation des préférences supposent que ces dernières ne sont jamais révélées, qu'elles sont connues *a priori*, qu'elles se révèlent progressivement ou qu'elles ne sont connues qu'*a posteriori*. Ces contextes de prise de décision sont fondamentaux en raison de leur influence directe sur la modélisation et sur les techniques d'optimisation.

*Des hypothèses supplémentaires sur les propriétés de la relation de préférence sont nécessaires dans certaines situations. Elles feront l'objet d'une discussion au Chapitre 1 lorsque le concept d'utilité sera introduit.

Le premier modèle est trivial, aucune information sur les préférences du preneur de décision n'est révélée. Un critère global doit alors être développé et une solution optimale est déterminée en fonction de ce critère. Évidemment, cela nécessite de faire plusieurs suppositions relatives aux préférences du preneur de décision, ce qui constitue un désavantage majeur de ce genre de modèle.

Dans le modèle *a priori*, les préférences sont connues avant le début de la résolution du problème. Autrement dit, elles font partie des données du problème. L'information qu'elles véhiculent peut être de type cardinal, c'est-à-dire que les niveaux de préférence des différents objectifs sont exprimés explicitement. Dans ce cas, il est possible d'élaborer un critère unique et d'obtenir ainsi un problème monocritère. Les préférences peuvent aussi être de type ordinal, auquel cas les objectifs sont simplement ordonnés selon leur importance relative ce qui permet d'appliquer des techniques d'optimisation lexicographique. Finalement, il arrive que l'information disponible soit mixte, c'est-à-dire à la fois cardinale et ordinale. Mais quel que soit le type d'information véhiculée par les préférences, ces dernières, lorsqu'exprimées *a priori*, permettent de transformer un problème multicritère en un problème monocritère.

Le modèle *interactif* (ou progressif) prévoit l'introduction des préférences au cours du processus de résolution. Le preneur de décision est en fait assisté d'un outil d'optimisation avec lequel il interagit afin de déterminer une solution. C'est lors de cette interaction que les préférences sont progressivement révélées. L'information qu'elles véhiculent consiste en un compromis entre différentes solutions proposées au preneur de décision. Comme cette thèse n'abordera pas le modèle interactif, nous référons le lecteur au livre d'Eschenauer, Koski et Osyczka (1990) [35] pour une présentation détaillée des différentes méthodes associées au modèle interactif.

Le modèle *a posteriori* suppose que les préférences sont inconnues au moment de la résolution du problème. Le preneur de décision fait son choix parmi un ensemble de solutions équivalentes (également désirables) identifiées par un processus d'optimisation. Bien qu'aucune supposition sur les préférences ne soit nécessaire, ce modèle a un désavantage important : l'ensemble des solutions efficaces est souvent très grand, ce qui rend particulièrement difficile la tâche de déterminer toutes ces solutions et éventuellement d'en choisir une.

Cette thèse aborde des problèmes de tournées multicritère dans des graphes où les préférences sont connues *a priori* ou *a posteriori*. Deux problèmes seront traités : le problème de planification de voyage et le problème du voyageur de commerce avec profits. Le premier sera étudié dans un contexte où les préférences sont connues *a priori* et le second sera abordé dans les contextes de préférences révélées *a priori* et *a posteriori*. Dans les deux cas, nous nous intéresserons exclusivement à des méthodes exactes.

Le problème de planification de voyage consiste à déterminer le plan de voyage optimal, comprenant billets d'avion et réservations d'hôtels, en fonction des préférences du voyageur, révélées *a priori*. Nous assumons la connaissance d'un critère global permettant d'agréger les préférences en un coût généralisé et donc d'aborder le problème comme s'il était monocritère. Notre modèle est basé sur un graphe orienté des villes et des liaisons aériennes entre ces dernières. Les délais et les coûts de déplacement, tous deux associés aux arcs, sont dépendants du moment auquel débute le déplacement. Le problème consiste à trouver le plan de voyage de coût minimum permettant au voyageur de visiter une séquence ordonnée et prédéterminée de villes, chaque séjour devant avoir une durée minimale précise. Notre approche de résolution est basée sur une décomposition du problème permettant d'utiliser efficacement un algorithme de plus courts chemins dans le graphe spatio-temporel issu du graphe original.

Le problème du voyageur de commerce avec profits (PVCP) est une variante relativement connue du problème du voyageur de commerce classique. Dans cette variante, la contrainte obligeant la visite de tous les sommets est levée et une valeur est associée à chacun des sommets. En plus de minimiser les coûts de déplacement, la somme des valeurs amassées (souvent appelée *profit*) doit être maximisée. Le voyageur de commerce avec profits est donc un problème d'optimisation combinatoire bi-critère. Selon le contexte de prise de décision, il peut être transformé en un problème monocritère, soit en agrégeant coûts et profits en un critère unique ou en imposant une contrainte sur un des deux objectifs (une borne supérieure sur les coûts ou une borne inférieure sur les profits). Au chapitre 3, nous nous intéressons à une variante monocritère du problème où les préférences sont connues avant le processus d'optimisation : le *Prize Collecting Traveling Salesman Problem* (qui n'a pas de dénomination clairement acceptée en français). Ensuite, au chapitre 4, nous étudions le PVCP dans un contexte où les préférences ne sont connues qu'*a posteriori* et où il est donc impossible de se rabattre sur une simplifica-

tion monocritère. Dans ce dernier cas, le problème servira d'illustration à une méthode exacte applicable à un large éventail de problèmes d'optimisation combinatoires bi-critères.

Cette thèse est divisée en quatre chapitres. Le premier est consacré à une revue de la littérature sur les méthodes exactes en optimisation multicritère pour les deux contextes de prise de décision que nous avons étudiés (*a priori* et *a posteriori*). Nous y mettons aussi en contexte les deux problèmes de tournées qui seront abordés dans les chapitres suivants. Le second chapitre présente nos travaux entourant le problème de planification de voyage, le chapitre 3 décrit nos travaux sur le *Prize Collecting Traveling Salesman Problem* et le chapitre 4 discute d'une approche générale pour les problèmes d'optimisation combinatoire bi-critère, méthode qui est ensuite appliquée au PVCP.

Notons finalement que cette thèse reproduit intégralement trois articles publiés ou soumis à des revues scientifiques internationales avec comité de lecture. Puisqu'ils apparaissent dans leur langue originale, les chapitres correspondants (2, 3 et 4) sont en anglais. Aussi avons-nous choisi de faire précéder chacun de ces chapitres d'un résumé en français.

# Chapitre 1

# Méthodes exactes en optimisation multicritère

Ce chapitre présente les principales techniques de résolution exacte développées pour les problèmes d'optimisation multicritères (POM). L'objectif est de fournir au lecteur un portrait général de l'état actuel de cette littérature afin de faciliter la compréhension des chapitres suivants, tous dédiés à des problèmes de tournées multicritères résolus par des méthodes exactes. La présentation qui suit se veut donc sommaire, d'autant plus qu'il existe de nombreux ouvrages présentant des revues détaillées de la littérature et des principales techniques de résolution. Parmi ceux-ci, mentionnons les ouvrages généraux de Chankong et Haimes (1983) [20], de Miettinen (1999) [76], de Sawaragi, Nakayama et Tanino (1985) [91], de Steuer (1989) [95] et de Yu (1985) [101]. Plusieurs revues de la littérature traitant d'optimisation multicritère ont aussi été publiées, parmi lesquelles on retrouve celles de Collette et Siarry (2004) [23], de Ehrgott (2000) [31], de Ehrgott et Gandibleux (2002) [32] et de Steuer, Gardiner et Gray (1996) [96]. Mentionnons finalement que Figueria, Salvatore et Ehrgott (2005) [37] ont édité un ouvrage consacré à une revue de l'état de l'art en prise de décision multicritère.

Ce chapitre débute avec la présentation de méthodes exactes destinées à résoudre des problèmes où les préférences sont connues *a priori* ou *a posteriori*. Bien entendu, classifier les approches de résolution en fonction du moment où les préférences sont révélées est

quelque peu simpliste. En fait, toutes les tentatives de classification que nous avons répertoriées ([23, 31, 76, 32]) sont nettement plus élaborées ; mais elles sont loin de se compléter, au contraire. Notre intention n'est donc pas de contribuer aux efforts de classification, mais simplement d'introduire les méthodes exactes les plus utilisées pour résoudre des POM. Nous présentons ensuite deux exemples de POM : le problème de planification de voyages et le problème du voyageur de commerce avec profits. Le premier fera ensuite l'objet du chapitre 2 alors que le second sera traité sous des angles différents dans les chapitres 3 et 4.

## 1.1 POM avec préférences révélées *a priori*

Des quatre contextes de prise de décision présentés en introduction, celui où les préférences sont connues avant le processus d'optimisation se rapproche des situations étudiées par les économistes du dix-neuvième siècle qui ont construit la théorie de l'utilité. En ajoutant des hypothèses supplémentaires sur les propriétés de la relation de préférence, ils ont développé le concept d'*utilité*, un critère global résumant l'ensemble des préférences d'un individu. Ce concept d'utilité peut servir à construire une fonction, la *fonction d'utilité*, agrégeant l'ensemble des critères sur lesquels le preneur de décision exprime ses préférences. Le problème d'optimisation devient alors un problème monocritère de maximisation de l'utilité. Une telle transformation du problème original est possible lorsqu'une information cardinale sur les préférences est disponible, mais il arrive parfois que l'information véhiculée par les préférences soit purement ordinale. Une approche lexicographique est alors envisageable. Le preneur de décision peut aussi révéler ses préférences en associant une valeur cible à chacun des critères, tout en ordonnant ceux-ci selon un ordre de priorité (information mixte). Le problème peut alors être abordé de plusieurs façons différentes, comme nous le verrons plus loin. Cette section fait d'abord une brève introduction à la théorie de l'utilité avant de présenter les principales approches de résolution applicables lorsque les préférences du preneur de décision sont connues avant le processus d'optimisation.

### 1.1.1 Introduction à la théorie de l'utilité

La théorie de l'utilité repose essentiellement sur la relation de préférence permettant de comparer deux solutions réalisables (nous écrirons $\vec{x}_a \succeq \vec{x}_b$ pour signifier que la solution $\vec{x}_a$ est préférée à la solution $\vec{x}_b$). Définies en introduction comme une relation de pré-ordre complet, les préférences satisfont toujours les propriétés suivantes :

**Définition 1.** *La relation de préférence est une* ***relation complète*** *si et seulement si* $\forall \vec{x}_a, \vec{x}_b \in \mathcal{X}$, *soit* $\vec{x}_a \succeq \vec{x}_b$ *ou* $\vec{x}_b \succeq \vec{x}_a$.

**Définition 2.** *La relation de préférence est une* ***relation réflexive*** *si et seulement si* $\forall \vec{x} \in \mathcal{X}$, $\vec{x} \succeq \vec{x}$.

**Définition 3.** *La relation de préférence est une* ***relation transitive*** *si et seulement si* $\forall \vec{x}_a, \vec{x}_b, \vec{x}_c \in \mathcal{X}$, *si* $\vec{x}_a \succeq \vec{x}_b$ *et* $\vec{x}_b \succeq \vec{x}_c$, *alors* $\vec{x}_a \succeq \vec{x}_c$.

Afin de pouvoir définir une fonction d'utilité continue $U : \mathbb{R}^k \rightarrow \mathbb{R}$ (rappelons que l'équation (1), dans l'introduction, a défini un POM comme un problème à $k$ objectifs), il faut faire l'hypothèse que les préférences sont continues et fortement monotones. Mas-Colell, Whinston et Green (1995) [75] définissent formellement ces propriétés et font, par ailleurs, une présentation détaillée de la théorie de l'utilité. Les fonctions d'utilité permettent de définir le concept d'optimalité dans un contexte multicritère. Si $\vec{z}^* \in \mathcal{Z}$ maximise $U$, alors $\vec{z}^*$ est un point optimal dans l'espace des objectifs et tout $\vec{x} \in \mathcal{X}$ tel que $(f_1(\vec{x}), ..., f_k(\vec{x})) = \vec{z}^*$ est un point optimal dans l'espace des variables de décision.

Mais il ne faudrait pas croire que la meilleure façon de résoudre un POM est nécessairement de maximiser la fonction d'utilité. D'abord, parce qu'il est très difficile, voire impossible, de construire une fonction d'utilité représentant parfaitement les préférences. Ensuite, parce que les fonctions d'utilité sont presque toujours non linéaires, ce qui en complexifie la maximisation.

### 1.1.2 Une méthode exacte en situation d'information cardinale

Nous considérons d'abord une méthode applicable lorsqu'une information de nature cardinale sur les préférences est disponible : la *somme pondérée des fonctions objectifs*. Il s'agit d'une des premières techniques appliquées aux POM et aussi sans doute la plus répandue. Le principe de la somme pondérée est d'optimiser une combinaison convexe des fonctions objectifs en assignant une valeur scalaire aux solutions réalisables. Le problème monocritère ainsi obtenu est ensuite résolu par des méthodes d'optimisation classiques. Soit $\lambda$ un vecteur de poids tel que $\lambda \in \mathbb{R}^k$, $\lambda_i > 0$ et $\sum_{i=1}^{k} \lambda_i = 1$. La technique de la somme pondérée consiste à optimiser :

$$\sum_{j=1}^{k} \lambda_j z_j, \qquad \text{avec } \vec{z} \in \mathcal{Z} \tag{1.1}$$

Cette approche a un avantage important par rapport à plusieurs autres méthodes : la structure du problème multicritère original est maintenue, seul l'objectif est modifié. Ainsi, pour tous les POM issus d'un problème monocritère classique, les algorithmes développés pour le problème original sont applicables. Cela explique sans doute la grande popularité de cette approche. Ehrgott et Gandibleux (2002) [32] citent d'ailleurs plusieurs applications de la somme pondérée pour des POM de chemins plus courts, de transport, d'affectation, de flot dans les réseaux, d'arbre de recouvrement minimal, de sac-à-dos et de localisation.

Le principal inconvénient de cette approche survient lorsqu'on l'utilise dans le but d'énumérer l'ensemble des solutions également désirables ; nous y reviendrons dans la section suivante. Lorsqu'on cherche simplement à optimiser une fonction d'utilité, la méthode elle-même peut difficilement être critiquée directement. Sa faiblesse se situe au niveau du choix de la fonction d'utilité et de sa capacité à représenter adéquatement les préférences du preneur de décision. Des tentatives ont d'ailleurs été faites afin d'améliorer le réalisme de la fonction objectif, notamment par Keeney et Raiffa (1976) [63] qui proposent une fonction d'utilité permettant d'intégrer certaines non linéarités à l'objectif. L'utilisation de ce type de fonction demeure toutefois marginale par rapport à celle de la somme pondérée.

### 1.1.3 Une méthode exacte en situation d'information ordinale

L'approche *lexicographique* s'avère un choix naturel dans une situation où l'information sur les préférences permet seulement d'ordonner les critères du plus important au moins important. Avec cette méthode, les objectifs sont considérés à tour de rôle et un problème monocritère, auquel des contraintes sont ajoutées graduellement, est optimisé pour chaque objectif. On optimise par rapport à un premier objectif $(i)$ et le résultat de cette optimisation, $((\vec{z})_i^*)$ est introduit sous la forme d'une contrainte d'égalité $((\vec{z})_i = (\vec{z})_i^*)$ que la solution du sous-problème suivant devra satisfaire. On procède ainsi jusqu'à ce que les $k$ critères aient été pris en compte.

De la même manière que pour la paramétrisation d'une fonction d'utilité, l'ordonnancement des objectifs s'avère souvent être une décision arbitraire, et donc contestable. Encore une fois, cette critique ne s'attaque pas à la méthode elle-même mais plutôt aux décisions subjectives sous-jacentes.

### 1.1.4 Méthodes exactes en situation d'information mixte

Lorsque l'information véhiculée par les préférences est à la fois cardinale et ordinale, trois approches sont habituellement utilisées : la programmation par buts, la méthode $\epsilon$-*constraint* et les méthodes basées sur la distance avec un objectif de référence.

#### Méthodes basées sur la distance avec un objectif de référence

Aussi connue sous le nom de méthode du compromis [32, 101], cette technique consiste à minimiser la distance avec un point de référence représentant les préférences du preneur de décision. Deux mesures de distances sont utilisées : la distance absolue et la distance relative. Soit $\vec{z}_u$ le vecteur correspondant au point de référence. La distance absolue s'écrit alors :

$$L_r(\vec{z}) = \left[ \sum_{i=1}^{k} |(\vec{z}_u)_i - (\vec{z})_i|^r \right]^{\frac{1}{r}} \tag{1.2}$$

avec $1 \leq r < \infty$. La méthode plus répandue utilisant la distance absolue est la méthode dite de *min-max* ou de *Tchebychev*. On pose alors $r = \infty$ et la fonction de distance s'écrit :

$$L_\infty(\vec{z}) = \max_{i \in \{1,\ldots,k\}} ((\vec{z}_u)_i - (\vec{z})_i). \tag{1.3}$$

Quant à la mesure de distance relative, elle est formulée :

$$L_r^{rel}(\vec{z}) = \left[ \sum_{i=1}^{k} \left| \frac{(\vec{z}_u)_i - (\vec{z})_i}{(\vec{z}_u)_i} \right|^r \right]^{\frac{1}{r}}. \tag{1.4}$$

Le fait qu'elles reposent sur le choix d'un point de référence constitue la principale faiblesse de ces méthodes. Bien qu'il puisse être plus facile pour le preneur de décision d'exprimer ses préférences sous la forme d'un objectif idéal à atteindre que sous la forme d'une fonction d'utilité, il n'en demeure pas moins que le choix de ce point de référence aura un impact majeur sur la qualité des solutions.

### Méthode $\epsilon$-*constraint*

La méthode $\epsilon$-*constraint* [20, 76] (aussi connue sous le nom de méthode de compromis [23]) repose sur la transformation de $k - 1$ des $k$ objectifs en contraintes afin d'obtenir un problème monocritère. Un objectif prioritaire doit donc être identifié ; ce sera l'unique objectif du problème transformé. Un vecteur $\epsilon$ contenant les bornes supérieures sur chacun des autres objectifs est ensuite utilisé pour générer les nouvelles contraintes d'inégalité. Le problème transformé (appelé $\epsilon$-*constraint problem*) s'écrit :

$$\min_{\vec{x} \in \mathcal{X}} f_i(\vec{x}) \text{ sujet à } f_j(\vec{x}) \leq \epsilon_j \quad j \neq i \tag{1.5}$$

Le principal inconvénient de cette méthode vient de la complexité du $\epsilon$-*constraint problem* dont la résolution requiert souvent beaucoup de temps de calcul. Par exemple, dans le cas des POM combinatoires, le $\epsilon$-*constraint problem* est souvent $NP$-complet. Cette complexité a évidemment tendance à croître avec le nombre d'objectifs.

**Programmation par buts**

À l'instar des méthodes de distances, la programmation par but repose sur l'atteinte de valeurs cibles identifiées par un vecteur $\vec{z}_u$. On associe à chaque objectif $i$ des variables d'écart $d_i^+$ et $d_i^-$ dont l'une et/ou l'autre est ensuite minimisée sous les contraintes suivantes :

$$\vec{x} \in \mathcal{X} \tag{1.6}$$

$$(\vec{z})_1 = (\vec{z}_u)_1 + d_1^+ - d_1^- \tag{1.7}$$

$$\vdots$$

$$(\vec{z})_k = (\vec{z}_u)_k + d_k^+ - d_k^-$$

$$d_i^+ \geq 0, d_i^- \geq 0 \text{ et } d_i^+ \cdot d_i^- = 0 \quad \forall i \in \{1, ..., k\} \tag{1.8}$$

Évidemment, si le preneur de décision préfère que la valeur du $i$ième objectif soit toujours au-dessus de la valeur cible $(\vec{z}_u)_i$, on choisira d'inclure uniquement la variable $d_i^+$ dans l'objectif alors que s'il préfère que la valeur de cet objectif demeure en-dessous de la valeur cible, on minimisera plutôt $d_i^-$. Lorsque la valeur cible doit être atteinte exactement, on minimisera la somme des deux variables d'écart correspondantes : $d_i^+ + d_i^-$. Le vecteur de variables à minimiser peut donc s'exprimer de différentes manières dépendemment des préférences du preneur de décision. La programmation par but est parfois considérée comme un domaine d'étude distinct de celui de l'optimisation multicritère. Aussi y a-t-il des ouvrages complets qui lui sont dédiés, tels que ceux de Ignizio (1976) [55] et de Lee (1972) [71].

## 1.2 POM avec préférences révélées *a posteriori*

Lorsque les préférences ne sont pas connues au moment de l'optimisation, il n'existe pas de valeur optimale unique. L'objectif est alors de déterminer un ensemble de solutions également désirables à partir duquel le preneur de décision pourra ensuite faire son choix. Avant de discuter des méthodes de résolution permettant d'identifier ces solutions équivalentes, certains concepts issus de la littérature économique doivent encore être définis.

### 1.2.1 L'efficacité au sens de Pareto

La notion d'*efficacité de Pareto** est l'équivalent multicritère du concept de solution optimale en optimisation monocritère. De la même manière qu'une solution optimale n'est pas nécessairement unique, il existe potentiellement plusieurs solutions efficaces. Mais contrairement aux solutions optimales qui partagent toutes la même valeur, les solutions efficaces peuvent être associées à des vecteurs d'évaluation différents. Il est donc nécessaire de définir une relation entre les vecteurs d'évaluation induisant un ordonnancement partiel de l'espace des objectifs. En supposant que le preneur de décision a des préférences complètes, transitives, réflexives et monotones croissantes, il est possible de définir une relation de dominance entre les vecteurs d'évaluation. C'est cette relation entre les éléments de l'espace des objectifs qui permet de définir une relation d'efficacité (au sens de Pareto) entre les éléments de l'espace des variables de décision.

**Définition 4 (La relation de dominance).** *Soient deux vecteurs d'évaluation* $\vec{z}$, $\vec{z}\,' \in \mathbb{R}^k$ *associés à un problème de minimisation. On dit que* $\vec{z}$ *domine* $\vec{z}\,'$ *si et seulement si* $\forall i \in 1..k$, $\vec{z}_i \leq \vec{z}\,'_i$ *et* $\exists i \in 1..k$ *tel que* $\vec{z}_i < \vec{z}\,'_i$.

**Définition 5 (La relation de non-dominance).** *Soit un vecteur d'évaluation* $\vec{z} \in \mathcal{Z}$. *On dira que* $\vec{z}$ *est non dominé dans* $\mathcal{Z}$ *si et seulement s'il n'existe aucun vecteur* $\vec{z}\,' \in \mathcal{Z}$ *tel que* $\vec{z}\,'$ *domine* $\vec{z}$.

**Définition 6 (Efficacité au sens de Pareto).** *Une solution* $\vec{x} \in \mathcal{X}$ *est dite Pareto efficace dans* $\mathcal{X}$ *si et seulement si* $\nexists \vec{x}\,' \in \mathcal{X}$ *tel que* $f(\vec{x}\,')$ *domine* $f(\vec{x})$.

**Définition 7 (Ensemble efficace).** *L'ensemble efficace* $E = \{\vec{x} \in \mathcal{X} \mid \vec{x}$ *est efficace dans* $\mathcal{X}\}$.

**Définition 8 (Frontière de Pareto).** *La frontière de Pareto* $F = \{\vec{z} \in \mathcal{Z} \mid \vec{z}$ *est non dominé dans* $\mathcal{Z}\}$.

La frontière de Pareto comprend donc tous les vecteurs d'évaluation non dominés et sa projection dans l'espace des variables de décision correspond à l'ensemble des solutions efficaces. Puisqu'il est possible que deux solutions efficaces différentes partagent un même vecteur

*Du nom de l'économiste italien Vilfredo Pareto (1848-1923) dont l'interprétation de la théorie de l'équilibre général (*Manuel d'Économie Politique (1906)*) guide les travaux de nombreux économistes depuis les années 30.

d'évaluation, donc un même point de la frontière de Pareto, la cardinalité de $E$ est nécessairement supérieure ou égale à la cardinalité de $F$. Pour cette raison, on cherchera habituellement à déterminer exactement ou à approximer la frontière de Pareto plutôt que l'ensemble efficace. L'identification des points de cette frontière constitue donc le but ultime de l'optimisation multicritère lorsque les préférences du preneur de décision sont inconnues.

Pour terminer, deux points particuliers de l'espace des objectifs doivent être définis. Lorsqu'on considère un problème de minimisation, les points *idéal* et *Nadir* ($\vec{z}_I \in \Lambda$ et $\vec{z}_N \in \Lambda$) sont respectivement des bornes inférieure et supérieure sur la valeur des objectifs. En supposant que l'ensemble efficace est non vide, stable (tout point non efficace est dominé par un point efficace) et compact (les valeurs extrêmes pouvant donc être atteintes), ces bornes peuvent être définies comme suit :

**Définition 9 (Point idéal).** *Le point idéal* $\vec{z}^{\,I}$ *est tel que* $z_i^I = \min_{\vec{z} \in \Lambda} z_i, \forall i \in 1..k.$

**Définition 10 (Point Nadir).** *Le point Nadir* $\vec{z}^{\,N}$ *est tel que* $z_i^N = \max_{\vec{z} \in \Lambda} z_i, \forall i \in 1..k.$

### 1.2.2 Méthodes exactes pour déterminer la frontière de Pareto

Cette section passe en revue les principales techniques permettant de déterminer la frontière de Pareto exacte d'un POM. Parmi celles-ci, on retrouve certaines des méthodes applicables lorsque les préférences sont connues *a priori*. Une fois modifiées, ces dernières permettent théoriquement d'identifier tous les points de la frontière de Pareto.

#### Somme pondérée des fonctions objectifs

Optimiser l'équation (1.1) pour tous les vecteurs de poids possibles permet d'énumérer l'ensemble des points de la frontière de Pareto lorsque la projection du domaine réalisable dans l'espace des objectifs est convexe. On distingue trois sous-ensembles de points dans $\mathcal{Z}$ : les points *supportés et extrêmes* correspondant aux points extrêmes de l'enveloppe convexe de $\mathcal{Z}$, les points *supportés et non extrêmes* correspondant aux points sur l'enveloppe convexe de $\mathcal{Z}$ et pouvant être exprimés comme une combinaison linéaire de deux points extrêmes, et les points

*non supportés* correspondant à tous les points à l'intérieur de l'enveloppe convexe de $\mathcal{Z}$. On peut montrer (voir [95]) que seuls les points supportés, extrêmes et non dominés peuvent être générés par une optimisation de l'équation (1.1) pour toutes les valeurs de $\lambda$. Les points non supportés et non dominés, qui existent seulement lorsque $\mathcal{Z}$ n'est pas convexe, constituent la principale difficulté puisqu'ils sont impossibles à trouver à l'aide d'une somme pondérée. En fait, ils sont introuvables parce qu'ils sont dominés par des points de l'enveloppe convexe de $\mathcal{Z}$, qui ne font pas partie de $\mathcal{Z}$.

Koski (1985) [66] démontre que les solutions efficaces sont généralement plus nombreuses que les solutions supportées. Il n'en demeure pas moins qu'il arrive parfois qu'une somme pondérée des objectifs soit utilisée pour résoudre des POM non convexes, tels que des POM combinatoires. Ehrgott et Gandibleux (2002) [32] donnent des références pour des problèmes de plus courts chemins, de flots dans les réseaux, d'arbre de recouvrement minimal et de sac à dos.

### Méthodes basées sur la distance avec un objectif de référence

Il est possible de pondérer la fonction de distance et de faire varier les poids afin d'énumérer la frontière de Pareto. La distance de Tchebychev est communément évoquée pour ce genre d'approche. La version pondérée de l'équation (1.3) s'écrit :

$$L_\infty(\vec{z}) = \max_{i \in \{1,\ldots,k\}} \lambda_i |(\vec{z}_u)_i - (\vec{z})_i| \tag{1.9}$$

Contrairement à la somme pondérée des objectifs qui ne garantit pas de trouver toutes les solutions non dominées, l'utilisation d'une distance de Tchebychev permet théoriquement d'énumérer la frontière de Pareto exacte, même pour des problèmes discrets [91]. Toutefois, Sayin et Kouvelis (2005) [92] soulignent que l'application de ces résultats théoriques n'a toujours pas été réalisée avec succès.

### Méthode *$\epsilon$-constraint*

Outre la somme pondérée des objectifs, la méthode *$\epsilon$-constraint* est certainement la plus répandue en optimisation multicritère, et plus particulièrement en optimisation bi-critère. Il est

en effet possible, sous certaines conditions, d'énumérer la totalité de la frontière de Pareto en faisant varier les valeurs du vecteur $\epsilon$ (voir équation (1.5)). Cette méthode a été appliquée à des problèmes d'affectation, d'arbre de recouvrement minimal, de commis voyageur, de sac-à-dos et de création d'horaires [32]. Une revue plus détaillée de la littérature entourant cette approche est présentée au chapitre 4.

**Optimisation séquentielle**

L'optimisation séquentielle[†] fonctionne bien pour les problèmes bi-critères. Le principe consiste à trouver successivement la meilleure solution, la deuxième meilleure solution, la troisième meilleure solution et ainsi de suite, toujours par rapport au même objectif. Par exemple, on commencera par trouver la meilleure solution pour laquelle $z_i = z_i^I$. On augmentera ensuite progressivement la valeur de $z_i$ jusqu'à ce que $z_i = z_i^N$. L'ensemble des solutions ainsi obtenues peut inclure quelques solutions dominées. Ces dernières sont toutefois facilement identifiables et peuvent donc être exclues sans problème. Clìmaco et Martins (1982) [22] démontrent l'exactitude de cette approche pour un problème de plus courts chemins à deux critères.

L'optimisation séquentielle demeure toutefois difficile à appliquer à des problèmes de plus de 3 critères car le point Nadir devient alors très difficile à calculer, comme en témoigne la discussion de Korhonen, Salo et Steuer (1997) [65]. Des méthodes d'approximation sont nécessaires, ce qui limite la qualité des solutions ; le point Nadir utilisé étant approximatif. Les problèmes de *max-ordering* font exception à cette règle. Ehrgott et Skriver (2003) [33] proposent d'ailleurs une méthode générale adaptée à ce type de problème.

**Programmation dynamique**

La programmation dynamique est souvent appliquée à des variantes multicritères de problèmes monocritères pour lesquels il existe des algorithmes de programmation dynamique performants. La formule récursive associée au problème original doit alors être adaptée à un contexte impliquant plusieurs objectifs. Cela a été fait, entre autres, pour le problème de trans-

[†]Traduction libre de *ranking methods*

port, de plus courts chemins et de sac-à-dos [32]. Bertsekas (1987) [13] fait une présentation détaillée de la programmation dynamique.

**Séparation et évaluation progressive**

La technique de séparation et évaluation progressive (*branch-and-bound*) est largement utilisée pour la résolution de problèmes d'optimisation combinatoires. Comme son appellation française l'indique, cette approche est basée sur une séparation du problème original en une multitude de sous-problèmes mutuellement disjoints mais permettant toujours une énumération exhaustive du domaine réalisable. L'évaluation progressive de ces sous-problèmes permet d'améliorer des bornes sur la valeur de la solution optimale jusqu'à ce que cette dernière ait pu être identifiée.

L'adaptation de cette méthode aux POM pose un défi de taille : les bornes, qui étaient des valeurs scalaires dans le cas monocritère, deviennent des vecteurs jouant le rôle de points Nadir pour les sous-problèmes. Comme nous le mentionnions plus tôt, ces points sont très difficiles à calculer en présence de plus de deux critères. Il existe tout de même quelques applications au problème du sac-à-dos, de *max-ordering*, de plus courts chemins et de *cutting stock* [32].

**Méthode à deux phases**

La méthode à deux phases proposée par Ulungu (1993) [98] et Ulungu et Teghem (1995) [99] constitue un cadre de résolution exact pour des problèmes bi-critères. Durant la première phase, l'ensemble des solutions efficaces supportées sont trouvées à l'aide de l'approche par somme pondérée, les poids étant calculés au fur et à mesure. Les solutions efficaces non supportées sont trouvées au cours de la seconde phase selon des techniques propres au problème à résoudre. Cette méthode est très répandue et constitue, selon Ehrgott et Gandibleux (2002) [32], l'approche de loin la plus utilisée pour énumérer tous les points de la frontière de Pareto.

## 1.3 Études de cas

Cette section présente une revue de la littérature entourant les deux problèmes de tournées multicritères dans des graphes qui seront étudiés dans les chapitres suivants : le problème de planification de voyage et le problème du voyageur de commerce avec profits.

### 1.3.1 Problème de planification de voyage

Le problème de planification de voyage n'est pas un problème classique clairement défini comme l'est le problème du voyageur de commerce avec profits. De manière générale, nous considérons qu'un problème de planification de voyage consiste à établir un itinéraire répondant le mieux possible aux exigences d'un voyageur. L'objectif de cette section est de dresser un portrait général de la littérature traitant de problèmes similaires, sans toutefois nous attarder à notre problème particulier et aux méthodes de résolution que nous avons employées. Cette mise en contexte plus pointue sera faite au début du chapitre 2.

Notre problème de planification de voyage est inspiré de la problématique étudié dans le mémoire de maîtrise de Bérubé (2003) [14]. Cette recherche portait sur un système de planification de voyage destiné à être utilisé par des agents informatiques intelligents évoluant dans un environnement dynamique et incertain. Un modèle d'incertitude appliqué au domaine du voyage est présenté, ainsi qu'une modélisation des préférences des voyageurs. Une stratégie de planification permettant de construire des plans (avions, hôtels) complets et optimaux conformes à ces préférences est ensuite proposée. Cette approche combine le formalisme des planificateurs classiques à une recherche de type $A^*$.

D'autres chercheurs se sont intéressés à des problèmes semblables. Dunstall et al. (2004) [30] présentent l'*Electronic Travel Planner*, un système utilisant des heuristiques dans le but de fournir des itinéraires touristiques adaptés aux préférences des voyageurs. Les auteurs n'expliquent toutefois pas le fonctionnement de leur algorithme. Ils font plutôt une description détaillée de l'architecture de leur système, qui permet par ailleurs au voyageur d'exprimer des préférences complexes.

Crainic, Ricciardi et Strochi (2004) [25] se sont quant à eux intéressés à l'identification d'itinéraires respectant les contraintes temporelles et les préférences d'un touriste visitant une ville comportant de nombreux monuments historiques (l'exemple de Rome est suggéré par les auteurs). Un ensemble de caractéristiques permettant d'attribuer un score aux différents attraits touristiques sont proposées. Cette fois encore, les auteurs demeurent silencieux sur les techniques d'optimisation utilisées. Ils mentionnent toutefois que l'algorithme de recherche $A^*$ fait partie des stratégies qu'ils utilisent.

Mentionnons finalement les travaux de Caramia et al. (1999) [17] qui modélisent une problématique reliée à l'industrie du tourisme comme une variante du problème du voyageur de commerce avec profits. Nous en discuterons dans la section suivante qui porte justement sur ce problème.

### 1.3.2 Problème du voyageur de commerce avec profits

Le problème du voyageur de commerce avec profits est une variante bi-critère du problème du voyageur de commerce (PVC) classique qui consiste à déterminer un cycle hamiltonien de coût minimum dans un graphe complet. L'allégorie vient de la confection d'un trajet de coût minimum permettant à un voyageur de commerce de visiter tous ses clients une et une seule fois avant de revenir à son point de départ. Mais l'intérêt pour ce problème d'optimisation combinatoire va bien au-delà de la problématique du voyageur de commerce cherchant à faire sa tournée à moindre coût. En fait, à peu près toutes les problématiques de prise de décision relatives au séquencement de diverses actions, tâches ou opérations peuvent être formulées comme des instances particulières du PVC.

La formulation mathématique du problème est basée sur un graphe $G = (V, E)$, orienté ou non, sur lequel on définit pour chaque arête $e \in E$ un coût $c_e$ représentant la distance entre ses deux sommets incidents. Le PVC consiste à trouver un cycle hamiltonien dans $G$ tel que la somme des coûts des arêtes qui le composent soit minimale. Nous pouvons supposer sans perte de généralité que $G$ est un graphe complet puisque, s'il ne l'est pas, il suffit de remplacer les arêtes manquantes par des arêtes ayant un coût très élevé.

Cette section présente une variante du PVC : le PVC avec profits (PVCP). Le PVCP se distingue du PVC d'abord parce que tous les sommets du graphe n'ont pas à être visités pour qu'une solution soit réalisable, et ensuite parce qu'une valeur (appelée profit) est associée à chacun des sommets. Le PVCP consiste à maximiser le profit amassé tout en minimisant les coûts de déplacement. Avant de discuter du PVCP, nous ferons ressortir les principales variantes du PVC afin de situer le PVCP dans le vaste paysage des variantes du PVC, qui fait par ailleurs l'objet d'un livre édité par Gutin et Punnen (2002) [49].

**Les principales variantes du PVC**

Nous mentionnions plus tôt que le graphe sur lequel le problème est défini peut être orienté ou non. Lorsque $G$ est orienté, c'est habituellement parce que le problème est *asymétrique*, c'est-à-dire que $c_{ij}$ est potentiellement différent de $c_{ji}$. Autrement, le problème est *symétrique* et suppose normalement que les coûts respectent l'inégalité du triangle. Ces propriétés du graphe sont indépendantes de ce que nous appelons les variantes du PVC original, c'est-à-dire que chacune d'elles peut être symétrique ou asymétrique et avoir ou non une structure de coûts respectant l'inégalité du triangle.

Afin de mettre un peu d'ordre dans la myriade de problèmes issus du PVC original, nous les avons classés selon trois catégories, allant de la plus spécifique à la plus générale. Les contraintes du PVC original définissent une catégorie très spécifique de PVC. En plus du PVC original, celle-ci inclut des problèmes tels que le *Maximum PVC* [11] qui consiste à maximiser les coûts de déplacement et le *Bottleneck PVC* [59] où l'on cherche à minimiser le coût le plus élevé parmi l'ensemble des coûts $c_{ij}$ d'une solution. La classe des PVC avec sélection inclut toutes les variantes dont les solutions réalisables ne visitent pas nécessairement tous les sommets du graphe. C'est notamment le cas du *PVC généralisé* [39] et du *Covering Tour Problem* [43]. Dans le *PVC généralisé*, on sépare l'ensemble des sommets $V$ en $k$ sous-groupes disjoints d'au moins trois sommets de telle sorte que $V = V_1 \cup V_2 \cup ... \cup V_k$. Une solution est réalisable seulement si chaque sous-groupe est visité au moins une fois. Le *Covering Tour Problem* est plus général, les sous-groupes n'étant pas nécessairement disjoints.

Nous utilisons la même nomenclature que Feillet, Dejax et Gendreau (2005) [36] en re-

groupant sous la bannière *PVC avec profits (PVCP)* les variantes du PVC avec sélection qui associent une valeur aux sommets visités. Le problème consiste alors à choisir les sommets à visiter puis à trouver le cycle hamiltonien de coût minimum dans le sous-graphe composé de ces sommets. Deux objectifs fortement contradictoires doivent donc être optimisés. D'un côté, on cherche à maximiser le profit engendré par la visite des sommets, et d'un autre côté, on voudrait minimiser les coûts de déplacement du voyageur. Alors que le premier objectif pousse le voyageur à se déplacer, le second l'incite plutôt à limiter ses déplacements. Bien qu'il s'agisse clairement d'un problème à deux objectifs, presque toutes les recherches à ce jour ont été faites sur des versions à un seul critère ; soit en combinant les deux critères dans une somme pondérée, soit en contraignant l'un des deux.

**Les principales variantes du PVCP**

Il existe de nombreuses variantes du PVCP et l'absence d'une appellation unique pour chacune d'elles crée parfois une certaine confusion. Afin de s'y retrouver, nous reprenons ici la classification en trois variantes proposée dans [36]. Les auteurs y distinguent le *Profitable Tour Problem (PTP)*, le *Orienteering Problem (OP)* et le *Prize Collecting TSP (PCTSP)*.

Présenté par Dell'Amico (1995) [27], le PTP combine les deux objectifs du PVCP (maximisation du profit et minimisation des coûts) en une seule fonction. On cherche à maximiser la différence entre la somme des profits amassés et la somme des coûts encourus. Dans ce problème, le choix du mot *profit* pour identifier les valeurs associées aux sommets est discutable. Il serait plus juste de parler de gains associés aux sommets. La différence des gains et des coûts représente alors véritablement un profit à maximiser. Le PTP est aussi connu sous le nom de *Simple Cycle Problem* [39].

Le *Orienteering Problem (OP)* consiste à trouver un circuit qui maximise le profit total sous contrainte que les coûts n'excèdent pas une certaine valeur. Ce problème a été introduit par Tsiligirides (1984) [97] dans une étude sur les compétitions d'*orienteering*. Celles-ci ont lieu dans les montagnes ou en forêt et consistent à se déplacer d'un point de contrôle initial à un point de contrôle final en passant par divers points de contrôle intermédiaires auxquels sont associés différents pointages. Le gagnant est celui qui accumule le score le plus élevé sans

dépasser le temps alloué pour atteindre le point de contrôle final. Le OP est aussi présenté sous le nom de *Selective TSP* dans [68] et de *Maximum Collection Problem* dans [60].

Le *Prize Collecting TSP (PCTSP)*, défini par Balas (1989) [6], consiste à minimiser le coût total sous la contrainte que le profit soit au moins égal à une valeur minimale. Le problème original présenté dans [6] associe une pénalité aux sommets non visités. La somme de ces pénalités est ajoutée aux coûts de déplacement. Le PCTSP est présenté sous le nom de *Quota TSP* par Awerbuch, Azar, Blum et Vempala (1998) [4].

**Formulation du PVCP dans un graphe non orienté**

Nous avons choisi de ne présenter que la formulation non orientée du PVCP car les problèmes auxquels nous nous intéresserons ultérieurement sont tous des problèmes symétriques dans lesquels les coûts respectent l'inégalité du triangle.

Lorsque le graphe n'est pas orienté, on définit pour tous les sous-ensembles de sommets $S \subset V$ un ensemble $\delta(S)$ composé des arêtes reliant un sommet de $S$ à un sommet de $V \setminus S$. On définit aussi une variable $x_e$ pour chaque arête $e \in E$ et une variable $y_v$ pour chaque sommet $v \in V$. Les variables $x_e$ sont égales à 1 lorsque l'arête correspondante fait partie de la solution et les variables $y_v$ sont égales à 1 lorsque le sommet correspondant fait partie de la solution. Le profit associé au sommet $v$ est noté $p_v$ et le coût associé à l'arête $e$ est noté $c_e$. Les contraintes du PVCP s'écrivent alors :

$$\sum_{e \in \delta(\{v\})} x_e = 2y_v \qquad (\forall\, v \in V) \tag{1.10}$$

$$\sum_{e \in \delta(S)} x_e \geq 2y_v \qquad (\forall\, S \subset V :\ 1 \in S,\ v \in V \setminus S) \tag{1.11}$$

$$y_1 = 1 \tag{1.12}$$

$$x_e \in \{0,1\} \qquad (\forall\, e \in E) \tag{1.13}$$

$$y_v \in \{0,1\} \qquad (\forall\, v \in V) \tag{1.14}$$

La contrainte (1.10) est une contrainte de degré obligeant chaque sommet faisant partie d'une

solution à avoir deux arêtes incidentes. Les contraintes d'élimination de sous-tours sont traduites par la contrainte (1.11). La contrainte (1.12) force la visite d'un dépôt (le sommet 1), point de départ et d'arrivée de toutes les solutions.

Dans le modèle précédent, nous avons omis la fonction objectif car celle-ci est différente d'une variante à l'autre. Nous présentons ici les formulations pour les trois variantes présentées précédemment. L'objectif du PTP s'écrit :

$$\min \left[ \sum_{e \in E} c_e x_e - \sum_{v \in V} p_v y_v \right] \tag{1.15}$$

sous les contraintes (1.10) à (1.14). La formulation du OP est :

$$\max \sum_{v \in V} p_v y_v \tag{1.16}$$

sous les contraintes (1.10) à (1.14) ainsi que sous la contrainte de budget (aussi appelée contrainte de *knapsack*) :

$$\sum_{e \in E} c_e x_e \leq c_{max} \tag{1.17}$$

Finalement, on formule le PCTSP comme :

$$\min \left[ \sum_{e \in E} c_e x_e - \sum_{v \in V} (y_v - 1) l_v \right] \tag{1.18}$$

où $l_v$ représente la pénalité associée au sommet $v$. Les solutions au PCTSP doivent satisfaire les contraintes (1.10) à (1.14) ainsi que la contrainte de profit (aussi appelée contrainte de quota) :

$$\sum_{v \in V} p_v y_v \geq p_{min} \tag{1.19}$$

**Complexité**

Le PVCP étant une généralisation du PVC classique, son appartenance à la classe des problèmes $NP$-difficiles est évidente. Il existe néanmoins des preuves formelles de la complexité du PVCP pour chacune de ses variantes. On peut prouver que le PTP est $NP$-difficile en montrant que le PVC est un cas particulier du PTP. Pour ce faire, on attribue un très grand profit à chacun des sommets de telle sorte que la solution optimale les visite nécessairement tous. La preuve pour le PCTSP est toute aussi simple : il suffit de modifier la contrainte de profit de telle

sorte que le profit minimal soit égal à la somme des profits du graphe. C'est ce que font, entre autres, Fischetti et Toth (1988) [40]. L'appartenance du OP à la classe des problèmes $NP$-difficiles peut se faire par réduction. Par exemple, Golden, Levy et Vohra (1987) [46] montrent que le OP est un cas particulier du PVC. De la même manière, il est possible de réduire le OP à un problème de circuit hamiltonien [68]. Certaines instances du PVCP peuvent être résolues en temps polynomial. C'est notamment le cas dans un article de Balas (1999) [8] où des contraintes d'ordonnancement sont introduites dans le PCTSP.

**Applications**

Le PVCP compte de nombreuses applications [36], principalement issues de situations que le PVC classique ne permet pas de modéliser de manière satisfaisante. Ceci inclut, par exemple, les applications du PVC où il n'est pas possible de satisfaire complètement la demande, c'est-à-dire qu'il est impossible de visiter tous les sommets. Ce genre de problèmes a été étudié par Gensch (1978) [45] dans le contexte de l'industrie de l'acier et par Pekny et Miller (1990) [85] dans le contexte de l'industrie des produits chimiques.

Il arrive aussi que le PVCP apparaisse comme un sous-problème dans des méthodes de résolution pour d'autres types de problèmes. C'est le cas, notamment, dans les travaux de Göthe-Lundgren et al. [50] sur le *vehicle routing cost allocation problem* de même que dans la méthode proposée par Helmberg (1999) [53] pour le *m-Cost ATSP* (PVC avec $m$ voyageurs encourant des coûts distincts). Enfin, une procédure de résolution heuristique au problème de routage de véhicules basée sur la résolution itérative de PVCPs est présentée par Noon, Mittenthal et Pillai (1994) [79].

En terminant, mentionnons que Caramia et al. (1999) [17] modélisent un problème relié à l'industrie du tourisme comme une variante du PVCP (sans toutefois en faire mention explicitement). Les auteurs présentent un algorithme de recherche tabou produisant des itinéraires passant par les principaux attraits touristiques d'une ville (musées, églises, etc.). Ces itinéraires ont pour objectif de maximiser une mesure de l'intérêt associé aux différents monuments tout en minimisant les temps de déplacement du touriste. Des contraintes temporelles sont imposées et une fluidité du flot de touristes est garantie par l'ajout de contraintes de capacité associées

aux sommets. Un ensemble de solutions, non dominées entre elles, sont obtenues par l'optimisation d'une somme pondérée des objectifs.

### Ensemble efficace et frontière de Pareto du PVCP

Le PVCP est un problème d'optimisation combinatoire bi-critère n'ayant, à notre connaissance, jamais fait l'objet d'une résolution exacte dans un contexte où les préférences du preneur de décision sont révélées *a posteriori*. Dans tous les travaux que nous avons répertoriés, les auteurs transforment le problème en une variante à un seul objectif : le PTP, le OP ou le PCTSP. À notre connaissance, seuls Keller et Goodchild (1988) [64] essaient d'approximer l'ensemble efficace en abordant le problème, qu'ils baptisent *Multiobjective Vending Problem*, avec une stratégie semblable aux méthodes $\epsilon$-*constraint*. En fait, ils fixent la valeur d'un des objectifs et résolvent le problème monocritère qui en résulte à l'aide d'heuristiques. En répètant cela pour un nombre fixe de valeurs du critère qu'ils contraignent, ils obtiennent un estimé de la frontière de Pareto. Leurs résultats ne sont pas comparés avec la véritable frontière et se limitent à des problèmes de très petite taille (25 sommets).

Les variantes monocritères du PVCP, telles que le PTP, PCTSP et le OP, ont toutefois déjà été résolues exactement. Les approches exactes se résument essentiellement à des procédures de séparation et évaluation progressive développées pour le PVC et adaptées au PVCP. Feillet, Dejax et Gendreau (2005) [36] font un inventaire des différents types de relaxation. Une première façon de faire consiste à relaxer les contraintes d'élimination de sous-tours (1.11) et à résoudre le problème d'affectation qui en résulte afin d'obtenir une borne. On discute de cette stratégie pour le PTP dans [27]. Pour les variantes contraintes du PVCP, le problème d'affectation contraint issu de la relaxation des contraintes d'élimination de sous-tours doit être résolu. Afin de faciliter le calcul des bornes, on résout la plupart du temps la relaxation linéaire du problème. La borne associée à cette relaxation peut aussi être obtenue en utilisant la relaxation lagrangienne de la contrainte de ressource [40, 85]. Lorsqu'une relaxation linéaire est utilisée, la borne peut être renforcée grandement en ajoutant une série d'inégalités valides au programme linéaire. De telles inégalités sont proposées par Fischetti, Salazar-González et Toth (1998) [38], par Gendreau, Laporte et Semet (1998) [44] et par Leifer et Rosenwein (1994)

[72]. Ces algorithmes de séparation et coupes, en particulier [38] et [44], sont d'ailleurs les plus performants selon [36].

Les algorithmes de séparation et coupes, mieux connus sous leur nom anglais *branch-and-cut*, sont issus d'une hybridation entre le principe de séparation et évaluation progressive et les méthodes de plans sécants (*cutting plane methods*). Ces dernières consistent à ajouter des hyperplans dont le demi-espace coupe un ou des points de l'espace des solutions. Dans les problèmes de programmation en nombres entiers, ces hyperplans permettent de couper des solutions non entières de la relaxation linéaire du problème. Afin de générer des coupes, un problème de séparation, souvent $NP$-difficile, doit être résolu. L'intégration d'une méthode de plans sécants à une procédure de séparation et évaluation progressive permet de poursuivre la recherche lorsque la génération de nouvelles coupes devient trop difficile. On ajoute donc un certain nombre de coupes à chaque noeud de l'arbre de recherche, après quoi une opération de branchement est effectuée si la solution issue de la procédure de plans sécants n'est pas entière. Afin de faciliter la résolution du programme linéaire, plusieurs implantations utilisent une stratégie de *pricing* qui consiste à éliminer certaines variables du modèle, choisies à l'aide d'heuristiques ou encore en fonction de leur coûts réduits, afin d'en simplifier la résolution. Évidemment, la solution issue du modèle simplifié (aussi appelé graphe éparse) n'est pas nécessairement valide pour le graphe original. Une procédure de *pricing* est donc exécutée : toutes les variables non actives dont le coût réduit est négatif (positif dans le cas d'un problème de maximisation) sont réactivées. Si aucune variable n'a à être réactivée, alors la solution associée au modèle simplifié est valide pour le modèle complet. Pour plus de détails, nous référons le lecteur au livre de Jünger, Reinelt et Rinaldi [58] qui font une description détaillée des algorithmes de séparation et coupes pour le PVC.

Une autre approche consiste à relaxer les contraintes de degré (1.10), relaxation qui s'avère efficace pour les problèmes symétriques. Le problème d'arbre-1 couvrant minimal[‡] résultant peut être résolu en temps polynomial. Ce type de relaxation est utilisé dans [40] pour le PCTSP asymétrique, dans [61, 87] pour le OP asymétrique, et dans [27] pour le PTP. Bien que la relaxation des contraintes de degré permette généralement d'obtenir de meilleures bornes que la

[‡]Le problème d'arbre-1 couvrant minimal est un arbre couvrant minimal enraciné au dépôt auquel une arête est ajoutée afin de permettre un retour à la racine.

relaxation des contraintes d'élimination de sous-tours, les algorithmes de séparation et coupes relaxant les contraintes d'élimination de sous-tours [38, 44] résolvent de plus grandes instances [36]. Finalement, l'approche additive proposée pour le PCTSP dans [40] consiste à utiliser séquentiellement différentes bornes et ce de manière additive. Cette technique s'est avérée très efficace sur des instances symétriques, mais comme les résultats datent de près de vingt ans, il est difficile de les comparer avec des stratégies plus récentes.

# Chapitre 2

# Time-dependent shortest paths through a fixed sequence of nodes : Application to a travel planning problem

Ce chapitre modélise un problème de planification de voyage sous la forme d'un problème de plus courts chemins dans un graphe dépendant du temps. Notre problème de planification de voyage est défini par une séquence ordonnée de villes à visiter avec, pour chacune, une durée minimale de séjour. Pour être admissible, une solution doit respecter la séquence de visites et permettre au voyageur de revenir à son point de départ. Nous assumons la connaissance *a priori* d'une fonction d'utilité traduisant l'ensemble des préférences du voyageur. L'objectif du problème consiste donc à maximiser cette fonction d'utilité sous les contraintes définies par la séquence de visite. Afin de travailler avec un problème de minimisation, nous remplaçons l'objectif de maximisation de l'utilité par une minimisation du *coût d'utilité*. Ce dernier correspond à la différence entre l'utilité du plan de voyage courant et celle du plan idéal, pour lequel tous les critères seraient à leur niveau d'appréciation maximal.

L'environnement est modélisé par un graphe orienté où les sommets correspondent aux

villes et où les arcs représentent autant de liaisons aériennes entre deux villes. Le délai de déplacement de même que le coût associé à chaque arc sont dépendants du temps, c'est-à-dire qu'ils dépendent de l'instant auquel le voyageur s'engage sur l'arc. En supposant que l'horizon temporel est fini et discret, le graphe original peut être étendu en un graphe spatio-temporel. Tous les sommets du graphe original y sont représentés à chacun des instants de l'horizon temporel. Quant aux arcs du graphe original, le nombre d'arcs spatio-temporels associés à chacun dépend de la fréquence des départs entre les deux villes qu'ils relient. Notre modèle inclut aussi des sommets et des arcs artificiels permettant de modéliser l'attente à l'aéroport ou le séjour à l'hôtel.

La solution optimale d'un problème de planification de voyage correspond à un plus court chemin dans le graphe spatio-temporel. Il ne s'agit toutefois pas d'un problème de plus courts chemins classique puisque la solution doit inclure une visite dans chacune des villes de la séquence de visites. De plus, en raison de la composante temporelle du problème, l'application d'un algorithme de plus courts chemins entre chaque paire de villes de la séquence de visites ne garantit pas la production d'une solution réalisable. En fait, il est possible d'obtenir une solution réalisable si on calcule les plus courts chemins entre chaque paire de villes et ce pour chaque instant de l'horizon temporel. Mais, ce genre de décomposition nécessite la résolution d'un nombre de plus courts chemins qui croît exponentiellement avec la longueur de la séquence de visites.

Nous proposons une stratégie de décomposition permettant de résoudre exactement le problème de planification de voyage en résolvant un nombre de problèmes de plus courts chemins qui croît linéairement avec la taille de la séquence de visites. De plus, nous améliorons significativement les performances de notre stratégie de décomposition en explorant progressivement la composante temporelle, ce qui nous permet d'obtenir des solutions réalisables sans avoir à considérer la totalité de l'horizon temporel. Les valeurs de ces solutions servent de bornes sur la valeur de la solution optimale et permettent ainsi d'ignorer plusieurs chemins potentiels dans le reste de l'horizon temporel.

Nous présentons des résultats sur des instances de 200 sommets et 2368 arcs. En considérant un horizon temporel de 750 heures (environ 1 mois), ceci correspond à un graphe spatio-

temporel de 300 000 sommets et de plus de 100 millions d'arcs. La plupart des instances sur ce type de réseau, pour une séquence de visites de longueur 3, sont résolues en moins de 2 minutes sur un Pentium IV de 3GHz. Le plus gros problème que nous ayons résolu (en moins de 2 heures) comportait une séquence de visites de 20 villes définie sur un graphe spatio-temporel (implicite) de 5 millions de sommets et de plus de $50 \times 10^{12}$ arcs.

# Time-dependent shortest paths through a fixed sequence of nodes : application to a travel planning problem

Jean-François Bérubé

Jean-Yves Potvin

Jean Vaucher

Département d'informatique et de recherche opérationnelle,

Université de Montréal,

C.P. 6128, succursale Centre-ville,

Montréal, Québec, Canada H3C 3J7

Article paru dans

*Computers & Operations Research*, Volume 33, 2006. Pages 1838-1856.

**Abstract**

In this paper, we introduce a travel planning problem which is solved by computing time-dependent shortest paths through a fixed sequence of nodes. Given a predetermined itinerary, our travel planning problem consists in finding the best travel plan, involving planes and hotels, based on the traveler's preferences. Our time-dependent framework therefore models plane flights, hotels, stays in each city as well as global time constraints. Given the large size of time-dependent networks, an exact decomposition algorithm is devised to solve instances of realistic size in reasonable computation times.

**Keywords.** Shortest paths, fixed nodes, time-dependent, travel planning.

## 2.1 Introduction

Various time-dependent shortest path problems (TDSPPs) have been studied in the literature. However, none of them address shortest paths that must go through a given sequence of nodes. In this case, the sequence specifies new nodes in addition to the usual origin and destination. The goal is to find a shortest path that goes through the sequence, including intermediate nodes that must be visited to go from one element to the next in the sequence.

This paper is motivated by planning problems found in the travel industry. In particular, we address planning problems where, given a predetermined itinerary and the traveler's preferences, an optimal travel plan is searched for in a deterministic environment. This applies to situations where the travel plan is constrained by particular events. For example, business trips or leisure trips where the traveler wants to see short-period exhibitions, festivals, etc. Only a few papers have been devoted to this topic : we know only about [17, 25], where the focus is on the construction of itineraries within a city, based on the traveler's preferences. The proposed models are mostly variants of the classical Traveling Salesman Problem and do not address time-dependent issues.

In our travel planning problem, a *sequence of visits*, that is, an ordered list of cities to be visited within a given time horizon, is specified. Planes are then used to travel between the cities, while hotels are used to stay in foreign cities. Each element in the sequence of visits $V$ is a pair $(v_k, d_k)$, $k = 0, ..., |V| - 1$, where $v_0$ is the departure city, $v_k$ is the $k$th city in the sequence and $d_k$ is the minimal stay duration in city $v_k$. It is assumed that the traveler can wait and stay at no cost from his departure city (city $v_0$) and that the minimal stay duration in this city is null.

A solution to the problem is a closed travel plan schedule, starting and ending at the departure city, that follows the sequence of visits $V$ and includes a stay of at least $d_k$ time units in each city $v_k$. There is absolutely no constraint on intermediate cities visited between two consecutive elements in $V$. The traveler can even go through a city $v_k \in V$ during the transit between two other consecutive cities in the sequence, $v_p$ and $v_q$ $(p, q \neq k)$. For example, the sequence of visits $V = \{(1, 0), (4, 60), (9, 80)\}$ represents a trip that must visit city 4 during

at least 60 time units and city 9 during at least 80 time units, before returning to city 1, the initial departure node. As long as they are feasible and satisfy the minimum stay durations, the sequences of cities *{1,4,6,8,9,2,1}* , *{1,3,4,1,8,9,2,4,3,1}* might be solutions to this problem.

The optimality of a solution depends on the traveler's preferences, measured in *utility* units* that aggregate the traveler's preferences. The solution that maximizes the utility is the optimal solution with regard to the traveler's preferences. This maximization problem can be transformed into a minimization problem by considering the difference between the utility of an ideal travel plan and the actual plan utility. We will refer to this difference as the *utility cost* of a solution, which is often called a *generalized cost*. Note that this paper does not address the issue of designing these costs or modeling the traveler's preferences. We assume that these costs are given.

Due to the fixed sequence of visits, a sequential solution approach looks quite natural. However, one cannot blindly apply classical search strategies to this problem. For example, a best-first search will tend to explore a large number of paths that lead to some city $v_k \in V$ before proceeding to the next city $v_{k+1} \in V$. This happens because most paths that reach $v_k$ are shorter than those that reach $v_{k+1}$. Therefore, if there are $C$ possible paths from one city to the next in the sequence, the best-first search is likely to expand $O(C^{|V|})$ paths. Exploiting look-ahead heuristics to prune the solution space might be a good way to speed up the search process. For example, the $A^*$ algorithm [51, 52, 84] can find a path of minimum cost in many Euclidean graphs in $O(n)$ [93]. This approach has been used in [14], but the lack of good heuristics to compute tight lower bounds on the cost of a solution has prevented an efficient pruning of the search space. The number of partial plans thus increased quickly with the number of cities in the sequence of visits.

The contributions of this paper are the following. First, we model our travel planning problem as a TDSPP through a fixed sequence of nodes. Taking advantage of the particular structure of this model, we propose a decomposition approach that allows us to solve problems of realistic size to optimality, by sequentially solving $|V|$ subproblems of much smaller size. All

*In economics, utility is a measure of the satisfaction gained by the consumption of a package of goods or services.

these subproblems have the same complexity and the impact of $|V|$ on the running time is thus linear. Finally, a partition of the time horizon into subintervals is proposed to speed up the computations even further.

In Section 2.2, a brief literature review on deterministic discrete TDSPPs is presented. Then, we explain how our travel planning problem can be modeled as a TDSPP through a fixed sequence of nodes. We suggest an exact approach to solve it and present computational results on large size instances (even larger than what is found in practice), based on a discretization of time in units of one hour. A conclusion follows.

## 2.2 Deterministic discrete time-dependent shortest path problems

In this section, we present an overview of deterministic discrete time-dependent shortest path problems. However, we limit our discussion to the fundamental concepts needed to understand our problem-solving approach.

### 2.2.1 Static shortest path problems

The static shortest path problem (SSPP) is worth mentioning because, as observed later, SSPP algorithms can be used to solve TDSPPs. Let $G = (N, A)$ be a graph where $N$ is the set of $n$ nodes and $A$ is the set of $m$ arcs. A cost $c_{ij}$ is associated with each arc $(i, j) \in A$. This cost may be a monetary cost, a travel time (delay) or any other resource consumption cost. The aim of the SSPP is to find the shortest paths, measured in some unique resource units, like minutes or dollars, between either one origin and one destination (one-to-one problem), one origin and all possible destinations (one-to-all problem) or all possible origins and a single destination (all-to-one problem). SSPPs are usually solved with label correcting algorithms, like those of Bellman, Ford and Moore [12, 41, 77], or the label setting algorithm of Dijkstra [28] for graphs with non negative arc costs. A heap-based implementation of the latter provides a run time of $O(m \log(n))$. The reader is referred to [3, 83], among others, for surveys on the SSPP.

### 2.2.2 Time-dependent shortest path problems

Time-dependent shortest path problems, also known as *dynamic shortest path problems*, are defined on a graph $G = (N, A)$ where $N$ is the set of $n$ nodes and $A$ is the set of $m$ arcs. A delay $d_{ij}$ is associated with each arc that represents the time needed to travel from node $i$ to node $j$, $(i, j) \in A$. This delay is time-dependent as it depends on the departure time from node $i$. The function $d_{ij}(t)$ returns the delay to travel from $i$ to $j$ when leaving $i$ at time $t$. Consequently, one will arrive at node $j$ at time $t + d_{ij}(t)$. The delay function might be discrete or continuous and, in both cases, might be stochastic. In the following, we will restrict ourselves to deterministic discrete time problems, where the number of delays on each arc is finite and known with certainty.

A cost $c_{ij}$ can be associated with each arc, representing the resource consumption (other than time) needed to travel on arc $(i, j) \in A$. As for delays, a deterministic discrete cost function $c_{ij}(t)$ returns the traveling cost on $(i, j)$ when leaving node $i$ at time $t$. To address some variants of these problems, a *waiting cost function* $w_i(t)$ is also needed, that returns the cost of waiting one time unit at node $i$ from time $t$ to $t + 1$. For example, waiting at $i$ from time $t_1$ to $t_q$ will cost $\sum_{z=1}^{q} w_i(t_z)$. The aim of the TDSPP is to find *minimum delay paths (fastest paths)*, *minimum cost paths* or *multicriteria shortest paths* in $G$.

Orda and Rom [80, 81] provide a theoretical analysis of TDSPPs and discuss the properties of some of these algorithms. Transportation applications are reported in [83].

#### The space-time network

From any discrete time-dependent graph $G = (N, A)$, we can build a *space-time network* (also known as a *time-expanded network*) where each node of $G$ is divided into as many nodes as there are discrete time steps. If $\mathcal{T} = \{t_0, ..., t_{T-1}\}$ is the set of $T$ discrete time steps, then the space-time network of $G$ has exactly $nT$ nodes and at most $mT$ arcs. The latter is denoted $G^* = (N^*, A^*)$ with node set $N^* = \{i_h : i \in N, 0 \leq h \leq T - 1\}$ and arc set $A^* = \{(i_h, j_k) : (i, j) \in A, t_h + d_{ij}(t_h) = t_k, 1 \leq h < k \leq T - 1\}$. Given that the space-time network is static, the application of a static shortest path algorithm identifies a solution that is

also valid in the original time dependent network [19].

### The FIFO and cost consistency properties

An arc $(i, j) \in A$ is said to be FIFO if, by leaving node $i$ earlier, we are guaranteed to arrive no later at node $j$. Formally, $t + d_{ij}(t) \leq (t + k) + d_{ij}(t + k), \forall t$ and $k \geq 1$. We say that a graph is FIFO if the FIFO property is satisfied on every arc.

The cost consistency property is to the costs what the FIFO property is to delays. That is, an arc $(i, j)$ is cost consistent if leaving node $i$ earlier does not cost more than waiting to leave later. The cost consistency therefore applies only when waiting is allowed.

Both the FIFO and cost consistency properties can be exploited to improve TDSPP algorithms [83]. Unfortunately, these properties do not hold in our travel planning problem, as we will see later.

### Classification of TDSPP variants

Chabini and Dean proposed a classification for different TDSPP variants in [18]. We have tried to synthesize and extend their work to some other variants. Table 2.1 shows this classification to which the bi-criteria and multicriteria variants were added. Note that the last network property in Table 2.1 states that one can travel beyond the time horizon $T$. The delays and costs for $t > T$ are then equal to the delays and costs at time $T$.

Many variants of the TDSPP, most of which are reviewed in [18], can be obtained from the characteristics found in Table 2.1. They are summarized below.

- The simple all-to-one TDSPP consists in computing shortest paths when waiting is prohibited and all costs and delays are strictly positive [24, 102]. The general all-to-one TDSPP is obtained when waiting is allowed and costs may be negative [18].
- For the general one-to-all TDSPP, an important result states that any static shortest path algorithm can solve it, if the FIFO property holds. A generalization of SSPP algorithms was thus suggested in 1969 by Dreyfus [29] to address this variant. It was later demons-

| Criteria | Variant |
|---|---|
| Objective | *Minimum cost path* ;<br>*Fastest path* : minimum cost path with $c_{ij}(t) = d_{ij}(t)$ ;<br>*Bi-criteria path* : optimization with two criteria (e.g., cost and time) ;<br>*Multicriteria path* : optimization with more than two criteria ; |
| Waiting constraints | *Forbidden waiting* : waiting at a node is forbidden ;<br>*Unbounded waiting* : waiting is allowed and is unbounded ;<br>*Bounded waiting* : waiting is allowed and is bounded ; |
| Waiting costs | *Memoryless* : waiting costs are independent of waiting duration ;<br>*With memory* : waiting costs depend on waiting duration ; |
| Source and destination | *One-to-all* : compute paths from one source to all destinations ;<br>*All-to-one* : compute paths from all sources to one destination ;<br>*One-to-one* : compute paths from a single source to a single destination ; |
| Network properties | *FIFO network* : FIFO property is satisfied ;<br>*Periodic network* : $d_{ij}(t)$ and $c_{ij}(t)$ follow periodic patterns ;<br>*Zero-delay arcs* : arcs with zero-time delays are allowed ;<br>*Travel beyond time horizon* $T$ :<br>$d_{ij}(T+k) = d_{ij}(T)$ and $c_{ij}(T+k) = c_{ij}(T), \forall k \geq 1$. |

TAB. 2.1 – Classification of time-dependent shortest path problems

trated that this generalization is valid only when the FIFO property holds [2, 62]. Some algorithms have also been developed for situations where the FIFO property does not hold [16, 83, 18].

- The one-to-one TDSPP has been less studied than the all-to-one and one-to-all TDSPPs. In [62], the authors suggest to use label-setting algorithms and to stop the search when the destination is reached. This may not lead to an efficient search given that label-setting algorithms tend to search in "circles" around the origin node, due to a lack of heuristic information to prune the solution space. In [19], adaptations of the $A^*$ algorithm are proposed to compute fastest paths in deterministic discrete time-dependent FIFO networks.
- Multicriteria variants have also been studied. For example, exact and approximate algorithms have been developed for bi-criteria TDSPPs in [78, 83, 90, 100].

## 2.3 A TDSPP framework for the travel planning problem

This section gives a TDSPP model for our travel planning problem. We start by introducing the time-dependent network of cities and flight connections. We then expand the network to obtain a discrete space-time network, to which we add the possibility to wait at an airport or in a hotel. The aim of this framework is to model the problem so that it can be solved by applying a shortest path algorithm in the corresponding space-time network.

### 2.3.1 The network of cities and flight connections

Let $N = \{1, ..., n\}$ be the set of cities, $A = \{(i,j),\ i,j \in N\}$ be the set of airline connections, where $(i,j)$ reads "flight from city $i$ to city $j$". That is, there are flights that leave city $i$ for city $j$ at one or more time points. We can therefore model the physical environment with a connected directed graph $G = (N, A)$.

Traveling on arc $(i,j) \in A$ incurs some delay $d_{ij}$ and cost $c_{ij}$. Delays are measured in discrete time units, like hours, and costs in utility units (as discussed in the introduction). A natural way to model delays and costs is to consider them as flight-dependent. For example, flight number 3 that leaves city $i$ for $j$ at time $t$ may take 4 hours at a cost of 100 utility units. But, these delays and costs can also be considered as *time-dependent*, if we assume that each flight leaves city $i$ at a different time. In the previous example, instead of associating the 4 hours delay and the 100 cost units with flight number 3, we can associate them with the departure time $t$. The assumption that no more than one flight leaves city $i$ for city $j$ at a given time is not unrealistic, assuming that the discrete time steps are small enough.

Due to time-dependent delays and costs, the network of cities and flight connections can be modeled as a directed and connected graph $G = (N, A)$, where the $n$ nodes represent the cities and the $m$ arcs represent time-dependent flight connections between cities. We will use the traditional notation for time-dependent delays and costs : the travel delay between cities $i$ and $j$ when the departure time is $t$ is noted $d_{ij}(t)$ and the generalized cost of traveling from city $i$ to city $j$ when the departure time is $t$ is noted $c_{ij}(t)$.

### 2.3.2 The space-time network of cities and flight connections

As for any time-dependent graph, we can expand $G = (N, A)$ to obtain its associated space-time network $G^* = (N^*, A^*)$ with $n^*$ nodes and $m^*$ arcs. In $G^*$, the nodes of the original network are split into as many nodes as the number of time steps over the finite time horizon. With $T$ time steps, the $n$ original nodes will expand into $nT$ nodes and so $n^* = nT$. In the following, we will note $i_t$ the node that corresponds to city $i$ at time $t$.

The expansion of arcs (flights) depends on the frequency of departures on each flight connection. If the average frequency over the whole graph is $\lambda$, the average number of flights on a connection will be $\lfloor \frac{T}{\lambda} \rfloor$. So, each arc in $A$ will generate $\lfloor \frac{T}{\lambda} \rfloor$ arcs on average. Each arc will leave a node $i_k$, $k \in [0, .., T]$, to a node $j_l$, $l = k + d_{ij}(k)$. The mean value of $m^*$ is thus $m\lfloor \frac{T}{\lambda} \rfloor \leq mT$. Figure 2.1 shows the expansion of a flight connection between cities $i$ and $j$ with $T = 5$ and $\lambda = 2$. Note that the two flights do not have the same delays nor the same costs : $d_{ij}(2) = 2$, $d_{ij}(4) = 1$, $c_{ij}(2) = 100$ and $c_{ij}(4) = 120$.

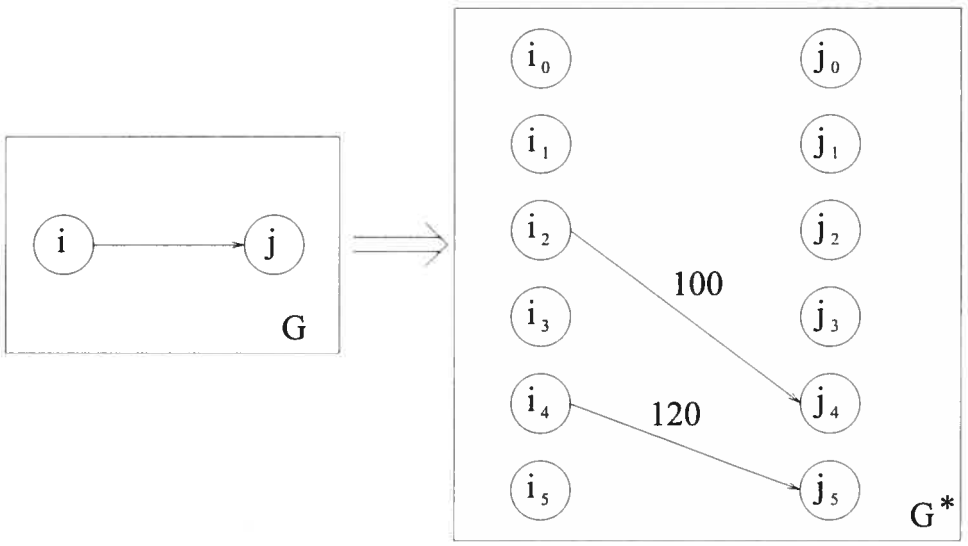


FIG. 2.1 – Space-time expansion : the left part shows the original graph and the right part shows its space-time expansion.

In any transportation system, a traveler is likely to wait. This reality should thus be incorporated into travel plans. In our case, we allow the traveler to wait at the airport for short periods

of time or, otherwise, to book a hotel room. This is explained in the following.

### 2.3.3 Allowing travelers to wait

The travelers may decide to wait at a node $i$ in order to take a flight that leaves later. Waiting is allowed through *waiting arcs* that link a space-time network node $i_t$ to another node representing the same city later in time ($i_{t+k}$ if the traveler waits for $k$ time steps). That is, an arc from each node $i_t$ to each node $i_{t+k}$ is added to the space-time network for all values of $k \in [1, T-t]$. A cost is associated with waiting, and the latter depends on the waiting duration and whether the traveler waits at the airport or in a hotel, as it is explained below.

Since the waiting cost is typically non linear and depends on the waiting duration, we cannot simply add a link from each node associated with a city to each node associated with the same city later in time. If we do so, the traveler will never use a long and costly waiting arc. He will rather use successive waiting arcs of one time unit, because the total cost of those arcs will be less than the cost of the long waiting arc. To prevent this, we will do as in [18] and duplicate each node into a waiting and a non-waiting node. The arrival in a city will always be at the waiting node, while a departure will always be from a non waiting node.

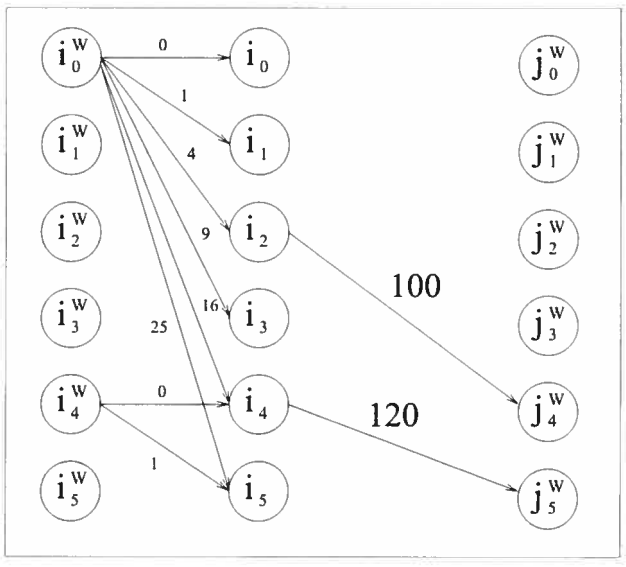


FIG. 2.2 – Space-time expansion example with waiting and non-waiting nodes

Each waiting node has outgoing arcs to non waiting nodes for the same city later in time, thus allowing the traveler to "wait" in that city. Figure 2.2 illustrates the node duplication process using our previous example with $T$=5 and $\lambda$=2. In the figure, the nodes labeled with $W$ are waiting nodes and the waiting cost function is $k^2$, where $k$ is the waiting duration. It should be noted that only waiting arcs that leave at times 0 and 4 are shown for clarity. In the following, two different ways of waiting will be

**Waiting at the airport**

The idea here is to use an increasing marginal cost function to model the increasing annoyance of waiting one more time step at an airport. For example, waiting one time unit may entail one cost unit, but waiting two time units may cost more than two units. An example based on a quadratic function is illustrated in Figure 2.3a.

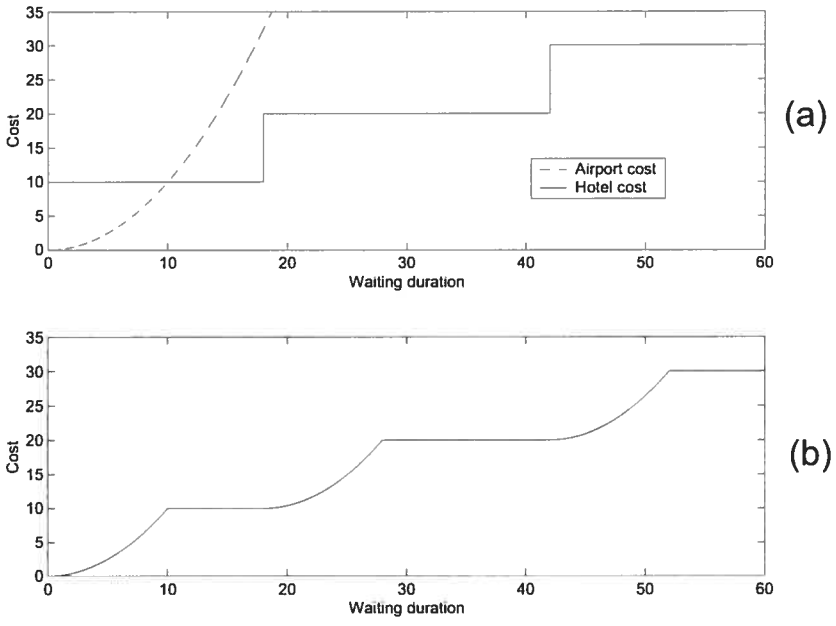


FIG. 2.3 – Example of airport and hotel waiting cost functions

**Waiting in a hotel**

Instead of waiting at the airport, the traveler may choose to wait in a hotel, which appears to be more appropriate for a stay of more than a few hours. The waiting cost function in this case would typically be very different from the one used at the airport. In the case of hotels, in particular, the marginal cost is null until the reservation is extended for another night. Figure 2.3a illustrates a case where the hotel waiting cost function follows a step function.

Using step functions to model hotel costs appear to be a reasonable choice. Let us suppose that hotel rooms are available for blocks of $B$ time steps and that the first block starts at time 0. Obviously, the traveler does not need to be in the city for the total length of his hotel reservation. He can have a reservation from time $B$ to time $3B$, but arrive at time $B + a$ and leave at time $3B - b$, $(a, b < B)$. This way of dividing hotel availability into blocks of fixed duration implies that the waiting cost is the same as long as its length fits within a single reservation block. Typically, $B$ would represent one day. In Figure 2.3, each block is 24 hours long. Note that, in this example, the current block has started 6 hours ago and the next one will thus start at time 18.

Simple functions can be devised for waiting in a hotel. The following function $f(\cdot)$ returns the cost when the traveler arrives at the hotel at time $k$ and leaves at time $l$, if the price for a single reservation block is $P$ :

$$f(k, l, B, P) = \left( \left\lceil \frac{l}{B} \right\rceil - \left\lceil \frac{k}{B} \right\rceil + 1 \right) \times P. \tag{2.1}$$

As we can see in Figure 2.3a, waiting at the airport is cheaper for short durations but, for longer durations, it is better for the traveler to wait at the hotel. The point where both functions crosses corresponds to the duration threshold where waiting at the hotel becomes the best option. In general, the waiting cost functions at the airport and in a hotel must be merged together to produce the "true" cost function on a given waiting arc. Figure 2.3b illustrates this function. The steps come from the hotel waiting function and the quadratic parts from the airport waiting function. A quadratic shape appears at the beginning of each hotel block to allow the traveler to wait at the airport for a short period of time before leaving a city (short

enough for the airport waiting cost to be cheaper than the hotel waiting cost) and avoid the high cost associated with an additional hotel block.

### 2.3.4 Classifying the TDSPP framework

We have now completed the specification of the environment graph $G = (N, A)$ and the corresponding space-time network. At this point, it is important to note that the FIFO property does not hold for graph $G$. Leaving a city node earlier does not guarantee that arrival will take place no later, because the speed of different planes is not necessarily the same. The cost consistency property also does not hold since there is no guarantee that the cost between two nodes will be lower for earlier departure times. That is, we can not take advantage of the strategies developed for FIFO and cost consistent networks.

According to the previous discussion, the travel planning problem can be solved as a time-dependent shortest path problem where the arcs are labeled with generalized costs. With regard to the TDSPP classification presented in 2.2.2, the travel planning problem is characterized by the following elements :

- *goal :* generalized cost minimization ;
- *waiting constraints :* unbounded waiting with memory ;
- *source and destination :* one-to-one "through many" ;
- *network properties :* non-FIFO, non-periodic network with zero-delay arcs and forbidden travel beyond time horizon.

The objective, waiting characteristics and network property characteristics are similar to what is found in the literature. For the source and destination attributes, however, we had to introduce the terms *through many* to underline the fact that we are looking for a one-to-one optimal path that must go through a given sequence of cities and stay in each city for a minimal duration. To the best of our knowledge, no such problem has been reported in the literature thus far.

As demonstrated by our travel planning problem, this variant is both of theoretical and practical interest. Given that our shortest paths are time-dependent and must go through a fixed

sequence of cities, the solution to this problem is far from trivial. In particular, we cannot compute a shortest path between each pair of consecutive cities in the sequence of visits and combine them, because these problems are not independent and the solution obtained is likely to be suboptimal, or even infeasible.

## 2.4 An algorithm for the travel planning problem

In this section, we present an exact algorithm for solving our travel planning problem. But first, we explain how the search space is represented.

### 2.4.1 The search space

As for any shortest path problem, the search space consists of all possible paths that start at the source nodes. In our problem, given a sequence of visits $V = \{(v_0, d_0), (v_1, d_1), (v_2, d_2), ...\}$, the source nodes are the nodes $v_{0t} \in V, \forall t \in [0, T]$. From this combinatorial number of paths, those that satisfy the traveler's visit constraints are feasible, and the optimal solution is a least cost one among those. The quality of an algorithm depends on its capacity to prune the search space efficiently, while maintaining a minimum number of pending solutions.

### 2.4.2 Decomposition of the search space

We suggest a decomposition of the search space that leads to subproblems that can be efficiently solved. Since the travel planning problem is defined through a sequence of visits, we can divide the whole problem into shortest path subproblems between consecutive pairs of cities in the sequence. The first subproblem is thus to find shortest paths from $v_{00}$ to $v_{1t}$, $\forall t \in [0, T]$, when such paths exist. This first subproblem is a one-to-many shortest path problem in $G^*$, as there are no waiting costs at the initial departure city. That is, these shortest paths are identical to those that would have been obtained from all nodes $v_{0t}$, $t \in [0, T]$. The solution of the first subproblem provides the shortest paths from the initial departure city to the first city to be visited for all possible departure and arrival times and corresponds to *solution set* $S_1$.

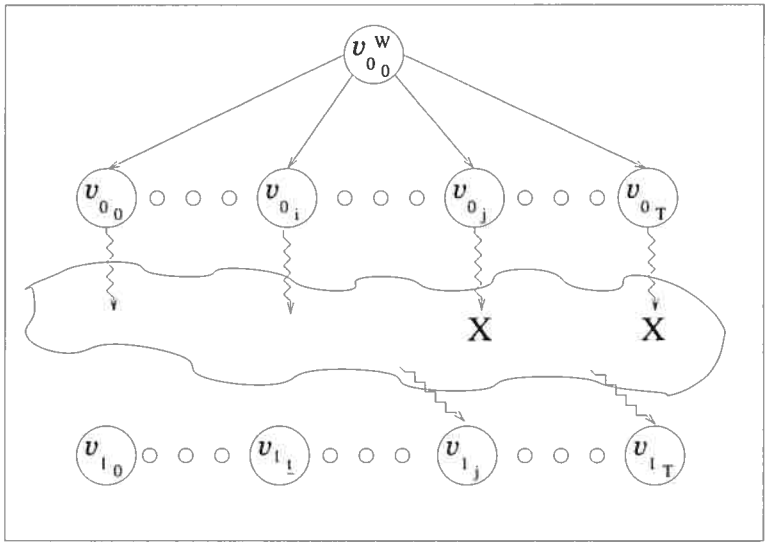


FIG. 2.4 – First subproblem obtained from the search space decomposition

Figure 2.4 shows a solution set for the first subproblem. Each path in the set starts with the initial node $v_{0_0}^W$ that represents the first city in the sequence of visits $V$ at time 0. The initial node is a waiting node (labeled with a $W$) since the traveler might decide to leave $v_0$ later. The figure depicts four possible departure times, the end points of the time horizon (times 0 and $T$) and two arbitrary times $i$ and $j$. The paths marked with an "X" are partial paths that cannot be completed. Clearly, a path that starts at time $T$ will never be completed as travel beyond time $T$ is forbidden. As there are no feasible paths from some departure times, there might also be no feasible paths to some arrival times. In the figure, there are no paths to nodes between $v_{1_0}$ and $v_{1_{\underline{t}}}$, where time $\underline{t}$ denotes the first feasible arrival time. Finally, the end node of the solutions are always *non-waiting nodes*, because the subproblems are aimed at finding a path that includes the stay in city $v_1$. To take into account the visit duration constraint, the shortest path algorithm does not consider arcs $(v_{1_k}^W, v_{1k+l})$, $\forall l < d_1$, where $d_1$ is the minimal visit duration in $v_1$.

Solutions to the subproblems are not disjoint in general, that is, a single space-time node might appear in several solutions, but *at most one* shortest path will be found for each arrival node. Although there might be many different shortest paths, we only need to find one. The solution to our first subproblem can thus be represented by set $S_1 = \{..., (p_t, c_t), ...\}$, where $p_t$ is the shortest path that arrives in $v_1$ at time $t$ at a cost $c_t$. As there might not be any feasible

arrival for some time points, $S_1$ does not necessarily contain a pair for every $t \in [0, T]$.

Through the principle of optimality, [†] one of the solutions of the first subproblem will be part of the optimal solution of the original problem. This is true because the final solution must go through $v_1$ and the solution of the first subproblem includes the best feasible paths from $v_0$ to $v_1$ for all possible time points. We can thus extend these paths to obtain the shortest paths from $v_0$ (for all departure times) to $v_2$ (for all possible arrival times).

Like the first subproblem, the second subproblem can be solved as a one-to-many shortest path problem by adding a new node, called the *surrogate* node, to $G^*$. The surrogate node has an outgoing arc to the last node of each path in $S_1$. The travel cost on these arcs is the total cost of the corresponding path. Formally, the surrogate node is linked to the end node of every path $p_t \in S_1$ with a zero-delay arc of cost $c_t$. Finding the shortest paths from the surrogate node to $v_{2t}, \forall t \in [0, T]$ is then a one-to-many shortest path problem.

This decomposition can be extended to every visit in the sequence. Let $S_k$ be the set of paths to the $k$th city in the sequence of visits. The paths in $S_k$ are necessarily the shortest paths from the initial departure city (for all departure times) to the $k$th city in the sequence for all possible arrival times within the time horizon. Moreover, these paths provide a waiting duration of at least $d_l$ in all cities $v_l \in V, l \in [0, k-1]$. Knowing $S_k$, we can add a surrogate node to $G^*$ and create arcs from this node to the end node of every path $p_t \in S_k$. Those arcs are zero-delay arcs of cost $c_t$. Figure 2.5 depicts the general subproblem, using $D_k$ to denote the surrogate node. As in Figure 2.4, the earliest arrival time is noted $\underline{t}$. Thus, there are no arcs from $D_k$ to nodes $v_{kt}, t < \underline{t}$.

The best solution to the last subproblem (the one associated with the return to the departure city $v_0$) is the optimal solution to the original problem. The travel planning problem is thus solved through a sequence of static one-to-many shortest path problems.

[†]The principle of optimality states that an optimal path is composed of optimal subpaths ; it is the foundation of dynamic programming.

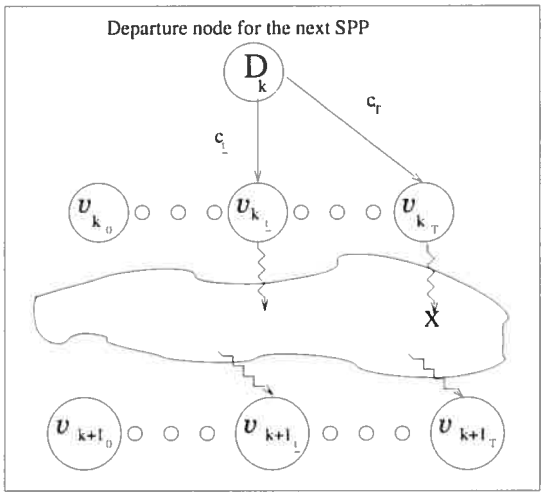


FIG. 2.5 – General subproblem of the search space decomposition

### 2.4.3 The algorithm

We will describe our algorithm by first introducing the main loop which performs the search space decomposition. We will then explain our strategy to solve the subproblems.

#### The main loop

The main loop in our algorithm performs the decomposition previously described. The idea is to compute the shortest paths between every pair of consecutive cities in the sequence of visits for all possible departure and arrival times. These shortest paths must be computed by following the sequence, that is, by starting with paths between $v_0$ and $v_1$, then between $v_1$ and $v_2$, etc.

The *findShortestPaths()* method is explained in the next subsection. The *createSurrogateNode()* method creates the surrogate node and its outgoing arcs from solution set $S$. It is presented below.

**Algorithm 1** The main loop

$S \Leftarrow \emptyset$

**for all** $v \in V$ **do**

$d \Leftarrow createSurrogateNode(S)$

$S \Leftarrow findShortestPaths(d, v)$

**end for**

{The best path in $S$ is the optimal solution}

**Algorithm 2** createSurrogateNode

Node $n$

$addNode(n)$ {add node to the space-time graph}

**for all** $(p_t, c_t) \in S$ **do**

$l \Leftarrow last(p_t)$ {$l$ is assigned to the end node of path $p_t$}

$addArc(n, l, c_t)$ {add arc of cost $c_t$ going from $n$ to $l$ }

**end for**

**return** $n$

### Solving the one-to-many SSPP

A heap implementation of Dijkstra's algorithm is used to solve the one-to-many SSPP between every pair of consecutive cities in the sequence of visits. The algorithm stops when the arrival nodes have all been reached. We have also modified the classical algorithm to introduce upper bounds on the optimal solution. The next subsection explains how these bounds have been generated.

### Iterative exploration of the time component

The algorithm can be improved by an iterative exploration of the time component, that is, by considering successive subsets of time points over the time horizon. Instead of computing the shortest paths to some city $v \in V$ for all possible arrival times, we can divide this interval into several subintervals. The idea is to obtain a complete feasible solution early during the search procedure, and to use its cost as an upper bound when subsequent time windows are explored.

The time horizon $[0, T]$ is thus split into $\gamma$ time windows : $[0, \frac{T}{\gamma}], [\frac{T}{\gamma}, 2\frac{T}{\gamma}], ..., [(\gamma - 1)\frac{T}{\gamma}, T]$.

The main loop is first executed on the first time interval : $[0, \frac{T}{\gamma}]$. At the end of this first execution, we have the best solution to the original problem, but only for paths that take place within $[0, \frac{T}{\gamma}]$. The solution cost is then used as an upper bound during the subsequent time intervals to prune paths with a larger cost. If there is no solution that contains the whole sequence of visits in the first time interval, the algorithm will continue with the subsequent interval without any upper bound. As soon as a feasible solution will be found, its cost will become an upper bound. Once a bound is found, it will be tightened by the cost of each feasible solution found during the search process.

It should be noted that all shortest paths computed within a given time interval will still be optimal when considering subsequent time intervals. This is true because no path starting in a subsequent time interval can lead to an arrival in a previous time interval. Such paths could only exist if some arcs would have negative delays, which is not the case here. However, a shortest path to a node with arrival in a given time interval may depart from a previous time interval. As the time intervals are not completely independent, some computations need to be done more than once.

Partitioning time into subintervals should therefore be used with care. With too many time intervals ($\gamma$ high) the performance might degrade, due to "computational overlapping". On the other hand, too few time intervals ($\gamma$ small) might be grossly suboptimal. Our empirical results will illustrate this phenomenon.

### 2.4.4 Complexity analysis

Applying a static shortest path algorithm to the space-time network without our decomposition approach is impracticable. Let $SPA$ be the worst-case complexity of the shortest path algorithm used on the space-time network and let $K$ be the length of the sequence of visits. Without the iterative exploration of the time component and without problem decomposition, the worst-case complexity will be $O(T^{K-1} \times SPA)$. One will have to compute the shortest paths to the first city of the sequence for all possible arrival times ($O(SPA)$). For every path found,

the paths to the second city for all possible arrival times will then be computed in $O(T \times SPA)$, etc. Overall, we thus have $O(SPA + T \times SPA + T^2 \times SPA + T^3 \times SPA + ... + T^{K-1} \times SPA)$.

Our decomposition approach reduces the search space exploration to a linear function of the space-time network size and sequence length. The first subproblem can be solved in $O(SPA)$. Since the second subproblem is also a one-to-many shortest path problem, it can also be solved in $O(SPA)$, as for every other subproblem. The worst-case complexity is therefore $O(K \times SPA)$. Since our implementation uses the heap-based Dijkstra's algorithm, its worst-case complexity is $O(K \times m^* \log(n^*))$. Note that the iterative exploration of the time component reduces the average case complexity but does not change the worst case (as long as the number of time windows $\gamma$ is set to appropriate values). As shown in the next section, our approach leads to good results in practice, even on problems of large size.

## 2.5 Computational results

In this section, we present empirical results obtained with our algorithm. We have measured the number of *nodes explored* and the time needed to obtain the optimal solution on problems of different sizes. The number of nodes explored is the number of nodes labeled by the modified Dijkstra's algorithm during the entire computational process.

In the first subsection, we describe the environment in which the simulations took place and in the second subsection, we show the results. We cannot compare our algorithm with others because we have not found any that applies to our travel planning problem.

### 2.5.1 The simulation

The tests presented here have been made on a graph of 200 cities which is approximately the number of airport cities in North America. Our graph also has 2368 flight connections. Each city is connected to at least 9 cities and at most 72 cities ; the average node degree is 24. We used time steps of one hour and considered 750 time steps (i.e., 31.25 days). All flight connections offer departures twice a day. The space-time graph has thus 150 000 nodes and

146 635 arcs, excluding the waiting nodes and waiting arcs. If we add the latter, we obtain a space-time graph with 300 000 nodes and 112 796 635 arcs. It should be noted, however, that most waiting arcs remain virtual as they are created only when they are explored.

The airport and hotel waiting cost functions are $\frac{(l-k)^2}{3}$ and $\left\lceil \frac{(l-k)}{24} \right\rceil \times 100$, respectively. The rational behavior is thus to wait during at most 12.5 hours before taking a hotel room. The problem is to find a travel plan that goes through three random cities, spending at least one day in the first city, two days in the second and three days in the third city. Finally, parameter $\gamma$ was set to 5.

We will modify the graph characteristics in the following to study how the algorithm behaves in different situations.

### 2.5.2 Results

In the following subsections, the results obtained in different simulation environments are reported. These tests have been run on a 3GHz Pentium IV processor. In all cases, the computation times are under 16 minutes. Furthermore, in most cases, the optimal solution is found within 2 minutes of CPU time.

#### Time horizon

Extending the time horizon $T$ impacts both the computation time and the number of explored nodes. Figure 2.6 shows a linear increase in both measures as $T$ increases from 200 to 5000[‡].

The value of $T$ impacts the size of the space-time network in two different ways, as the number of nodes is $nT$ and the number of arcs is in the order of $O(nT^2)$. The relation observed in Figure 2.6 indicates that the number of nodes is the dominant factor. This is not a surprise, given that many arcs in the space-time network are "artificial" waiting arcs.

[‡]Assuming steps of one hour, $T = 5000$ represents a planning horizon of 7 months.

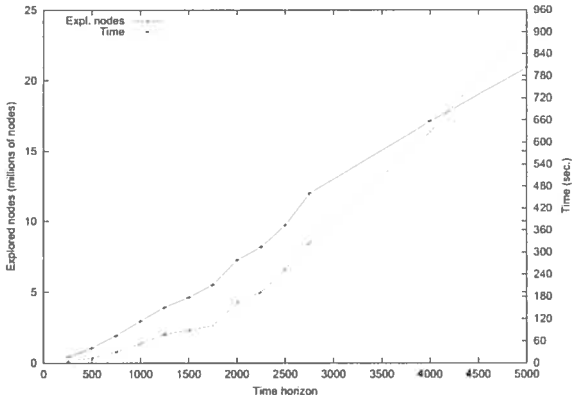


FIG. 2.6 – Impact of time horizon on number of explored nodes and computation time

**Number of cities**

The number of cities is also a very important factor with regard to the size of the space-time network. In terms of the number of cities $n$, the number of arcs in the space-time network grows in the order of $O(nT^2)$. However, as in subsection 2.5.2, a linear relation is observed in Figure 2.7. This indicates that the dominant factor is again the number of nodes. In this experiment, we have considered a time horizon of $T = 750$. Thus, each city is represented by 1500 waiting and non waiting nodes.

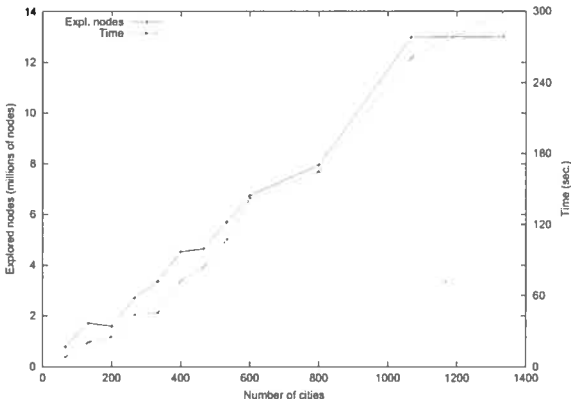


FIG. 2.7 – Impact of number of cities on number of explored nodes and computation time

### Graph density

We define the "graph density" as the number of flight connections divided by the maximum number of feasible connections. With this measure, the density of the original graph is 0.059.

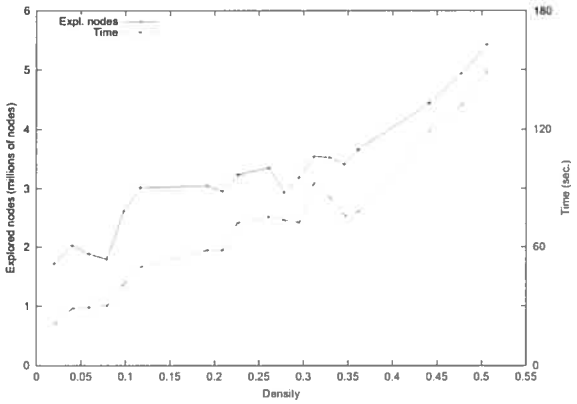


FIG. 2.8 – Impact of density on number of explored nodes and computation time

As observed in Figure 2.8, a low density graph is easier to manage, given that the computation time increases linearly with density (at least for density values below 0.5). By increasing density, we increase the number of flight connections, but we do not change the number of nodes nor the number of waiting arcs. Here, with $T = 750$, $n = 200$ and 2 flights a day on each flight connection, every increase of 0.01 unit in density adds 25 000 flight arcs.

### Flight frequency

The flight frequency is the average number of flights per day on a single flight connection. As shown in Figure 2.9, flight frequency has an impact on computation time. Clearly, the search time can only increase with more opportunities to travel between two cities.

### Sequence of visits

Here, cities were added to the sequence of visits, with a minimal stay duration of 24 hours associated with each new city. As shown in Figure 2.10, this increase has a limited impact on

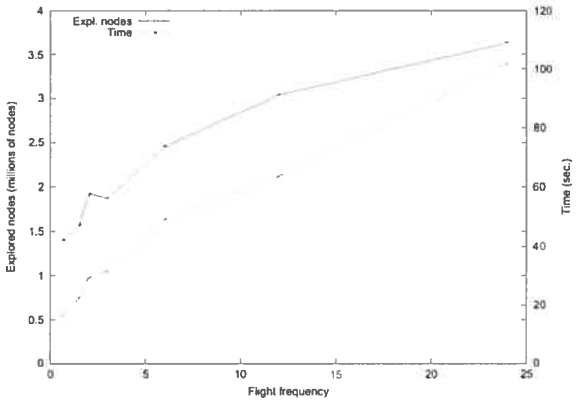


FIG. 2.9 – Impact of flight frequency on number of explored nodes and computation time

computation time. This is due to our search space decomposition that alleviates the "combinatorial explosion" associated with an increase in the number of visits.

Note that we also applied an algorithm that works on the entire search space. An $A^*$ algorithm that exploits a heuristic (under)estimation of the costs to guide the search process was implemented. This approach was also used in a previous work for a similar problem [14]. We used a transformation of the Euclidean distance between two cities to compute the estimate. We soon realized that the search space was really too large, even for toy problems, and this avenue was quickly abandoned [§]. This demonstrates the importance of the decomposition approach presented in subsection 2.4.2.

Note that no feasible solutions exist for sequences of visits with more than 13 cities (with a time horizon of 750 hours or approximately 30 days), because each visit implies a stay of at least one day, and there are also additional waiting and traveling times to get from one city to the other. In any case, a sequence of visits with more than 13 cities exceeds what would be found in most travel plans. In section 2.5.2, where we test the limits of our algorithm, results are reported for longer sequences (using a large time horizon).

[§] In fact, the memory needed to store all potential paths explodes when the length of the sequence of visits is greater than 2.

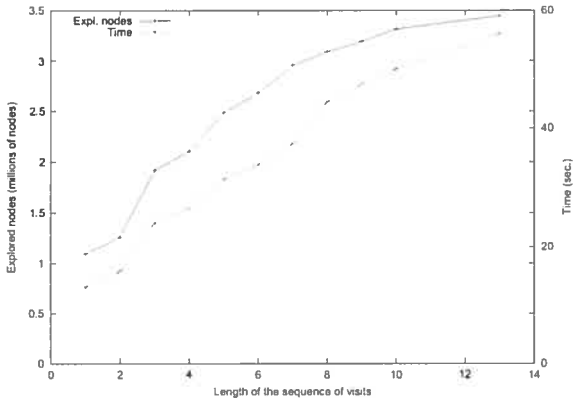


FIG. 2.10 – Impact of sequence of visits on number of explored nodes and computation time

### $\gamma$ parameter

As previously mentioned, parameter $\gamma$ determines the number of time subintervals. Values of $\gamma$ from 1 (i.e. the whole time horizon is considered) to 8 were tried on a time horizon of 1000. As observed in Figure 2.11, $\gamma = 2.75$ is the best choice for this particular problem. The shape of the curve is also typical of the parameter's impact. For low values of $\gamma$, the computation times first tend to decrease until a minimum is reached. Then, the computation times start to increase again. This underlines the importance of calibrating $\gamma$ with care.

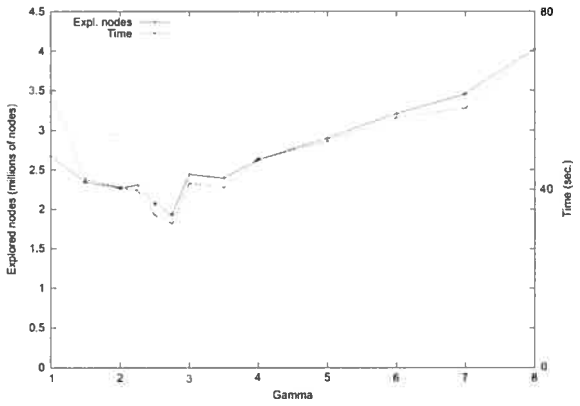


FIG. 2.11 – Impact of $\gamma$ on number of explored nodes and computation time

**Results on large problems**

We have tried to find the largest problem that could be solved with our algorithm. To do so, a large number of cities and large time horizon were used, based on a sequence of visits of length 20. In these experiments, parameter $\gamma$ was fixed at 3.

Basically, we observed that problem size was limited by the available memory (1 Gb). Although the amount of memory needed by the shortest path algorithm did not increase much, the limitations came from the space-time network, which grows very quickly.

The largest problem that we successfully solved was based on a graph of 1000 cities with a time horizon of 2500. The space-time network had 5 000 000 nodes and more than $50 \times 10^{12}$ arcs (mostly virtual waiting arcs). It was solved in 1 hour and 43 minutes, and 170 743 454 nodes in the search space were explored.

## 2.6 Conclusion

In this paper, we have shown that a travel planning problem can be modeled as a time-dependent shortest path problem through a fixed sequence of nodes. It is thus possible to apply classical shortest path algorithms to solve it, although within a proper decomposition scheme that reduces the overall complexity to the sum of simpler subproblems.

This decomposition strategy allowed us to manage large problems (a naive application of classical SSPP algorithms would not have achieved this without sacrificing optimality). The results show that our approach can solve problems of realistic size within a few minutes.

Our work can be extended in several ways. As mentioned before, the use of a generalized or aggregated cost is not necessarily the best approach to solve multicriteria optimization problems. In the case of the travel planning problem, it might be hard to build utility functions that adequately model the traveler's preferences. Therefore, a true multicriteria problem-solving approach might be considered.

Finally, we want to address a more general problem framework that allows a traveler to

specify a partially ordered sequence of visits. This would lead to additional degrees of freedom when designing a travel plan. However, searching for the best order of visits is a much more complex problem that would require the use of heuristic or approximate methods.

# Chapitre 3

# A branch-and-cut algorithm for the undirected prize collecting traveling salesman problem

Ce chapitre s'intéresse au problème du voyageur de commerce avec profits dans un contexte où les préférences du preneur de décision sont connues *a priori*. Plutôt que d'optimiser une fonction d'utilité comme nous l'avons fait au chapitre précédent, nous considérons une variante monocritère du problème dans laquelle un des deux objectifs, en l'occurrence la maximisation du profit, est transformé en contrainte : le *prize collecting traveling salesman problem (PCTSP)*. Étant donné un graphe non orienté où des coûts sont associés aux arêtes et où des profits sont associés aux noeuds, le PCTSP consiste à trouver un cycle simple de coût minimum permettant de satisfaire une contrainte de profit minimum.

Nous proposons un algorithme de séparation et coupes (*branch-and-cut*) permettant de résoudre exactement le PCTSP. Il s'agit, à notre connaissance, du premier algorithme exact pour ce problème, exception faite de l'approche additive de Fischetti et Toth (1988) [40]. Le principe général d'un algorithme de séparation et coupes est d'ajouter des inégalités valides au modèle de programmation linéaire qui est résolu à chacun des noeuds d'un algorithme de séparation et évaluation progressive. Ces inégalités ont pour objectif d'améliorer la borne associée aux

noeuds dans l'espoir de réduire le nombre de branches qui devront être explorées avant d'en arriver à une solution optimale.

Les inégalités que nous proposons proviennent du polytope du problème du sac-à-dos, du polytope du PVC, ainsi que d'une combinaison des contraintes d'élimination de sous-tours et de la contrainte de profit. La plupart sont issues des résultats théoriques sur le PCTSP démontrés par Balas [6, 7, 9] ou sont des adaptations d'inégalités valides développées pour le problème d'*orienteering*. Outre la présentation formelle de six familles d'inégalités valides pour le PCTSP, un certain nombre de considérations algorithmiques sont abordées. Des algorithmes de séparation permettant d'identifier les inégalités violées par les problèmes relaxés sont présentés, de même que les détails des principales composantes de notre algorithme de séparation et coupes.

Nous présentons des résultats pour une série d'instances obtenues en transformant des instances classiques du PVC disponibles dans la TSPLIB de Reinelt (1991) [88] en utilisant les transformations proposées par Fischetti, Salazar-Gonzalez et Toth (1998) [38]. Ces résultats permettent d'analyser le comportement des différentes familles d'inégalités en fonction de la nature et de la complexité des instances. Par exemple, nous avons comparé différentes façons d'ajouter les inégalités violées afin d'identifier la séquence de séparation la plus performante, c'est-à-dire l'ordre dans lequel il est préférable d'ajouter les inégalités violées afin de résoudre les problèmes le plus rapidement possible. Notre algorithme a pu résoudre des instances de plus de 500 sommets en moins de 4 heures.

# A branch-and-cut algorithm for the undirected prize collecting traveling salesman problem

Jean-François Bérubé

Michel Gendreau

Jean-Yves Potvin

Département d'informatique et de recherche opérationnelle,

Université de Montréal,

C.P. 6128, succursale Centre-ville,

Montréal, Québec, Canada H3C 3J7

Article soumis à la revue *Networks*

le 12 décembre 2006

**Abstract**

Given an undirected graph with edge costs and vertex prizes, the aim of the Prize Collecting Traveling Salesman Problem (PCTSP) is to find a simple cycle minimizing the total edge cost while collecting at least a minimum amount of prizes. In this paper, we present the first branch-and-cut algorithm to solve the PCTSP. We have adapted valid inequalities from cuts designed for the Orienteering Problem and we have implemented some existing polyhedral results for the PCTSP. Computational results on instances with more than 500 vertices are reported.

**Keywords.** Prize Collecting Traveling Salesman Problem, Branch-and-Cut, Valid Inequalities.

## 3.1 Introduction

The Prize Collecting Traveling Salesman Problem (PCTSP) was introduced as a model for scheduling the daily operations of a steel rolling mill [6]. Given an undirected graph with edge costs and vertex prizes, the aim of the PCTSP is to find a simple cycle minimizing the total edge cost while collecting at least a minimum total prize. Also known as the Quota TSP [4], the PCTSP has been classified as a Traveling Salesman Problem with Profits [36], along with the the Profitable Tour Problem (PTP) and the Orienteering Problem (OP). The PTP, introduced in [27], aims at maximizing the difference between the collected prizes and the travel costs ; it is also known as the Simple Cycle Problem [39]. In the OP, one must find a tour that maximizes the total collected prize while maintaining the costs under a fixed value. It has been introduced in a study on orienteering competitions [97] and is also known as the Selective TSP [68] and the Maximum Collection Problem [60].

Although there are several polyhedral results for the PCTSP on directed graphs [6, 7, 9], they have not been implemented in practice yet. To our knowledge the only exact approach for the PCTSP is a branch-and-bound algorithm due to Fischetti and Toth [40]. However, there are many publications on exact methods to solve the OP [36]. In particular, the two branch-and-cut algorithms devised for the symmetric OP in [38, 44] are currently the most effective procedures.

The two main contributions of this paper are (1) to propose the first branch-and-cut algorithm for the PCTSP, based on valid inequalities derived from the associated knapsack and Traveling Salesman (TS) polytopes, and (2) to analyze, through extensive computational results, how these inequalities behave in practice. Since such inequalities have never been used to solve the PCTSP, we believe that our computational results will help in the development of future exact algorithms for the PCTSP.

The paper is organized as follows. In Section 1, the mathematical model is presented, along with some notation. Section 2 describes the valid inequalities while separation procedures are discussed in Section 3. The overall branch-and-cut algorithm is described in Section 4. Finally, Section 5 reports computational results with different application orders for the separation pro-

cedures, using instances generated from the Traveling Salesman Library TSPLIB [88]. Note that throughout this text, we assume that the reader is familiar with the branch-and-cut methodology, as described in [58].

## 3.2 Mathematical model

Let $G = (V, E)$ be a complete undirected graph, with edge set $E$ and vertex set $V = \{1, 2, ...\}$, among which vertex 1 stands for the depot. The nonnegative integer prizes are denoted $p_v$ for each $v \in V$ ($p_1 = 0$) and the minimum collected prize is denoted $\bar{p}$. For each $e \in E$, the nonnegative integer travel costs $c_e$ satisfy the triangle inequality. We define $E(S) = \{(u, v) \in E : u \in S, v \in S\}$ and $\delta(S) = \{(u, v) \in E : u \in S, v \notin S\}$ for $S \subset V$. To simplify the notation, we write $\delta(v)$ instead of $\delta(\{v\})$ for each $v \in V$. We also define $V' = V \setminus \{1\}$ and $V(T) = \{v \in V : T \cap \delta(v) \neq \emptyset\}$ for $T \subseteq E$. The decision variables are :

$$x_e = \begin{cases} 1 & \text{if edge } e \in E \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

$$y_v = \begin{cases} 1 & \text{if vertex } v \in V \text{ is visited} \\ 0 & \text{otherwise} \end{cases}$$

Finally, we define $y(S) = \sum_{v \in S} y_v$, $p(S) = \sum_{v \in S} p_v$ for $S \subseteq V$, $x(T) = \sum_{e \in T} x_e$ and $c(T) = \sum_{e \in T} c_e$ for $T \subseteq E$. The PCTSP can then be formulated as the following 0-1 integer linear program :

$$\min \sum_{e \in E} c_e x_e \tag{3.1}$$

subject to

$$\sum_{v \in V} y_v p_v \geq \bar{p} \tag{3.2}$$

$$x(\delta(v)) = 2y_v \qquad (\forall\, v \in V) \tag{3.3}$$

$$x(\delta(S)) \geq 2y_v \qquad (\forall\, S \subset V :\ 1 \in S,\ v \in V \setminus S) \tag{3.4}$$

$$y_1 = 1 \tag{3.5}$$

$$x_e \in \{0,1\} \qquad (\forall\, e \in E) \tag{3.6}$$

$$y_v \in \{0,1\} \qquad (\forall\, v \in V') \tag{3.7}$$

Constraint (3.2) imposes a minimum bound on the collected prize. It can also be formulated as

$$\sum_{v \in V} p_v(1 - y_v) \leq U \tag{3.8}$$

where $U = p(V) - \bar{p}$. The degree constraints (3.3) insure that a feasible solution goes exactly once through each visited vertex. Constraints (3.4) are the Subtour Elimination Constraints (SEC). They require, for each visited vertex $v \in V'$ of a feasible solution, to be reachable from the depot by two edge-disjoint paths. These constraints can also be written as :

$$x(E(S)) \leq y(S) - y_v \quad (\forall\, S \subset V :\ 1 \in S,\ v \in V \setminus S) \tag{3.9}$$

or

$$x(E(S)) \leq y(S) - y_v \quad (\forall\, S \subset V :\ 1 \in V \setminus S,\ v \in S) \tag{3.10}$$

as long as the degree constraints (3.3) hold. Constraint (3.5) forces the depot to be visited and constraints (3.6) and (3.7) impose that all variables be 0-1. Note that the model forces all feasible solutions to visit at least three vertices. Since solutions with less than three vertices can be easily found by explicit enumeration, we can assume, without loss of generality, that an optimal solution contains at least three vertices.

## 3.3 Valid inequalities for the PCTSP

We use inequalities obtained from the associated knapsack and TS polytopes, and from a combination of the minimum prize constraint and the SECs. Most of them are derived from the polyhedral results of Balas for the asymmetric PCTSP [6, 7, 9] or are adaptations of valid OP inequalities found in branch-and-cut procedures [38, 44]. To our knowledge, none of these inequalities has ever been adapted and implemented to solve the PCTSP.

### 3.3.1 Inequalities from the associated knapsack polytope

We consider two types of inequalities based on knapsack constraints. The first is based on the minimum prize constraint (3.8) and is directly related to the associated knapsack polytope, while the second uses an upper bound on the optimal cost value to define a cost-based knapsack constraint. We will refer to these inequalities as the *cover* and *cost-cover* inequalities, respectively.

#### Cover inequalities

From the minimum prize constraint (3.8), we define the associated knapsack polytope $KP = conv\{y \in \{0,1\}^V : y \text{ satisfies (3.8)}\}$, where $conv$ stands for the convex hull. Among the known classes of valid inequalities for the knapsack problem, we consider the cover inequalities :

$$\sum_{v \in S}(1 - y_v) \leq |S| - 1 \tag{3.11}$$

where $S$ is a minimal cover for (3.8), that is, $S$ is a minimal subset of $V$ such that $p(S) > U$. Equation (3.11) can be strengthened as :

$$\sum_{v \in (S \cup S')} (1 - y_v) \leq |S| - 1 \tag{3.12}$$

where $S' = \{v \in V \setminus S : p_v \geq \max_{w \in S} p_w\}$ [5]. The coefficients of the $y$ variables can be lifted to obtain *lifted-cover* inequalities that reinforce equation (3.12).

**Theorem 1 ([5, 10]).** *Let $S$ be a minimal cover for (3.8), $S' = \{v \in V \backslash S : p_v \geq \max_{w \in S} p_w\}$, and $S_h$ the set of the first $h$ elements of $S$ (which are assumed to be sorted in non increasing order of prize), $h = 1, ..., |S|$. Let $V$ be partitioned into $V_0, V_1, .., V_q$, $q = |S| - 1$, where :*

$$\begin{aligned} V_h = \left\{ v \in (S \cup S') : p(S_h) \leq p_v < p(S_{h+1}) \right\}, \quad h = 2, ..., q \\ V_1 = (S \cup S') \setminus \cup_{h=2}^{q} V_h, \\ V_0 = V \setminus (S \cup S') \end{aligned} \tag{3.13}$$

*and define :*

$$\pi_v = h, \quad \forall v \in V_h, \quad h = 0, ..., q. \tag{3.14}$$

*Then the inequality :*

$$\sum_{v \in S} (1 - y_v) + \sum_{v \in V \backslash S} \pi_v (1 - y_v) \leq |S| - 1 \tag{3.15}$$

*is satisfied by all $y \in KP$.*

Since the PCTS polytope is included in $KP$ [6], the lifted-cover inequalities are also valid for the PCTSP.

**Cost-cover inequalities**

Let $c_U$ be the upper bound on an optimal solution. Then $\sum_{e \in E} c_e x_e \leq c_U$ defines a knapsack constraint in terms of the costs that can be used to derive valid inequalities. Let $S \subseteq V$, $1 \in S$ and $\sigma_S$ a lower bound on the optimal TSP value on $S$. Then, if $\sigma_S > c_U$ and if the costs satisfy the triangle inequality,

$$y(S) \leq |S| - 1 \tag{3.16}$$

is valid for the PCTSP. We are interested in special cases of equation (3.16) that are easier to separate because they do not require bounding the TSP to find $\sigma_S$. When $|S| = 3$ :

$$y_u + y_v \leq 1 \quad \forall u, v \in V' \quad \text{such that} \quad c_{(1,u)} + c_{(u,v)} + c_{(v,1)} > c_U \tag{3.17}$$

and when $|S| = 2$ :

$$y_v = 0 \quad \forall v \in V' \quad \text{such that} \quad 2c_{(1,v)} > c_U \tag{3.18}$$

### 3.3.2 Inequalities from SECs and knapsack constraints

We now introduce the *cycle-cover* and *conditional* inequalities that both use a knapsack constraint (either on the collected prizes or on the costs) to formulate strengthened versions of the SECs.

#### Cycle-cover inequalities

The cycle-cover inequalities exploit the minimum prize constraint and the fact that a feasible solution must be a cycle. Let $S \subset V$, $1 \in S$ such that $p(S) < \bar{p}$, then

$$x(E(S)) \leq y(S) - 1 \tag{3.19}$$

is a valid inequality for the PCTSP. Since $1 \in S$, at least one vertex of $S$ (the depot) must be visited by any feasible solution. Also, since $p(S) < \bar{p}$, a feasible solution cannot be completely contained in $S$. Therefore, the optimal solution will visit a subset of $1 < n^* < |S|$ vertices of $S$. Considering the optimal solution, we have $y^*(S) = n^*$. Moreover, since the optimal solution is not completely contained in $S$, $x(E(S)) \leq n^* - 1$. Hence $x(E(S)) \leq y(S) - 1$.

#### Conditional inequalities

An upper bound $c_U$ on the objective value can be used to derive inequalities similar to the cycle-cover ones, but based on the selected edges. Although they are not guaranteed to be valid, these inequalities can be conditionally used in a cutting-plane context. Let $T \subseteq E$ such that $c(T) > c_U$, then

$$x(T) \leq y(V(T)) - 1 \tag{3.20}$$

is valid for the PCTSP if no feasible solution of value lower than $c_U$ is contained in $T$, since $x(T) \leq y(V(T))$ holds for every feasible solution. This occurs, in particular, when $T$ defines a simple cycle that goes through the depot and for which $c(T) > c_U$.

### 3.3.3 Comb inequalities

Many valid inequalities for the TSP can be adapted to the PCTSP. Among those, we consider comb inequalities [47, 48] which have been adapted to the PCTSP in [7, 9]. Let us consider two sets of vertices, the handle $H \subset V$ and the teeth $T_j \subset V$ $(j = 1, .., t)$. The general comb inequalities are formulated as :

$$x(E(H)) + \sum_{j=1}^{t} x(E(T_j)) \leq y(H) + \sum_{j=1}^{t} |T_j| - \frac{3t+1}{2} \tag{3.21}$$

for all $H, T_1, ..., T_t$ satisfying :

$$|T_j \cap H| \geq 1 \qquad j = 1, .., t \tag{3.22}$$

$$|T_j \setminus H| \geq 1 \qquad j = 1, ..., t \tag{3.23}$$

$$T_i \cap T_j = \emptyset \qquad 1 \leq i < j \leq t \tag{3.24}$$

$$t \geq 3 \text{ and odd} \tag{3.25}$$

In the special case where

$$|T_j \cap H| = 1 \tag{3.26}$$

for all $j$, the inequalities are referred to as *simple comb* inequalities (or Chvátal inequalities) [21]. Simple comb inequalities become *2-matching* inequalities if

$$|T_j \setminus H| = 1 \tag{3.27}$$

for all $j$.

### 3.3.4 Logical inequalities

Obviously, if an edge $e \in \delta(v)$ is part of a solution, the vertex $v$ must be visited, hence the following logical inequality :

$$x_e \leq y_v \quad \forall e \in \delta(v), \quad v \in V' \tag{3.28}$$

Note that logical inequalities are a special case of the SECs (3.4) when $e = (j, v) \notin \delta(1)$.

### 3.3.5 Summary

Table 3.1 summarizes the valid inequalities (other then the SECs) used in our branch-and-cut algorithm. As mentioned earlier, most of them come from previous polyhedral results on the asymmetric PCTSP or have been adapted from valid OP inequalities.

| **Category** | **Description** | **References** | **Equations** |
|---|---|---|---|
| Knapsack polytope | Lifted-cover (LC) | [5, 10] | (3.15) |
| | Cost-cover (CC) | [44, 72] | (3.17), (3.18) |
| SEC and knapsack | Cycle-cover (CY) | [38] | (3.19) |
| | Conditional (CO) | [38] | (3.20) |
| TS polytope | 2-Matching (2M) | [7, 9, 38, 44] | (3.21) with (3.26), (3.27) |
| | Simple comb (SC) | [7, 9] | (3.21) with (3.26) |
| Other | Logical (LO) | [38, 44, 72] | (3.28) |

TAB. 3.1 – Summary of valid inequalities

- *LC.* The cover inequalities are introduced in [5, 10] for the knapsack problem and are used in [38] to solve the symmetric OP. The formulation is analog to (3.12), but is expressed in terms of edge variables ($x$) because the knapsack constraint is defined on the total travel cost in the OP. The lifted-cover inequalities are also presented in [5, 10], but have never been implemented to solve neither the PCTSP nor the OP.
- *CC.* Cost-cover inequalities are used to solve the symmetric OP in [44, 72] with an upper bound based on the maximum cost constraint.
- *CY.* A formulation similar to our cycle-cover inequalities is used to solve the OP in [38]. The knapsack constraint is defined on the costs and $S$ is defined as a set of edges that do not satisfy the maximum cost constraint.
- *CO.* Conditional inequalities are introduced in [38], again to solve the OP. A lower bound on the total collected prize is used to find infeasible sets of vertices.
- *2M, SC.* Adaptations of the comb inequalities to the PCTSP can be found in [7, 9]. Only the 2-matching inequalities have been implemented in the branch-and-cut algorithms for the symmetric OP [38, 44].

– *LO*. Logical inequalities are introduced in [72] and have been used to solve the symmetric OP in [38, 44, 72].

## 3.4 Separation algorithms

In this section, we describe the separation procedures used in our branch-and-cut algorithm to identify violated inequalities. In the following, we denote $G^* = (V^*, E^*)$ the graph associated with the solution $(x^*, y^*)$ obtained through the linear relaxation LP, where $V^* = \{v \in V : y_v^* > 0\}$ and $E^* = \{e \in E : x_e^* > 0\}$.

### 3.4.1 Subtour elimination constraints

Let us consider $x_e^*$ as the capacity value of edge $e \in E^*$. For any $v \in V^* \setminus \{1\}$, a minimum-capacity$(1, v)$-cut is found in $G^*$ through a maximum-flow algorithm, where the vertices $v \in V^* \setminus \{1\}$ are considered in decreasing order of the corresponding $y_v^*$. We used the highest label preflow-push algorithm [3] which runs in $O(|V^*|^2\sqrt{|E^*|})$ for any given vertex. Hence, we obtain an overall complexity of $O(|V^*|^3\sqrt{|E^*|})$ to find a most violated SEC for each $v \in V^*\setminus\{1\}$. Let $(S_v, V^*\setminus S_v)$ be the minimum-capacity cut obtained for $v$, with $1 \in (V^*\setminus S_v)$. The set $S_v$ gives a potentially violated SEC (3.10). When a violated SEC is found, $y_v^*$ is increased by $2 - \sum_{e\in\delta(S_v)} x_e^*$ to prevent the same cut from being generated in subsequent iterations. The set $(V^* \setminus S_v)$ is used to produce potentially violated cycle-cover inequalities (3.19), as explained in section 3.4.4. This separation algorithm is very similar to the one used in [38], although ours has a lower complexity on sparse graphs like $G^*$, due to the maximum-flow algorithm that we use.

### 3.4.2 Lifted-cover inequalities

Lifted-cover inequalities (3.15) result from the lifting procedure described in Theorem 1. A most-violated cover inequality is first produced from a minimum cover set $S \subset V$ such that $p(S) > U$, and its coefficients are lifted afterward. To obtain a violated cover inequality, the

set $S$ should minimize $\sum_{v \in S} y_v^*$, hence the following 0-1 knapsack problem :

$$\alpha^* = \min \sum_{v \in V} y_v^* z_v \tag{3.29}$$

subject to :

$$\sum_{v \in V} p_v z_v \geq U \tag{3.30}$$

$$z_v \in \{0,1\} \qquad (\forall\, v \in V) \tag{3.31}$$

Obviously, variables $z_v$ for which the corresponding $y_v^*$ is equal to 0 can be fixed to 1 and those for which the corresponding $y_v^*$ is equal to 1 can be fixed to 0. If $\alpha^* < 1$, $S$ gives a most-violated cover inequality. However, the latter can be strengthened by making $S$ a minimal cover set, which can be done with a simple greedy algorithm. We efficiently solve the 0-1 knapsack problem with our implementation of Martello and Toth specialized branch-and-bound algorithm [73, 74].

### 3.4.3 Cost-cover inequalities

Violated cost-cover inequalities (3.17) are identified through simple enumeration with complexity $O(|E|)$. Inequalities (3.18) are all added to the LP during its initialization at the root node of the branching tree through another enumeration procedure that runs in $O(|V|)$.

### 3.4.4 Cycle-cover inequalities

The separation algorithm for cycle-cover inequalities (3.19) is embedded within the separation algorithm for the SECs. As already mentioned, we use the minimum-capacity cut produced by the maximum-flow algorithm to obtain an hopefully violated cycle-cover inequality. Therefore, one simply needs to check for violation before adding a cycle-cover inequality.

### 3.4.5 Conditional inequalities

The heuristic separation algorithm for the conditional inequalities (3.20) enumerates the simple cycles of $G^*$ that pass through the depot. Whenever the cost of a cycle is greater then

the best known upper bound on the optimal solution, the cycle is considered as a potential candidate for $T$ and the inequality is checked for violation. Obviously, enumerating all simple cycles of $G^*$ might be very long. We thus limit the number of enumerated cycles to 100.

### 3.4.6 Comb inequalities

We consider both 2-matching and simple comb inequalities. The 2-matching inequalities are separated through the heuristic procedure suggested in [38] while violated simple comb inequalities are identified through the heuristic procedure 5.3 of Padberg and Rinaldi [82]. Although the 2-matching inequalities are a special case of the simple comb inequalities, it is worth having two distinct separation procedures due to their heuristic nature. With two different heuristics, it is hoped that different 2-matching inequalities will come up.

### 3.4.7 Logical inequalities

Logical inequalities are found by complete enumeration with a complexity of $O(|E^*|)$.

## 3.5 Overall branch-and-cut algorithm

Although there exists generic branch-and-cut frameworks [1], we decided to implement our own to have as much control as possible on the problem-solving process. Figure 3.1 depicts the overall branch-and-cut algorithm, where the main phases are represented with boxes. Explanations for each phase follow, along with details on some parameters.

### 3.5.1 Initial solution

At the root node of the branching tree, a feasible solution is computed. The cost of this solution gives an upper bound on the optimal solution value ($UB$) and the solution itself will be used to remove some variables from the initial LP as explained later. The initial solution is found through a heuristic algorithm based on the *GENIUS* [42] and *Extension and*

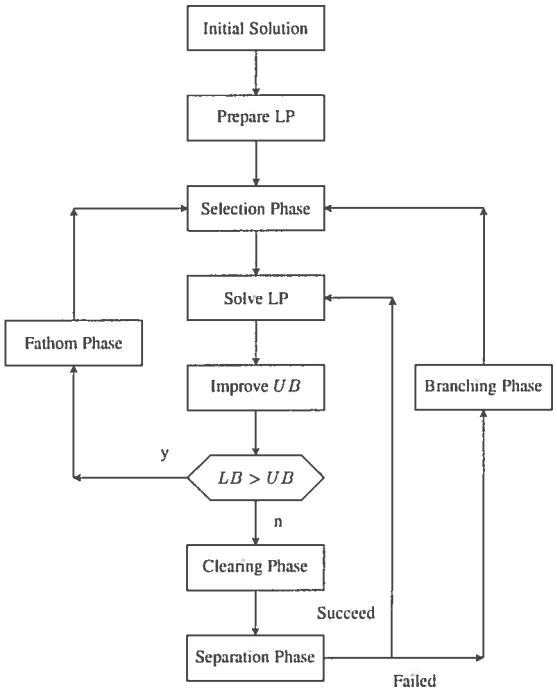


FIG. 3.1 – Overall branch-and-cut algorithm : the algorithm begins with the construction of the initial solution and ends when the selection phase failed, that is, when there is no more branching node to explore.

*Collapse* [26] procedures. Developed for the TSP, *GENIUS* is made of two phases : a tour construction phase based on generalized insertions (*GENI*) and a post-optimization phase (*US*). The *Extension and Collapse* heuristic is an improvement procedure for the PCTSP based on an iterative execution of two consecutive phases : the *Extension* phase and the *Collapse* phase. In the *Extension* phase, the collected prize of a feasible solution is improved by adding vertices to the current cycle. Vertices are sorted in decreasing order according to their prize to insertion cost ratio ($R_v$) and they are added as long as $R_v > \bar{R}$, where $\bar{R}$ is the average of $R_v$ over all the vertices. The $R_v$ ratios are updated each time a vertex is added, but not $\bar{R}$. The cost of the resulting extended solution is then reduced in the *Collapse* phase by removing vertices that lead to the largest savings in solution cost. To do so, we select the shortest path in the current solution that starts at vertex $i$, goes through the depot, and does not satisfies the minimum prize constraint. All the vertices visited by the current solution that are not on this path are removed. Finally, the cycle is closed by adding the vertex with the lowest insertion cost among those that

can be added to satisfy the minimum prize constraint. This procedure is repeated for all vertices $i$ in the current solution and the best collapsed solution is kept.

Our heuristic algorithm starts with a set $V_S = \{v \in V : p(V_S) \geq \bar{p}\}$ found through a greedy procedure that selects the vertices (including the depot) in random order until the minimum prize constraint is satisfied. The following steps are then applied :

1. A feasible solution $S_k$ visiting all vertices in $V_S$ is constructed and improved with *GENIUS*.
2. Starting with $S_k$, the following steps are applied until no more improvement is possible :
   (a) *A*pply the *Extension phase* as described in [26] to obtain solution $S_e$.
   (b) Apply *GENIUS* on the vertices of $S_e$ to obtain $S'_e$ and keep the best of $S_e$ and $S'_e$.
   (c) Apply the *Collapse phase* as described in [26] to obtain $S_c$.
   (d) Set $S_k$ as the best of $S_k$ and $S_c$.
3. Apply *GENIUS* on the vertices of the best known solution $S_k$ to obtain $S'_k$ and keep the best of $S_k$ and $S'_k$.

### 3.5.2 Initial linear program

The initial LP (*Prepare LP* box of figure 1) includes equations (3.1), (3.2), (3.3) and (3.5) along with all the vertex variables ($y$), but only a fraction of the edge variables ($x$). This reduces the size of the LP and thus helps to solve it efficiently. Edge variables visited by the initial solution are included in the LP along with the five lowest cost edges incident to each vertex visited in the initial solution.

Because some edge variables are removed from the initial LP, we cannot fathom branching nodes for which the LP is infeasible because it might be feasible if some previously excluded variables were present. In order to prevent the LP from being infeasible, we add a virtual vertex $\omega$ and artificial edges from $\omega$ to each vertex $v \in V$. In the objective of the LP, the cost associated with artificial edge $a$ is given by : $c_a = c(E) + 1$. The artificial variables ($z_a$) must also be included in the degree constraints (3.3) and may take integer values 0, 1 or 2. Suppose, for example, that a vertex $v$ is visited ($y_v > 0$) while all variables that correspond to its incident

edges are not part of the LP. Let us assume that $0.5 < y_v \leq 1$. Then, the corresponding artificial edge variable $(v, \omega)$ needs to be : $1 < z_a \leq 2$ to satisfy the degree constraint.

### 3.5.3 Selection phase

The iterative part of the algorithm starts with the selection phase where the next branching node to be explored in selected. A priority queue maintains the nodes in non-decreasing order of their associated LP solution value ($LB$). The selection consists in removing a node with the lowest $LB$ from the queue. Usually, the latter node is selected but, in some cases, the $y$ variables selected for branching are such that the maximum collected prize is lower then $\bar{p}$ and so the node is automatically fathomed. When the queue is empty, the algorithm stops and the best feasible solution is guaranteed to be an optimal solution.

### 3.5.4 Solving phase

The LP is solved with the CPLEX commercial software. However, it is important to observe that the LP solution value may not be a valid lower bound because some variables have been excluded from the model (see *Initial LP* and *Clearing Phase*). A pricing operation is performed to check if some excluded variables should be integrated into the LP : excluded variables with negative reduced costs are added to the LP before it is re-optimized. When all excluded variables have a positive reduced cost, the solution value of the LP is guaranteed to be a valid lower bound. Note that no more than 100 variables are added to the LP in a single pricing operation to keep the LP as small as possible.

### 3.5.5 Improving the upper bound ($UB$)

Because the $UB$ allows us to prune the branching tree, its improvement might speed up the algorithm by reducing the number of explored nodes. To improve the quality of this bound, the following heuristic procedure is applied after five consecutive executions of the separation phase.

1. Node selection : nodes are sorted in decreasing order of their corresponding $y^*$ value and are selected in turn, until $\bar{p}$ is reached.
2. Solution construction : a solution is then constructed with *GENIUS* [42].

If this new feasible solution is better then the best known feasible solution, $UB$ is updated.

### 3.5.6 Clearing phase

The aim of the clearing phase is to keep the size of the LP as small as possible by reducing the number of edge variables ($x$). To do so, we remove all variables $x_e$ with $\lfloor LB + 0.2r_e \rfloor \geq UB$, where $LB$ is the current LP value and $r_e$ is the reduced cost of $x_e$. Another way to reduce the size of the LP is to remove inactive constraints. We do so for constraints that have not been tight for the last 30 iterations. All removed constraints are stocked in a constraint pool.

### 3.5.7 Separation phase

This is where violated inequalities are added to the LP. If no violated cut is found in the constraint pool, the separation algorithms described in the previous section are applied sequentially until one succeeds. The violated cuts are then added and the separation phase ends. The order in which the separation algorithms are executed varies. We have tried several sequences which are described in the section on computational results. However, we should mention that all separation sequences are not possible because some separation algorithms assume that all violated SECs have been added to the LP. This is the case of the lifted-cover, cycle-cover, 2-matching, simple comb and conditional inequalities.

Adding violated cuts does not always improve the LP solution value. This phenomenon often called *tailing off* can be reduced by forcing the algorithm to branch even if some more violated cuts might be found. We force the execution of the branching phase whenever the LP solution value does not improve by at least 0.01 in the last five iterations on the same branching node.

### 3.5.8 Branching phase

The algorithm always branch on the most fractional $y$ variable, that is the $y^*$ which is closer to 0.5. If all the $y$ variables are integer, then we branch on the most fractional $x$ variable.

Because some cuts might be removed in the clearing phase, it is possible that the LP-solver finds a solution with a value lower than the lower bound associated with the branching node. To prevent this, each branching node keeps trace of the active cuts that have produced its lower bound. These cuts are added to the LP-model (if they are not already there) when a node is selected in the selection phase.

### 3.5.9 Fathom phase

The fathom phase corresponds to fathoming the current branching node.

## 3.6 Computational results

Polyhedral theory and existing branch-and-cut algorithms for the TSP, OP and knapsack problems provide insight on the types of inequalities that should be considered and on useful features of a good branch-and-cut implementation for the PCTSP. However, an analysis of the behavior of the various inequality families should also help in the development of future cutting-plane algorithms for the PCTSP. To this end, results are reported with different variants of the separation phase, as it is explained below. The branch-and-cut algorithm was implemented in C++ and was run on an AMD Opteron 2.2 Ghz processor. The LPs were solved using CPLEX 9.3 and a time limit of 4 hours (14400 seconds) was imposed on each instance.

This section starts with a description of our PCTSP instances and the five separation sequences tested. Summary results and a brief analysis on the behavior of each sequence follows. Finally, general results are reported, using the best separation sequence for each problem.

### 3.6.1 Instances

We transformed 5 VRP and 46 TSP instances of the TSPLIB [88] into PCTSP instances using the rules provided in [38]. We considered all instances with 70 to 532 vertices for which the vertex coordinates were available. For VRP instances, the demands are interpreted as the vertex prizes. For TSP instances, the prizes $p_v$ $(v \in V')$ are generated in three different ways :

- Generation 1 : $p_v = 1$ ;
- Generation 2 : $p_v = 1 + (7141v + 73) \bmod 100$ ;
- Generation 3 : $p_v = 1 + \lfloor 99\frac{c_{(1,v)}}{\theta} \rfloor$, where $\theta = \max_{w \in V'} c_{(1,w)}$.

Instances of generation 1 are in general easy because all prizes are the same. Generation 2 produces instances with pseudo-random prizes while generation 3 produces hard problems where larger prizes are associated with vertices that are further from the depot. For each instance, we search for the optimal solution of problems with a minimum collected prize $\bar{p} = \alpha p(V)$, with $\alpha \in \{0.25, 0.50, 0.75\}$.

### 3.6.2 Separation sequences

Many versions of the separation phase have been tried out to determine the best sequence. A separation sequence is defined by the order in which the inequality families are separated. In the next sections, we present results for the five most interesting separation sequences corresponding to letters A, B, C, D and E, where :

A : LO, CY* ;

B : LO, CY, CC, LC ;

C : LO, CY, 2M, SC ;

D : LO, CY, CC, LC, 2M SC ;

E : LO, CY, CC, LC, 2M, SC, CO.

*In the following, CY includes both SECs and cycle cover inequalities because they are separated through the same algorithm.

### 3.6.3 Empirical analysis of the separation sequences

For each separation sequence and for each value of $\alpha$, Table 3.2 shows summary statistics on the computation times. For each group of instance (VRP and TSP generation 1, 2 and 3), the following statistics are reported :

- SOLVED : the number of problems solved within the time limit (among the 5 VRP and 46 TSP original instances) ;
- TIME_AV : the average computation time (in seconds) spent on the problems solved within the time limit ;
- TIME_GAP : the average gap (in seconds) over each instance of a group, between the computation time of the current sequence[†] and the fastest time.

In the following, we analyse the results reported in Table 3.2, grouped per instance type and per $\alpha$ value. In both cases, we ignore separation sequence A which is clearly ill suited for almost all problems.

For VRP instances, the best separation sequences include lifted and cost-cover inequalitites (B, D, E) while sequence C, that includes only comb inequalities, is the worst. We observe the opposite for TSP generation 1 instances where sequences B and C are the worst and best sequences, respectively. The first one solves only 37 problems, while the second one solves more problems than all other sequences (39) in reasonable computation times. The under-achievement of the cover inequalities is not a surprise since they are always satisfied when all prizes are equal, which is the case for generation 1 instances. The results for TSP generation 2 instances suggest that cover inequalities alone are of no use, as sequence B shows results similar to sequence A. The largest number of instances solved is obtained when comb inequalities are added (C). The inclusion of both comb and cover inequalities (D) clearly provides the best results for generation 2 instances. The conclusions are similar for generation 3 instances where the best sequences are D and E.

Trends also emerge when we look at the data in terms of the $\alpha$ parameter instead of the instance type. When $\alpha = 0.25$, the cover inequalities (sequence B) perform well, but the best sequence is E, which includes all types of inequalities. When $\alpha$ goes up to 0.5, however,

[†]No gap is computed for the instances that were not solved within the time limit.

sequence E does not solve the maximum number of instances within the time limit. The best results are obtained by removing the conditional inequalities (sequence D). Finally, for $\alpha = 0.75$, sequences D and E both show good results.

In general, separation sequences A, B and C do not seem appropriate for the PCTSP, because of their highly unpredictable behavior. Sequence D, which combines both cover and comb inequalities, is much more robust. Separation sequence E (sequence D plus conditional inequalities) shows results similar to sequence D, although the latter is slightly better in most cases.

| $\alpha$ | 0.25 | | | | | 0.50 | | | | | 0.75 | | | | | Averages per instance type | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sequence | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E | A | B | C | D | E |
| VRP instances | | | | | | | | | | | | | | | | | | | | |
| SOLVED | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| TIME_AV | 105 | 88 | 142 | 92 | 92 | 187 | 111 | 172 | 134 | 116 | 68 | 74 | 106 | 59 | 69 | 120 | 91 | 140 | 95 | 92 |
| TIME_GAP | 17 | 0 | 54 | 4 | 5 | 78 | 1 | 62 | 24 | 7 | 10 | 16 | 48 | 1 | 10 | 35 | 6 | 55 | 10 | 7 |
| TSPP generation 1 instances | | | | | | | | | | | | | | | | | | | | |
| SOLVED | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 41 | 40 | 39 | 34 | 34 | 36 | 36 | 36 | 37 | 37 | 39 | 38 | 38 |
| TIME_AV | 1017 | 945 | 785 | 843 | 758 | 661 | 675 | 1269 | 1036 | 701 | 568 | 528 | 545 | 550 | 569 | 749 | 716 | 866 | 810 | 676 |
| TIME_GAP | 107 | 34 | 51 | 110 | 23 | 150 | 165 | 198 | 287 | 69 | 405 | 365 | 48 | 54 | 74 | 221 | 188 | 99 | 150 | 55 |
| TSPP generation 2 instances | | | | | | | | | | | | | | | | | | | | |
| SOLVED | 39 | 38 | 40 | 39 | 39 | 31 | 31 | 36 | 36 | 35 | 31 | 32 | 35 | 35 | 35 | 34 | 34 | 37 | 37 | 36 |
| TIME_AV | 1207 | 964 | 1227 | 895 | 905 | 934 | 847 | 1148 | 1014 | 997 | 1385 | 1718 | 1526 | 1203 | 1270 | 1175 | 1176 | 1300 | 1037 | 1057 |
| TIME_GAP | 192 | 192 | 190 | 94 | 104 | 317 | 231 | 278 | 145 | 199 | 703 | 846 | 468 | 146 | 212 | 404 | 423 | 312 | 128 | 172 |
| TSPP generation 3 instances | | | | | | | | | | | | | | | | | | | | |
| SOLVED | 25 | 29 | 24 | 29 | 28 | 26 | 28 | 32 | 33 | 33 | 27 | 26 | 31 | 32 | 31 | 26 | 28 | 29 | 31 | 31 |
| TIME_AV | 1567 | 1249 | 1139 | 1303 | 1021 | 1967 | 2063 | 2170 | 1972 | 2081 | 2749 | 2269 | 2206 | 2008 | 1752 | 2094 | 1860 | 1838 | 1761 | 1618 |
| TIME_GAP | 377 | 68 | 395 | 123 | 222 | 847 | 862 | 539 | 384 | 493 | 1630 | 1436 | 688 | 124 | 234 | 951 | 789 | 541 | 210 | 316 |
| Averages per value of $\alpha$ | | | | | | | | | | | | | | | | | | | | |
| SOLVED | 27 | 28 | 27 | 28 | 28 | 25 | 26 | 29 | 29 | 28 | 24 | 24 | 27 | 27 | 27 | 25 | 26 | 28 | 28 | 28 |
| TIME_AV | 974 | 812 | 823 | 783 | 694 | 937 | 924 | 1190 | 1039 | 974 | 1193 | 1147 | 1096 | 955 | 915 | 1035 | 961 | 1036 | 926 | 861 |
| TIME_GAP | 173 | 74 | 173 | 83 | 89 | 348 | 315 | 269 | 210 | 192 | 687 | 666 | 313 | 81 | 133 | 403 | 352 | 252 | 125 | 138 |

TAB. 3.2 – Empirical analysis of five different separation sequences

### 3.6.4 General results associated with the fastest separation sequences

General statistics for each instance and each $\alpha$ value, using the fastest separation sequence, are given in Tables 3.3, 3.4, 3.5 and 3.6. The columns correspond to :

- TIME : the total computation time ;
- NDS : the number of branching nodes explored ;
- CUTS : the number of cuts added to the LP ;
- GAP : the gap (%) between the upper and lower bounds at the root ;
- %UB : the gap (%) between the upper bound obtained at the root node and the optimal

solution value ;

– OPT : the optimal value.

For instances still unsolved after 4 hours (t.l.), we give the best feasible solution value in the $OPT$ column and the gap (%) between the best upper bound and the best lower bound (value of the best known feasible solution) in the $GAP$ column.

The results confirm that TSP instances of generation 3 are the most difficult to solve while generation 1 instances are the easiest. The average computation time increases from 757 to 1563 seconds between generation 1 and 3. Moreover, an average of 91 branching nodes are explored for generation 1 instances against 855 nodes for generation 3 instances. The quality of the upper and lower bounds at the root is also worse for generation 3 (average gap of 10.4%) as compared to generations 1 (7.4%) and 2 (6.2%). We observe that the average computation time is about the same for all three values of $\alpha$ even if the average number of explored nodes rise from 237 to 452 between $\alpha = 0.25$ and $\alpha = 0.75$. This increase might be due to the heuristics used to find feasible solutions (see sections 3.5.2 and 3.5.5) that seem to find better solutions when $\alpha$ is high. Actually, the average gap at the root is 10.9%, 5.7% and 4.5% for $\alpha = 0.25, 0.50$ and 0.75, respectively. Moreover, the quality of the upper bound at the root node is 6.2%, 4.0% and 3.1% for $\alpha = 0.25$, 0.50 and 0.75, respectively. Among the 455 instances considered, 30 where solved at the root node of the branching tree and 85 where still unsolved after 4 hours of computation. For these instances, the final gap between the lower bound and the best feasible solution found is 4.3%, on average. One should also note that 10 unsolved problems show a final gap under 0.1%.

| $\alpha$ | 0.25 | | | | | | 0.50 | | | | | | 0.75 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | TIME | NDS | CUTS | GAP | %UB | OPT | TIME | NDS | CUTS | GAP | %UB | OPT | TIME | NDS | CUTS | GAP | %UB | OPT |
| att48 | 0.52 | 3 | 207 | 10.15 | 9.84 | 6896 | 0.45 | 3 | 122 | 3.87 | 3.65 | 11734 | 0.44 | 1 | 96 | 0.0 | 0.0 | 20685 |
| eil51 | 0.29 | 1 | 74 | 0.0 | 0.0 | 76 | 1.3 | 37 | 253 | 2.48 | 0.0 | 161 | 3.03 | 69 | 370 | 2.33 | 1.55 | 254 |
| eilA76 | 4.11 | 24 | 382 | 14.15 | 11.32 | 94 | 2.97 | 13 | 317 | 4.5 | 4.0 | 192 | 4.88 | 32 | 290 | 4.75 | 4.75 | 321 |
| eilA101 | 7.33 | 12 | 662 | 18.58 | 17.7 | 93 | 8.53 | 5 | 740 | 0.97 | 0.49 | 205 | 4.99 | 21 | 271 | 0.59 | 0.3 | 336 |
| gil262 | 425.49 | 35 | 4400 | 1.03 | 0.26 | 387 | 534.61 | 31 | 4236 | 2.79 | 2.43 | 803 | 278.45 | 26 | 2989 | 0.84 | 0.76 | 1305 |

TAB. 3.3 – Statistics on VRP instances

| $\alpha$ | 0.25 | | | | | | 0.50 | | | | | | 0.75 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | TIME | NDS | CUTS | GAP | %UB | OPT | TIME | NDS | CUTS | GAP | %UB | OPT | TIME | NDS | CUTS | GAP | %UB | OPT |
| st70 | 0.6 | 1 | 170 | 0.0 | 0.0 | 129 | 0.85 | 1 | 289 | 0.0 | 0.0 | 260 | 3.14 | 22 | 395 | 10.71 | 10.08 | 428 |
| eil76 | 0.8 | 1 | 123 | 0.0 | 0.0 | 102 | 0.94 | 1 | 166 | 0.0 | 0.0 | 235 | 0.78 | 1 | 99 | 0.0 | 0.0 | 335 |
| pr76 | 1.31 | 1 | 313 | 0.0 | 0.0 | 23450 | 2.39 | 5 | 312 | 0.31 | 0.02 | 41248 | 109.56 | 389 | 1412 | 5.22 | 3.89 | 64142 |
| gr96 | 5.21 | 1 | 814 | 0.0 | 0.0 | 10460 | 38.05 | 125 | 1001 | 13.54 | 12.51 | 20688 | 12.5 | 11 | 778 | 3.33 | 3.03 | 31343 |
| rat99 | 2.87 | 1 | 420 | 0.0 | 0.0 | 293 | 14.3 | 29 | 719 | 2.2 | 1.53 | 581 | 11.67 | 9 | 629 | 1.95 | 1.83 | 856 |
| kroA100 | 13.07 | 25 | 825 | 18.73 | 14.42 | 4931 | 9.11 | 7 | 811 | 5.6 | 5.32 | 9184 | 10.83 | 9 | 738 | 3.7 | 3.39 | 14379 |
| kroB100 | 10.16 | 13 | 775 | 10.23 | 9.18 | 4305 | 5.72 | 1 | 700 | 0.0 | 0.0 | 9096 | 7.42 | 1 | 685 | 0.0 | 0.0 | 14680 |
| kroC100 | 16.99 | 29 | 1164 | 8.78 | 6.27 | 4964 | 18.26 | 19 | 1092 | 8.11 | 6.62 | 9457 | 10.11 | 7 | 740 | 3.65 | 3.29 | 13892 |
| kroD100 | 16.11 | 23 | 1168 | 15.98 | 10.46 | 4762 | 6.3 | 9 | 626 | 0.64 | 0.4 | 8719 | 6.99 | 1 | 549 | 0.0 | 0.0 | 14058 |
| kroE100 | 6.75 | 13 | 587 | 1.28 | 0.0 | 3905 | 6.87 | 5 | 643 | 1.27 | 1.18 | 9097 | 7.12 | 3 | 524 | 0.57 | 0.5 | 14622 |
| rd100 | 4.57 | 3 | 610 | 2.97 | 2.97 | 1438 | 6.53 | 7 | 661 | 4.56 | 3.8 | 3168 | 9.52 | 19 | 617 | 3.96 | 3.39 | 5094 |
| eil101 | 4.59 | 8 | 441 | 1.77 | 0.88 | 112 | 4.36 | 5 | 450 | 5.33 | 4.92 | 232 | 3.1 | 2 | 261 | 1.26 | 1.26 | 391 |
| lin105 | 7.79 | 5 | 658 | 25.12 | 8.61 | 2718 | 168.02 | 735 | 1547 | 11.38 | 8.09 | 5920 | 103.17 | 93 | 1740 | 2.18 | 1.61 | 9120 |
| pr107 | 2.34 | 1 | 404 | 0.0 | 0.0 | 8678 | 6.87 | 1 | 802 | 0.0 | 0.0 | 18311 | 16.23 | 9 | 991 | 2.1 | 1.96 | 36292 |
| pr124 | 30.38 | 15 | 1896 | 6.37 | 1.99 | 14325 | 13.3 | 1 | 1226 | 0.0 | 0.0 | 22998 | 42.17 | 7 | 1485 | 6.15 | 5.79 | 39008 |
| bier127 | 6.5 | 7 | 516 | 1.34 | 0.46 | 10986 | 4.49 | 1 | 402 | 0.0 | 0.0 | 26347 | 11.1 | 19 | 455 | 3.14 | 2.8 | 49680 |
| ch130 | 20.4 | 31 | 1122 | 9.96 | 7.94 | 1183 | 8.64 | 1 | 726 | 0.0 | 0.0 | 2408 | 69.65 | 245 | 1011 | 4.33 | 3.61 | 3876 |
| pr136 | 20.21 | 3 | 1383 | 13.63 | 13.53 | 21116 | 71.88 | 13 | 2788 | 27.28 | 13.24 | 46167 | 24.83 | 1 | 857 | 0.0 | 0.0 | 67910 |
| gr137 | 17.58 | 9 | 1128 | 9.97 | 9.79 | 17723 | 10.62 | 1 | 912 | 0.0 | 0.0 | 29575 | 23.3 | 1 | 1098 | 0.0 | 0.0 | 44100 |
| pr144 | 29.09 | 1 | 1503 | 0.0 | 0.0 | 14327 | 84.26 | 23 | 2188 | 14.68 | 13.43 | 27424 | 138.9 | 93 | 1670 | 13.03 | 11.55 | 41136 |
| ch150 | 20.31 | 11 | 1108 | 3.1 | 1.7 | 1332 | 22.95 | 27 | 928 | 0.61 | 0.29 | 2760 | 132.18 | 57 | 1511 | 1.52 | 1.09 | 4366 |
| kroA150 | 21.35 | 11 | 1189 | 4.03 | 3.31 | 5786 | 2137.76 | 2003 | 5036 | 6.36 | 4.49 | 11496 | 329.77 | 549 | 2090 | 0.65 | 0.12 | 17395 |
| kroB150 | 39.98 | 9 | 1571 | 4.29 | 3.53 | 5896 | 36.53 | 5 | 1637 | 1.11 | 1.09 | 11357 | 25.68 | 5 | 941 | 1.45 | 1.42 | 16879 |
| pr152 | 44.14 | 5 | 1740 | 0.55 | 0.0 | 20029 | 68.85 | 33 | 1938 | 5.71 | 5.29 | 36333 | 161.66 | 197 | 2153 | 1.19 | 1.0 | 51826 |
| u159 | 90.94 | 23 | 4259 | 0.85 | 0.63 | 9266 | 570.01 | 251 | 3241 | 8.37 | 7.2 | 18511 | 48.3 | 1 | 2011 | 0.0 | 0.0 | 27096 |
| rat195 | 102.23 | 25 | 1627 | 2.62 | 2.45 | 558 | 156.26 | 56 | 1544 | 5.62 | 5.28 | 1112 | 211.16 | 19 | 1496 | 2.81 | 2.63 | 1665 |
| d198 | 46.58 | 1 | 1568 | 0.0 | 0.0 | 4994 | 1366.88 | 359 | 3715 | 6.26 | 5.69 | 6913 | 222.57 | 19 | 2300 | 19.61 | 11.93 | 9091 |
| kroA200 | 197.99 | 27 | 2779 | 2.71 | 2.32 | 6154 | 118.79 | 26 | 2029 | 4.85 | 4.49 | 12372 | 63.84 | 3 | 1382 | 0.49 | 0.46 | 19211 |
| kroB200 | 64.5 | 9 | 1881 | 2.37 | 1.76 | 6016 | 351.33 | 109 | 2451 | 9.7 | 8.67 | 12338 | 61.29 | 1 | 1522 | 0.0 | 0.0 | 18898 |
| gr202 | 30.33 | 1 | 1425 | 0.0 | 0.0 | 8221 | 328.51 | 311 | 1686 | 4.51 | 3.94 | 13790 | 108.26 | 13 | 1481 | 4.18 | 4.08 | 20914 |
| ts225 | 6531.68 | 1933 | 16040 | 13.56 | 11.68 | 29121 | t.l. | | | 2.55 | | 57995 | t.l. | | | 0.46 | | 85949 |
| tsp225 | 187.05 | 19 | 2727 | 14.42 | 12.91 | 924 | 317.29 | 79 | 2944 | 26.66 | 11.43 | 1721 | 1522.61 | 388 | 3634 | 4.76 | 4.54 | 2629 |
| pr226 | 410.39 | 29 | 4674 | 65.22 | 7.78 | 20158 | 5429.52 | 351 | 5959 | 68.25 | 15.8 | 36720 | 573.84 | 9 | 4452 | 59.24 | 27.46 | 47142 |
| gr229 | 83.23 | 13 | 2021 | 5.98 | 5.72 | 18823 | 154.54 | 9 | 2674 | 3.05 | 2.95 | 39875 | 143.54 | 7 | 1721 | 2.45 | 2.36 | 64959 |
| gil262 | 179.51 | 7 | 3321 | 3.58 | 3.22 | 541 | 165.14 | 18 | 2835 | 4.28 | 4.09 | 986 | t.l. | | | 0.39 | | 1552 |
| pr264 | 24.12 | 1 | 678 | 0.0 | 0.0 | 9232 | 532.23 | 13 | 2358 | 4.71 | 4.63 | 22644 | 177.51 | 3 | 2539 | 43.25 | 13.38 | 34346 |
| a280 | 1765.66 | 271 | 14298 | 21.12 | 10.97 | 649 | 303.65 | 17 | 3086 | 4.21 | 4.13 | 1231 | 461.14 | 7 | 3149 | 10.7 | 8.82 | 1841 |
| pr299 | 562.18 | 17 | 4140 | 14.08 | 13.72 | 11209 | t.l. | | | 2.89 | | 23089 | 672.98 | 3 | 4312 | 5.47 | 5.45 | 32674 |
| lin318 | t.l. | | | 2.68 | | 8787 | 2355.21 | 29 | 8415 | 12.9 | 12.63 | 15913 | 1860.79 | 285 | 3875 | 13.86 | 13.47 | 25288 |
| rd400 | 483.4 | 1 | 6143 | 0.0 | 0.0 | 3084 | 2110.73 | 45 | 6413 | 4.67 | 4.48 | 6284 | 11466.7 | 217 | 9426 | 2.81 | 2.59 | 9906 |
| fl417 | t.l. | | | 30.68 | | 1975 | t.l. | | | 11.64 | | 5754 | t.l. | | | 3.88 | | 8149 |
| gr431 | 883.06 | 51 | 2757 | 2.74 | 2.46 | 14641 | 14285.7 | 583 | 6483 | 2.96 | 2.42 | 35222 | t.l. | | | 1.98 | | 74952 |
| pr439 | t.l. | | | 7.59 | | 21334 | 1483.28 | 8 | 5745 | 1.11 | 1.09 | 35297 | t.l. | | | 5.21 | | 60803 |
| pcb442 | t.l. | | | 0.63 | | 11067 | t.l. | | | 0.01 | | 22281 | t.l. | | | 0.31 | | 34305 |
| d493 | 11933.1 | 320 | 15499 | 37.02 | 19.33 | 9355 | 1943.26 | 44 | 8215 | 1.19 | 1.16 | 13582 | t.l. | | | 6.22 | | 22338 |
| att532 | 4521.65 | 14 | 10366 | 4.3 | 4.17 | 3812 | 10280.1 | 69 | 11485 | 3.68 | 3.54 | 8943 | t.l. | | | 7.60 | | 17406 |

TAB. 3.4 – Statistics on TSP generation 1 instances

| $\alpha$ | 0.25 | | | | | | 0.50 | | | | | | 0.75 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | TIME | NDS | CUTS | GAP | %UB | OPT | TIME | NDS | CUTS | GAP | %UB | OPT | TIME | NDS | CUTS | GAP | %UB | OPT |
| st70 | 0.96 | 5 | 177 | 4.65 | 3.88 | 124 | 1.31 | 3 | 324 | 1.98 | 1.98 | 247 | 7.5 | 19 | 598 | 5.63 | 4.93 | 405 |
| eil76 | 1.38 | 1 | 235 | 0.0 | 0.0 | 99 | 6.44 | 31 | 531 | 6.64 | 5.21 | 200 | 5.21 | 31 | 467 | 1.59 | 0.95 | 312 |
| pr76 | 5.87 | 29 | 734 | 7.03 | 5.77 | 22157 | 22.13 | 173 | 824 | 5.76 | 5.41 | 38330 | 188.18 | 977 | 2201 | 1.06 | 0.0 | 58419 |
| gr96 | 34.76 | 105 | 1091 | 10.15 | 4.79 | 10991 | 34.6 | 95 | 1311 | 10.39 | 9.07 | 19380 | 62.01 | 117 | 1352 | 1.97 | 1.38 | 28725 |
| rat99 | 17.75 | 159 | 518 | 12.12 | 9.76 | 268 | 61.5 | 196 | 1275 | 9.43 | 7.83 | 518 | 24.67 | 129 | 746 | 1.3 | 0.65 | 767 |
| kroA100 | 38.38 | 55 | 1575 | 18.87 | 13.44 | 4716 | 29.7 | 79 | 1226 | 7.64 | 4.24 | 8519 | 96.42 | 141 | 1759 | 4.34 | 3.87 | 13115 |
| kroB100 | 7.01 | 11 | 619 | 5.31 | 2.31 | 3590 | 41.7 | 173 | 1076 | 8.92 | 7.0 | 7794 | 35.41 | 71 | 1070 | 2.56 | 2.18 | 13441 |
| kroC100 | 68.63 | 103 | 1805 | 3.47 | 0.97 | 4881 | 41.16 | 35 | 1834 | 9.52 | 6.96 | 9060 | 29.02 | 77 | 857 | 2.17 | 1.48 | 12732 |
| kroD100 | 75.45 | 139 | 1507 | 14.77 | 10.11 | 4777 | 30.74 | 113 | 878 | 12.2 | 11.36 | 8267 | 40.37 | 147 | 933 | 3.37 | 3.2 | 12897 |
| kroE100 | 12.23 | 79 | 522 | 14.07 | 12.76 | 3397 | 17.9 | 33 | 899 | 14.05 | 13.73 | 7644 | 91.46 | 249 | 1069 | 2.43 | 1.98 | 12803 |
| rd100 | 19.87 | 25 | 1148 | 7.16 | 6.97 | 1508 | 22.29 | 73 | 1029 | 3.08 | 2.23 | 2892 | 44.44 | 115 | 916 | 0.64 | 0.37 | 4350 |
| eil101 | 3.73 | 1 | 432 | 0.0 | 0.0 | 108 | 9.11 | 23 | 558 | 3.69 | 2.76 | 211 | 71.52 | 89 | 1480 | 2.55 | 1.7 | 347 |
| lin105 | 5.48 | 35 | 326 | 2.72 | 2.18 | 2515 | 716.02 | 1397 | 4376 | 6.97 | 5.09 | 5614 | 143.45 | 391 | 1368 | 8.71 | 7.14 | 8604 |
| pr107 | 22.7 | 31 | 1060 | 0.59 | 0.11 | 8322 | 76.69 | 321 | 1688 | 9.4 | 9.23 | 26372 | 212.35 | 305 | 2919 | 3.19 | 2.85 | 33592 |
| pr124 | 130.58 | 77 | 2381 | 14.45 | 10.72 | 14261 | 162.39 | 93 | 3017 | 1.01 | 0.51 | 23150 | 390.93 | 147 | 2947 | 8.74 | 8.26 | 37407 |
| bier127 | 50.33 | 173 | 795 | 6.84 | 3.79 | 10525 | 37.55 | 113 | 768 | 1.27 | 0.95 | 24478 | 298.98 | 401 | 2192 | 4.63 | 3.82 | 48015 |
| ch130 | 79.37 | 139 | 1557 | 9.93 | 5.7 | 1159 | 83.66 | 147 | 1467 | 7.16 | 6.45 | 2220 | 37.41 | 43 | 1096 | 2.07 | 1.91 | 3552 |
| pr136 | 19.61 | 39 | 988 | 3.85 | 2.68 | 18143 | t.l. | | | 1.64 | | 40241 | 308.91 | 371 | 2077 | 5.79 | 5.63 | 58571 |
| gr137 | 75.21 | 131 | 1392 | 8.96 | 7.53 | 17252 | 366.15 | 281 | 1805 | 2.19 | 1.72 | 28242 | 441.08 | 309 | 3044 | 1.73 | 1.4 | 42644 |
| pr144 | 56.94 | 5 | 2558 | 1.75 | 1.71 | 13991 | 284.38 | 155 | 2965 | 3.96 | 3.51 | 27073 | 963.33 | 705 | 2775 | 6.88 | 5.53 | 40589 |
| ch150 | 35.57 | 31 | 1236 | 10.62 | 9.96 | 1238 | 541.81 | 285 | 3433 | 5.1 | 3.66 | 2476 | 36.02 | 31 | 1093 | 8.03 | 7.7 | 3922 |
| kroA150 | 172.63 | 81 | 2494 | 16.71 | 14.74 | 5306 | 60.85 | 43 | 1580 | 9.26 | 8.85 | 9968 | 6748.62 | 3793 | 10563 | 1.81 | 0.84 | 15658 |
| kroB150 | 56.77 | 37 | 1606 | 0.2 | 0.0 | 4958 | 469.95 | 115 | 3270 | 7.91 | 6.27 | 10278 | 238.58 | 123 | 2181 | 3.33 | 2.92 | 14971 |
| pr152 | 121.46 | 47 | 2405 | 1.17 | 0.96 | 19864 | 249.75 | 173 | 2640 | 2.61 | 2.28 | 34474 | 4299.57 | 5887 | 16911 | 6.44 | 5.17 | 50520 |
| u159 | 320.51 | 147 | 3599 | 13.07 | 9.96 | 8953 | 763.28 | 169 | 5006 | 4.6 | 3.22 | 17161 | 2569.05 | 535 | 6641 | 1.92 | 1.75 | 25377 |
| rat195 | 129.58 | 31 | 2292 | 2.89 | 2.7 | 505 | 112.81 | 13 | 1874 | 3.24 | 3.14 | 988 | 543.85 | 25 | 2773 | 5.88 | 5.63 | 1510 |
| d198 | 141.14 | 109 | 1935 | 7.84 | 7.65 | 4819 | 2579.62 | 470 | 4814 | 2.44 | 2.22 | 6653 | t.l. | | | 0.01 | | 8835 |
| kroA200 | 1469.1 | 109 | 4403 | 15.91 | 14.07 | 5898 | 2278.69 | 386 | 6196 | 6.98 | 5.91 | 11219 | t.l. | | | 0.53 | | 17305 |
| kroB200 | 223.31 | 55 | 2770 | 16.4 | 14.25 | 5490 | 415.38 | 175 | 3417 | 12.98 | 12.53 | 11250 | t.l. | | | 0.05 | | 17210 |
| gr202 | 169.37 | 95 | 2194 | 3.31 | 2.81 | 7930 | 753.52 | 373 | 3675 | 3.05 | 2.64 | 12804 | 636.21 | 450 | 2431 | 2.06 | 1.79 | 19165 |
| ts225 | 576.52 | 207 | 3332 | 9.43 | 7.8 | 27478 | 907.77 | 127 | 3928 | 19.69 | 18.86 | 53102 | 2078.69 | 301 | 6973 | 4.86 | 4.33 | 79841 |
| tsp225 | 341.86 | 73 | 4046 | 8.86 | 8.42 | 848 | 1803.93 | 351 | 4565 | 7.09 | 6.32 | 1585 | 670.7 | 315 | 3249 | 2.54 | 2.46 | 2416 |
| pr226 | 498.88 | 23 | 4201 | 3.07 | 1.68 | 19956 | 8186.06 | 652 | 9643 | 13.96 | 11.15 | 36190 | 1846.79 | 137 | 8244 | 1.4 | 1.34 | 46096 |
| gr229 | t.l. | | | 1.45 | | 17215 | 3478.96 | 565 | 6979 | 4.58 | 3.97 | 35856 | 1254.23 | 319 | 3713 | 0.73 | 0.71 | 57295 |
| gil262 | t.l. | | | 1.20 | | 500 | 165.21 | 21 | 2971 | 0.12 | 0.12 | 865 | 4055.05 | 717 | 8187 | 2.24 | 1.95 | 1356 |
| pr264 | 3447.13 | 213 | 4585 | 3.73 | 2.97 | 8721 | t.l. | | | 5.01 | | 23660 | 1311.34 | 269 | 4456 | 0.3 | 0.24 | 33247 |
| a280 | 1413.6 | 513 | 6805 | 0.7 | 0.52 | 572 | t.l. | | | 1.92 | | 1143 | 561.59 | 20 | 3858 | 6.86 | 6.8 | 1658 |
| pr299 | t.l. | | | 0.06 | | 10292 | t.l. | | | 3.37 | | 20613 | t.l. | | | 0.27 | | 29980 |
| lin318 | t.l. | | | 3.52 | | 8275 | 3394.98 | 43 | 7538 | 1.82 | 1.49 | 14909 | 4222.08 | 390 | 7555 | 6.78 | 6.77 | 23571 |
| rd400 | t.l. | | | 0.04 | | 2784 | 3102.53 | 83 | 8235 | 3.23 | 2.78 | 5590 | 6745.16 | 107 | 9128 | 1.65 | 1.49 | 8915 |
| fl417 | 5488.45 | 37 | 13755 | 7.56 | 6.92 | 1748 | t.l. | | | 13.95 | | 5971 | t.l. | | | 0.01 | | 7647 |
| gr431 | 1882.38 | 241 | 3602 | 2.29 | 1.94 | 13555 | t.l. | | | 0.03 | | 31725 | t.l. | | | 1.94 | | 69195 |
| pr439 | 10242.4 | 261 | 10022 | 0.33 | 0.24 | 18214 | t.l. | | | 0.15 | | 33110 | t.l. | | | 8.89 | | 61024 |
| pcb442 | 4992.98 | 71 | 9479 | 0.7 | 0.57 | 9548 | t.l. | | | 0.02 | | 19165 | t.l. | | | 2.94 | | 31462 |
| d493 | t.l. | | | 1.16 | | 8832 | t.l. | | | 1.21 | | 12835 | t.l. | | | 2.24 | | 20014 |
| att532 | 9026.86 | 253 | 9897 | 2.49 | 2.16 | 3536 | t.l. | | | 1.42 | | 8213 | t.l. | | | 4.07 | | 15535 |

TAB. 3.5 – Statistics on TSP generation 2 instances

| $\alpha$ | 0.25 | | | | | | 0.50 | | | | | | 0.75 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | TIME | NDS | CUTS | GAP | %UB | OPT | TIME | NDS | CUTS | GAP | %UB | OPT | TIME | NDS | CUTS | GAP | %UB | OPT |
| st70 | 13.05 | 47 | 949 | 15.67 | 8.76 | 198 | 7.71 | 24 | 661 | 6.19 | 4.64 | 308 | 20.62 | 41 | 1015 | 1.82 | 1.14 | 434 |
| eil76 | 32.68 | 175 | 960 | 19.86 | 10.64 | 126 | 9.05 | 24 | 703 | 4.27 | 3.32 | 204 | 1.71 | 7 | 161 | 2.8 | 2.8 | 312 |
| pr76 | 1121.31 | 7385 | 1802 | 21.3 | 9.72 | 29811 | 24.02 | 159 | 867 | 1.11 | 0.25 | 42200 | 147.24 | 711 | 1947 | 0.63 | 0.0 | 63348 |
| gr96 | 137.2 | 347 | 2051 | 13.31 | 9.5 | 15384 | 50.48 | 99 | 1669 | 8.62 | 8.14 | 22491 | 272.77 | 717 | 2078 | 4.29 | 3.5 | 31844 |
| rat99 | 922.85 | 1281 | 3332 | 25.21 | 15.04 | 401 | 55.05 | 101 | 1412 | 8.17 | 5.39 | 579 | 37.84 | 45 | 955 | 0.37 | 0.12 | 808 |
| kroA100 | 254.08 | 623 | 1811 | 22.07 | 9.44 | 5803 | 17.47 | 21 | 1205 | 1.97 | 1.4 | 8325 | 117.14 | 255 | 1565 | 8.94 | 7.7 | 13241 |
| kroB100 | 200.34 | 265 | 2661 | 30.59 | 11.07 | 6434 | 42.17 | 101 | 1142 | 4.31 | 3.68 | 8768 | 31.02 | 53 | 1089 | 0.19 | 0.08 | 13447 |
| kroC100 | 37.85 | 51 | 1843 | 16.84 | 12.52 | 5714 | 86.21 | 169 | 1622 | 6.44 | 4.61 | 9283 | 179.66 | 563 | 1531 | 1.64 | 1.03 | 13156 |
| kroD100 | 117.75 | 249 | 1958 | 18.92 | 14.44 | 5936 | 57.52 | 167 | 1340 | 7.59 | 5.56 | 8998 | 42.84 | 125 | 944 | 0.24 | 0.0 | 12731 |
| kroE100 | 3376.34 | 4743 | 5081 | 28.54 | 14.66 | 6968 | 41.76 | 37 | 1600 | 12.84 | 8.03 | 9313 | 34.07 | 113 | 1083 | 0.63 | 0.31 | 12740 |
| rd100 | 28.77 | 59 | 1184 | 5.09 | 4.26 | 2068 | 51.72 | 157 | 1157 | 4.69 | 4.06 | 3377 | 72.26 | 165 | 1257 | 3.27 | 2.94 | 4925 |
| eil101 | 12.51 | 19 | 754 | 13.64 | 12.12 | 116 | 17.55 | 41 | 619 | 1.33 | 0.89 | 223 | 4.79 | 11 | 302 | 0.53 | 0.53 | 373 |
| lin105 | t.l. | | | 7.37 | | 4952 | 423.4 | 676 | 2887 | 2.47 | 0.88 | 6547 | 1142.09 | 1879 | 4687 | 3.28 | 2.32 | 9482 |
| pr107 | 86.48 | 159 | 3047 | 8.89 | 4.41 | 15553 | t.l. | | | 0.25 | | 27198 | 6307.42 | 13873 | 1944 | 7.77 | 6.69 | 34064 |
| pr124 | 113.99 | 79 | 2526 | 25.77 | 10.77 | 17479 | 206.93 | 129 | 5114 | 10.22 | 6.6 | 26375 | 278.58 | 113 | 2730 | 5.49 | 3.85 | 37799 |
| bier127 | 53.41 | 115 | 1229 | 11.7 | 10.99 | 20912 | 4654.12 | 13859 | 3399 | 8.41 | 6.86 | 42358 | 1119.78 | 1787 | 3082 | 1.7 | 1.3 | 70777 |
| ch130 | 145.18 | 111 | 2190 | 29.76 | 22.02 | 1381 | 60.85 | 109 | 1309 | 5.0 | 3.88 | 2305 | 17.73 | 17 | 619 | 1.38 | 1.3 | 3719 |
| pr136 | t.l. | | | 1.56 | | 40241 | 4564.78 | 3097 | 3788 | 3.72 | 3.0 | 42179 | 3913.4 | 2535 | 3889 | 1.28 | 0.94 | 60109 |
| gr137 | 232.62 | 249 | 2971 | 13.45 | 12.5 | 21810 | t.l. | | | 2.18 | | 34023 | 83.93 | 51 | 1212 | 0.86 | 0.78 | 44203 |
| pr144 | t.l. | | | 4.44 | | 22462 | t.l. | | | 4.98 | | 30033 | 3369.23 | 2139 | 2797 | 10.87 | 9.71 | 38267 |
| ch150 | 272.98 | 141 | 3375 | 27.65 | 16.63 | 1725 | 132.36 | 32 | 2357 | 1.55 | 1.29 | 2675 | 198.65 | 159 | 2003 | 0.48 | 0.28 | 3980 |
| kroA150 | 254.04 | 103 | 3455 | 15.16 | 7.98 | 6102 | 78.72 | 69 | 1694 | 0.3 | 0.11 | 9409 | t.l. | | | 0.42 | | 15801 |
| kroB150 | 286.07 | 179 | 3001 | 10.89 | 8.15 | 6276 | 256.38 | 95 | 3158 | 8.47 | 7.89 | 10392 | 359.08 | 389 | 2004 | 2.57 | 2.2 | 15520 |
| pr152 | 362.71 | 203 | 4186 | 9.61 | 8.88 | 26410 | t.l. | | | 4.61 | | 40937 | 1798.74 | 937 | 3530 | 14.9 | 14.21 | 51130 |
| u159 | t.l. | | | 0.20 | | 12619 | 328.3 | 95 | 5423 | 3.25 | 2.8 | 17631 | 3066.63 | 1091 | 4343 | 0.71 | 0.46 | 26524 |
| rat195 | t.l. | | | 13.02 | | 722 | 776.12 | 177 | 4534 | 3.69 | 3.01 | 999 | 1223.75 | 613 | 3054 | 6.84 | 6.65 | 1471 |
| d198 | 7423.25 | 3136 | 6765 | 8.66 | 8.31 | 5474 | 1974.04 | 757 | 3531 | 2.24 | 2.11 | 7388 | 4646.09 | 1495 | 5839 | 7.35 | 7.01 | 10241 |
| kroA200 | 1295.99 | 181 | 5658 | 11.84 | 9.08 | 6975 | 803.92 | 229 | 3843 | 0.75 | 0.5 | 11987 | t.l. | | | 2.00 | | 18659 |
| kroB200 | 1554.23 | 167 | 6221 | 20.33 | 10.89 | 6988 | 1398.61 | 230 | 4362 | 1.73 | 1.51 | 10752 | 423.13 | 307 | 2077 | 1.4 | 1.23 | 16920 |
| gr202 | 1831.86 | 1301 | 3744 | 7.67 | 7.22 | 8832 | 5085.5 | 2477 | 6101 | 3.91 | 3.68 | 14377 | t.l. | | | 0.09 | | 21506 |
| ts225 | t.l. | | | 23.57 | | 40971 | t.l. | | | 0.69 | | 53414 | t.l. | | | 1.45 | | 80656 |
| tsp225 | t.l. | | | 12.35 | | 1198 | t.l. | | | 0.12 | | 1649 | 4433.22 | 1151 | 4707 | 6.01 | 5.82 | 2474 |
| pr226 | 1043.21 | 69 | 6407 | 5.11 | 4.62 | 27424 | t.l. | | | 0.77 | | 39091 | 8540.59 | 829 | 10264 | 4.29 | 4.25 | 45451 |
| gr229 | 405.21 | 117 | 4842 | 15.33 | 15.11 | 25258 | 4004.14 | 405 | 6602 | 2.06 | 1.89 | 46791 | t.l. | | | 0.46 | | 76314 |
| gil262 | 753.32 | 30 | 7554 | 9.77 | 9.31 | 594 | 387.02 | 12 | 4379 | 0.62 | 0.41 | 961 | 3848.21 | 675 | 5092 | 1.99 | 1.39 | 1486 |
| pr264 | t.l. | | | 25.21 | | 17932 | t.l. | | | 4.72 | | 23264 | 1310.94 | 87 | 6482 | 0.88 | 0.83 | 29928 |
| a280 | t.l. | | | 14.38 | | 786 | 4324.35 | 185 | 8929 | 1.72 | 1.63 | 1084 | t.l. | | | 0.94 | | 1596 |
| pr299 | t.l. | | | 19.97 | | 16278 | 5129.46 | 189 | 10549 | 1.14 | 1.06 | 20317 | t.l. | | | 1.24 | | 30269 |
| lin318 | t.l. | | | 5.57 | | 10816 | 6867.39 | 831 | 7978 | 1.42 | 1.21 | 16401 | 13249.4 | 683 | 8802 | 4.12 | 4.05 | 25892 |
| rd400 | 11875.9 | 262 | 17326 | 11.56 | 9.33 | 3050 | 2602.41 | 111 | 7750 | 0.49 | 0.38 | 5700 | t.l. | | | 0.58 | | 9181 |
| fl417 | t.l. | | | 11.92 | | 4295 | t.l. | | | 8.59 | | 5740 | t.l. | | | 0.99 | | 6557 |
| gr431 | t.l. | | | 19.93 | | 36760 | t.l. | | | 2.27 | | 56484 | t.l. | | | 2.94 | | 93835 |
| pr439 | t.l. | | | 0.75 | | 22222 | 7880.73 | 87 | 12269 | 6.85 | 6.54 | 35771 | t.l. | | | 6.58 | | 63974 |
| pcb442 | t.l. | | | 12.51 | | 12836 | t.l. | | | 0.46 | | 19632 | t.l. | | | 1.41 | | 30631 |
| d493 | t.l. | | | 0.15 | | 9360 | t.l. | | | 0.02 | | 13480 | t.l. | | | 3.14 | | 21469 |
| att532 | t.l. | | | 24.61 | | 7647 | t.l. | | | 0.94 | | 10258 | t.l. | | | 4.13 | | 17380 |

TAB. 3.6 – Statistics on TSP generation 3 instances

## 3.7 Conclusion

We have adapted inequalities from the knapsack and TS polytopes to implement the first branch-and-cut procedure for the PCTSP. We have also presented separation algorithms for those inequalities as well as a heuristic procedure to find feasible solutions to the PCTSP. Our branch-and-cut algorithm has been tested on problems generated from TSP and VRP instances

of the TSPLIB [88] ranging from 70 to 532 vertices. Our results show that subtour elimination and logical inequalities alone are not sufficient to tackle hard problems. Whereas adding cover inequalities appear to be enough for problems with small minimum collected prize $\bar{p}$ ($\alpha = 0.25$), the comb inequalities should be added when $\bar{p}$ is large ($\alpha = 0.75$). For intermediate values of $\bar{p}$ ($\alpha = 0.50$), both the cover and comb inequalities are required to obtain good results.

## Acknowledgments

Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT). This support is gratefully acknowledged.

# Chapitre 4

# An exact $\epsilon$-constraint method for bi-objective combinatorial optimization problems : Application to the Traveling Salesman Problem with Profits

Ce chapitre aborde le problème du voyageur de commerce avec profits (PVCP) dans un contexte où les préférences sont inconnues au moment de l'optimisation. Notre objectif est donc d'énumérer la frontière de Pareto exacte plutôt que de déterminer une solution optimale. La stratégie que nous avons développée pour y arriver n'est pas valide seulement pour le PVCP. En fait, nous proposons dans ce chapitre une méthode *$\epsilon$-constraint* pour les problèmes d'optimisation combinatoires bi-critères où les deux objectifs sont évalués par des nombres entiers.

Nous faisons évoluer le vecteur $\epsilon$ de telle sorte que la résolution séquentielle des *$\epsilon$-problems* correspondants énumère toujours tous les points sur la frontière de Pareto. Dans le cas du PVCP, cette approche se résume à la résolution d'une séquence de PCTSPs où le profit minimum à

amasser constitue l'unique valeur du vecteur $\epsilon$. Cette séquence débute avec un profit minimum de 0 qui augmente ensuite progressivement jusqu'à ce que tous les sommets du graphe doivent être visités pour satisfaire la contrainte de profit. Nous utilisons l'algorithme de séparation et coupes présenté au chapitre précédent pour résoudre les PCTSP. Bien que cet algorithme soit spécialisé, la méthodologie globale demeure générique et peut être appliquée avec succès à tout problème d'optimisation bi-critère dont les objectifs prennent des valeurs entières et pour lesquels il existe un algorithme performant permettant de résoudre les *$\epsilon$-problems*.

Afin d'améliorer les performances de l'algorithme, nous proposons deux heuristiques permettant de tirer profit des informations générées lors de la résolution d'un *$\epsilon$-problem*. La première de ces heuristiques exploite la connaissance croissante du polytope d'un *$\epsilon$-problem* à l'autre en conservant certaines des inégalités séparées lors d'exécutions précédentes de l'algorithme de séparation et coupes. Cette astuce permet d'éviter l'exécution abusive des algorithmes de séparation et d'améliorer les bornes plus rapidement. Évidemment, cela est possible seulement lorsque les inégalités demeurent valides d'un *$\epsilon$-problem* à l'autre. Nous proposons aussi l'ajout de nouvelles inégalités issues de la solution optimale du sous-problème précédent. La deuxième heuristique sert à améliorer la solution réalisable calculée à la racine de l'arbre de branchement en utilisant la solution optimale du *$\epsilon$-problem* précédent. Comme cette solution de départ permet d'élaguer l'arbre de branchement, plus elle est de qualité, plus la solution optimale risque d'être trouvée rapidement. Nos résultats indiquent une amélioration de plus de 40% du temps de calcul, en moyenne, lorsque ces heuristiques sont utilisées.

Nous avons testé notre algorithme sur les mêmes instances du PVCP qu'au chapitre précédent. Ainsi, nous avons généré les premiers résultats exacts pour le PVCP sur des instances accessibles à tous et déjà utilisées dans la littérature. Nous avons réussi à énumérer la frontière de Pareto exacte de problèmes allant jusqu'à 150 sommets en moins de 72 heures de temps de calcul sur des ordinateurs standards.

Finalement, nous proposons une modification de notre algorithme permettant d'obtenir des solutions approximatives. Il s'agit simplement d'ajouter une tolérance sur l'écart entre la meilleure borne inférieure connue et la valeur de la meilleure solution réalisable connue. Rappelons que cette dernière agit comme borne supérieure sur la valeur optimale. Nos tests sur

le PVCP montrent que l'ajout d'une telle tolérance permet d'obtenir d'excellents résultats, et ce très rapidement. Par exemple, avec une tolérance de 10%, on obtient en moyenne une frontière de Pareto suffisamment proche de la frontière exacte pour que les points de cette dernière soient en moyenne à une distance relative de 2% du point le plus proche sur la frontière approximative, et ce en à peine plus de 1% du temps nécessaire pour calculer la frontière exacte.

# An exact $\epsilon$-constraint method for bi-objective combinatorial optimization problems: application to the Traveling Salesman Problem with Profits

Jean-François Bérubé
Michel Gendreau
Jean-Yves Potvin

Jean-François Bérubé
Michel Gendreau
Jean-Yves Potvin

Département d'informatique et de recherche opérationnelle,
Université de Montréal,
C.P. 6128, succursale Centre-ville,
Montréal, Québec, Canada H3C 3J7

Article soumis à la revue *European Journal of Operational Research*
le 28 juin 2007

## Abstract

This paper describes an exact $\epsilon$-constraint method for bi-objective combinatorial optimization problems with integer objective values. This method tackles multi-objective optimization problems by solving a series of single objective subproblems, where all but one objective are transformed into constraints. We show in this paper that the Pareto front of bi-objective problems can be efficiently generated with the $\epsilon$-constraint method. Furthermore, we describe heuristics based on information gathered from previous subproblems that significantly speed up the method. This approach is used to find the exact Pareto front of the Traveling Salesman Problem with Profits, a variant of the Traveling Salesman Problem in which a profit or prize value is associated with each vertex. The goal here is to visit a subset of vertices while addressing two conflicting objectives: maximize the collected prize and minimize the travel costs. We report the first exact results for this problem on instances derived from classical vehicle routing and traveling salesman problem instances with up to 150 vertices. Results on approximations of the Pareto front obtained from a variant of our exact algorithm are also reported.


**Keywords.** Bi-objective Combinatorial Optimization, $\epsilon$-Constraint Problem, Pareto front, Branch-and-Cut, Traveling Salesman Problem with Profits.

## 4.1 Introduction

Decision making issues can rarely rely on a single well defined criterion. Although the multiple facets of a decision process can be aggregated into a single objective function, this simplification involves arbitrary rules that can hardly capture adequately the complexity of real world decision issues. Thereby, the interest for multi-criteria decision making has continually grown during the last decades, as attested by the number of books and surveys on the topic (see [20, 34, 76, 96], among others). It comes as no surprise if more and more publications address combinatorial issues given that many real world applications involve discrete decisions or events. The reader is referred to [32] for a review of the literature on multi-objective combinatorial optimization (MOCO) problems.

This paper addresses a special case of MOCO problems, namely bi-objective combinatorial optimization (BOCO) problems with integer objective values. BOCO problems are often considered independently of MOCO problems because of their particular nature: going from many to two objectives corresponds to a significant simplification of the problem (*"Three is more than two plus one."* [32]). General BOCO problems are formulated as:

$$\min\ f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x})) \qquad \text{subject to: } \vec{x} \in \mathcal{X} \tag{4.1}$$

where $\mathcal{X}$ is the set of feasible solutions, or *solution space*. We denote each evaluation vector $f(\vec{x})$ as $\vec{z}$ and $(\vec{z})_i$ stands for the value of the $i$th objective. To simplify the notation, we write $z_i$ instead of $(\vec{z})_i$ up to Theorem 4. From there, $(\vec{z})_i$ is used to avoid any ambiguity..

The *objective space* is defined by $\mathcal{Z} = \{\vec{z} = (z_1, z_2) : z_i = f_i(\vec{x}),\ \forall \vec{x} \in \mathcal{X},\ i = 1, 2\}$. Since no solution optimizes simultaneously both objectives, one will search for an acceptable trade-off instead of an optimal solution. This compromise must be such that no strictly better solution exists, even though some solutions might be considered as equivalent. This involves a partial order of the objective space, defined by a *dominance relation*. The latter is used to characterize *Pareto efficiency*, a concept that replaces the optimal solution of single objective optimization problems.

**Definition 1 (Dominance relation).** *Let $\vec{z}$ and $\vec{z}' \in \mathcal{Z}$. We say that $\vec{z}$ dominates $\vec{z}'$ ($\vec{z} \succ \vec{z}'$) if and only if $z_1 \leq z'_1$ and $z_2 \leq z'_2$ where at least one inequality is strict.*

**Definition 2 (Pareto efficiency).** *A solution $\vec{x} \in \mathcal{X}$ is (Pareto) efficient in $\mathcal{X}$, if and only if $\nexists \vec{x}' \in \mathcal{X}$ such that $f(\vec{x}') \succ f(\vec{x})$.*

**Definition 3 (Efficient set).** *The efficient set $\mathcal{E} = \{\vec{x} \in \mathcal{X} : \vec{x}$ is Pareto efficient in $\mathcal{X}\}$.*

**Definition 4 (Pareto front).** *The Pareto front $\mathcal{F} = \{f(\vec{x}) : \vec{x} \in \mathcal{E}\}$.*

The *Efficient set* ($\mathcal{E}$) and *Pareto front* ($\mathcal{F}$) contain all the Pareto efficient solutions and all the non-dominated points in the objective space, respectively. Since the efficient set is defined on the solution space while the Pareto front is defined on the objective space, the cardinality of $\mathcal{E}$ is always greater than or equal to the cardinality of $\mathcal{F}$. That is, there might be many feasible solutions that correspond to the same point in the objective space. Multi-objective optimization can approximate the Pareto front to provide a set of equivalent solutions to the decision maker who will then be aware of many equivalent tradeoffs. This should help him to take a decision. In some special cases, when the size of the efficient set is reasonable, it is even possible to provide the exact Pareto front to the decision maker. This paper addresses that kind of problem.

Among exact methods to find the Pareto front of MOCO problems, weighted sum scalarization is the most popular according to [32]. This method solves different single objective subproblems generated by a linear scalarization of the objectives. By varying the weights of this linear function, all supported* non-dominated points can be found. It is worth noting that the subproblems are as easy to solve as the corresponding mono-criterion problems. On the other hand, linear scalarization cannot find unsupported points and is therefore ill-suited for non-convex objective spaces such as those associated with MOCO problems. This drawback can be overcome with the Two-Phase Method [99] that finds all supported points of $\mathcal{F}$ through a weighted sum scalarization in the first phase, while non-supported points are found during the second phase with problem specific methods. Most algorithms that find the exact Pareto front of MOCO problems are variants of the Two-Phase Method [32], although other parametric approaches based on weighted scalarizations can find the exact Pareto front of BOCO problems [67, 86, 92].

*The supported points are those found on the convex envelope of the objective space.

Besides weighting sum algorithms, the $\epsilon$-constraint method [20, 76] is the best known approach for solving MOCO problems, according to [32]. This method generates single objective subproblems, called $\epsilon$-constraint problems, by transforming all but one objectives into constraints. The upper bounds of these constraints are given by the $\epsilon$-vector and, by varying it, the exact Pareto front can theoretically be generated. In practice, because of the high number of subproblems and the difficulty to establish an efficient variation scheme for the $\epsilon$-vector, this approach has mostly been integrated within heuristic and interactive schemes. It can however generate the exact Pareto front in particular situations, such as BOCO problems, as we will see later.

This paper focuses on BOCO problems for which no polynomial time algorithm exists for solving the corresponding single objective problems, but where the latter can still be efficiently solved through branch-and-cut. Many problems share these characteristics, including the Bi-Objective Covering Tour problem [57] and bi-objective variants of the Traveling Salesman Problem (TSP), such as the Bi-Objective Traveling Purchaser Problem [89] and the Traveling Salesman Problem with Profits (TSPP) [36]. For these problems, $\epsilon$-constraint methods are particularly attractive because the addition of new constraints through a branch-and-cut procedure is quite natural.

The first contribution of this paper is to show the correctness of an efficient variant of the $\epsilon$-constraint method for BOCO problems, where exactly one $\epsilon$-constraint problem is solved for each point on the Pareto front. The second contribution is the introduction of heuristic improvements based on the exploitation of information gathered from previous problems that provides significant speed-ups. The proposed method is then used to solve instances of the TSPP, a variation of the TSP in which a profit or prize value is associated with each vertex. We report the first exact Pareto fronts for TSPP instances obtained from classical VRP and TSP instances available in the TSPLIB [88].

The paper is organized as follows. Our general problem-solving approach is presented in Section 1. Then, Section 2 introduces the improvement heuristics. Section 3 describes the TSPP and explains how our general algorithm can be adapted to solve it. Finally, Section 4 reports computational results on several TSPP instances.

## 4.2 Exact $\epsilon$-constraint method for BOCO problems

The $\epsilon$-constraint method has been developed for general multi-objective problems. It solves $\epsilon$-constraint problems $P_k(\epsilon)$ obtained by transforming one of the objectives into a constraint. For the bi-objective case, the problems $P_1(\epsilon_2)$ and $P_2(\epsilon_1)$ are:

$$\min f_1(\vec{x}) \tag{4.2}$$

$$\text{subject to:} \quad \vec{x} \in \mathcal{X}, \tag{4.3}$$

$$f_2(\vec{x}) \leq \epsilon_2, \tag{4.4}$$

and

$$\min f_2(\vec{x}) \tag{4.5}$$

$$\text{subject to:} \quad \vec{x} \in \mathcal{X}, \tag{4.6}$$

$$f_1(\vec{x}) \leq \epsilon_1, \tag{4.7}$$

respectively.

**Theorem 2.** *$\vec{x}^*$ is an efficient solution of a BOCO problem if and only if $\exists\ \epsilon_2$ such that $\vec{x}^*$ solves $P_1(\epsilon_2)$ or $\exists\ \epsilon_1$ such that $\vec{x}^*$ solves $P_2(\epsilon_1)$.*

**Theorem 3.** *If $\vec{x}^*$ solves $P_1(\epsilon_2)$ or $P_2(\epsilon_1)$ and if this solution is unique, then $\vec{x}^*$ is an efficient solution of a BOCO problem.*

Theorems 2 and 3 are proved for general multi-objective problems (see [20, 76]) and are therefore valid for BOCO problems. These theorems mean that efficient solutions can always be found by solving $\epsilon$-constraint problems, as long as $\epsilon_2$ is such that $P_1(\epsilon_2)$ is feasible or $\epsilon_1$ is such that $P_2(\epsilon_1)$ is feasible. Moreover, Theorem 2 indicates that for any efficient solution $\vec{x}^*$, one can find an $\epsilon_j$ such that $\vec{x}^*$ solves $P_1(\epsilon_j)$ or $P_2(\epsilon_j)$. In other words, the exact Pareto front can be found by solving $\epsilon$-constraint problems, as long as we know how to modify $\epsilon_j$ to generate at least one solution for every point of $\mathcal{F}$. This issue has recently been addressed for the general multi-objective case in [69], but it remains an important drawback of the $\epsilon$-constraint method. However, the particularities of BOCO problems yield to a simple variation

scheme for $\epsilon$ that can be numerically implemented. The idea is to construct a sequence of $\epsilon$-constraint problems based on a progressive reduction of $\epsilon_j$. Let $\vec{z}^{\,I} = (z_1^I, z_2^I)$ with

$$z_1^I = \min_{\vec{z} \in \mathcal{Z}} z_1 \quad \text{and} \quad z_2^I = \min_{\vec{z} \in \mathcal{Z}} z_2, \tag{4.8}$$

be the ideal point and let $\vec{z}^{\,N} = (z_1^N, z_2^N)$ with

$$z_1^N = \min_{\vec{z} \in \mathcal{Z}} \{z_1 : z_2 = z_2^I\} \quad \text{and} \quad z_2^N = \min_{\vec{z} \in \mathcal{Z}} \{z_2 : z_1 = z_1^I\} \tag{4.9}$$

be the Nadir point that defines lower and upper bounds on the value of efficient solutions, respectively. Algorithm 3 finds the Pareto front of BOCO problems with integer objective values through a sequence of $\epsilon$-constraint problems. Throughout the algorithm, $\epsilon_j$ is decreased

**Algorithm 3** : Exact Pareto front of BOCO problems with integer objective values.

1. Set $i = 1$, $j = 2$ or $i = 2$, $j = 1$.
2. Compute the ideal and Nadir points.
3. Set $\mathcal{F} = \{(z_i^I, z_j^N)\}$ and $\epsilon_j = z_j^N - \Delta$ ($\Delta = 1$).
4. While $\epsilon_j \geq z_j^I$, do:
   (a) Solve $P_i(\epsilon_j)$ through branch-and-cut and add the optimal solution value $(z_i^*, z_j^*)$to $\mathcal{F}$.
   (b) Set $\epsilon_j = z_j^* - \Delta$.
5. Remove dominated points from $\mathcal{F}$ if required (as explained later, some dominated points might be found throughout this procedure).

by a constant value $\Delta$ (currently set to 1). As explained later, $\Delta$ may sometimes be larger to strengthen the $\epsilon$-constraint. Note that a similar approach is used without proof in [57] for the Bi-Objective Covering Tour problem.

This strategy is somehow related to ranking methods, another exact problem-solving scheme for bi-objective problems. The idea of ranking methods is to start from a feasible solution $\vec{x}$ such that $f_1(\vec{x}) = z_1^I$ (or $f_2(\vec{x}) = z_2^I$) and to find the second best, third best, ..., feasible solutions based on the first (or second) objective, until the Nadir point in reached. Among the

resulting solutions, there is a set of efficient solutions that represent all points on the Pareto front. This approach has been introduced in the early 80's to solve the bi-objective shortest path problem [22]. It relies on $k$-best algorithms [70], which have been developed for many problems such as the shortest path and minimum spanning tree problems. To prove the correctness of Algorithm 3, we first state two lemmas similar to those in [22]. However, the latter are specific to the shortest path problem while our discussion takes place in the context of general BOCO problems.

**Lemma 1.** *$(z_1^I, z_2^N) \in \mathcal{F}$ and $(z_1^N, z_2^I) \in \mathcal{F}$.*

*Proof.* Suppose that $(z_1^I, z_2^N) \notin \mathcal{F}$. Then, $\exists (z_1, z_2) \in \mathcal{Z} : (z_1, z_2) \succ (z_1^I, z_2^N)$. Thus according to definition 1, either:

1) $z_1 < z_1^I$ and $z_2 < z_2^N$ or

2) $z_1 < z_1^I$ and $z_2 = z_2^N$ or

3) $z_1 = z_1^I$ and $z_2 < z_2^N$.

Since 1) and 2) contradict the definition of an ideal point and because 3) contradicts the definition of a Nadir point, then $(z_1^I, z_2^N) \in \mathcal{F}$. The proof that $(z_1^N, z_2^I) \in \mathcal{F}$ is similar. □

**Lemma 2.** *$\forall (z_1, z_2) \in \mathcal{Z}$, if $(z_1, z_2) \in \mathcal{F}$, then $z_1^I \leq z_1 \leq z_1^N$ and $z_2^I \leq z_2 \leq z_2^N$.*

*Proof.* By Lemma 1, $(z_1^I, z_2^N) \in \mathcal{F}$, thus it is non-dominated. Since $z_1^I = \min_{z \in \mathcal{Z}} z_1$, $z_1 \geq z_1^I$, $\forall (z_1, z_2) \in \mathcal{F}$. Also, if $z_2 > z_2^N$, $(z_1^I, z_2^N) \succ (z_1, z_2)$ and $(z_1, z_2) \notin \mathcal{F}$. Hence, $z_1 \geq z_1^I$ and $z_2 \leq z_2^N$, $\forall (z_1, z_2) \in \mathcal{F}$. The proofs for $z_2 \geq z_2^I$ and $z_1 \leq z_1^N$ are similar. □

Lemma 2 defines a region of the objective space that contains the Pareto front. Let us define the set

$$\mathcal{Z}^+ = \{(z_1, z_2) \in \mathcal{Z} : z_1^I \leq z_1 \leq z_1^N \text{ and } z_2^I \leq z_2 \leq z_2^N\} \tag{4.10}$$

which is depicted in Figure 4.1 by the rectangular region formed by the ideal and Nadir points. This region can be used to characterize the interval of possible values for $\epsilon_j$, that is $[z_j^I, z_j^N]$,

$j = 1, 2$. Let $\epsilon_j^s$ be the $s$th value on this interval and let us define the following subsets of $\mathcal{Z}^+$

$$Z_{\epsilon_j^s}^+ = \{\vec{z} \in \mathcal{Z}^+ : z_i = f_i(\vec{x}^*) \text{ where } \vec{x}^* \text{ is a solution of } P_i(\epsilon_j^s)\}. \tag{4.11}$$

Figure 4.1 (a) depicts a typical point $\vec{z}^* = a$ that minimizes the second objective among the points of $\mathcal{Z}_{\epsilon_2^s}^+ = \{a, b, c, d\}$ and shows the preferred and dominated regions according to this point (note that the preferred region is empty). If $\vec{z}^* \in \mathcal{F}$, the other points of the Pareto front must be in the two unidentified regions of $\mathcal{Z}^+$. The correctness of Algorithm 3 is shown by the following theorem.

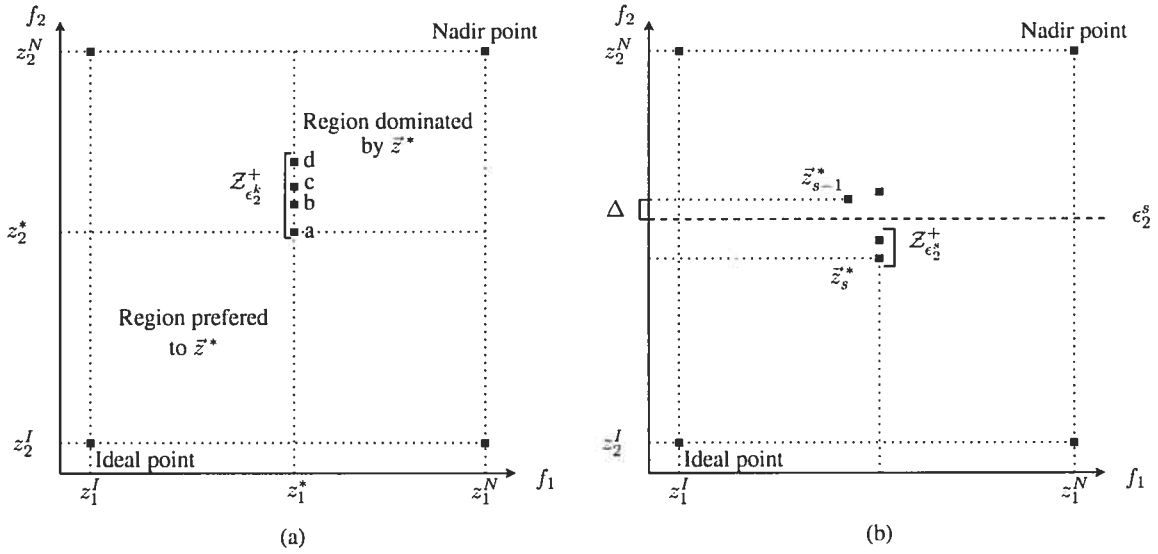


Figure 4.1: (a) Illustration of the dominance relation among elements of $\mathcal{Z}^+$. (b) Illustration of two consecutive points in the sequence defined by Theorem 4.

**Theorem 4.** *A sequence of $\epsilon$-constraint problems $P_i(\epsilon_j)$ defined by $\epsilon_j^1, ..., \epsilon_j^s, ..., \epsilon_j^S$ where:*

*(a)* $\epsilon_j^1 = z_j^N$, $\epsilon_j^S = z_j^I$,

*(b)* $\epsilon_j^s = (\vec{z}_{s-1}^*)_j - \Delta$, *with* $\vec{z}_{s-1}^*$ *the value of a solution to* $P_i(\epsilon_j^{s-1})$ *and* $\Delta = 1$,

*generates one feasible solution for each point of the Pareto front.*

*Proof.* Let $\vec{z}_1^*, ..., \vec{z}_s^*, ..., \vec{z}_S^*$ be the sequence of solutions corresponding to the sequence of $\epsilon$-constraint problems defined by (a) and (b). Let us show that if $\vec{z} \in \mathcal{Z} \setminus \{\vec{z}_1^*, ..., \vec{z}_s^*, ..., \vec{z}_S^*\}$,

then $\vec{z} \notin \mathcal{F}$. Assume that there is a solution $\vec{z}' \in \mathcal{Z} \setminus \{\vec{z}_1^*, ..., \vec{z}_s^*, ..., \vec{z}_{|\mathcal{F}|}^*\}$ such that $\vec{z}' \in \mathcal{F}$. By Lemma 2, $z_i^I \leq z_i' \leq z_i^N$. Then, either:

1) $z_i' = (z_s^*)_i$ (for a given $s$, $s = 1, ..., S$), or

2) $(\vec{z}_{s-1}^*)_i < z_i' < (\vec{z}_s^*)_i$ and $(\vec{z}_{s-1}^*)_j < z_j' \leq (\vec{z}_s^*)_j$ (for a given $s$, $s = 1, ..., S$).

In case 1), $z_j'$ must be lower than $(\vec{z}_s^*)_j$ for $\vec{z}'$ to be efficient. But since $\Delta$ equals 1 and since the objective values are integers, $\epsilon_j$ will eventually reach a value for which the optimum of the corresponding $\epsilon_j$-constraint problem is $\vec{z}'$, that is $\vec{z}' \in \{\vec{z}_{s+1}^*, ..., \vec{z}_S^*\}$, which contradicts the hypothesis. Case 2) is impossible because $\vec{z}_s^*$ is the optimal value of $P_i(\epsilon_j^{s-1} - \Delta)$, with $\Delta = 1$ and integer objectives.

□

***Handling the dominated points.*** Because there might exist many solutions to $P_i(\epsilon_j^s)$ with different values for objective $j$ (i.e. $|Z_{\epsilon_j^s}^+| > 1$), some dominated points might be generated by the sequence of $\epsilon$-constraint problems defined by Theorem 4. For example, in Figure 4.1(a), the points $b$, $c$ and $d$ are dominated by point $a$. Nevertheless, since all non-dominated points will be found, one can simply exclude the non-efficient solutions to obtain the exact Pareto front, as it is done with the $k$-best solutions obtained from ranking methods. Another possibility is to solve both $P_1(\epsilon_2^s)$ and $P_2(\epsilon_1^s)$ (see Theorem 2). This can be done implicitly by modifying the branch-and-cut algorithm used to solve the $\epsilon$-constraint problems. Let $\vec{z}_s^*$ be the optimal value for $P_i(\epsilon_j^s)$. Then, the lower bound of all pending nodes in the branching tree are greater than or equal to $(\vec{z}_s^*)_i$, and other optimal solutions might be found by processing the pending nodes with a lower bound equal to $(\vec{z}_s^*)_i$. Moreover, since a feasible solution such that $(\vec{z}_s^*)_j \leq \epsilon_j^s$ is known, $\epsilon_j^s$ can be decreased to $(\vec{z}_s^*)_j - \Delta$. Doing so until no more feasible solution exists will lead to a unique optimal solution for the strengthened $\epsilon$-constraint problem, which satisfies Theorem 3.. This strategy reduces the number of subproblems to the exact number of points on the Pareto front, but those problems might be harder to solve.

***The efficient set.*** A similar modification of the branch-and-cut algorithm produces the efficient

set in addition to the Pareto front. The idea is to fathom only the nodes with a lower bound strictly greater than the value of the best known feasible solution. The value of $\epsilon_j^s$ should be updated to the maximum of $(\vec{z}^*)_j$ and $\epsilon_j$, for each optimal value $\vec{z}_s^*$ found. Moreover, inequalities that cuts all known optimal solutions must be added.

***Approximation.*** One should also observe that the algorithm can be modified to generate approximations of the Pareto front while reducing the computation time. Since the branch-and-cut procedure runs until the gap between the best known feasible solution value (upper bound) and the best linear relaxation value (lower bound) is reached, a tolerance on this gap will produce approximate solutions for the $\epsilon$-constraint problems and thus generate an approximate Pareto front. As shown in Section 4, this strategy significantly reduces the computation time while providing good approximations of the Pareto front.

***Strengthening the $\epsilon$-constraint.*** Until now, we have considered that the constant $\Delta$ equals 1, but in some cases, it might be possible to use a larger value, which obviously strengthens the $\epsilon$-constraint. We show that $\Delta$ can be set to the greatest common divisor among the possible values of objective $j$ ($\Omega_j$), assuming integer objective values.

**Proposition 1.** *$\forall \vec{z}_1, \vec{z}_2 \in \mathcal{Z}$ such that $(\vec{z}_1)_j \neq (\vec{z}_2)_j$, $|(\vec{z}_1)_j - (\vec{z}_2)_j| \geq \Omega_j$*

*Proof.* Suppose there exist some points $\vec{z}_1$ and $\vec{z}_2 \in \mathcal{Z}$ such that $|(\vec{z}_1)_j - (\vec{z}_2)_j| < \Omega_j$, then

$$\frac{|(\vec{z}_1)_j - (\vec{z}_2)_j|}{\Omega_j} < 1 \tag{4.12}$$

Since objective values are positive integers, the left-hand side of equation (4.12) must be null. Therefore, $(\vec{z}_1)_j = (\vec{z}_2)_j$ which contradicts the hypothesis. □

## 4.3 Improvement Heuristics

The sequence of $\epsilon$-constraint problems is defined by a progressive reduction of $\epsilon_j$ that leads to a progressive increase in objective $i$. This suggests that the structure of consecutive subproblems

might be similar. In the following, two classes of heuristics are proposed to take advantage of these similarities in order to improve the branch-and-cut algorithm when solving $P_i(\epsilon_j)$. Both heuristics exploit information gathered from previous subproblems to solve future subproblems faster. The first heuristic exploits knowledge of the polytope, while the second one improves the quality of the initial solution. Later on, we will explain how those heuristics can be applied in the case of the TSPP.

***Exploiting knowledge of the polytope.*** The general principle underlying cutting-plane algorithms is to reduce the size of the relaxed solution space by adding valid cuts. Due to structural similarities between two consecutive $\epsilon$-constraint problems, and since separation of violated inequalities is often hard, it is quite natural to keep some constraints from one $P_i(\epsilon_j)$ problem to the next, as long as these constraints remain valid. This can be implemented quite easily when the branch-and-cut algorithm already maintains a cut-pool. Typically, the latter contains inequalities that have been removed to reduce the size of the model. Since previously removed cuts can be violated again later on, they are kept in a pool which is explored at the beginning of the separation phase... We thus suggest to initialize the cut-pool of $P_i(\epsilon_j^s)$ with the active cuts at the optimum of $P_i(\epsilon_j^{s-1})$ (another way to manage a cut-pool is proposed in [89] for the Bi-objective Traveling Purchaser problem). Knowledge of the polytope can also be exploited to generate valid inequalities. In the next section, we describe an inequality for the TSPP based on the optimal solution of the previous problem.

***Improving the initial feasible solution*** . Similarities between consecutive solutions can be exploited by the heuristic that generates the initial solution. A better initial solution increases the upper bound on the optimal solution value and thus allows to prune branches early and reduce the number of explored nodes.

## 4.4 The Traveling Salesman Problem with Profits

To empirically validate our $\epsilon$-constraint method for BOCO problems, we applied it to the TSPP. This section describes the problem and explains how our $\epsilon$-constraint method can solve it.

### 4.4.1 Problem description

Among the multiple variants of the TSP [49], the TSPP belongs to the *selective TSP* class where a feasible solution is not required to visit all vertices. We use the classification in [36], where variants of the selective TSP in which values are associated with vertices are considered to be TSPP. The latter is a BOCO problem where two strongly conflicting objectives must be optimized. Namely, one must find a Hamiltonian cycle over a subset of vertices such that the collected prize is maximized while the travel cost is minimized. The prize collection maximization implies that the traveler should visit a large number of vertices, while the cost minimization has the opposite effect.

The scope of our discussion will be restricted to the undirected TSPP which can be mathematically formulated as follows. Let $G = (V, E)$ be an undirected complete graph, with edge set $E$ and vertex set $V$, among which vertex 1 stands for the depot. The nonnegative integer prizes are denoted $p_v$ for each $v \in V$ ($p_1 = 0$). For every $e \in E$, the nonnegative integer travel cost $c_e$ satisfies the triangle inequality. We define $E(S) = \{(u, v) \in E : u \in S, v \in S\}$ and $\delta(S) = \{(u, v) \in E : u \in S, v \notin S\}$ for $S \subset V$, and $V(T) = \{v \in V : T \cap \delta(\{v\}) \neq \emptyset\}$ for $T \subseteq E$. We also define, $V' = V \setminus \{1\}$ and for each $v \in V$, we write $\delta(v)$ instead of $\delta(\{v\})$. The decision variables are :

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

$$y_v = \begin{cases} 1 & \text{if vertex } v \text{ is visited} \\ 0 & \text{otherwise} \end{cases}$$

Finally, we define $y(S) = \sum_{v \in S} y_v$ and $p(S) = \sum_{v \in S} p_v$ for $S \subseteq V$, and $x(T) = \sum_{e \in T} x_e$

and $c(T) = \sum_{e \in T} c_e$ for $T \subseteq E$. The TSPP can then be formulated as the following 0-1 integer linear program (LP):

$$\max \sum_{v \in V} p_v y_v \tag{4.13}$$

$$\min \sum_{e \in E} c_e x_e \tag{4.14}$$

subject to:

$$x(\delta(v)) = 2y_v \qquad (\forall\, v \in V) \tag{4.15}$$

$$x(\delta(S)) \geq 2y_v \qquad (\forall\, S \subset V :\ 1 \in S,\ v \in V \setminus S) \tag{4.16}$$

$$y_1 = 1 \tag{4.17}$$

$$x_e \in \{0,1\} \qquad (\forall\, e \in E) \tag{4.18}$$

$$y_v \in \{0,1\} \qquad (\forall\, v \in V') \tag{4.19}$$

The degree constraints (4.15) insure that a feasible solution goes exactly once through each visited vertex. The subtour elimination constraints (SEC) (4.16) require that each visited vertex $v \in V'$ of a feasible solution be reachable from the depot by two edge-disjoint paths. Constraint (4.17) forces the depot to be visited and constraints (4.18) and (4.19) impose that all variables be 0-1. Note that the model forces all feasible solutions to visit at least 3 vertices. Since solutions with less than 3 vertices can be easily found by explicit enumeration, we assume, without loss of generality, that optimal solutions contain at least 3 vertices.

In spite of its bi-objective nature, the literature focuses on mono-criterion variants of the TSPP. Three variants have been extensively studied up to now: the Profitable Tour Problem (PTP), the Orienteering Problem (OP) and the Prize Collecting TSP (PCTSP). The PTP, introduced in [27], maximizes the difference between the collected prizes and the travel cost; it is also known as the Simple Cycle Problem [39]. In the OP, one must find a tour that maximizes the total collected prize while maintaining the traveling cost under a fixed value. It has been introduced in a study on orienteering competitions [97] and it is also known as the Selective TSP [68] and as the Maximum Collection Problem [60]. Finally, the PCTSP was introduced as a model for scheduling the daily operations of a steel rolling mill [6]. Given an undirected graph

with edge costs and node prizes, the aim of the PCTSP is to find a simple cycle minimizing the total edge cost while collecting a minimum total prize. The PCTSP is also known as the Quota TSP [4].

The bi-criteria nature of the TSPP has been considered in [64] where the efficient set is approximated for problems with less than 25 vertices. This algorithm dates back to 1988 and could be used to solve much larger problems nowadays, but it remains a heuristic or approximate method. To the best of our knowledge, the bi-objective TSPP has never been solved exactly. However, there is some literature on exact algorithms for mono-criterion variants of the TSPP which are mostly adaptations of branch-and-bound procedures developed for the TSP (see [36] for a complete survey).

### 4.4.2 Finding the exact Pareto front of the TSPP

The TSPP, as defined by equations (4.13) to (4.19), is a BOCO problem where both objectives take integer values. Theorem 4 shows how to apply the $\epsilon$-constraint method to find the exact Pareto front of such problems. One must first decide which of the cost minimization or the collected prize maximization should be kept in the objective. In other words, one should decide if the $\epsilon$-constraint problems will be OPs (collected prize maximization) or PCTSPs (cost minimization). In this work, the PCTSP was chosen for a number of reasons. First, an algorithm for solving the PCTSP was already available to us. Second, the implementation of the $\epsilon$-constraint method is quite easy in this case. The collected prize is set to 0 and is then gradually increased up to a maximum value, which is the summation of the price values over all vertices. In the case of the OP, one must start with a cost that corresponds to the optimal solution value of a TSP. Finally, when the greatest common divisor among all prize values is greater than 1, the PCTSP formulation allows one to set $\Delta$ to a value greater than one (see Proposition 1). By doing so, the minimum prize constraint is strengthened by forcing its upper bound ($\epsilon$) to be as close as possible to the collected prize of the optimal PCTSP solution. In the case of the OP formulation, a $\Delta$ value greater than one can hardly be considered, as it requires the calculation of the greatest common divisor among all possible Hamiltonian cycle costs.

The mathematical formulation of the PCTSP is similar to the one for the TSPP (equations

4.13 to 4.19), except for the first objective (4.13) which is replaced by the minimum collected prize constraint:

$$\sum_{v \in V} y_v p_v \geq \bar{p} \tag{4.20}$$

where $p_v$ is the prize associated with vertex $v$ and $\bar{p}$ is a constant corresponding to the minimum prize to be collected. Equation (4.20) can also be formulated as

$$\sum_{v \in V} p_v(1 - y_v) \leq U \tag{4.21}$$

where $U = p(V) - \bar{p}$. This corresponds to the $\epsilon$-constraint of $P_i(\epsilon_j)$, where $\epsilon_j$ is represented by $U$. The sequence of PCTSPs starts with the worst $\bar{p}$ value ($\bar{p} = 0$). The latter is then progressively increased until the largest possible value ($\bar{p} = p(V)$) is reached. Note that when $\bar{p}$ is null, the optimal PCTSP solution stays at the depot. This zero cost solution corresponds to the point $(c^I, p^N)$. On the other hand, the point $(c^N, p^I)$ is associated with the PCTSP with $\bar{p} = p(V)$, which is a classical TSP.

The PCTSPs are efficiently solved through the branch-and-cut procedure described in [15]. The latter has been modified to generate a solution with the greatest collected prize among the optimal solutions of $P_i(\epsilon_j)$, as explained under *Handling dominated points* in Section 1. Thereby, the number of PCTSPs to be solved is limited to the number of points on the Pareto front. Our experiments show that this actually reduces the number of subproblems by 10% on average. However, because those subproblems are harder to solve, we observe an increase of 159% in computation time, on average. We thus decided to use the original version of the branch-and-cut algorithm, since it is more efficient overall.

From all the valid PCTSP inequalities used in this algorithm (see Appendix 4.7), only the cost-cover and conditional inequalities are not guaranteed to be valid for any $\epsilon_j$. Both cuts are based on a set of vertices or on a cycle that leads to solutions with a cost greater than the best known feasible solution value. Since the sequence of $P_i(\epsilon_j)$ problems is such that the solution values increase, the upper bound for a subproblem is no longer valid for the next subproblems. That is, we do not separate the cost-cover and conditional inequalities. The lifted-cover inequalities are still valid in subsequent subproblems, although they lose their strength because the cover might not remain minimal. Cycle-cover inequalities also remain

valid because the minimum collected prize increases from one subproblem to the next. Simple comb, 2-matching and logical inequalities are clearly valid for the whole sequence of PCTSPs.

The computation time of the branch-and-cut algorithm that solves the PCTSPs can be substantially reduced by using the heuristic improvements presented in Section 2. In the following, we show how those heuristics have been implemented.

*Exploiting knowledge of the polytope.* In addition to reusing previously separated inequalities, the knowledge of the polytope can be improved by adding the following inequality. Since the minimum collected prize increases from one subproblem to the next, at least one unvisited vertex in $P_i(\epsilon_j^s)$ must be visited in $P_i(\epsilon_j^{s+1})$. Hence, we have the following valid *visit inequality*:

$$y(V \setminus V_s^*) \geq 1 \tag{4.22}$$

where $V_s^*$ is the set of visited vertices in an optimal solution of $P_i(\epsilon_j^s)$. Such an inequality is added to the LP of each new subproblem.

*Improving the initial feasible solution* . A good feasible solution can be obtained through a modification to the optimal solution of the previous $\epsilon$-constraint problem. Adding any vertex to the optimal solution of $P_i(\epsilon_j^{s-1})$ actually produces a feasible solution to $P_i(\epsilon_j^s)$. Let $\vec{x}_{s-1}^*$ be an optimal solution to $P_i(\epsilon_j^{s-1})$. Then, the following heuristic procedure generates a feasible solution $\tilde{x}_s$ to $P_i(\epsilon_j^s)$ :

1. Compute a feasible solution $\tilde{x}_s$ with the heuristic algorithm used in [15] to find an initial feasible solution.
2. For every vertex $v \in V \setminus V_{s-1}^*$ :
   a) Construct solution $\hat{x}_s$ by inserting $v$ in $\vec{x}_{s-1}^*$ at a location that minimizes the detour.
   b) If $c(\tilde{x}_s) > c(\hat{x}_s)$, set $\tilde{x}_s = \hat{x}_s$.

## 4.5 Computational results

We transformed VRP and TSP instances of the TSPLIB [88] into TSPP instances using the rules provided in [15] and [38]. We considered instances for which the node coordinates were available. For VRP instances, the demands are interpreted as the node prizes. For TSP instances, the prizes $p_v$ $(v \in V')$ are generated in three different ways:

- Generation 1: $p_v = 1$;
- Generation 2: $p_v = 1 + (7141v + 73) \bmod 100$;
- Generation 3: $p_v = 1 + \lfloor 99\frac{c_{(1,v)}}{\theta} \rfloor$, where $\theta = \max_{w \in V'} c_{1,w}$.

Instances of generation 1 are in general easy problems since all prizes are the same and fixed to 1. Generation 2 produces instances with pseudo-random prizes between 1 and 100, while generation 3 produces hard problems where larger prizes are associated with vertices that are further from the depot. The greatest common divisor among the prize values is equal to 1 in all those instances, except for the VRP instances eil22, eil30 and eil33 (see Table 4.2), where it is equal to 100, 25 and 10, respectively. The algorithm was implemented in C++ and was run on a AMD Opteron 2.4 Ghz processor. The LPs were solved using CPLEX 9.3.

This section starts with the performance analysis of our improvement heuristics before showing results for all solved instances. Finally, we analyse the quality of the approximate Pareto fronts obtained, as explained in Section 1.

### 4.5.1 Performance of the improvement heuristics

Table 4.1 shows data on the performance of the improvement heuristics on a sample of instances of different sizes (the size is at the end of each instance identifier). The computation times, in seconds, of the standard algorithm without any improvement heuristic are given in column *STD*. The three next columns give the relative improvement (in percent) of the computation time due to each of the three improvement heuristics:

- *ACH*: keep active cuts for subsequent subproblems,

- *VSI*: visit inequalities,
- *ISH*: initial solution heuristic.

The columns *ALL* and *AIH* give the computation time and the relative improvement, respectively, when all three improvement heuristics are enabled. For each instance, the best improvement is identified in bold face.

| Instance | Type | STD | ACH | VSI | ISH | ALL | AIH |
|---|---|---|---|---|---|---|---|
| eil33 | vrp | 85.41 | 24.39 | 3.47 | 11.57 | 54.89 | **35.73** |
| eilA76 | vrp | 4528.03 | 20.88 | 12.04 | 23.14 | 2605.27 | **42.46** |
| att48 | tspp1 | 19.08 | 46.54 | -1.00 | 2.31 | 9.82 | **48.53** |
| eil76 | tspp1 | 105.43 | 52.74 | -0.16 | 24.92 | 37.19 | **64.73** |
| rd100 | tspp1 | 829.61 | 53.61 | -0.85 | 7.32 | 365.53 | **55.94** |
| pr144 | tspp1 | 40002.63 | 50.63 | 8.06 | 15.47 | 17481.92 | **56.3** |
| ch150 | tspp1 | 10362.8 | 46.65 | 1.51 | 16.66 | 3706.72 | **64.23** |
| ulysses22 | tspp2 | 21.65 | 30.02 | 11.69 | 3.19 | 13.6 | **37.18** |
| att48 | tspp2 | 1215.76 | 15.14 | 2.08 | 2.00 | 866.16 | **28.76** |
| berlin52 | tspp2 | 1386.49 | **20.21** | 2.47 | 6.54 | 1321.63 | 4.68 |
| eil76 | tspp2 | 6508.44 | 18.26 | 2.47 | 8.3 | 5158.31 | **20.74** |
| rd100 | tspp2 | 79563.53 | 41.23 | -0.81 | 6.88 | 43100.31 | **45.83** |
| ulysses22 | tspp3 | 21.27 | 28.40 | 8.04 | 1.41 | 14.04 | **33.99** |
| att48 | tspp3 | 1788.56 | 14.66 | -3.01 | 1.67 | 1334.41 | **25.39** |
| berlin52 | tspp3 | 2276.94 | 24.77 | 0.79 | 1.88 | 1292.27 | **43.25** |
| st70 | tspp3 | 17528.77 | 38.02 | 0.17 | 5.75 | 9258.1 | **47.18** |
| eil76 | tspp3 | 7986.51 | **40.52** | 4.14 | 17.1 | 4971.95 | 37.75 |
| Averages | | | 33.33 | 3.01 | 9.18 | | **40.75** |

Table 4.1: Performance of the improvement heuristics

The results show that keeping the active inequalities from one PCTSP to the next significantly reduce the computation time (33% in average). We have no strong evidence that the visit inequalities reduce the computation time, although a small improvement is observed on average. On 5 instances, however, the introduction of visit inequalities slightly increases the computation time. Since the improvement is sometimes around 10% while the computation times never increase by more than 3%, we decided to keep them for the exhaustive tests reported later. Using the solution of the previous PCTSP to find a feasible initial solution for the next subproblem improves the computation time by 1.4% to 24.9% (9.18% in average). Combining the three heuristics almost always give the best results. Two exceptions are reported in Table 4.1 where one should have used only the *ACH* heuristic.

### 4.5.2 Results for exact Pareto fronts

Tables 4.2, 4.3, 4.4, and 4.5 show results for VRP instances and TSP generation 1, 2 and 3 instances, respectively. There are fewer instances in Tables 4.4 and 4.5, when compared with Table 4.3, because TSP generation 2 and 3 instances are more difficult to solve than generation 1 instances. The reported results clearly show that the largest generation 2 and 3 instances cannot be addressed within the time limit. We have thus decided to put aside instances with more than 130 vertices (8 instances). The three improvement heuristics were enabled in all cases. The columns correspond to:

- TIME: the total computation time, in seconds;
- $|\mathcal{F}|$: the size of the Pareto front;
- N: the number of $\epsilon$-constraint problems solved[†];
- $\bar{t}$: the average computation time of the $\epsilon$-constraint problems;
- $\sigma_t$: the standard deviation of the $\epsilon$-constraint problems' computation time;
- L-5%: the percentage of the computation time spent on the 5% harder problems;
- S-50%: the percentage of the computation time spent on the 50% easier problems.

The letters *t.l.* (time limit) indicates that the instance was still unsolved after a time limit of 72 hours (259,200 seconds). For those instances, the column $|\mathcal{F}|$ gives the ratio $\alpha = \frac{p}{p(V)}$, which is an indication of the portion of the Pareto front that has been found before the time limit was reached.

| Instance | TIME | $\lvert\mathcal{F}\rvert$ | N | $\bar{t}$ | $\sigma_t$ | L-5% | S-50% |
|---|---|---|---|---|---|---|---|
| eil22 | 7.96 | 67 | 71 | 0.11 | 0.05 | 9.67 | 32.04 |
| eil23 | 8.37 | 75 | 77 | 0.11 | 0.07 | 9.8 | 21.74 |
| eil30 | 27.3 | 125 | 141 | 0.19 | 0.18 | 18.79 | 19.08 |
| eil33 | 54.89 | 159 | 228 | 0.24 | 0.21 | 16.82 | 20.13 |
| att48 | 11.47 | 48 | 47 | 0.24 | 0.2 | 15.34 | 22.58 |
| eil51 | 539.6 | 223 | 254 | 2.12 | 2.28 | 21.65 | 16.36 |
| eilA76 | 2605.27 | 355 | 458 | 5.69 | 7.15 | 24.62 | 14.6 |
| eilA101 | 5046.78 | 498 | 701 | 7.2 | 9.7 | 27.13 | 10.39 |
| gil262 | t.l. | *29.48* | - | - | - | - | - |

Table 4.2: Statistics on the VRP instances

[†]The $\epsilon$-constraint problems are PCTSPs that visit at least 3 vertices. The trivial solutions containing 1 or 2 vertices are found by enumeration and are therefore not included in N.

| Instance | TIME | $\lvert\mathcal{F}\rvert$ | N | $\bar{t}$ | $\sigma_t$ | L-5% | S-50% |
|---|---|---|---|---|---|---|---|
| burma14 | 0.14 | 14 | 12 | 0.01 | 0.01 | 21.43 | 35.71 |
| ulysses16 | 0.2 | 16 | 14 | 0.01 | 0.01 | 15.0 | 25.0 |
| ulysses22 | 0.51 | 22 | 21 | 0.02 | 0.01 | 11.76 | 25.49 |
| att48 | 9.82 | 48 | 47 | 0.21 | 0.15 | 14.56 | 24.75 |
| eil51 | 10.53 | 51 | 50 | 0.21 | 0.29 | 23.84 | 13.49 |
| berlin52 | 10.18 | 52 | 51 | 0.2 | 0.17 | 13.16 | 18.86 |
| st70 | 61.57 | 70 | 69 | 0.89 | 1.71 | 36.09 | 6.11 |
| eil76 | 37.19 | 76 | 75 | 0.5 | 0.59 | 21.65 | 18.93 |
| pr76 | 175334.47 | 76 | 75 | 2337.79 | 10774.27 | 83.3 | 0.02 |
| rat99 | 233.78 | 99 | 98 | 2.39 | 3.03 | 23.57 | 16.26 |
| kroA100 | 341.15 | 100 | 99 | 3.45 | 4.81 | 23.22 | 9.18 |
| kroB100 | 1075.63 | 100 | 99 | 10.86 | 33.79 | 54.69 | 2.55 |
| kroC100 | 303.73 | 100 | 99 | 3.07 | 3.58 | 19.94 | 14.0 |
| kroD100 | 178.9 | 100 | 99 | 1.81 | 2.53 | 23.41 | 11.1 |
| kroE100 | 837.43 | 100 | 99 | 8.46 | 25.39 | 52.05 | 3.42 |
| rd100 | 365.53 | 100 | 99 | 3.69 | 10.8 | 59.36 | 5.01 |
| eil101 | 90.37 | 101 | 100 | 0.9 | 0.84 | 19.58 | 20.52 |
| lin105 | 5558.73 | 105 | 104 | 53.45 | 152.07 | 61.17 | 1.66 |
| pr107 | 74.12 | 107 | 106 | 0.7 | 0.76 | 20.67 | 16.99 |
| pr124 | 2990.82 | 124 | 123 | 24.32 | 44.57 | 34.47 | 3.32 |
| bier127 | 1073.62 | 127 | 126 | 8.52 | 18.13 | 43.96 | 6.68 |
| ch130 | 719.24 | 130 | 129 | 5.58 | 12.87 | 42.61 | 6.25 |
| pr136 | 64590.76 | 136 | 135 | 478.45 | 1627.46 | 68.69 | 0.13 |
| gr137 | 3354.58 | 137 | 136 | 24.67 | 64.72 | 48.83 | 3.95 |
| pr144 | 17481.92 | 144 | 143 | 122.25 | 675.57 | 80.94 | 1.56 |
| ch150 | 3706.72 | 150 | 149 | 24.88 | 51.03 | 37.53 | 3.14 |
| kroA150 | 81024.71 | 150 | 149 | 543.79 | 1066.54 | 39.86 | 1.52 |
| kroB150 | 68089.94 | 150 | 149 | 456.98 | 1309.99 | 54.31 | 0.39 |
| pr152 | t.l. | *93.38* | - | - | - | - | - |
| u159 | t.l. | *39.87* | - | - | - | - | - |

Table 4.3: Statistics on the TSP generation 1 instances

Our algorithm was able to solve instances of 150 vertices for easy instances (TSP generation 1) and up to about 100 vertices for harder instances. Among those that remained unsolved after 72 hours, 39% were almost solved ($\alpha > 0.8$) while there was still a lot to do for 28% of them ($\alpha < 0.2$). Observe that the latter are all very hard instances (TSP generation 3).

Two factors characterize hard TSPP instances: the size of the Pareto front and the difficulty of the subproblems (PCTSPs). An empirical evidence of the first factor is a correlation coefficient of 0.74 between the size of the Pareto front and the total computation time. We observed that all instances solved in more than 24 hours are from generations 2 and 3 (except for one), which is not a surprise since both generations are designed to produce instances with a lot of efficient solutions. On average, the ratio of $|\mathcal{F}|$ over the number of vertices is 8.08 and 7.84 for generations 2 and 3, respectively, while it is 1.0 for generation 1 and 3.78 for VRP instances.

| Instance | TIME | $\|\mathcal{F}\|$ | N | $\bar{t}$ | $\sigma_t$ | L-5% | S-50% |
|---|---|---|---|---|---|---|---|
| burma14 | 1.73 | 59 | 60 | 0.03 | 0.02 | 15.03 | 27.17 |
| ulysses16 | 4.31 | 102 | 101 | 0.04 | 0.02 | 10.21 | 34.8 |
| ulysses22 | 13.6 | 130 | 130 | 0.1 | 0.05 | 9.78 | 32.65 |
| att48 | 866.16 | 435 | 438 | 1.98 | 1.21 | 13.1 | 26.85 |
| eil51 | 627.13 | 225 | 269 | 2.33 | 2.26 | 19.24 | 16.81 |
| berlin52 | 1321.63 | 406 | 411 | 3.22 | 2.75 | 15.97 | 16.84 |
| st70 | 13892.93 | 503 | 643 | 21.61 | 38.21 | 34.48 | 4.53 |
| eil76 | 5158.31 | 386 | 538 | 9.59 | 10.81 | 22.45 | 16.28 |
| pr76 | t.l. | *96.29* | - | - | - | - | - |
| rat99 | 31524.89 | 662 | 779 | 40.47 | 76.59 | 38.31 | 11.11 |
| kroA100 | t.l. | 87.88 | - | - | - | - | - |
| kroB100 | 186395.45 | 1332 | 1363 | 136.75 | 337.95 | 50.42 | 7.33 |
| kroC100 | 120664.66 | 1311 | 1333 | 90.52 | 129.24 | 29.95 | 18.99 |
| kroD100 | 53819.04 | 1128 | 1129 | 47.67 | 33.95 | 15.43 | 25.2 |
| kroE100 | 82149.58 | 1068 | 1086 | 75.64 | 119.62 | 32.46 | 9.26 |
| rd100 | 43100.31 | 920 | 962 | 44.8 | 30.01 | 14.31 | 26.92 |
| eil101 | 34953.71 | 515 | 838 | 41.71 | 45.84 | 21.87 | 13.5 |
| lin105 | 203727.18 | 1043 | 1329 | 153.29 | 291.15 | 36.25 | 10.71 |
| pr107 | t.l. | *49.36* | - | - | - | - | - |
| pr124 | t.l. | *29.24* | - | - | - | - | - |
| bier127 | t.l. | *87.65* | - | - | - | - | - |
| ch130 | t.l. | *93.87* | - | - | - | - | - |

Table 4.4: Statistics on the TSP generation 2 instances

| Instance | TIME | $\|\mathcal{F}\|$ | N | $\bar{t}$ | $\sigma_t$ | L-5% | S-50% |
|---|---|---|---|---|---|---|---|
| burma14 | 1.99 | 70 | 68 | 0.03 | 0.01 | 9.05 | 31.16 |
| ulysses16 | 3.81 | 92 | 88 | 0.04 | 0.02 | 9.45 | 32.28 |
| ulysses22 | 14.04 | 128 | 126 | 0.11 | 0.05 | 10.19 | 31.98 |
| att48 | 1334.41 | 438 | 440 | 3.03 | 1.83 | 13.01 | 26.95 |
| eil51 | 1196.46 | 267 | 299 | 4.0 | 6.16 | 29.71 | 10.51 |
| berlin52 | 1292.27 | 439 | 446 | 2.9 | 2.12 | 15.61 | 24.26 |
| st70 | 9258.1 | 452 | 546 | 16.96 | 17.35 | 19.27 | 12.52 |
| eil76 | 4971.95 | 383 | 468 | 10.62 | 8.87 | 16.68 | 20.9 |
| pr76 | t.l. | *82.52* | - | - | - | - | - |
| rat99 | t.l. | *5.70* | - | - | - | - | - |
| kroA100 | 137168.33 | 815 | 820 | 167.28 | 209.37 | 25.77 | 13.12 |
| kroB100 | t.l. | *16.75* | - | - | - | - | - |
| kroC100 | 128195.9 | 1223 | 1228 | 104.39 | 106.56 | 20.87 | 17.14 |
| kroD100 | 71826.93 | 1063 | 1068 | 67.25 | 73.97 | 20.6 | 24.61 |
| kroE100 | t.l. | *10.38* | - | - | - | - | - |
| rd100 | 180959.84 | 1513 | 1551 | 116.67 | 428.34 | 65.3 | 6.9 |
| eil101 | 38227.19 | 499 | 697 | 54.85 | 92.39 | 33.78 | 7.62 |
| lin105 | t.l. | *9.77* | - | - | - | - | - |
| pr107 | t.l. | *2.16* | - | - | - | - | - |
| pr124 | t.l. | *3.66* | - | - | - | - | - |
| bier127 | t.l. | *16.09* | - | - | - | - | - |
| ch130 | t.l. | *88.35* | - | - | - | - | - |

Table 4.5: Statistics on the TSP generation 3 instances

The impact of the subproblems' toughness on the computation time is partially shown by a correlation coefficient of 0.51 between the average subproblem computation time and the total

computation time. Although this correlation is significant, it does not tell the whole story. One should observe that the subproblems are not equally hard. In fact, the computation times are mostly due to a few PCTSPs. This phenomenon is observed on hard instances with a relatively small Pareto front (that is, the latter cannot explain the instance's toughness), such as *pr76, kroA150, kroB150* and *pr136* generation 1 instances. The $\sigma_t$ statistic of those instances is very high and 40% to 83% of the computation time is spent on 5% of the subproblems. Moreover, less than 2% of the computation is spent on 50% of the subproblems (0.02% for *pr76* instance).

### 4.5.3 Results for approximate Pareto fronts

As explained in Section 1, our algorithm can produce an approximation of the Pareto front. One simply has to introduce a tolerance ($\rho$) on the minimal gap between the upper and lower bound for a solution to be accepted by the branch-and-cut procedure. Tables 4.6 and 4.7 show results for $\rho$ of 0.01 and 0.10, respectively, on a sample of hard instances for which the exact Pareto front is known.

There is no consensus on the quality metrics that should be used for multi-criteria approximation algorithms. We decided to use two categories of metrics reported in [56]. The first category is made of distance based metrics while the second is made of ratios on the size of the exact and approximated Pareto fronts. More precisely, the columns of Tables 4.6 and 4.7 correspond to:

- $t_e$ : computation time for the exact Pareto front ($\mathcal{F}$);
- $t_a$ : computation time for the approximate Pareto front ($\bar{\mathcal{F}}$);
- $d_p = \frac{1}{|\mathcal{F}|} \sum_{z \in \mathcal{F}} \min_{z' \in \bar{\mathcal{F}}} \frac{|z_p - z'_p|}{z_p}$, where $z_p$ stands for the collected prize associated with point $z$;
- $d_c = \frac{1}{|\mathcal{F}|} \sum_{z \in \mathcal{F}} \min_{z' \in \bar{\mathcal{F}}} \frac{|z_c - z'_c|}{z_c}$, where $z_c$ stands for the travel cost associated with point $z$;
- $d_z = \frac{1}{|\mathcal{F}|} \sum_{z \in \mathcal{F}} \min_{z' \in \bar{\mathcal{F}}} \max\left(\frac{|z_p - z'_p|}{z_p}, \frac{|z_c - z'_c|}{z_c}\right)$[‡];
- $d_z^{max} = \max_{z \in \mathcal{F}} \min_{z' \in \bar{\mathcal{F}}} \max\left(\frac{|z_p - z'_p|}{z_p}, \frac{|z_c - z'_c|}{z_c}\right)$;

[‡]This corresponds to an average of the best relative Chebyshev distances. The latter defines the distance between the points $(x_1, y_1)$ and $(x_2, y_2)$ as: $\max(|x_2 - x_1|, |y_2 - y_1|)$.

- $Q_1 = \frac{\mathcal{F} \cap \bar{\mathcal{F}}}{|\mathcal{F}|}$;
- $Q_2 = \frac{\mathcal{F} \cap \bar{\mathcal{F}}}{|\mathcal{F}|}$.

For $\rho = 0.01$, the algorithm is 5.7 times faster than the original exact version. It produces a very good approximation of the Pareto front for every instance of the sample. Even though only an average of 38% of the non dominated points are found, each non dominated point is on average at a distance of 0.002 of a point on the approximated front, according to the relative Chebyshev distance metric. When the tolerance is increased to 0.10, the algorithm runs 83 times faster than the original version and still finds a good approximation of the Pareto front. The average relative Chebyshev distance between each point of $\mathcal{F}$ and the nearest point of $\bar{\mathcal{F}}$ is 0.018.

One should observe that for both values of $\rho$, $Q_1$ is always lower than $Q_2$. For example, an average of 38.0% and 62.3 % of the points on $\mathcal{F}$ are also on $\bar{\mathcal{F}}$ when $\rho = 0.01$ and $\rho = 0.10$, respectively. This suggests that the approximated non dominated set is smaller than the exact one. Actually, the size of the approximated front corresponds to 50.5% ($\rho = 0.01$) and 22.3% ($\rho = 0.10$) of $|\mathcal{F}|$, on average. Those averages exclude TSP generation 1 instances for which both the exact and approximate fronts have about the same size. Finally, although there is no theoretical guarantee on the performance of the approximation algorithm, the results show relatively small variations in $d_z$. This suggests that the strategy is quite robust.

## 4.6 Conclusion

We have shown that the $\epsilon$-constraint method can be used efficiently to find the exact Pareto front of BOCO problems with integer objective values. We also provide improvement heuristics devised to speed up the resolution of the $\epsilon$-constraint problems when the latter are solved through branch-and-cut. Our $\epsilon$-constraint method and the improvement heuristics have been tested successfully on the TSPP. The results have shown the relevance of the improvement heuristics and provided the first exact solutions for TSPP instances. Because the TSPP is a very hard problem, the instances that have been solved are quite small. Obviously, exact algorithms cannot run very fast on BOCO problems, but we believe that our solutions will be useful

| Instance | Type | $t_e$ | $t_a$ | $t_a/t_e$ | $d_p$ | $d_e$ | $d_z$ | $d_z^{max}$ | $Q_1$ | $Q_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| eilA76 | vrp | 2605.27 | 734.63 | 0.282 | 0.001 | 0.002 | 0.002 | 0.026 | 0.603 | 0.903 |
| eilA101 | vrp | 5046.78 | 1847.79 | 0.366 | 0.001 | 0.001 | 0.001 | 0.010 | 0.538 | 0.736 |
| pr76 | tspp1 | 175334.47 | 28747.08 | 0.164 | 0.000 | 0.002 | 0.002 | 0.010 | 0.461 | 0.461 |
| pr136 | tspp1 | 64590.76 | 4742.81 | 0.073 | 0.000 | 0.001 | 0.002 | 0.009 | 0.500 | 0.504 |
| pr144 | tspp1 | 17481.92 | 5819.35 | 0.333 | 0.000 | 0.001 | 0.002 | 0.019 | 0.465 | 0.479 |
| kroA150 | tspp1 | 81024.71 | 2516.27 | 0.031 | 0.000 | 0.002 | 0.003 | 0.010 | 0.253 | 0.253 |
| kroB150 | tspp1 | 68089.94 | 22744.47 | 0.334 | 0.000 | 0.002 | 0.003 | 0.010 | 0.407 | 0.407 |
| st70 | tspp2 | 13892.93 | 1061.9 | 0.076 | 0.001 | 0.001 | 0.002 | 0.018 | 0.475 | 0.685 |
| kroB100 | tspp2 | 186395.45 | 8089.15 | 0.043 | 0.001 | 0.001 | 0.002 | 0.029 | 0.261 | 0.645 |
| kroC100 | tspp2 | 120664.66 | 5988.84 | 0.050 | 0.001 | 0.001 | 0.002 | 0.041 | 0.250 | 0.659 |
| rd100 | tspp2 | 43100.31 | 3712.49 | 0.086 | 0.001 | 0.001 | 0.002 | 0.032 | 0.262 | 0.623 |
| lin105 | tspp2 | 203727.18 | 31861.07 | 0.156 | 0.001 | 0.001 | 0.002 | 0.081 | 0.320 | 0.742 |
| st70 | tspp3 | 9258.1 | 2192.75 | 0.237 | 0.002 | 0.001 | 0.003 | 0.032 | 0.473 | 0.751 |
| kroA100 | tspp3 | 137168.33 | 17569.18 | 0.128 | 0.002 | 0.001 | 0.003 | 0.050 | 0.329 | 0.786 |
| kroD100 | tspp3 | 71826.93 | 12683.36 | 0.177 | 0.002 | 0.001 | 0.003 | 0.039 | 0.249 | 0.639 |
| rd100 | tspp3 | 180959.84 | 33950.47 | 0.188 | 0.002 | 0.001 | 0.003 | 0.143 | 0.171 | 0.617 |
| eil101 | tspp3 | 38227.19 | 10434.01 | 0.273 | 0.001 | 0.002 | 0.002 | 0.017 | 0.439 | 0.709 |
| Averages | | | | **0.176** | **0.001** | **0.001** | **0.002** | **0.034** | **0.380** | **0.623** |

Table 4.6: Results for approximate Pareto fronts with $\rho = 0.01$

| Instance | Type | $t_e$ | $t_a$ | $t_a/t_e$ | $d_p$ | $d_e$ | $d_z$ | $d_z^{max}$ | $Q_1$ | $Q_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| eilA76 | vrp | 2605.27 | 22.83 | 0.009 | 0.007 | 0.008 | 0.019 | 0.211 | 0.085 | 0.345 |
| eilA101 | vrp | 5046.78 | 83.48 | 0.017 | 0.005 | 0.004 | 0.014 | 0.141 | 0.084 | 0.228 |
| pr76 | tspp1 | 175334.47 | 17.67 | 0.000 | 0.002 | 0.008 | 0.028 | 0.083 | 0.118 | 0.122 |
| pr136 | tspp1 | 64590.76 | 354.62 | 0.005 | 0.001 | 0.005 | 0.017 | 0.070 | 0.206 | 0.215 |
| pr144 | tspp1 | 17481.92 | 599.07 | 0.034 | 0.002 | 0.007 | 0.018 | 0.050 | 0.139 | 0.157 |
| kroA150 | tspp1 | 81024.71 | 164.27 | 0.002 | 0.002 | 0.006 | 0.020 | 0.083 | 0.073 | 0.078 |
| kroB150 | tspp1 | 68089.94 | 367.13 | 0.005 | 0.000 | 0.006 | 0.018 | 0.083 | 0.127 | 0.129 |
| st70 | tspp2 | 13892.93 | 39.33 | 0.003 | 0.007 | 0.006 | 0.018 | 0.113 | 0.074 | 0.272 |
| kroB100 | tspp2 | 186395.45 | 163.33 | 0.001 | 0.004 | 0.004 | 0.014 | 0.178 | 0.035 | 0.208 |
| kroC100 | tspp2 | 120664.66 | 224.53 | 0.002 | 0.006 | 0.005 | 0.015 | 0.155 | 0.039 | 0.291 |
| rd100 | tspp2 | 43100.31 | 1623.16 | 0.009 | 0.006 | 0.003 | 0.014 | 0.243 | 0.052 | 0.226 |
| lin105 | tspp2 | 203727.18 | 528.95 | 0.003 | 0.005 | 0.005 | 0.025 | 0.068 | 0.023 | 0.122 |
| st70 | tspp3 | 9258.1 | 460.18 | 0.050 | 0.007 | 0.004 | 0.022 | 0.143 | 0.084 | 0.297 |
| kroA100 | tspp3 | 137168.33 | 1211.63 | 0.009 | 0.008 | 0.005 | 0.015 | 0.356 | 0.058 | 0.333 |
| kroD100 | tspp3 | 71826.93 | 1894.98 | 0.026 | 0.006 | 0.004 | 0.016 | 0.125 | 0.027 | 0.149 |
| rd100 | tspp3 | 180959.84 | 1623.16 | 0.009 | 0.006 | 0.003 | 0.013 | 0.143 | 0.032 | 0.214 |
| eil101 | tspp3 | 38227.19 | 1805.84 | 0.047 | 0.007 | 0.006 | 0.019 | 0.113 | 0.066 | 0.234 |
| Averages | | | | **0.012** | **0.005** | **0.006** | **0.018** | **0.138** | **0.078** | **0.213** |

Table 4.7: Results for approximate Pareto fronts with $\rho = 0.10$

benchmarks to evaluate the quality of future approximation algorithms for the TSPP. Besides, we have shown that good approximations of the Pareto front might be found relatively quickly through a simple modification of our exact algorithm.

## Acknowledgments

Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT). This support is gratefully acknowledged.

## 4.7 Appendix - Valid inequalities for the PCTSP

This appendix summarizes the valid inequalities used by the branch-and-cut algorithm presented in [15] for the model defined by equations (4.14) to (4.20). They are obtained either from the associated knapsack polytope, from a combination of the SECs and the minimum prize constraint, or from the associated traveling salesman polytope.

### 4.7.1 Inequalities from the associated knapsack polytope

Two types of inequalities based on knapsack constraints are considered. They are referred as the *lifted-cover* and *cost-cover* inequalities.

#### Lifted-cover inequalities

Let $S$ be a minimal cover for (4.21), i.e., $S$ is a minimal subset of $V$ such that $p(S) > U$. The cover inequality:

$$\sum_{v \in (S \cup S')} (1 - y_v) \leq |S| - 1 \tag{4.23}$$

where $S' = \{v \in V \setminus S : p_v \geq \max_{w \in S} p_w\}$, is valid for the knapsack problem [5]. The coefficients of the $y$ variables can be lifted to obtain *lifted-cover* inequalities that reinforce equation (4.23). Let $S' = \{v \in V \setminus S : p_v \geq \max_{w \in S} p_w\}$, and $S_h$ the set of the first $h$ elements of $S$ ($\forall i \in S$, $p_i \geq p_{i+1}$ is assumed), $h = 1, ..., |S|$. Let V be partitioned into

$V_0, V_1, .., V_q$, $q = |S| - 1$, where:

$$V_h = \{v \in (S \cup S') : p(S_h) \leq p_v < p(S_{h+1})\}, \quad h = 2, ..., q$$
$$V_1 = (S \cup S') \setminus \cup_{h=2}^{q} V_h, \tag{4.24}$$
$$V_0 = V \setminus (S \cup S')$$

and define:

$$\pi_v = h, \quad \forall v \in V_h, \quad h = 0, ..., q. \tag{4.25}$$

Then, the lifted-cover inequality is written:

$$\sum_{v \in S}(1 - y_v) + \sum_{v \in V \setminus S} \pi_v (1 - y_v) \leq |S| - 1 \tag{4.26}$$

It has been shown that (4.26) is valid for all $y \in KP$, where $KP$ is the convex hull of $\{y \in \{0,1\} : y$ satisfies (4.21) $\}$ [5]. Since the PCTS polytope is included in $KP$ [6], the lifted-cover inequalities are also valid for the PCTSP.

**Cost-cover inequalities**

Let $c_U$ be the upper bound on an optimal solution. Then $\sum_{e \in E} c_e x_e \leq c_U$ defines a knapsack constraint in terms of costs that can be used to derive valid inequalities. Let $S \subseteq V$, $1 \in S$ and $\sigma_S$ a lower bound on the optimal TSP value on $S$. Then, if $\sigma_S > c_U$ and if the costs satisfy the triangle inequality, the following *cost-cover* inequalities are valid for the PCTSP. We consider only special cases that are easy to separate, namely, when $|S| = 3$ :

$$y_u + y_v \leq 1 \quad \forall u, v \in V' \quad \text{such that} \quad c_{(1,u)} + c_{(u,v)} + c_{(v,1)} > c_U \tag{4.27}$$

and when $|S| = 2$ :

$$y_v = 0 \quad \forall v \in V' \quad \text{such that} \quad 2c_{(1,v)} > c_U \tag{4.28}$$

### 4.7.2 Inequalities from the SEC and knapsack constraint

*Cycle-cover* and *conditional* inequalities both use a knapsack constraint to strengthen the SEC (4.16).

**Cycle-cover inequalities**

The cycle-cover inequalities exploit the minimum prize constraint and the fact that a feasible solution must be a cycle. Let $S \subset V, 1 \in S$ such that $p(S) < \bar{p}$, then

$$x(E(S)) \leq y(S) - 1 \tag{4.29}$$

is a valid inequality for the PCTSP, as shown in [15].

**Conditional inequalities**

An upper bound $c_U$ on the objective value can be used to derive inequalities similar to the cycle-cover, but based on the selected edges. Although they are not guaranteed to be valid, these inequalities can be conditionally used in a cutting-plane context. Let $T \subseteq E$ such that $c(T) > c_U$, then

$$x(T) \leq y(V(T)) - 1 \tag{4.30}$$

is valid for the PCTSP if no feasible solution of value lower than $c_U$ is contained in $T$, since $x(T) \leq y(V(T))$ holds for every feasible solution. This occurs, in particular, when $T$ defines a simple cycle that goes through the depot and for which $c(T) > c_U$.

### 4.7.3 Comb inequalities

The well known comb inequalities can be adapted from the TSP to the PCTSP [7]. Let us consider two sets of vertices, the handle $H \subset V$ and the teeth $T_j \subset V$ $(j = 1, .., t)$. The general comb inequalities are formulated as:

$$x(E(H)) + \sum_{j=1}^{t} x(E(T_j)) \leq y(H) + \sum_{j=1}^{t} |T_j| - \frac{3t+1}{2} \tag{4.31}$$

for all $H, T_1, ..., T_t$ satisfying:

a) $|T_j \cap H| \geq 1$, with $j = 1, .., t$;

b) $|T_j \setminus H| \geq 1$, with $j = 1, ..., t$;

c) $T_i \cap T_j = \emptyset$, with $1 \leq i < j \leq t$; and

d) $t \geq 3$ and odd.

In the special case where $|T_j \cap H| = 1$ for all $j$, the inequalities are referred to as *simple comb* inequalities. Simple comb inequalities become *2-matching* inequalities if $|T_j \setminus H| = 1$ for all $j$.

### 4.7.4 Logical inequalities

Obviously, if an edge $e \in \delta(v)$ is part of a solution, the vertex $v$ must be visited, hence the following logical inequality:

$$x_e \leq y_v \quad \forall e \in \delta(v), \quad v \in V' \tag{4.32}$$

# Conclusion

Cette thèse s'inscrit dans la mouvance actuelle où l'on cherche à exploiter la performance grandissante des ordinateurs en modélisant des problèmes complexes de façon réaliste et en mettant en oeuvre des résultats théoriques jusqu'ici très peu utilisés en raison des ressources de calcul qu'ils exigent. Dans ce contexte, nous avons contribué, certes bien modestement, à l'avancement des connaissances sur deux problèmes de tournées multicritères : le problème de planification de voyage et le problème du voyageur de commerce avec profits (PVCP). Dans un premier temps, nous avons étudié ces deux problèmes en faisant l'hypothèse que les préférences du preneur de décision sont connues avant le processus d'optimisation. Cela nous a permis, d'une part, d'exploiter l'existence d'une fonction d'utilité, et d'autre part, de formuler le problème original en un problème monocritère où certains objectifs sont transformés en contraintes. Nous avons ensuite abordé le PVCP sans faire d'hypothèse sur les préférences, mais plutôt en élaborant une stratégie d'utilisation de la méthode $\epsilon$-*constraint* afin d'énumérer efficacement tous les points de la frontière de Pareto, et ce pour un vaste ensemble de problèmes bi-critères.

Motivé par l'importance grandissante du commerce électronique dans l'industrie du tourisme, notre problème de planification de voyage modélise la construction d'un plan de voyage optimal (avions et hôtels) satisfaisant les contraintes d'un voyageur et correspondant le mieux possible à ses préférences. Nous avons proposé un modèle permettant de résoudre ce problème efficacement à l'aide d'algorithmes de plus courts chemins dans un graphe spatio-temporel. Notre approche de résolution exacte repose sur une stratégie de décomposition où le nombre de sous-problèmes croît linéairement avec la longueur de l'itinéraire imposé par le voyageur. De plus, nous avons proposé une stratégie d'exploration progressive de la composante tempo-

relle permettant d'améliorer significativement les performances de l'algorithme. Bien que nos résultats témoignent de la performance de notre approche sur des problèmes d'assez grande taille, il est clair que nous sommes loin d'un prototype réaliste pour la confection de plans de voyage. Le modèle doit encore être raffiné afin d'être plus souple et de permettre au voyageur d'exprimer ses préférences de manière plus nuancée. Nous croyons toutefois que notre stratégie de résolution pourra être utilisée dans des contextes plus réalistes et plus complexes, dans la mesure où l'ordre de visite des villes (ou d'autres attraits touristiques) demeure prédéterminé.

Nos travaux sur le PVCP portent sur un algorithme de séparation et coupes pour le *prize collecting traveling salesman problem* (PCTSP) et sur une approche de type $\epsilon$-*constraint* permettant d'énumérer tous les points de la frontière de Pareto pour certains problèmes bi-critères. Notre algorithme pour résoudre le PCTSP repose sur six familles d'inégalités valides utilisées pour la première fois pour résoudre ce problème. Pour chacune de ces familles, nous avons présenté un algorithme de séparation efficace et des résultats détaillés permettant d'apprécier leur impact sur la performance globale de l'algorithme. Nous avons aussi analysé le comportement de différentes séquences d'ajout des coupes. Notre algorithme étant le premier algorithme exact pour le PCTSP, il n'existe malheureusement aucun point de comparaison. Nous sommes toutefois persuadés que notre analyse du comportement des différentes familles de coupes pourra être d'une grande utilité lors du développement de nouveaux algorithmes exacts pour le PCTSP nécessitant l'ajout d'inégalités valides.

La troisième partie de cette thèse se veut plus générale que les deux premières. En effet, le PVCP sert ici d'illustration à une méthode permettant de résoudre les problèmes d'optimisation combinatoire multicritères où les valeurs des objectifs sont entières. Nous avons démontré que la frontière de Pareto exacte de ces problèmes peut être énumérée par les solutions optimales d'une séquence de $\epsilon$-*problems*. Nous avons aussi proposé des heuristiques permettant de tirer profit des informations générées au cours de la résolution de ces sous-problèmes afin d'accélérer significativement l'énumération de la frontière de Pareto. Nos résultats sur le PVCP démontrent d'ailleurs très clairement la pertinence de ces heuristiques. Il serait maintenant intéressant de mettre nos idées à l'épreuve sur d'autres problèmes et, surtout, de les comparer avec d'autres stratégies, en particulier sur le PVCP. Les instances, pour lesquelles nous avons d'ailleurs obtenu les premiers résultats exacts, sont générées à partir d'instances classiques

du problème du commis voyageur. Nos résultats devraient donc être confrontés dans les prochaines années.

Une suite logique à nos travaux sur le PVCP serait d'adapter notre méthode exacte à des variantes plus élaborées du problème. Outre les extensions classiques telles que l'ajout de fenêtres de temps ou l'absence de dépôt prédéterminé, il serait intéressant d'étudier l'impact de profits marginaux décroissants associés à des sommets de même type. Par exemple, dans le contexte de l'industrie du tourisme, on pourrait modéliser les différentes attractions touristiques ou activiés à l'aide de différentes catégories de sommets. Il semble raisonnable de supposer que l'intérêt marginal pour un même type d'attraction touristique ait tendance à décroître. Par exemple, la visite d'une dixième cathédrale risque d'être beaucoup moins intéressante que ne le fut la visite de la première. Une variante du PVCP qui considérerait différentes catégories de sommets avec des profits marignaux décroissants serait considérablement plus difficile à résoudre que le problème original parce que la valeur des profits changerait au cours de la construction d'une solution.

Cette thèse n'a pas abordé les approches heuristiques en optimisation multicritère. Il s'agit toutefois d'une littérature importante, bien plus en fait que la littérature traitant de méthodes exactes. La raison évidente pour cela est bien entendu la complexité des problèmes multicritères qu'il est souvent impossible de résoudre de manière exacte. Les résultats de la variante heuristique de notre algorithme pour le PVCP témoignent des avantages des approches heuristiques. Une étude plus approfondies de ces dernières pour le PVCP serait d'ailleurs une suite intéressante à nos travaux sur les méthodes exactes, d'autant plus que les solutions que nous avons obtenues permettraient d'évaluer la qualité des méthodes heuristiques.

Il ne serait pas surprenant que les progrès technologiques contribuent à accroître l'intérêt, autant pour les méthodes exactes que pour les méthodes heuristiques. Chose certaine, dans le contexte actuel où la puissance des ordinateurs permet de résoudre des modèles de plus en plus réalistes, l'engouement pour l'optimisation multicritère continuera de croître dans les prochaines années.

# Bibliographie


[1] ACHTERBERG, T. «SCIP - a framework to integrate constraint and mixed integer programming». Tech. Rep. ZR-06-19, Konrad-Zuse-Zentrum fur Informationstchnik Berlin, 2004.

[2] AHN, B. H. et J. Y. SHIN, «Vehicle routing with time windows and time-varying congestion», *J. Opl. Res. Soc.*, vol. 42, 1991, pp. 393–400.

[3] AHUJA, R. K., T. L. MAGNANTI et J. B. ORLIN, *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[4] AWERBUCH, B., Y. AZAR, A. BLUM et S. VEMPALA, «New approximation guarantees for minimum-weight k-trees and prize-collecting salesman», *SIAM Journal on Computing*, vol. 28, 1998, pp. 254–262.

[5] BALAS, E., «Facets of the knapsack polytope», *Mathematical Programming*, vol. 8, 1975, pp. 146–164.

[6] BALAS, E., «The prize collecting traveling salesman problem», *Networks*, vol. 19, 1989, pp. 621–636.

[7] BALAS, E., «The prize collecting traveling salesman problem : II. Polyhedral results», *Networks*, vol. 25, 1995, pp. 199–216.

[8] BALAS, E., «New classes of efficiently solvable generalized traveling salesman problems», *Annals of Operations Research*, vol. 86, 1999, pp. 529–558.

[9] BALAS, E. «The prize collecting traveling salesman problem and its applications». Dans *The Traveling Salesman Problem and its Variations*, G. Gutin et A. P. Punnen, Eds. Kluwer Academic Publishers, 2002, pp. 663–695.

[10] BALAS, E. et E. ZEMEL, «Facets of the knapsack polytope from minimal covers», *SIAM Journal of Applied Mathematics*, vol. 34, 1978, pp. 119–148.

[11] BARVINOK, A., E. K. GIMADI et A. I. SERDYUKOV. «The maximum TSP». Dans *The Traveling Salesman Problem and Its Variations*, G. Gutin et A. Punnen, Eds. Kluwer Academic Publishers, 2002, pp. 585–608.

[12] BELLMAN, R., «On a routing problem», *Quaterly of Applied Mathematics*, vol. 16, 1958, pp. 87–90.

[13] BERTSEKAS, D., *Dynamic Programming*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[14] BÉRUBÉ, J.-F. «Planification pour agents dans un environnement dynamique et incertain». Mémoire de maîtrise, Université de Montréal, 2003.

[15] BÉRUBÉ, J.-F., M. GENDREAU et J.-Y. POTVIN. «A branch-and-cut algorithm for the undirected prize collecting traveling salesman problem». Tech. Rep. CRT-2006-30, Centre for Research on Transportation, Université de Montréal, 2006.

[16] CAI, X., T. KLOKS et C. K. WONG, «Time-varying shortest path problems with constraints», *Networks*, vol. 29, 1997, pp. 141–149.

[17] CARAMIA, M., N. RICCIARDI, A. SCOZZARI et G. STORCHI. «Tourist flow organization in an artistic town». Dans *Proceedings of CUPUM'99 : Computer in Urban Planning and Urban Management* (Venise, Italie, 1999).

[18] CHABINI, I. et B. DEAN. «Shortest path problems in discrete-time dynamic networks : Complexity, algorithms, and implementations». Tech. rep., Massachusetts Institute of Technology, Cambridge, MA, 1999.

[19] CHABINI, I. et S. LAN, «Adaptations of the $A^*$ algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks», *IEEE Transactions on Intelligent Transportation Systems*, vol. 3, 2002, pp. 60–74.

[20] CHANKONG, V. et Y. Y. HAIMES, *Multiobjective Decision Making : Theory and Methodology*. North-Holland, 1983.

[21] CHVÁTAL, V., «Edmonds polytopes and weakly hamiltonian graphs», *Mathematical Programming*, vol. 5, 1973, pp. 29–40.

[22] CLÍMACO, J. C. N. et E. Q. V. MARTINS, «A bicriterion shortest path algorithm», *European Journal of Operational Research*, vol. 11, 1982, pp. 399–404.

[23] COLLETTE, Y. et P. SIARRY, *Multiobjective Optimization*. Springer-Verlag, 2004.

[24] COOKE, K. L. et E. HALSEY, «The shortest route through a network with time-dependant internodal transit times», *Journal of Mathematical Analysis and Applications*, vol. 14, 1966, pp. 493–498.

[25] CRAINIC, T. G., N. RICCIARDI et G. STORCHI. «The on-demand tourist paths problem». Dans *Proceedings of TRISTAN V : Triennial Symposium on Transportation Analysis* (Le Gosier, Guadeloupe, Juin 2004).

[26] DELL'AMICO, M., F. MAFFIOLI et A. SCIOMACHEN, «A lagrangian heuristic for the prize collecting travelling salesman problem», *Annals of Operations Research*, vol. 81, 1998, pp. 289–305.

[27] DELL'AMICO, M., F. MAFFIOLI et P. VÄRBRAND, «On prize-collecting tours and the asymmetric travelling salesman problem», *International Transactions in Operational Research*, vol. 2, 1995, pp. 297–308.

[28] DIJKSTRA, E. W., «A note on two problems in connexion with graphs», *Numerishe Matematik*, vol. 1, 1959, pp. 269–271.

[29] DREYFUS, S. E., «An appraisal of some shortest path algorithms», *Operations Research*, vol. 17, 1969, pp. 395–412.

[30] DUNSTALL, S., M. E. T. HORN, P. KILBY, M. KRISHNAMOORTHY, B. OWENS, D. SIER et S. THIEBAUX, «An automated itinerary planning system for holiday travel», *Information Technology & Tourism*, vol. 6, 2004, pp. 195–210.

[31] EHRGOTT, M., «Approximation algorithms for combinatorial multicriteria optimization problems», *International Transactions in Operational Research*, vol. 7, 2000, pp. 5–31.

[32] EHRGOTT, M. et X. GANDIBLEUX. «Multiobjective combinatorial optimization - theory, methodology, and applications». Dans *Multiple Criteria Optimization : State of the Art Annotated Bibliographic Surveys*, M. Ehrgott et X. Gandibleux, Eds. Kluwer Academic Publishers, 2002, pp. 369–444.

[33] EHRGOTT, M. et A. J. V. SKRIVER, «Solving biobjective combinatorial max-ordering problems by ranking methods and a two-phases approach», *European Journal of Operational Research*, vol. 147, 2003, pp. 657–664.

[34] EHRGOTT, M. et M. M. WIECEK. «Multiobjective programming». Dans *Multiple Criteria Decision Analysis : State of the Art Surveys*, J. Figueira, S. Greco, et M. Ehrgott, Eds. Kluwer Academic Publishers, 2005, pp. 667–722.

[35] ESCHENAUER, H., J. KOSKI et A. OSYCZKA, *Multicriteria design optimization : Procedures and Applications*. Springer Verlag, 1990.

[36] FEILLET, D., P. DEJAX et M. GENDREAU, «Traveling salesman problems with profits», *Transportation Science*, vol. 39, 2005, pp. 188–205.

[37] FIGUERIA, J., S. GRECO et M. Ehrgott, EDS., *Multiple criteria decision analysis : state of the art surveys*. Kluwer Academic Publishers, 2005.

[38] FISCHETTI, M., J. J. SALAZAR-GONZÁLEZ et P. TOTH, «Solving the orienteering problem through branch-and-cut», *INFORMS Journal on Computing*, vol. 10, 1998, pp. 133–148.

[39] FISCHETTI, M., J. J. SALAZAR-GONZÁLEZ et P. TOTH. «The generalized traveling salesman and orienteering problems». Dans *The Traveling Salesman Problem and its Variations*, G. Gutin et A. P. Punnen, Eds. Kluwer Academic Publishers, 2002, pp. 609–662.

[40] FISCHETTI, M. et P. TOTH. «An additive approach for the optimal solution of the prize-collecting travelling salesman problem». Dans *Vehicule Routing : Methods and Studies*, B. L. Golden et A. A. Assad, Eds. Elsevier Science Publishers, B.V. North-Holland, 1988, pp. 319–343.

[41] FORD JR., L. R., *Network flow theory*. Rand Co., 1956.

[42] GENDREAU, M., A. HERTZ et G. LAPORTE, «New insertion and postoptimization procedures for the traveling salesman problem», *Operations Research*, vol. 40, 1992, pp. 1086–1094.

[43] GENDREAU, M., G. LAPORTE et F. SEMET, «The covering tour problem», *Operations Research*, vol. 45, 1997, pp. 568–576.

[44] GENDREAU, M., G. LAPORTE et F. SEMET, «A branch-and-cut algorithm for the undirected selective traveling salesman problem», *Networks*, vol. 32, 1998, pp. 263–273.

[45] GENSCH, D. H., «An industrial application of the traveling salesman's subtour problem», *AIIE Transactions*, vol. 10, 1978, pp. 362–370.

[46] GOLDEN, B. L., L. LEVY et R. VOHRA, «The orienteering problem», *Naval Research Logistics*, vol. 34, 1987, pp. 307–318.

[47] GRÖTSCHEL, M. et M. W. PADBERG, «On the symmetric travelling salesman problem I : inequalities», *Mathematical Programming*, vol. 16, 1979, pp. 265–280.

[48] GRÖTSCHEL, M. et M. W. PADBERG, «On the symmetric travelling salesman problem II : lifting theorems and facets», *Mathematical Programming*, vol. 16, 1979, pp. 281–302.

[49] GUTIN, G. ET A. P. PUNNEN, Eds., *The Traveling Salesman Problem and its Variations*. Kluwer Academic Publishers, 2002.

[50] GÖTHE-LUNDGREN, M., K. JÖRNSTEN et P. VÄRBRAND, «On the nucleolus of the basic vehicle routing game», *Mathematical Programming*, vol. 72, 1996, pp. 83–100.

[51] HART, P. E., N. J. NILSSON et B. RAPHAEL, «A formal basis for the heuristic determination of minimum cost paths», *IEEE Transactions of Systems Science and Cybernetics*, vol. SSC-4, 1968, pp. 100–107.

[52] HART, P. E., N. J. NILSSON et B. RAPHAEL, «Correction to : A formal basis for the heuristic determination of minimum cost paths», *SIGART Newsletter*, no. 37, 1972, pp. 28–29.

[53] HELMBERG, C. «The m-cost ATSP». Dans *Proceedings of the 7th International IPCO Conference on Integer Programming and Combinatorial Optimization*. Lecture Notes In Computer Science ; Vol. 1610, 1999, pp. 242–258.

[54] HWANG, C.-L. et A. S. M. MASUD, *Multiple Objective Decision Making, Methods and Applications*. Springer Verlag, 1979.

[55] IGNIZIO, J. P., *Goal programming and extensions*. Lexington Books, 1976.

[56] JASZKIEWICZ, A. «Evaluation of multiple objective metaheuristics». Dans *Metaheuristics for Multiobjective Optimization*, X. Gandibleux, M. Sevaux, K. Sörensen, et V. T'kindt, Eds. Springer-Verlag, 2004, pp. 65–89.

[57] JOZEFOWIEZ, N., F. SEMET et E.-G. TALBI, «The bi-objective covering tour problem», *Computers & Operations Research*, vol. 34, 2007, pp. 1929–1942.

[58] JÜNGER, M., G. REINELT et G. RINALDI. «The traveling salesman problem». Dans *Handbooks in Operations Research and Management Science, Network Models*, M. O. Balls, T. L. Magnanti, C. L. Monma, et G. Nemhauser, Eds., vol. 7. North Holland, Amsterdam, 1995, pp. 225–330.

[59] KABADI, S. N. et A. P. PUNNEN. «The bottleneck TSP». Dans *The Traveling Salesman Problem and Its Variations*, G. Gutin et A. Punnen, Eds. Kluwer Academic Publishers, 2002, pp. 697–736.

[60] KATAOKA, S. et S. MORITO, «An algorithm for single constraint maximum collection problem», *Journal of the Operations Research Society of Japan*, vol. 31, 1988, pp. 515–530.

[61] KATAOKA, S., T. YAMADA et S. MORITO, «Minimum directed 1-subtree relaxation for score orienteering problem», *European Journal of Operational Research*, vol. 104, 1998, pp. 139–153.

[62] KAUFMAN, D. E. et R. L. SMITH, «Fastest paths in time-dependant networks for intelligent-vehicle-highway systems application», *IVHS Journal*, vol. 1, 1993, pp. 1–11.

[63] KEENEY, R. L. et H. RAFFIA, *Decisions with Multiple Objectives : Preferences and Value Tradeoffs*. Wiley, Chichester, England, 1976.

[64] KELLER, C. P. et M. F. GOODCHILD, «The multiobjective vending problem : a generalization of the travelling salesman problem», *Environment and Planning B : Planning and Design*, vol. 15, 1988, pp. 447–460.

[65] KORHONEN, P., S. SALO et R. E. STEUER, «A heuristic for estimating nadir criterion values in multiple objective linear progarmming», *Operations Research*, vol. 45, 1997, pp. 751–757.

[66] KOSKI, J., «Defectiveness of weighting method in multicriterion optimization of structures», *Communications in Applied Numerical Mathematics*, vol. 1, 1985, pp. 333–337.

[67] KOUVELIS, P. et S. SAYIN, «Algorithm robust for the bicriteria discrete optimization problem : Heuristic variations and computational evidence», *Annals of Operations Research*, vol. 147, 2006, pp. 71–85.

[68] LAPORTE, G. et S. MARTELLO, «The selective travelling salesman problem», *Discrete Applied Mathematics*, vol. 26, 1990, pp. 193–207.

[69] LAUMANNS, M., L. THIELE et E. ZITZLER, «An efficient, adaptative parameter variation scheme for metaheuristics based on the epsilon-constraint method», *European Journal of Operational Research*, vol. 169, 2006, pp. 932–942.

[70] LAWLER, E. L., «A procedure for computing the $K$ best solutions to discrete optimization problems and its application to the shortest path problem», *Management Science*, vol. 18, 1972, pp. 401–405.

[71] LEE, S. M., *Goal Progamming for Decision Analysis*. Auerbach Publications, 1972.

[72] LEIFER, A. C. et M. B. ROSENWEIN, «Strong linear programming relaxations for the orienteering problem», *European Journal of Operational Research*, vol. 73, 1994, pp. 517–523.

[73] MARTELLO, S. et P. TOTH, «An upper bound for the zero-one knapsack problem and a branch and bound algorithm», *European Journal of Operational Research*, vol. 1, 1977, pp. 169–175.

[74] MARTELLO, S. et P. TOTH, *Knapsack Problems, Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

[75] MAS-COLELL, A., M. D. WHINSTON et J. R. GREEN, *Microeconomic Theory*. Oxford University Press, 1995.

[76] MIETTINEN, K. M., *Nonlinear Multiobjective Optimization*. Kluwer academic, 1999.

[77] MOORE, E. F. «The shortest path through a maze». Dans *Proc. International Symposium on Theory of Switching* (Havard University Press, 1959), vol. 2, pp. 285–292.

[78] MOTE, J., I. MURTHY et D. OLSON, «A parametric approach to solving bicriterion shortest path problems», *European Journal of Operational Research*, vol. 53, 1991, pp. 81–92.

[79] NOON, C. E., J. MITTENTHAL et R. PILLAI, «TSSP+1 decomposition strategy for the vehicle routing problem», *European Journal of Operational Research*, vol. 79, 1994, pp. 524–536.

[80] ORDA, A. et R. ROM, «Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length», *Journal of ACM*, vol. 37, no. 3, 1990, pp. 607–625.

[81] ORDA, A. et R. ROM, «Minimum weight paths in time-dependent networks», *Networks*, vol. 21, 1991, pp. 295–319.

[82] PADBERG, M. W. et G. RINALDI, «Facet identification for the symmetric traveling salesman polytope», *Mathematical Programming*, vol. 47, 1990, pp. 219–257.

[83] PALLOTTINO, S. et M. G. SCUTELLÀ. «Shortest path algorithms in transportation models : Classical and innovative aspects». Dans *Equilibrium and Advanced Transportation Modelling*, P. Marcotte et S. Nguyen, Eds. Kluwer Academic Publishers, 1998, pp. 245–281.

[84] PEARL, J., *Heuristics : Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, Massachusetts, 1984.

[85] PENKY, J. F. et D. L. MILLER. «An exact parallel algorithm for the resource constrained travelling salesman problem with application to scheduling with aggregate deadline». Dans *ACM Eighteenth Annual Computer Science Conference Proceedings* (Pittsburgh, 1990), pp. 208–214.

[86] RALPHS, T. K., M. J. SALTZMAN et M: M. WIECEK, «An improved algorithm for solving biobjective integer programs», *Annals of Operations Research*, vol. 147, 2006, pp. 43–70.

[87] RAMESH, R., Y.-S. YOON et M. H. KARWAN, «An optimal algorithm for the orienteering tour problem», *ORSA Journal on Computing*, vol. 4, 1992, pp. 155–165.

[88] REINELT, G., «TSPLIB, A traveling salesman problem library», *ORSA Journal on Computing*, vol. 3, 1991, pp. 376–384.

[89] RIERA-LEDESMA, J. et J. J. SALAZAR-GONZÁLEZ, «The biobjective travelling purchaser problem», *European Journal of Operational Research*, vol. 160, 2005, pp. 599–613.

[90] SAFER, H. M. et J. B. ORLIN. «Fast approximation schemes for multi-criteria combinatorial optimization». Tech. Rep. Working Paper 3756-95, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[91] SAWARAGI, Y., H. NAKAYAMA et T. TANINO, *Theory of Multiobjective Optimization*. Academic Press, Orlando, FL, 1985.

[92] SAYIN, S. et P. KOUVELIS, «The multiobjective discrete optimization problem : a weighted min-max two-stage optimization approach and a bicriteria algorithm», *Management Science*, vol. 51, 2005, pp. 1572–1581.

[93] SEDGEWICK, R. et J. S. VITTER, «Shortest path in euclidean graphs», *Algorithimica*, vol. 1, no. 1, 1986, pp. 31–48.

[94] STADLER, W., «A survey of multicriteria optimization or the vector maximum problem part I : 1776-1960», *Journal of Optimization Theory and Applications*, vol. 29, 1979, pp. 1–51.

[95] STEUER, R. E., *Multiple Criteria Optimization : Theory, Computation, and Application*. Robert E. Krieger Publishing Company, 1989.

[96] STEUER, R. E., L. R. GARDINER et J. GRAY, «A bibliographic survey of the activities and international nature of multiple criteria decision making», *Journal of Multi-Criteria Decision Analysis*, vol. 5, 1996, pp. 195–217.

[97] TSILIGIRIDES, T., «Heuristic methods applied to orienteering», *Journal of the Operational Research Society*, vol. 35, 1984, pp. 797–809.

[98] ULUNGU, E. L. *Optimisation combinatoire multicritère : détermination de l'ensemble des solutions efficaces et méthodes interactives*. Thèse de doctorat, Faculté des Sciences, Université de Mons-Hainaut, Mons, Belgique, 1993.

[99] ULUNGU, E. L. et J. TEGHEM, «The two-phases method : An efficient procedure to solve bi-objective combinatorial optimization problems», *Foundations of Computing and Decision Sciences*, vol. 20, 1995, pp. 149–165.

[100] WARBURTON, A. R., «Approximation of pareto optima in multiple-objective shortest-path problems», *Operations Research*, vol. 35, 1987, pp. 70–79.

[101] YU, P.-L., *Multiple criteria decision making : concepts, techniques and extensions*. Plenum Press, New York, 1985.

[102] ZILIASKOPOULOS, A. K. et H. S. MAHMASSANI, «Time-dependant shortest path algorithms for real-time intelligent vehicle highway system applications», *Transportation Research Record*, no. 1408, 1993, pp. 94–100.