Column Generation for Bi-Objective Integer Linear Programs : Application to Bi-Objective Vehicle Routing Problems

Boadu Mensah Sarpong

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *3/12/2013* **par :**

Boadu Mensah SARPONG

Column Generation for Bi-Objective Integer Linear Programs : Application to Bi-Objective Vehicle Routing Problems

---

École doctorale et spécialité :
EDSYS : Informatique 4200018

Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS)

Directeur(s) de Thèse :
Christian ARTIGUES et *Nicolas* JOZEFOWIEZ

JURY :

Dominique FEILLET	Professeur, Ecole des Mines de Saint-Etienne	Président du Jury
Daniel VANDERPOOTEN	Professeur, Université Paris Dauphine	Examinateur
Marco LÜBBECKE	Professeur, Univerisité RWTH Aachen	Rapporteur
Clarisse DHAENENS	Professeur, Université Lille 1	Rapporteur
Christian ARTIGUES	Directeur de Recherche, LAAS-CNRS, Toulouse	Directeur de Thèse
Nicolas JOZEFOWIEZ	Maître de Conférences, INSA de Toulouse	Co-Directeur de Thèse

Acknowledgements

I would like to thank my two PhD directors Christian and Nicolas for proposing this thesis topic to me and teaching me how to do academic research the right way. They still believed and stood by me during difficult times when I was making no progress in my work. I'm also grateful to the referees and other members of my PhD defense jury for their corrections, comments, and insightful questions. Special thanks to Professor Tolga Bektaş of the University of Southamption who hosted me for a period of ten weeks during the summer of 2012. The short time I spent with him taught me so much academically and also in developing a better working habit. Thanks to all past and present members of the ROC group at LAAS-CNRS who provided me with a friendly environment for my research. I would finally want to thank my family, friends and loved ones who helped me in diverse ways all of which I cannot state here. I really appreciate your support.

Abstract

Multi-objective optimization deals with finding solutions to problems for which several objectives (or criteria) are considered. Unlike in single objective optimization, the optimal value of a multi-objective problem is a set of points called "the nondominated set". Lower and upper bounds of a multi-objective problem can also be described using sets. For most practical problems, the variables considered in multi-objective optimization represent non fractionable items and thus we talk of multi-objective integer programs. In order to obtain good lower and upper bounds that can be used in the design of exact methods, some problems are usually formulated with an exponential number of decision variables and these problems are solved by column generation. The work of this thesis seeks to contribute to the study of the use of column generation in multi-objective integer linear programming. We do this by studying a bi-objective vehicle routing problem which may be seen as a generalization of several other vehicle routing problems. We propose mathematical formulations for this problem and also find ways to quickly compute lower bounds by column generation. Since the subproblems solved when computing lower bounds have similar structures, we propose intelligent ways of treating some of these subproblems simultaneously rather than independently.

Keywords: Integer linear programming, column generation, multi-objective optimization, combinatorial optimization, vehicle routing

Résumé

L'optimisation multi-objectif concerne la résolution de problèmes pour lesquels plusieurs objectifs (ou critères) contradictoires sont pris en compte. Contrairement aux problèmes d'optimisation ayant un seul objectif, un problème multi-objectif ne possède pas une valeur optimale unique mais plutôt un ensemble de points appelés "ensemble non dominé". Les bornes inférieures et supérieures d'un problème multi-objectif peuvent être également décrites par des ensembles. Dans la pratique, les variables utilisées en optimisation multi-objectif représentent souvent des objets non fractionnables et on parle alors de problèmes multi-objectif en nombres entiers. Afin d'obtenir de meilleures bornes qui peuvent être utilisées dans la conception de méthodes exactes, certains problèmes sont formulés avec un nombre exponentiel de variables de décision et ces problèmes sont résolus par la méthode de génération de colonnes. Les travaux de cette thèse visent à contribuer à l'étude de l'utilisation de la génération de colonnes en programmation linéaires en nombres entiers multi-objectif. Pour cela nous étudions un problème de tournées de véhicules bi-objectif qui peut être considéré comme une généralisation de plusieurs autres problèmes de tournées de véhicules. Nous proposons des formulations mathématiques pour ce problème et des techniques pour accélérer le calcul des bornes inférieures par génération de colonnes. Les sous-problèmes qui doivent être résolus pour le calcul des bornes inférieures ont une structure similaire. Nous exploitons cette caractéristique pour traiter simultanément certains sous-problèmes plutôt qu'indépendamment.

Mot-clés: Programmation linéaire en nombres entiers, génération de colonnes, optimisation multi-objectif, optimisation combinatoire, tournées de véhicules

Contents

List of Algorithms

List of Figures

List of Tables

General Introduction

Context

Many optimization problems encountered in practical applications concern two or more contradictory objectives (or criteria). These problems, called multi-objective optimization problems, are different from classical optimization problems in the sense that the optimal value of a multi-objective problem is a set of points (called "the nondominated set") rather than a unique optimal value. No member of the nondominated set is better than another member over all the objective funcions. Lower and upper bounds of a MOP can also be described using sets. For most practical problems, the variables considered in MOPs represent non fractionable items (eg. number of persons) and thus we talk of multi-objective integer programs (MOIP). MOIP are solved mainly by heuristics and metaheuristics and although these methods are effective, they provide no guarantee of finding the exact nondominated set. Few exact methods have been proposed in the literature for MOIPs having two objectives. Most exact methods for solving optimization problems work by computing a lower bound and an upper bound so that the optimal value of the problem lies between these two values. The quality of these bounds are improved until they are equal (in which case we have the exact optimal value) or until the gap between them becomes reasonably small (in this case, we have a good approximation for the optimal solution together with a measure of quality for the approximation). Due to the role that lower and upper bounds play in solving optimization problems, there is the need to develop good mathematical models and efficient ways of computing such bounds for multi-objective problems.

In the single objective case, a popular way of computing good lower and upper bounds for some classes of integer programs (eg. vehicle routing problems) is by formulating them with an exponential number of variables. It is impractical and sometimes impossible to explicitly list all of the variables (or columns) involved in such formulations and so they are solved by column generation methods. Column generation is an iterative method and the main idea of the method is to decompose an original problem into two main parts namely a restricted master problem and a subproblem. The restricted master problem is a copy of the original problem in which only a few of the variables are kept. The role of the subproblem is to propose new variables to be introduced into the master problem in order to prove the convergence of the method or possibly improve the current optimal value. An iteration of column generation involves solving the linear relaxation of the master problem and then solving the subproblem to verify if it will propose some new variables (through a pricing process) to be added to the master problem. This iterative process ends when the

subproblem proposes no new variables. Although designed for non-integer linear problems, column generation has been a very successful method for solving integer linear programs when it is integrated in a branch-and-bound framework yielding a branch-and-price scheme.

In spite of the importance of column generation, only a few published papers deal with its application to multi-objective problems. The objective of this thesis is to contribute to the study of column generation as applied to multi-objective integer linear programs. We do this through the study of a bi-objective vehicle routing problem. We seek to propose good mathematical formulations for this problem and also find ways of computing good lower bounds by column generation. More precisely, subproblems having similar structures need to be solved when computing lower bound so we seek to propose intelligent ways of treating some of these subproblems simultaneously rather than idependently.

Organization and Contributions

The manuscript is organized into four main chapters. Chapter 1 gives an overview of column generation as applied to single objective vehicle routing problems and our contribution on an original variant. After introducing the different terminologies and formulations for vehicle routing problems, the basic definitions and principles of column generation are discussed. Next, we discuss the different stages of solving the very basic variant of the vehicle routing problem by column generation. A section in this chapter discusses the elementary shortest path problem with resource constraints. Indeed, this problem appears as a subproblem when solving several variants of the vehicle routing problem by a column generation method. In the last section of this chapter, we present an original variant of the vehicle routing problem and a way to solve it by a column generation method. The main interest of this problem stems from the subproblem we encounter. We propose a new dominance relation when solving the subproblem by a dynamic programming algorithm.

In Chapter 2 we give a review of multi-objective optimization. We discuss the interest of multi-objective optimization problems, the different approaches used in managing the multiple objectives, as well as different solution approaches. In particular, we review some popular exact methods for bi-objective integer programs. We also present some quality measures used in evaluating approximation methods and the approximate solutions they produce.

Chapter 3 discusses how good lower and upper bounds can be computed for bi-objective integer linear programs. The principle is to use sets of points having certain properties in the definition of lower and upper bounds. We therefore refer to lower and upper bound sets. We propose a generic column generation algorithm for computing a lower bound set for a problem when it is formulated with an exponential number of columns. The main idea used is to first convert the bi-objective program into single objective through a scalarization method. We then solve linear relaxations of the resulting single objective problem several times by varying the necessary parameters. We show that regardless of the scalarization method used, the subproblems that need to be solved when computing the members of a lower bound set have similar structures. Due to this, we propose different strategies to take advantage of the similar subproblem structures.

In order to test the different ideas presented in Chapter 3, an application problem is presented in Chapter 4. The problem is a generalization of the covering tour problem

namely the bi-objective multi-vehicle covering tour problem. Two different formulations are presented for this problem and different column generation approaches adapted to each formulation are tested. Summary of the experiments performed and discussion of the results obtained are presented at the end of the chapter. The results show the quality of the proposed formulations and the interest of the intelligent column generation techniques designed in Chapter 3 on this problem.

The manuscript ends with some general concluding remarks as well as future research directions in the short term, middle term, and long term.

Notation

Throughout this manuscript, we will usually need to compare vectors of real numbers. In general, there is no canonical way of doing this and so we need to clarify the notations used. Given a positive integer $n \geq 2$ and any two arbitrary vectors of real numbers $x, y \in \mathbb{R}^n$, we will use the following notation :

- $x = y$ if $x_i = y_i$ for $i = 1, \ldots, n$,
- $x < y$ if $x_i < y_i$ for $i = 1, \ldots, n$,
- $x \leqq y$ if $x_i \leq y_i$ for $i = 1, \ldots, n$,
- $x \leq y$ if $x \leqq y$ but $x \neq y$.

We will also write :

- $\mathbb{R}^n_>$ for $\{x \in \mathbb{R}^n : x > 0\}$,
- $\mathbb{R}^n_\geq$ for $\{x \in \mathbb{R}^n : x \geq 0\}$,
- $\mathbb{R}^n_\geqq$ for $\{x \in \mathbb{R}^n : x \geqq 0\}$.

Chapter 1

Column Generation for Vehicle Routing Problems

1.1 Introduction

In this chapter, we discuss the application of column generation to vehicle routing problems (VRPs). In Section 1.2, we present terminologies for VRPs as well as the different formulations proposed in the literature. We also give an overview of some general methods used in solving VRPs. Section 1.3 follows with a discussion on column generation as applied to VRPs by using the basic variant of the VRP (the capacitated VRP or CVRP) as an example. The elementary shortest path problem with resources constraints (ESPPRC) which usually appears as a subproblem in vehicle routing problems solved by column generation is discussed in Section 1.4. We present a specific vehicle routing problem and discuss its solution by column generation in Section 1.5. The interest of this problem stems from the fact that the subproblem encountered presents a challenge to dynamic programming algorithms used in its solution. We propose some ideas that can be used to overcome this challenge. Section 1.6 ends the chapter with some concluding remarks.

Related Publications. The ideas presented in Section 1.5 is the core of an article which is currently under prepartion for submission to an international journal. These ideas were developed following a 10 week research visit to the University of Southampton in the United Kingdom. The research was performed at the Centre for Operational Research, Management Sciences and Information Systems (`http://www.southampton.ac.uk/cormsis/`). During this stay, the author worked under the supervision of Professor Tolga Bektaş at the School of Management Sciences (`http://www.southampton.ac.uk/management/about/staff/tb12v07.page#background`).

1.2 Vehicle Routing Problems

Vehicle Routing Problems (VRP) are concerned with the optimal routing of a fleet of vehicles from one or several depots in order to deliver services to a number of geographically scattered customers. The first of such problems was presented by Dantzig and Ramser

(1959). Their problem was to design optimal routes to deliver gasoline from a bulk terminal to a large number of service stations. Since that time, several variants of VRP have been proposed to address problems encountered in real-world transportation systems. The different variants stems from the different kind of services that may be offered as well as the different operational constraints that need to be respected. In general, two main types of services namely *delivery* and *collection* are provided. Numerous operational constraints appear in real applications. Some of the most common ones are limits on the capacity of a vehicle, maximum limits placed on the length of a route, specific periods during which a customer may be visited, an order in which customers may be visited and the type of service that may be provided for each customer (only delivery, only collection, both delivery and collection).

In the very basic version of the VRP (the capacitated VRP or CVRP), the main operational constraint is that the total amount of goods delivered by a single vehicle cannot exceed a fixed capacity. If in addition to delivering goods, the vehicle may also collect goods from the customers they visit, then we talk of the VRP with pick up and delivery (VRPPD). Other basic variants of the CVRP are the distance-constrained capacitated VRP (DCVRP) in which a maximum limit is placed on the length of each route, the VRP with time windows (VRPTW) in which each customer should be visited within a specific time period, and the VRP with backhauling (VRPB) in which for each customer a visiting vehicle either deliver goods, or collect goods, but not both. A representation of the interconnections between the basic variants of the VRP is given in Figure 1.1. Most times, we refer to the CVRP as the *classical* VRP because of the central role it plays in the classification of VRPs. Indeed, many algorithms developed for the CVRP can be adapted to take into account other complicated operational constraints Laporte (2007). The variants of VRP studied in recent times combine several of the constraints stated above and several other ones.

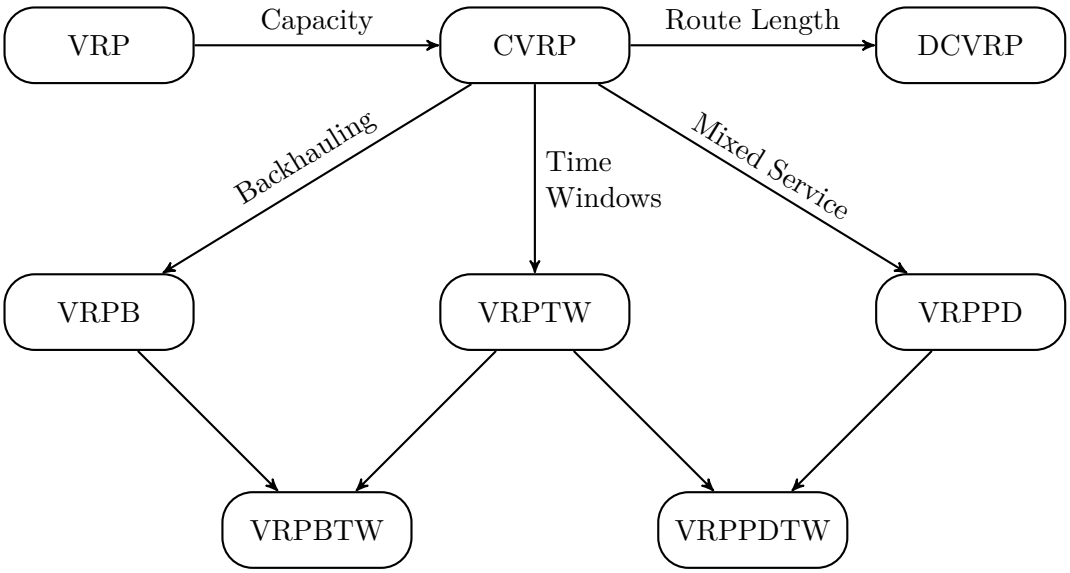


Figure 1.1: The basic variants of vehicle routing problems and their interconnections (Toth and Vigo, 2002).

The solution of a VRP is a set of routes (one for each vehicle) each of which starts and ends at a depot such that the demands (or requests) of all customers are satisfied and the global cost of transportation is minimised while respecting all operational constraints. The global cost of transportation may be measured based on different criteria such as the travel distance and/or time, and other costs linked to the use of a vehicle. The VRP is an NP-hard problem since it lies at the junction of two NP-hard problems namely the *Bin Packing Problem (BPP)* and the *Traveling Salesman Problem (TSP)*. For each vehicle (or route), we need to determine the customers it will visit (BPP) and also give an order in which they are visited (TSP). Sometimes, measuring the cost of a route is a difficult problem in itself.

Applications of VRPs exist in the domains of solid waste collection, street cleaning, school bus routing, dial-a-ride systems, transportation of handicapped persons, routing of salespeople, and maintenance of units (Toth and Vigo, 2002). Others are in the delivery of newspapers to retailers, of food and beverages to grocery stores and in the collection of milk products from dairy farmers (Golden et al., 2002). A general survey on VRPs can be found in Toth and Vigo (2002).

1.2.1 Formulations

There are three main types of model for VRPs found in the literature Toth and Vigo (2002). These are *vehicle flow formulations*, *commodity flow formulations* and *set partitioning formulations*. We may also differentiate between *two-index* and *three-index* flow formulations. We demonstrate the principles of vehicle flow formulations and set partitioning formulations by using the CVRP as an example. Most solution methods are based on one of these two. Only very few methods presented in the literature are based on commodity flow formulations. For this reason, commodity flow formulations are not discussed in this section.

Let $G = (V, A)$ be a complete graph where $V = \{v_0, \ldots, v_n\}$ is a set of nodes and $A = \{(v_i, v_j) : v_i, v_j \in V \text{ and } i \neq j\}$ is a set of arcs. Node v_0 is the depot where all routes must start and also end whereas nodes $v_1, \ldots, v_n$ represent n customer locations. A non-negative cost matrix $D = (d_{ij})$ which satisfies the triangle inequality is defined on set A. Each customer has a fixed demand of q_i which must fully be satisfied by a route that visits it. Let m be the total number of available vehicles. The CVRP consists in designing a set of at most m routes with total minimum cost such that each customer is visited by exactly one route and the sum of the demands of customers visited by a single route does not exceed the vehicle capacity, Q.

Vehicle Flow Formulations

In this type of formulation, a binary variable x_{ij} is used to indicate whether the arc (v_i, v_j) is used by a vehicle in the optimal solution ($x_{ij} = 1$), or not ($x_{ij} = 0$). Let u_i be the load of

a vehicle after visiting customer v_i. A vehicle flow formulation for the CVRP is given by :

$$\text{Minimize} \quad \sum_{(v_i,v_j)\in E} d_{ij}x_{ij} \tag{1.1}$$

$$\text{subject to:} \quad \sum_{(v_0,v_j)\in A} x_{0j} \leq m\,, \tag{1.2}$$

$$\sum_{(v_0,v_j)\in A} x_{0j} - \sum_{(v_i,v_0)\in A} x_{i0} = 0\,, \tag{1.3}$$

$$\sum_{v_i\in V} x_{ij} = 1 \qquad (v_j \in V\backslash\{v_0\})\,, \tag{1.4}$$

$$\sum_{v_j\in V} x_{ij} = 1 \qquad (v_i \in V\backslash\{v_0\})\,, \tag{1.5}$$

$$\sigma_i + q_j - \sigma_j + Qx_{ij} \leq Q \qquad (v_i, v_j \in V\backslash\{v_0\}, i \neq j)\,, \tag{1.6}$$

$$q_i \leq \sigma_i \leq Q \qquad (v_i \in V\backslash\{v_0\})\,, \tag{1.7}$$

$$\sigma_i \geq 0 \qquad (v_i \in V\backslash\{v_0\})\,, \tag{1.8}$$

$$x_{ij} \in \{0,1\} \qquad (v_i, v_j \in V)\,. \tag{1.9}$$

The objective of minimizing the total cost of the routes is given in (1.1). Constraint (1.2) limits the number of arcs that leaves the depot whereas (1.3) ensures that the number of arcs that enter the depot is the same as the number of those that leave it. Together, these two constraints ensure that not more than m routes are constructed. The *indegree* and *outdegree* constraints given by (1.4) and (1.5) respectively ensure that exactly one arc leaves and enter each customer node. Constraints (1.6) and (1.7) are subtour elimination constraints which were originally proposed for the TSP by Miller et al. (1960). These constraints eliminate routes that are not connected to the depot and also enforce the maximum capacity limits of vehicles. The domain of definition for the variables in σ and x are given by Constraints (1.8) and (1.9), respectively.

Vehicle flow formulations are perhaps the most used type of formulation for VRPs. They are most suited for cases where the cost of a route can be expressed as the sum of the costs of the edges it uses. They can, however, not be used for cases where the cost of a route depends not just on the individual edges it uses but rather on the total sequence of the edges.

Set Partitioning Formulations

A set partitioning formulation for a VRP uses an exponential number of variables each of which is associated to a feasible route. A route is said to be feasible if it satisfies all operational constraints. For the CVRP, a feasible route is simply a Hamiltonian cycle that connects the depot to a subset of customer nodes in such a way that the sum of the demands of customers on the cycle does not exceed the capacity of the vehicle. This type of model for the VRP was first proposed by Balinski and Quandt (1964).

Let Ω be the set of all feasible routes. Each feasible route $k \in \Omega$ has an associated cost c_k which is given by the cost of the arcs it uses. For each customer node v_i, we define a binary variable a_{ik} which takes a value of 1 if route k visits node v_i and $a_{ik} = 0$ if this is

not the case. A binary variable θ_k is defined for each route $k \in \Omega$ to determine if the route is selected in the optimal solution ($\theta_k = 1$) or not ($\theta_k = 0$). A set-partitioning formulation for the CVRP is given by :

$$\text{Minimize} \quad \sum_{k \in \Omega} c_k \theta_k \tag{1.10}$$

$$\text{subject to :} \quad \sum_{k \in \Omega} \theta_k \leq m \,, \tag{1.11}$$

$$\sum_{k \in \Omega} a_{ik} \theta_k = 1 \qquad (v_i \in V \backslash \{v_0\}) \,, \tag{1.12}$$

$$\theta_k \in \{0, 1\} \qquad (k \in \Omega) \,. \tag{1.13}$$

The solution of this formulation is a subset of feasible routes in Ω that minimizes the objective function (1.10) while ensuring that each customer node is visited by exactly one selected route (1.12). The cardinality of the subset must not exceed m as specified by Constraint (1.11). If m is reasonably large (eg. $m \geq n - 1$), then this constraint may be dropped. The above formulation is very general and can easily take into account several other operational constraints on a single route. We only need to redefine what a feasible route represents in such situations.

Since we assume that the cost matrix D satisfies the triangle inequality, we can transform the above set partitioning formulation into a *set covering* formulation by replacing (1.12) with

$$\sum_{k \in \Omega} a_{ik} \theta_k \geq 1 \quad (v_i \in V \backslash \{v_0\}) \,. \tag{1.14}$$

The optimal objective value of both the set partitioning and the set covering models are the same. Indeed, if the optimal solution of a set covering model contains two or more routes that visit the same customer then this customer may be kept in just one of these routes and removed from all the rest. The resulting solution will still be feasible, and thanks to the triangle inequality, the new objective value will be less than or equal to that of the previous solution. One main advantage of using a set covering formulation instead of a set partitioning formulation is that the dual space in a set covering formulation is much smaller since the dual variables associated to Constraints 1.14 are restricted to only nonnegative values. This means that methods that rely heavily on these dual variables becomes much faster.

A general disadvantage of both set partitioning and set covering models is that, the number of decision variables is huge. For example, in a simple instance of a CVRP with $n = 15$ we have about $15!/2 = 653,837,184,000$ routes (decision variables). It is, thus, impractical to list all of these variables and so methods that dynamically introduce the variables like *column generation* (discussed in Section 1.3) are employed.

1.2.2 Solution Methods

The Vehicle Routing Problem is one of the most studied combinatorial optimization problems. Different solution approaches ranging from exact methods, heuristics and metaheuristics have been proposed to tackle the numerous variants of the problem. Due to the difficulty of VRPs, most research effort concentrate on heuristics and metaheuristics

rather than on exact methods (Laporte, 2007). This is also probably because it is easier to adapt these methods to different variants.

The main exact approaches for VRPs rely on relaxations and implicit enumeration techniques like branch-and-bound. Some of the most popular exact algorithms are branch-and-cut in which cutting planes are combined with branch-and-bound, branch-and-price in which column generation is incorporated in a branch-and-bound framework, and Lagrangian relaxation. Among these, branch-and-cut algorithms are the most popular. Reviews on exact methods can be found in Laporte and Nobert (1987); Toth and Vigo (1998).

As already indicated, the literature on heuristics and metaheuristics for VRPs is enormous. We only list just a few of them here. Dantzig and Ramser (1959) proposed the first heuristic approach when they introduced the VRP. A greedy heuristic namely the savings algorithm was later proposed by Clarke and Wright (1964) and this has gone on to become a very popular heuristic. The popularity of the savings algorithms is not due to its accuracy but rather its speed and the simplicity in implementing it (Laporte, 2007). Different types of metaheuristics have also been proposed to solve the CVRP and its variants. Examples are, *local search* algorithms like *record-to-record travel* (Dueck, 1993), *tabu search* (Glover, 1986), *variable neighbourhood search* (Mladenović and Hansen, 1997), *very large neighbourhood search* (Ergun, 2001) and *adaptive large neighbourhood search* (Pisinger and Ropke, 2007). Some *population search* metaheuristics are *genetic algorithms* (Holland, 1975), *memetic algorithms* (Prins, 2004) and *adaptive memory procedure* (Rochat and Taillard, 1995). Other metaheuristics are *learning mechanisms* like *neural networks* and *ant colony optimization* (Schumann and Retzko, 1995; Ghaziri, 1991). Cordeau et al. (2005) presents a survey on metaheuristics for VRPs.

1.3 Column Generation

The birth of column generation traces back to the early 1960's (Dantzig and Wolfe, 1960; Gilmore and Gomory, 1961, 1963) with its first applications to VRPs appearing some years afterwards (Appelgren, 1969, 1971). Its role in integer programming is to compute dual bounds (i.e. lower bounds for minimization problems and upper bounds for maximization problems). In order to ensure the integrality of solutions, it may be necessary to embed column generation in a branch-and-bound framework. We thus obtain the solution approach called *branch-and-price*. In this section, we will concentrate mainly on computing lower bounds of VRPs by column generation. A complete discussion of column generation and branch-and-price can be found in Barnhart et al. (1998) or in a recent book on the subject (Desaulniers et al., 2005). A very good paper that describes the application of column generation to VRPs is Feillet (2010).

Lets consider the set covering model described by (1.10), (1.11), (1.13) and (1.14). Its

linear programming (LP) relaxation is given by :

$$\text{Minimize} \quad \sum_{k\in\Omega} c_k\theta_k \tag{1.15}$$

$$\text{subject to :} \quad \sum_{k\in\Omega} \theta_k \;\leq\; m\;, \tag{1.16}$$

$$\sum_{k\in\Omega} a_{ik}\theta_k \;\geq\; 1 \qquad (v_i \in V\backslash\{v_0\})\;, \tag{1.17}$$

$$\theta_k \;\geq \hat{\text{A}}\check{\text{a}}\; 0 \quad (k \in \Omega)\;. \tag{1.18}$$

Note: In the above formulating, we consider the variables in θ_k as being integers instead of binary in order avoid constraints of the form $\theta_k \leq 1$ in the dual formulation. It is clear, however, that this does not change the optimal value since any solution with $\theta_k \geq 2$ for a given $k \in \Omega$ will not be optimal.

Let π_0 be the non-positive dual variable associated with Constraint (1.16) and π_i for $v_i \in V\backslash\{v_0\}$ be non-negative dual variables associated with Constraints (1.17). The dual formulation of the relaxed set covering model is

$$\text{Maximize} \quad m\pi_0 + \sum_{v_i\in V\backslash\{v_0\}} \pi_i \tag{1.19}$$

$$\text{subject to :} \quad \pi_0 + \sum_{v_i\in V\backslash\{v_0\}} a_{ik}\pi_i \;\leq\; c_k \quad (k \in \Omega)\;, \tag{1.20}$$

$$\pi_0 \;\leq\; 0\;, \tag{1.21}$$

$$\pi_i \;\geq\; 0 \quad (v_i \in V\backslash\{v_0\})\;. \tag{1.22}$$

Suppose that the set of feasible routes Ω is manageable in the sense that all of its members can easily be listed and the resulting formulation can also be easily solved. Solving formulation (1.15 – 1.18) by the simplex algorithm requires that in each iteration, a non-basic variable with negative *reduced cost* is *priced-out* and enter the basis. The reduced cost of variable θ_k is defined as

$$\tilde{c}_k = c_k - \pi_0^* - \sum_{v_i\in V\backslash\{v_0\}} a_{ik}\pi_i^*\;, \tag{1.23}$$

where π_0^* and π_i^* for $v_i \in V\backslash\{v_0\}$ are optimal dual values at the current iteration. In selecting a non-basic variable to enter the basis, the simplex algorithm computes the reduced cost of all the non basic variables and picks the one having the most negative value. We call this technique *explicit pricing* and it is viable only when the set Ω is manageable. For problems in which there is a huge number of variables, explicit pricing becomes computationaly expensive and so we resort to a column generation method.

1.3.1 Basic Definitions and Principles

In column generation terminology, we refer to formulation (1.10), (1.11), (1.13) and (1.14) as the *integer programming master problem* (IPM) and its linear programming relaxation

given in (1.15 – 1.18) is called the *linear programming master problem* (LPM). We denote the dual formulation of LPM detailed in (1.19 – 1.22) as DLPM. The basic principle of column generation mimics the simplex algorithm but with two main differences. Firstly, since it is impractical to explicitly list all the members of Ω, we work with a reasonably small subset $\Omega_1 \subseteq \Omega$ for which LPM is primal feasible. The restriction of LPM to a subset $\Omega_1 \subseteq \Omega$ is called a *restricted linear programming master problem* (RLPM). Secondly, pricing of non-basic variables is now done implicitly by solving an auxilliary problem called the *subproblem* (SP). The subproblem is given by :

$$\text{minimize} \quad c_k - \pi_0^* - \sum_{v_i \in V \setminus \{v_0\}} a_{ik}\pi_i^* \quad \text{subject to} \quad k \in \Omega \setminus \Omega_1 \; . \tag{1.24}$$

The Algorithm

The algorithm starts by formulating a RLPM. At each iteration, the RLPM is first solved to obtained an optimal solution and a corresponding vector of dual values. Next, the subproblem is solved to see if any non-basic variables can be priced out in order to improve the current objective value. If the optimal value of the subproblem is non-negative then the current objective value of RLPM can not be improved and so an optimal solution of LPM has been found. If this is not the case then one or more columns having negative reduced costs are introduced into the RLPM and the process continues. Algorithm 1.1 summarizes a column generation method.

Algorithm 1.1 A Column Generation Algorithm

1: Generate an initial set of columns Ω_1 and formulate RLPM.
2: **repeat**
3: Solve RLPM(Ω_1).
4: Solve subproblem and let Λ be the set of columns found.
5: Set $\Omega_1 \leftarrow \Omega_1 \cup \Lambda$.
6: **until** $\Lambda = \emptyset$.

Primal and Dual Bounds

We remind ourselves that column generation works on a RLPM but not directly on LPM since the size of Ω is very large. Given that a RLPM is obtained by depriving the LPM of some of its variables, every feasible solution of RLPM is also a feasible for LPM and hence an upper bound on LPM. The situation is however different in the dual space. Removing some variables from LPM (to formulate RLPM) corresponds to removing some constraints from DLPM. As a result, a feasible (even an optimal) solution for DRLPM may not be feasible for DLPM. Thus if the algorithm starts with a feasible RLPM then LPM remains primal feasible throughout the whole process but its dual feasibility is proved only at convergence. Let $\bar{z}$ denote the optimal objective value of RLPM. If we have an upper bound κ on the optimal objective value z^*_{LPM} of LPM then a lower bound may also be computed at each iteration of the algorithm. Let $\tilde{c}^*$ be the optimal solution of the subproblem, then

the following inequality holds

$$\bar{z} + \kappa\tilde{c}^* \;\leq\; z^*_{LPM} \;\leq\; \bar{z}\,. \tag{1.25}$$

This means that the algorithm may be stopped earlier before it converges since it is possible to verify the solution quality at any given time.

Validity and Convergence of the Algorithm

The validity and convergence of column generation relies on the following simple property of the models involved (see Figure 1.2).

- The feasible space of RLPM is a subspace of the feasible space of LPM whereas the feasible space of DLPM is a subspace of that of DRLPM.

This means that any feasible solution for RLPM is also a feasible for LPM and the corresponding objective value is an upper bound on the optimal value of LPM. Also, the feasible space of DLPM is a relaxation of the feasible space of DRLPM and so not all feasible solutions of DRLPM are feasible for DLPM. The optimal value of DRLPM, however, provides a lower bound on the optimal value of DLPM. In addition, if the optimal solution of DRLPM is feasible for DLPM, then it is also the optimal value of DLPM. Introducing new variables (or columns) in the RLPM corresponds to adding constraints to the DRLPM. Given that there is a finite number of elements in Ω, the algorithm converges after a finite number of iterations and so it is an exact method for the LPM. The hope is that the LPM will become both primal and dual feasible after introducing a reasonable small number of columns. In worse case, however, the method will only converge after all feasible columns have been added.

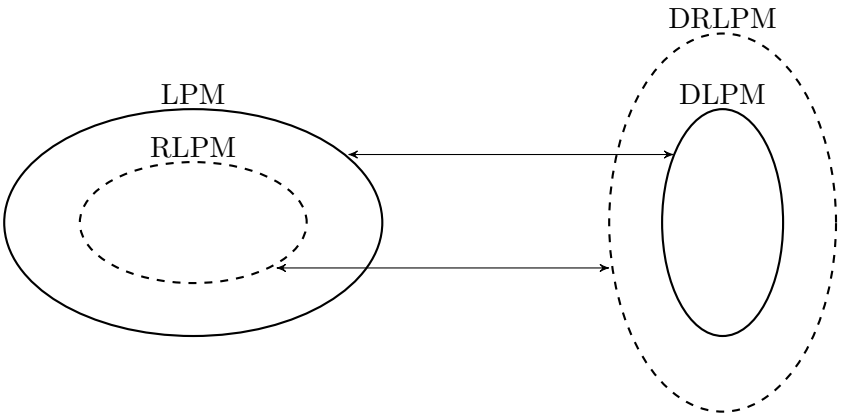


Figure 1.2: Interactions between LPM, RLPM, DLPM and DRLPM.
Notes: RLPM is a subspace of LPM and so the optimal value of RLPM is an upper bound on the optimal value on LPM. On the other hand, DLPM is a subspace of DRLPM and so the optimal value of DRLPM is a lower bound on the optimal value of DLPM.

1.3.2 Implementation and Other Issues

Column generation (branch-and-price when incorporated in a branch-and-bound framework) has become an important approach for solving vehicle routing problems. Yet, accurately implementing the method is often a difficult task due to the many computational tricks that can affect its the performance (Feillet, 2010). After formulating a master problem, an important aspect of column generation is to identify and model the subproblem in order to solve it efficiently. Indeed, solving the RLPM is often a relatively easier task than solving the corresponding subproblem and in most implementations, over 90% of the total computational time is spent on solving the subproblem. The specific subproblem encountered is problem dependent and sometimes it can be formulated differently for the same master problem. For example, the subproblems encountered when solving bin packing problems by column generation are usually variants of the knapsack problem whereas those encountered in vehicle routing problems are usually variants of the traveling saleman and the shortest path problems.

Convergence Issues

Column generation, just like several other iterative methods, suffers from slow convergence issues. Vanderbeck (2005) discusses three main convergence related issues in column generation namely the *heading-in effect*, the *plateau effect*, and the *tailing-off effect* (see Figure 1.3). The heading-in effect concerns the early stages of the algorithm. At these stages, the columns in the RLPM is usually not able to provide useful dual information for the subproblem and so irrelevant columns are added and poor lower bounds are produced during the first iterations. The plateau-effect is used to describe stages of the algorithm where the objective value of the RLPM fails to improve after several iterations. This is usually caused by degeneracy in the RLPM and hence multiple solutions for the DRLPM. The tailing-off effect which is the most serious of all the three effects happens towards the final stages of the algorithm. The objective value of RLPM improves only very slightly at each iteration. Different ideas have been proposed to address these issues and some of them are discussed in what follows.

Speedup Mechanisms

Without speedup mechanisms, it is almost impossible to obtain quality solutions by column generation in reasonable times (Desrosiers and Lübbecke, 2005). Over the years, researchers have proposed different strategies to tackle the main convergence issues discussed in the previous paragraph. Intelligently choosing an initial set of columns for the RLPM can help in reducing the heading-in effect. A good initial set of columns is one that is representative enough of the whole set of columns of the problem. In general, the initialization is done by using heuristics or artificial columns. For example, Lübbecke and Desrosiers (2005) propose the use of specially designed heuristics to compute the initial set of columns. We note however that even knowledge of the optimal solution of IPM does not provide very useful information for solving the LPM by column generation (Lübbecke and Desrosiers, 2005). In most cases, it is rewarding to invest in finding a good set of initial initial columns. Both the plateau and tailing-off effects result from the random oscillation of the dual

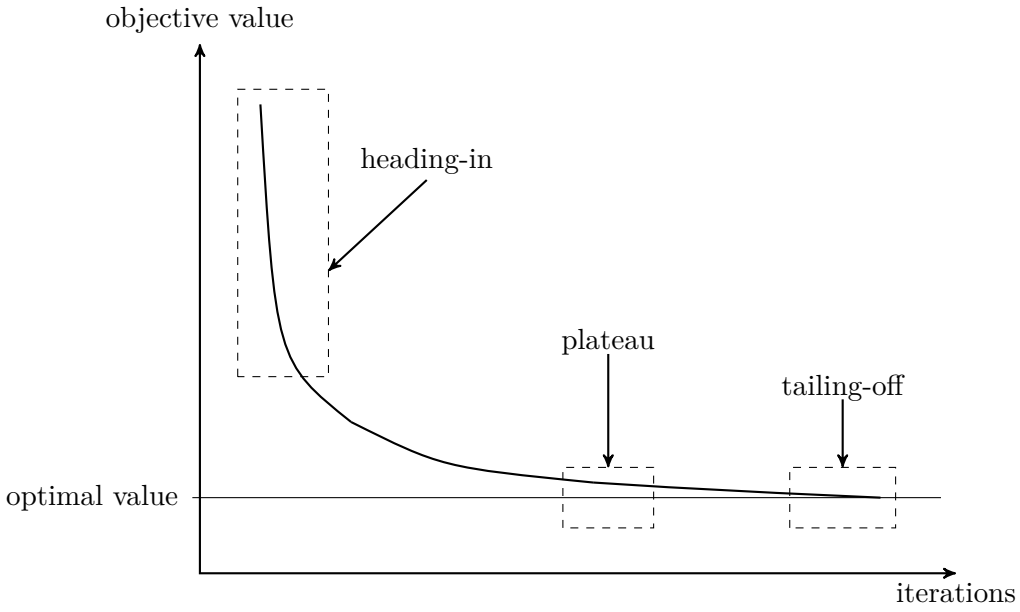


Figure 1.3: Some convergence related issues of a column generation method.

values of RLPM. For this reason, several strategies have been proposed to minimize the oscillations and thus *stabilize* the column generation method. A very popular way of achieving stabilization is to define boxes around previous dual values and also modify the RLPM in such a way that the feasible dual space is limited to the area defined by the boxes. This is the idea of the BOXSTEP method by Marsten et al. (1975). Another technique used to obtain stability is to adapt the RLPM in order to penalize the distance that separates a dual solution from the previous optimal dual solution (see Kim et al. (1995)). The stabilization technique proposed by du Merle et al. (1999) is a combination of the two previous techniques. Other speedup mechanisms concern the intelligent management of the columns. A general technique used to decrease the number of iterations needed by the algorithm to converge is to return several columns with negative reduced cost to the RLPM. This is the most widely used speedup mechaninsm and it is easy to implement when the subproblem is solved by dynamic programming (Desrosiers and Lübbecke, 2005). Sometimes it is useful to drop (delete) some columns from the RLPM when the number it contains becomes very huge. An extensive discussion of these speedup mechanisms and many more can be found in Desaulniers et al. (2002); Vanderbeck (2005); Desrosiers and Lübbecke (2005); Briant et al. (2008).

1.4 The Elementary Shortest Path Problem with Resource Constraints

The most common type of subproblem encountered when solving vehicle routing problems with column generation is a variant of the Shortest Path Problem (SPP) with some side constraints. The most common of these is the Elementary Shortest Path Problem with

Resource Constraints (ESPPRC). The ESPPRC consists in finding a path with minimum total cost within a graph in such a way that some limits on resource consumptions are respected and each node of the graph is visited at most once by the path.

More formally, let us consider an extension of graph G given by $G' = (V', A')$ where $V' = V \cup \{v_{n+1}\}$ is the new set of nodes obtained by adding a duplicate v_{n+1} of the depot v_0. The new set of arcs A' is obtained from the orignal set A by redirecting all arcs coming into the orginal depot v_0 to the duplicate depot v_{n+1}. Let $R \geq 1$ be the number of resources and $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_R)$ be a vector that limits the maximum consumption of each resource. For each arc $(v_i, v_j) \in A'$ we denote the arc cost by $\hat{d}_{ij} \in \mathbb{R}$ and the quantity of resources consumed along the arc by $w = (w_{ij}^1, \ldots, w_{ij}^R)$. For each resource r, we assume that the values of w_{ij}^r satisfy the triangle inequality. A possible mathematical model for the ESPPRC (presented by Feillet et al. (2004)) is the following :

$$\text{Minimize} \quad \sum_{(v_i,v_j)\in A} \hat{d}_{ij} x_{ij} \tag{1.26}$$

$$\text{subject to :} \quad \sum_{(v_0,v_j)\in A'} x_{0j} = 1 \,, \tag{1.27}$$

$$\sum_{(v_i,v_{n+1})\in A'} x_{i,n+1} = 1 \,, \tag{1.28}$$

$$\sum_{(v_i,v_j)\in A'} x_{ij} - \sum_{(v_j,v_i)\in A'} x_{ji} = 0 \qquad (v_i \in V \backslash \{v_0, v_{n+1}\}) \,, \tag{1.29}$$

$$t_i^r + w_{ij}^r - t_j^r + M x_{ij} \leq M \qquad r \in \{1, \ldots, R\},\ (v_i, v_j) \in A' \,, \tag{1.30}$$

$$0 \leq t_i^r \leq W_R \qquad r \in \{1, \ldots, R\},\ v_i \in V' \,, \tag{1.31}$$

$$x_{ij} \in \{0, 1\} \qquad (v_i, v_j) \in A' \,, \tag{1.32}$$

where x_{ij} variables represent flow in the graph, t_i^r is the total amount of the resource r consumed by a partial path after visiting node $v_i \in V'$, and M is a big number. The objective function (1.26) minimizes the total cost of the path. Constraint (1.27) ensures that only one arc leaves the source node whereas (1.28) ensures that only one arc enters the destination node. Flow conservation at the other nodes is achieved through (1.29). Constraints (1.30) update the total amount of consumed resources if an arc is selected in the path and also ensure sub-tour elimination, (1.31) are to ensure the consumtion of resources do not exceed their limits, and (1.32) are domain definitions. The ESPPRC is an NP-hard problem in the strong sense (Dror, 1994).

1.4.1 Overview of Solution Methods

In the literature, the ESPPRC has been solved by different approaches including branch-and-cut, constraint programming, Lagrangian relaxation and dynamic programming. Nevertheless, the preferred method when it is encountered as a subproblem in a column generation approach is by dynamic programming mainly because several columns having negative reduced costs can be returned at each iteration. Until recently, the shortest path problem with resource constraints (SPPRC) which is a relaxed version of the ESPPRC has been the main approach used in solution methods based on column generation. In some cases, these approaches resulted in optimal solutions in reasonable times. For example,

Desrochers et al. (1992) successfully applied it in solving the subproblem encountered in the VRPTW. In other cases, however, the elementary condition cannot be overlooked since relaxing it results in poor lower bounds that cannot be practically embedded in a branch-and-bound framework to produce optimal results. The main principle of dynamic programming algorithms used to solve the ESPPRC is to associate with each possible partial path, a label which indicates the consumption of resources and eliminate labels that cannot lead to the optimal solution with the help of *dominance rules*. Due to the exponential number of possible partial paths, the viability of these algorithms depends on their ability to identify labels that cannot lead to the optimal solution as early as possible. Two main classes of these algorithms are the *label setting* algorithms and *label correcting* algorithms. Label setting algorithms are based on the classical Dijkstra's algorthm whereas label correcting algorihms are extensions of the Bellman-Ford algorithm.

Feillet et al. (2004) proposed an exact algorithm for the ESSPRC which is based on an idea originally described by Beasley and Christofides (1989) on how to find elementary paths in the context of the SPPRC. Beasley and Christofides (1989) idea was to associate with each label an extra resource for each node of V'. The initial value of the resource in a label is 0, and it is set to a 1 when the label visits the node. In this way, it is possible to eliminate multiple visits to the same node since there will not be enough resources for more than one visit. Beasley and Christofides (1989) did not test this idea since they believed that with the increase in the number of resources (one for each node), the algorithm will be impractical for reasonable instances. Feillet et al. (2004) extended this idea by redefining the significance of the extra resource added to a label for each node. They set a value to 1 to indicate that a node is unreachable (can no longer be visited) whether because it has already been visited by the label or because other resource consumptions does not allow the label to visit it. In this way, they were able to identify unwanted partial labels more quickly and solved several VRPTW instances.

Instead of increasing the number of resources as it was done by Feillet et al. (2004), Chabrier (2006) rather simply forbids that a label revisits a node it has already visited. He then proposes a set of dominance rules that work at different levels in order to ensure that any label that can lead to an optimal path is not eliminated.

Recent improvements to dynamic programming algorithms for solving the ESPPRC have seen strategies like bi-directional search (Righini and Salani, 2006), and a dynamic management of associating resources to nodes in order to avoid multiple visits to a node (Boland et al., 2006; Righini and Salani, 2008). Although both of these have been proven to improve the performance of the algorithms, the later produces the better results. Next, we describe a dynamic programming algorithm which generalises the exact method by Feillet et al. (2004) and other methods developed for the ESPPRC.

1.4.2 The Decremental State Space Relaxation Algorithm

Righini and Salani (2008) developed the Decremental State Space Relaxation Algorithm (DSSR) around the same time that Boland et al. (2006) proposed the General State Space Augmenting Algorithm (GSSAA). In spite of the choice of names, both algorithms express the same idea. In this thesis, we will use the name chosen by Righini and Salani (2008) when refering to the algorithm. Instead of associating a resource to each node of V as

done by Feillet et al. (2004), the principle of DSSR is to do this for only a subset of nodes $V^* \subseteq V'$ (denoted as the *critical set*) which are likely to be visited more than once by an optimal path. That is, the nodes of V^* can be visited at most once on any given path whereas those in $V' \backslash V^*$ may be visited more than once. Thanks to the limited amounts of the resources, infinite loops are avoided. Naturally, the algorithm starts with $V^* = \emptyset$ which corresponds to solving the relaxed version, the SPPRC. If the optimal solution of this relaxed version is elementary (ie. visits no node more than once), then it is also the optimal solution of the elementary version. If this is not the case, then one or more nodes that are visited more than once by the optimal path are identified and added to V^*. The algorithm is repeated with the updated critical set and the process repeats until the optimal solution is elementary. In the best case scenario, DSSR finds an elementary path by solving the easier SPPRC (ie. when $V^* = \emptyset$). In the worst case scenario, an elementary path is found only when $V^* = V'$ and this corresponds to the exact dynamic programming algorithm by Feillet et al. (2004). The DSSR algorithm is described in Algorithm 1.2 and it uses the dynamic programming algorithm for the SPPRC by Desrochers et al. (1992) as a subroutine in Step 5.

Algorithm 1.2 DSSR - The Decremental State Space Relaxation Algorithm (Boland et al., 2006; Righini and Salani, 2008)

1: Initialization
2: Set $V^* \leftarrow \emptyset$.
3: **repeat**
4: Set $\Theta \leftarrow \emptyset$.
5: Solve SPPRC on graph with updated V^*.
6: **if** Optimal path is non-elementary **then**
7: Let Θ be a set new node(s) to be added to V^*.
8: Set $V^* \leftarrow V^* \cup \Theta$.
9: **end if**
10: **until** $\Theta = \emptyset$

Boland et al. (2006) proposed four different ways of updating the critical set after each iteration in which the optimal solution corresponds to a non-elementary path. The first strategy, *highest multiplicity on the optimal path (HMO)*, is to update V^* by adding only one of the nodes (there may be several of these) which was visited the most number of times. In the second strategy (*HMO-ALL*), all nodes visited the maximum number of times are added to V^*. The third strategy, *multiplicity greater than one on the optimal path - all nodes (MO-all)*, updates V^* by adding all nodes which were visited more than once on the optimal path. The fourth and final strategy, *multiplicity greater than one on some path - all nodes (M-all)*, is to add all nodes visited more than once by a an optimal path. From their results, HMO seems to be the best strategy. As proposed by Righini and Salani (2008), a possible way of improving the performance of DSSR is to warm-start V^* with an intelligent guess of critical nodes instead of starting with the empty set. This can eliminate some unecessary iterations during which critical nodes are identified.

1.5 The Minimum-Maximum Distance-Constrained CVRP

Traditional VRPs are concerned mainly with minimizing a costs function with little or no interest for the disparities that may exist between the routes making up a solution. This is natural since the main interest of employers or decision makers is to maximize their profits and they achieve this mainly by minimizing operational costs. Nevertheless, implementing the optimal solutions of such problems in real life can sometimes cause miscontent among employees and/or customers. In order to address this problem, some recent research have been dedicated to finding solutions to VRPs in which the routes need to be "balanced". The notion of *balancing* routes have been defined and treated differently in the literature. For example, both Lee and Ueng (1999) and Ribeiro and Ramalhinho Dias Lourenço (2001) try to quantify the total work load on all routes and share them fairly among employees. Other authors consider balanced routes by minimizing the difference between the maximal route length and the minimal route length in a solution (Jozefowiez et al., 2002; Pasia et al., 2007; Borgulya, 2008). Moreover, Kara and Bektaş (2005) ensures balanced routes by requiring a minimum load for any used vehicle whereas Bektaş (2012) considers route balancing by restricting the total number of customers visited by each route to lie within a predetermined range. These kinds of problems are referred to as VRPs with *load balancing* or *route balancing* and in this section we study the application of column generation to one of them. The problem considered is an extension of the CVRP in which the length of each route is required to lie within a predefined interval in addition to the usual objective of minimizing the sum of the lengths of the routes. We call this problem the minimum and maximum distance-constrained CVRP (MMDCVRP). A similar problem for the multiple traveling saleman problem has been treated by Rienthong et al. (2011). The authors adopted the integer program suggested by Kara and Bektaş (2005) and solved it with a commercial software in order to find practical (non-optimal) solutions to a real life problem. In addition to studying route balancing in vehicle routing, another interest of presenting this problem here lies in the subproblem encountered when solving it by column generation. We aim to discuss some issues concerning the application of column generation to problems which exhibits similar characteristics to the MMDCVRP. In particular, we investigate how the subproblem can be efficiently solved by dynamic programming.

1.5.1 Problem Description

The MMDCVRP is an extension of the CVRP in which it is required that the length of each route lies between a predetermined interval $[L_{\min}, L_{\max}]$. This problem can also be seen as an extension of the DCVRP which corresponds to the case for $L_{\min} = 0$. The interest of requiring the length of a route to lie within this interval is to ensure that the routes making up a solution are balanced. By description, the MMDCVRP looks similar to the VRPTW for which column generation has been a very successful solution method yet these two problems are very different. One main difference is that the length of a route is restricted by intervals at the depot and as well as customer nodes in the VRPTW whereas in the MMDCVRP the length of a route is restricted as a whole and not at individual customer nodes. Nevertheless, an instance of the MMDCVRP can be transformed to an instance of the VRPTW by defining intervals (time windows) $[a_i, b_i]$ for node $v_i \in G$ as follows:

- $[a_0, b_0] = [0, 0]$ when leaving the depot (v_0),
- $[a_0, b_0] = [L_{\min}, L_{\max}]$ when returning to depot (v_0),
- $[a_i, b_i] = [d_{0i}, L_{\max} - d_{i0}]$ for $v_i \in V \backslash \{v_0\}$.

Another major difference between the MMDCVRP and the VRPTW even after this transformation is that the arrival time at a node and the begining of service at the node should be the same. In other words, earlier arrival at a node is not permitted. This means that the consumption of exact resources should be modeled in the subproblem instead of the usual minimal resource consumptions. Gamache et al. (1998) explains how the exact consumption of a resource r may be modeled by the minimal consumption of two resources. These small differences have great impacts when solving the MMDCVRP and other similar problems by column generation. For example, when solving the ESPPRC subproblem by dynamic programming, the dominance rules used are well adapted and efficient for cases where only minimal resource consumptions are considered. If we are to consider exact resource consumption, then finding efficient dominance rules which are able to identify non-optimal partial paths as early as possible will not be an easy task.

1.5.2 Master Problem and Subproblem

We use the same notations as before. The main difference is that a feasible route $k \in \Omega$ is now defined as a hamiltonian cycle on a subset of V whose length lies between the interval $[L_{\min}, L_{\max}]$ and of total capacity not exceeding Q. Morever, there is no constraint on the number of vehicles available. The IMP for the MMDCVRP is, thus, defined by

$$\text{Minimize} \quad \sum_{k \in \Omega} c_k \theta_k \tag{1.33}$$

$$\text{subject to :} \quad \sum_{k \in \Omega} a_{ik} \theta_k \geq 1 \qquad (v_i \in V \backslash \{v_0\}) \,, \tag{1.34}$$

$$\theta_k \in \{0, 1\} \qquad (k \in \Omega) \,. \tag{1.35}$$

In the same way, the other models (LPM, RLPM, DLPM, DRLPM) can be easily defined as it was done for the CVRP.

Subproblem

The subproblem is to find feasible routes with negative reduced costs. Given the definition of a feasible route in this case, we obtain the following formulation:

$$\text{Minimize} \quad \sum_{(v_i, v_j) \in A'} \hat{d}_{ij} x_{ij} \tag{1.36}$$

$$\text{subject to :} \quad \sum_{(v_0, v_j) \in A'} x_{0j} = 1 \,, \tag{1.37}$$

$$\sum_{(v_i, v_{n+1}) \in A'} x_{i,n+1} = 1 \,, \tag{1.38}$$

$$\sum_{(v_i, v_j) \in A'} x_{ij} - \sum_{(v_j, v_i) \in A'} x_{ji} = 0 \qquad (v_i \in V \backslash \{v_0, v_{n+1}\}) \,, \tag{1.39}$$

$$\sigma_i + q_j - \sigma_j + Qx_{ij} \leq Q \qquad (v_i, v_j) \in A' , \tag{1.40}$$

$$l_i - l_j + (L_{\max} - d_{ij})x_{ij} \leq L_{\max} \qquad (v_i, v_j) \in A' , \tag{1.41}$$

$$\sum_{(v_i,v_j)\in A'} d_{ij}x_{ij} \geq L_{\min} , \tag{1.42}$$

$$0 \leq \sigma_i \leq Q \qquad v_i \in V' , \tag{1.43}$$

$$0 \leq l_i \leq L_{\max} \qquad v_i \in V , \tag{1.44}$$

$$x_{ij} \in \{0, 1\} \qquad (v_i, v_j) \in A' . \tag{1.45}$$

In this formulation, σ_i and l_i represent the load and the length of a route, respectively, after visiting node v_i. Without Constraint (1.42), the above problem is the usual ESPPRC with two types of resources. The first ressource constraint concerns the maximum load of a route which should not exceed Q. The second resource constraint states that the maximum length of a route should not exceed $L_{\max}$.

1.5.3 Solving the Subproblem

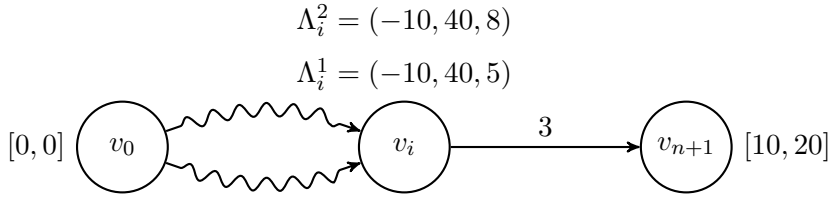
An important comment to make here is that we are not restricted in any way to solve the subproblem by any particular method. Any exact method (including dynamic programming, branch-and-cut, etc.) may be used once it is well adapted. In this section, however, we concentrate on solving the subproblem with dynamic programming since it looks somehow like ESPPRC with time windows which is solved efficiently by dynamic programming (see Feillet et al. (2004); Chabrier (2006); Boland et al. (2006); Righini and Salani (2008)). In all of these applications, minimal resource consumption is considered whereas we consider a combination of mininal resource as well as exact resource consumption. This means that an important component of the dynamic programming algorithm, the dominance rule, has to be modified.

The "usual" dominance rule

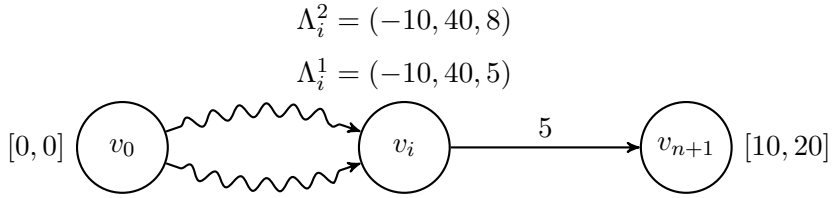
In what follows, we let the label $\Lambda_i = (\tilde{c}_i, \sigma_i, l_i)$ represent a partial path from the depot, v_0, to node v_i. The reduced cost up to this point is denoted by $\tilde{c}_i$ whereas σ_i and l_i represent the total load collected and the length, respectively. For simplicity, we have ommitted the other resources like those associated to critical nodes in the DSSR algorithm which ensures that an elementary route is found. The validity of what is discussed below is not affected in any way. Given two labels $\Lambda_i^1 = (\tilde{c}_i^1, \sigma_i^1, l_i^1)$ and $\Lambda_i^2 = (\tilde{c}_i^2, \sigma_i^2, l_i^2)$ on node v_i, the dominance rule for minimal resource consumption states that Λ_i^1 dominates Λ_i^2 if and only if $\Lambda_i^1 \leq \Lambda_i^2$. In this case, Λ_i^2 is rejected since for any feasible solution obtained from an extension of Λ_i^2, an equally better (if not strictly better) feasible solution can be obtained by extending Λ_i^1 in the same way.

In the case of exact resource consumption as it is for the MMDCVRP, the dominance rule just explained above does not work. This can be seen by considering the case in Figure 1.4a. By using the "usual" rule, Λ_i^2 is dominated by Λ_i^1 and so it is dropped at node v_i and never extended to node v_{n+1}. The extension of Λ_i^1 to node v_{n+1} is also rejected since it is not feasible (does not satisfy the minimum required length of a route). In this case, the algorithm finds no feasible routes although the extension of Λ_i^2 to node v_{n+1} is

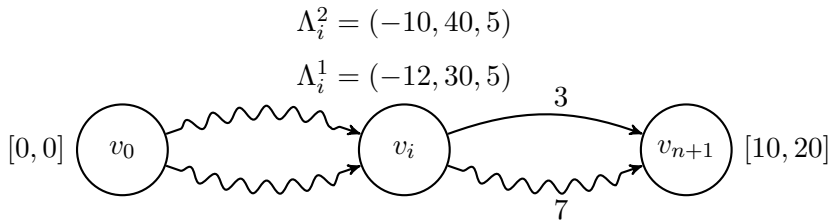
feasible. In order to avoid such an error, we can follow the idea poposed by Gamache et al. (1998) to consider the minimal (instead of exact) consumption of the resource l and also add the minimal consumption of another resource $\bar{l}$. The value of $\bar{l}$ is always equal to the negative of l (that is $\bar{l} = -l$). This rule ensures that no partial path that can lead to an optimal solution is eliminated. Indeed, Λ_i^1 can only dominate Λ_i^2 if $l_i^1 \leq l_i^2$ and $\bar{l}_i^1 \leq \bar{l}_i^2$. Since both l_i^1 and l_i^2 belong to the set of real numbers, we can deduce that if $l_i^1 \neq l_i^2$ then neither Λ_i^1 nor Λ_i^2 can dominate the other. In Figure 1.4a this rule will keep both labels on node v_i. The algorithm will try to extend both Λ_i^1 and Λ_i^2 to node v_{n+1} but will only keep the extension of Λ_i^2 since the other one is not feasible.



(a) No feasible labels are found if the "usual" rule is used. By following the rule of Gamache et al. (1998), a feasible label can be obtained by extending Λ_i^2 to v_{n+1}.



(b) If we use the rule of Gamache et al. (1998) then neither Λ_i^1 nor Λ_i^2 dominates the other on node v_i. If the newly proposed rule is used, then Λ_i^1 dominates Λ_i^2 on node v_i.



(c) The rule of Gamache et al. (1998) correctly identifies Λ_i^2 as dominated by Λ_i^1 on node v_i but the second condition of the newly proposed rule fails to do this.

Figure 1.4: Dominance rules for modeling exact resource consumption.

A new dominance rule

Although the rule by Gamache et al. (1998) ensures that no label that can lead to an optimal path is eliminated, it is clear that the rule is not very efficient for identifying dominated subpaths as early as possible. In fact, very few labels can be eliminated by this rule. This is not very good for the dynamic programming algorithm since its effectiveness greatly depends on the quality of the dominance rules. We propose the following dominance rule that is able to identify some dominated subpaths earlier than the rule proposed by Gamache et al. (1998). The basic idea is to add another condition to the rule proposed by Gamache et al. (1998) in order to strengthen it. We say that Λ_i^1 dominates Λ_i^2 if and only if at least one of the following two conditions is satisfied:

1. $\Lambda_i^1 \leq \Lambda_i^2$ and $l_i^1 = l_i^2$,
2. $\Lambda_i^1 \leq \Lambda_i^2$ and $l_i^1 + \delta_{i,n+1} \geq L_{\min}$.

In both conditions, the first part corresponds to the rule used when modeling minimal resource cosumptions. In addition, each condition has a second part to ensure that no partial path that can lead to an optimal solution is eliminated. Condition 1 is exactly the same as the one proposed by Gamache et al. (1998) whereas condition 2 is a new proposition to strengthen the previous one. The value of $\delta_{i,n+1}$ in this condition represents the minimum possible length from the current node v_i to the destination node (copy of depot) v_{n+1}. Since we assume that the consumption of all resources satisfy the triangle inequality, we have $\delta_{i,n+1} = d_{i,n+1}$. Condition 2 implies that Λ_i^1 can dominates Λ_i^2 only if we are sure that an extension of Λ_i^1 using the minimum possible quantity of an exact resource will still satisfy the conditions on this exact resource. These two conditions complement each other as either of them can identify some dominated paths in cases where the other cannot (see Figure 1.4b and 1.4c). So using them together guarantees that we have a stronger dominance rule. All dominated paths identified by the rule of Gamache et al. (1998) will be identified by condition 1 of this new rule whereas some dominated paths identified by condition 2 may not be identified with the rule of Gamache et al. (1998).

Even with this improvement in the form of the new dominace rule, the dynamic programming algorithm for solving problems with exact resource consumptions may not be as effective as we would expect as in the case of modeling minimal resource consumptions. For this reason we consider solving relaxations of the subproblem in order to find valid lower bounds for the MMDCVRP.

Relaxations for the MMDCVRP and its Subproblem

A first observation is that, the optimal value of the subproblem for any interval $[L_1, L_{\max}]$ where $0 \leq L_1 \leq L_{\min}$ provides a lower bound on the optimal value when we consider the interval $[L_{\min}, L_{\max}]$. In particular, when $L_1 = 0$ then we have a problem similar to the one we would encounter if we were solving the DCVRP (only a maximum limit on the length of a route). This means the optimal value of the DCVRP with an upper limit of $L_{\max}$ on the length of a route provides a lower bound on the optimal value of any MMDCVRP solved over the interval $[L_{\min}, L_{\max}]$ where $0 < L_{\min} < L_{\max}$. An advantage of solving this relaxation is that, we can use all known methods for the DCVRP in computing a lower

bound for the MMDCVRP. However, depending on how tight the interval $[L_{\min}, L_{\max}]$ is, the resulting bound may not be good enough to be used in a branch-and-price algorithm. In spite of the strength of the dominance rule used in this case, it may be more difficult to solve the subproblem since widening the interval increases the number of states a label can be in. A good idea, therefore, is to relax the interval just slightly and then apply the improved dominance rule described in the preceeding section.

1.5.4 Computational Experiments

We present some results from experiments performed to compare the quality of the lower bounds provided by each of the approaches discussed above (ie. the two dominance rules, and the relaxations). In order to give a good idea of how good each lower bound is, upper bounds are computed after computing a lower bound by removing any infeasible routes (in the case of relaxations) from the model and solving the corresponding integer program. We used some DCVRP instances from the VRP literature and these can be obtained from `http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html`. For each instance, the maximum capacity of a vehicle, Q, and the maximum length of a route, $L_{\max}$, are the original ones for the instance. In addition, we computed several values of minimum required length length of a route as $L_{\min} = \alpha \cdot L_{\max}$ where $\alpha \in \{0, 0.25, 0.50, 0.75\}$. All codes were written in C/C++ and the RLPM was solved with ILOG CPLEX 12.4. Tests were run on an Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz computer with a 2 GiB RAM.

The results of the experiments are summarized in Tables 1.1 and 1.2. Table 1.1 compares results obtained by using the dominance rule proposed by Gamache et al. (1998) to those obtained by using the new and improved dominance rule. The results obtained by relaxing the interval in conjunction with the improved rule are presented in Table 1.2. The column headings in both tables have the following meanings:

- Instance: name of the instance together with the original value of the maximum allowed length of a route for the instance
- L_{min}: minimum required length of a route
- L_1: a relaxation of the minimum required length of a route
- lb: value of the lower bound obtained
- ub: value of the upper bound obtained
- cpu: total computational time in cpu seconds
- %gap: IP gap for obtained lower and upper bounds

A time limit of 5000 cpu seconds was set for each instace. After this limit, the column generation algorithm is stopped and the best known lower and upper bounds are noted. For such a case, a value of > 5000 is recorded in the table.

Table 1.1: Comparison between Gamache et al's and New Dominance Relations

		Gamache et al.				Improved			
Instance	$L_{\min}$	cpu	lb	ub	%gap	cpu	lb	ub	%gap
	0	> 5000	342	375	8.80	0.51	375	375	0.00
D022-04g	53	> 5000	343	375	8.53	0.45	375	375	0.00
$L_{\max} = 210$	105	> 5000	413	439	5.92	0.21	432	440	1.82
	158	1124.80	640	662	3.44	0.15	640	665	3.76
	0	590.81	628	652	3.83	0.60	628	660	5.00
D023-03g	60	775.64	628	653	3.83	0.69	628	660	5.00
$L_{\max} = 240$	120	643.07	635	678	6.34	1.40	635	673	5.65
	180	1539.52	664	740	10.41	1.98	664	748	11.23
	0	> 5000	419	612	31.54	339.28	485	503	3.58
D030-03g	60	> 5000	423	619	31.66	438.53	485	519	6.55
$L_{\max} = 240$	120	> 5000	457	651	29.80	458.87	485	560	13.39
	180	> 5000	531	742	28.44	473.29	545	721	24.41
	0	> 5000	975	1130	13.72	1136.22	1039	1106	6.06
D033-04g	60	> 5000	986	1167	15.51	863.75	1039	1104	5.89
$L_{\max} = 240$	120	> 5000	1005	1115	9.87	570.86	1041	1102	5.54
	180	> 5000	1036	1115	7.09	590.68	1048	1118	6.70

Summary of Results. The results in Table 1.1 show that the newly proposed dominance rule really improves on the rule by Gamache et al. (1998). For example, all the algorithm converges for all instances when the new dominance rule is used whereas for most instances the the algorithm could not converge within the given time frame when we used the rule by Gamache et al. (1998). In cases where both approaches converge, the same lower bound is obtained. The upper bound is, however, different since different columns are added to the master problem for the different approaches. When both approaches converge, slightly better bounds are obtained when Gamache et al.'s rule is used.

For a fixed value of $L_{\max}$, the difficulty of an instance of the MMDCVRP increases with decreasing values of $L_{\max}$ (see Table 1.2). This is analogous to the VRPTW where an instance become more difficult to solve if the time windows are wider. For this reason, the overall performance of the relaxations investigated here is quite disappointing but not so surprising. Dropping the minimum required length of a feasible route in the subproblem enables us to use a more effective dominance rule. Nevertheless, the effectiveness of the whole algorithm however seems to worsen because a larger space of solutions needs to be explored by the subproblem. In this case, we have poorer lower and upper bounds as well as longer computational times. This type of relaxation does not help much and so it may be necessary to investigate other types of relaxations like those concerned with the routes being elementary.

Table 1.2: Performance of Relaxations

		Relaxation				
Instance	$L_{\min}$	L_1	cpu	lb	ub	%gap
	53	0	0.83	375	375	0.00
	53	53	0.45	375	375	0.00
	105	0	0.65	375	471	20.59
D022-04g	105	53	1.04	375	488	23.16
$L_{\max} = 210$	105	105	0.21	432	440	1.82
	158	0	0.49	375	666	43.69
	158	53	0.38	375	666	43.69
	158	105	0.29	429	666	35.74
	158	158	0.15	640	665	3.76
	60	0	0.83	628	660	5.00
	60	60	0.69	628	660	5.00
	120	0	0.91	628	673	6.84
D023-03g	120	60	1.19	628	673	6.84
$L_{\max} = 240$	120	120	1.40	635	673	5.65
	180	0	0.64	628	789	20.53
	180	60	0.94	628	777	19.31
	180	120	0.99	635	768	17.32
	180	180	1.98	664	748	11.23
	60	0	603.94	485	517	6.19
	60	60	438.53	485	519	6.55
	120	0	453.58	485	562	13.70
D030-03g	120	60	447.56	485	562	13.70
$L_{\max} = 240$	120	120	460.77	485	560	13.39
	180	0	354.24	485	726	33.20
	180	60	368.48	485	723	32.92
	180	120	476.59	486	636	23.74
	180	180	473.29	545	721	24.41
	60	0	1136.22	1039	1104	5.89
	60	60	863.14	1039	1104	5.89
	120	0	840.96	1039	1102	5.72
D033-04g	120	60	835.64	1039	1102	5.72
$L_{\max} = 240$	120	120	570.86	1041	1102	5.54
	180	0	754.92	1039	1164	10.74
	180	60	754.03	1039	1164	10.74
	180	120	700.05	1041	1163	10.49
	180	180	590.68	1048	1118	6.70

1.6 Conclusion

In this chapter, we have discussed the application of column generation to vehicle routing problems. Vehicle routing problems have several practical applications in different domains. A very popular solution method for VRPs is branch-and-price and this method combines column generation with branch-and-bound. Column generation solves a linear relaxation of the original problem in order to compute a lower (and an upper) bound whereas the branch-and-bound framework ensures that integer solutions are obtained. The success of column generation relies on the fact that the models involved usually have very good linear relaxations. The number of variables (or columns) in these models are, however, huge and so have to be introduced into the model in bits by solving a subproblem. The last section of the chapter presents how column generation can be used in computing a lower and upper bound for the minimum and maximum distance capacitated vehicle routing problem (MMDCVRP). Due to the perculiar nature of the subproblem for the MMDCVRP, its solution by a dynamic programming algorithm is quite challenging. We propose a new and stronger dominance rule based on one already proposed in the literature. We also consider a way to relax the subproblem and more precisely, relaxations for the subproblem that can help us compute lower bounds for the MMDCVRP. Different relaxations of the subproblems in a column generation method have been used with a great effect in the literature. Nevertheless, the type of relaxation considered here does not help much but rather worsens the performance of the algorithm. This means that although relaxations are usually a good way of finding practical solutions to a problem, one should be careful of the type of relaxation chosen since it can actually have negative effects.

Chapter 2

Multi-Objective Optimization

2.1 Introduction

Many problems arising in real life (e.g. in the domaine of logistics, transportation, etc.) present complex characteristics which are modeled and solved with the help of mathematical tools. Although these problems naturally present the "optimization" of several objectives or preferences, they are usually modelled with a single function together with several constraints which tries to take the most important objectives into account. This approach (called single objective optimization) may be sufficient in some cases. In most other cases, however, modeling a problem with just one function is impossible or may result in biased models since we may not have enough information about the objectives. Cohon (2004) actually states that:

> "There are no ifs, ands, or buts about it: all public decision making problems are *multiobjective*, and if we don't think about and analyze them with that in mind we may do serious damage to the quality of the decisions that we make."

The goal of multi-objective optimization is to tackle challenging problems by using more than one function in order to properly account for all (or most) objectives. In spite of the clear advantages that can be gained by using this approach, the resulting models are more difficult to solve due to the several objectives which contracdict each other. Two objectives are said to contradict each other if improving one of them results in degrading the other. Multi-objective optimization has its roots in the fields of economics and management science (Samuels, 1981; Pareto, 1896).

The aim of this chapter is to recall some basic notions of multi-objective optimization that will be necessary in order to better understand the rest of the manuscript. For a more detailed presentation in this field, one may consult a book like Ehrgott (2005). We start the chapter by giving the basic definitions and principles of multi-objective optimization. Given that the terminology used in the field varies from one author to another, this will serve as an opportunity to introduce the ones we use in this work. We also give an overview of the different solution approaches and methods as well as the different measures used in evaluating the quality of these methods and approximations.

2.2 Basic Definitions and Principles

A multi-objective optimization problem (MOP) can be defined as follows:

$$(MOP): \quad \underset{x \in \mathcal{X}}{\text{Minimize}} \quad F(x) = (f_1(x), f_2(x), \ldots, f_r(x)) \,, \tag{2.1}$$

where $r \geq 2$ is the number of objective functions, $x = (x_1, x_2, \ldots, x_n)$ is the decision variable vector or solution, $\mathcal{X}$ is the feasible solution set, and $F(x)$ is the objective vector. The set $\mathcal{Y} = F(\mathcal{X})$ corresponds to the images of the feasible solutions in the objective space, and $y = (y_1, y_2, \ldots, y_r)$, where $y_i = f_i(x)$, is a point of the objective space.

One main difference between a MOP and a classical optimization problem with only one objective is that a MOP does not have (in general) a unique optimal solution which is better than all other solutions. This means, we need a different way of comparing solutions in order to to be able to define optimality. This is done by using a dominace relation. Although there are several possible relations that may be used, the most commonly used and accepted one is *Pareto dominance*. The Pareto dominance between two solutions is defined as follows:

Definition 2.1 (Pareto Dominance)**.** *A solution x' dominates ($\preceq$) another solution x'' in the Pareto sense if and only if $\forall\ i \in \{1, \ldots, r\},\ f_i(x') \leq f_i(x'')$ and $\exists\ i \in \{1, \ldots, r\}$, such that $f_i(x') < f_i(x'')$.*

In Figure 2.1 we have two objective functions (f_1 and f_2) both of which are to be minimized over the set of feasible solutions $\{A, B, C, D, E\}$. We can see that solution B is dominated by solution C whereas solution D is dominated by both solutions C and E. A feasible solution dominated by no other feasible solution is said to be *efficient* or *Pareto optimal* and its image in the objective space is said to be *nondominated*. The set of all efficient solutions is called the *efficient set* (denoted by $\mathcal{X}_E$) and the set of all nondominated points is the *nondominated set* (denoted by $\mathcal{Y}_N$). In Figure 2.1, the solutions not dominated by any other solutions are A, B, and E and so the set of Pareto optimal solutions is $\{A, C, E\}$. Computing the set of efficient solutions is usually a very difficult task for most practical problems. Due to this, we usually search for a set of approximate solutions to the exact efficient set. An approximation to the efficient set is a set of solutions found by a heuristic that are not dominated by another solution found by the same heuristic. A solution in an approximate set is said to be *potentially efficient*.

Other notions of dominance and optimality beside those by Pareto may be used in defining preferences among solutions. One of these is *lexicographic dominance* which is defined as follows:

Definition 2.2 (Lexicographic dominance)**.** *A solution x' lexographically dominates ($\preceq_{\text{lex}}$) another solution x'' if there exists an index $k^* \leq n$ such that $x'_k = x''_k$ for $k = 1, \ldots, (k^* - 1)$ and $x'_k < x''_k$.*

Consequently, a feasible solution is said to be *lexicographically optimal* if it is not lexicographically dominated by any other feasible solution. If we order the two objective functions in Figure 2.1 as $F = (f_1, f_2)$, then the most preferred solution is A since it has the smallest value with respect to f_1. Changing the order of f_1 and f_2 results in E being

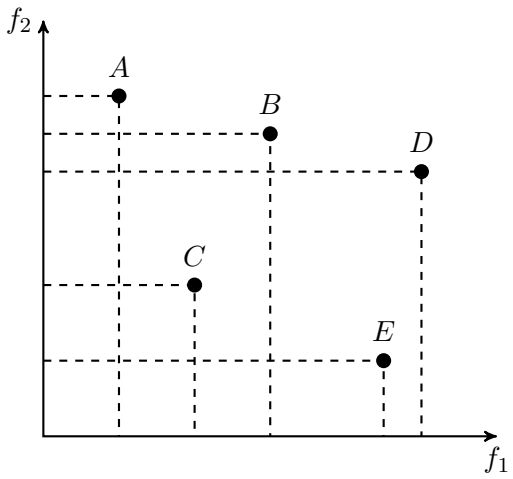
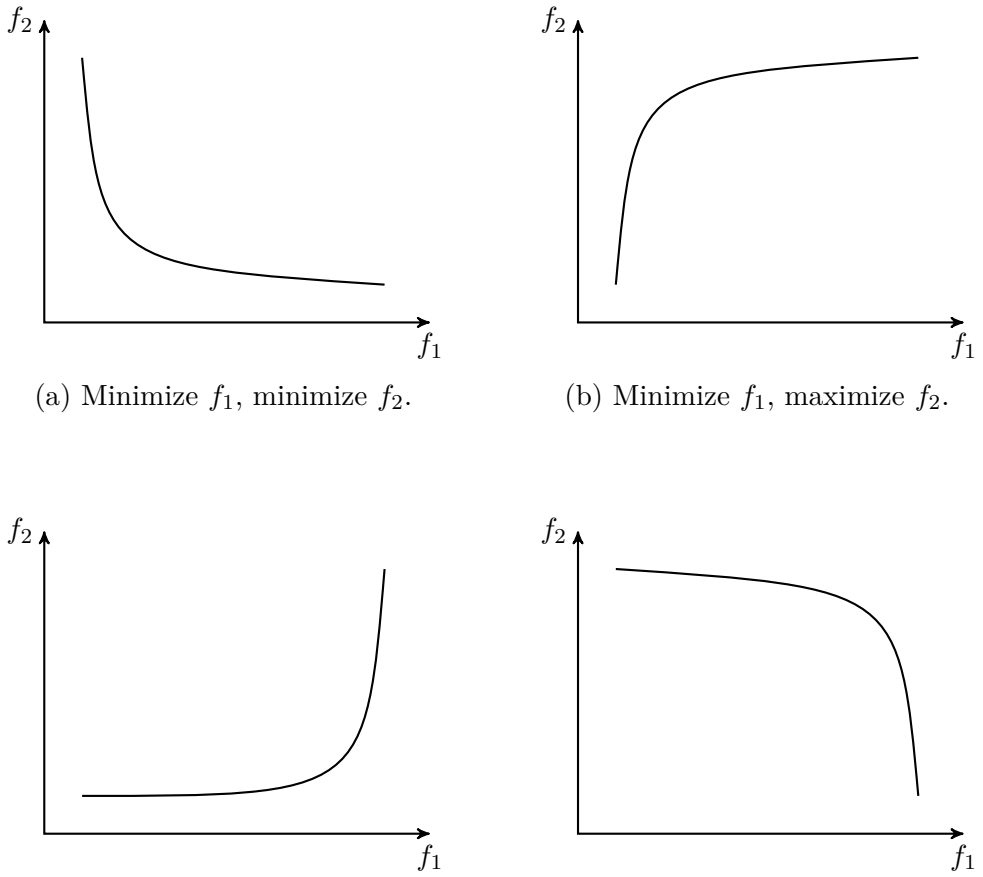


Figure 2.1: Pareto dominance between solutions.

the most preferred solution since it has the smallest value with respect to f_2. So the set of lexicographically optimal solutions is $\{A, E\}$. Other common dominance relations are *max-ordering dominance* and *cone dominance* (see Collette and Siarry (2004)). It is clear that the set of most preferred solutions that can be found by using a dominance relation other than Pareto dominance will always be a subset of the efficient set (set of Pareto optimal solutions). The main interest of the other dominance relations is that they allow some degree of freedom for a decision maker to influence the search for solutions (Collette and Siarry, 2004). For example, it is possible for a decision maker to order the components of the objective function $F(x) = (f_1(x), f_2(x), \ldots, f_r(x))$ in such a way that $i < j$ implies the objective function $f_i(x)$ is more important than $f_j(x)$. By so doing, he is sure to find a solution that best satisfies his most important objective. Throughout this work, we will use the notion of Pareto dominance when comparing solutions unless otherwise explicitely stated.

The set of nondominated points of a MOP describes what is known as a *tradeoff surface* or *Pareto frontier*. The shape of this surface depends on the number of objectives and whether each of them is to be minimized or maximized. Possible shapes of the tradeoff surface for problems consisting of only two objectives (*bi-objective problems*) is given in Figure 2.2. These just give an idea of the form of the shapes. In general, the tradeoff surface is not a perfect smooth curve or convex as portrayed. A nondominated point that lies on the convex part of the tradeoff surface is said to be *supported*. In the same way, a *supported efficient solution* is an efficient solution that maps to a supported nondominated point. Due to the difficult nature of MOPs, we often do not seek the exact nondominated set but rather a good approximation of it. In these cases, we search for a set of points corresponding to a set of solutions which do not dominate each other. One desirable quality of such an approximation is its ability to quickly converge to the true nondominated set (*intensification*). Another quality is its ability to better estimate the whole range of the tradeoff surface (*diversification*). These two qualities are represented in Figure (2.3). A good approximation is one that has a good balance of both intensification



(a) Minimize f_1, minimize f_2.

(b) Minimize f_1, maximize f_2.

(c) Maximize f_1, minimize f_2.

(d) Maximize f_1, maximize f_2.

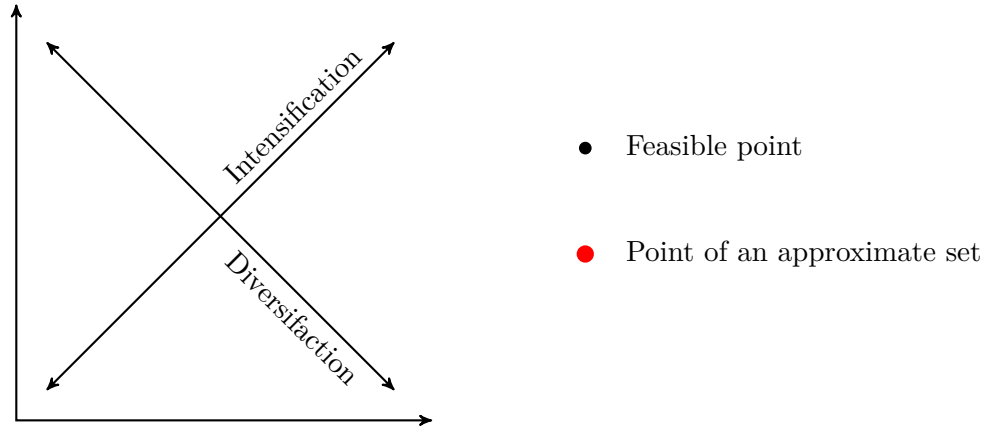
Figure 2.2: Common shapes of the tradeoff surface for a bi-objective problem.

and diversification.

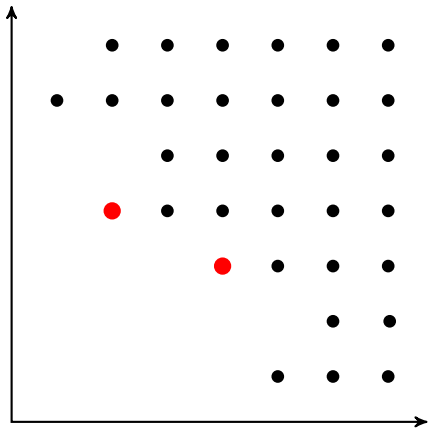
For most practical problems, the feasible set of solutions $\mathcal{X}$ represents a finite set. In particular, the components of a solution vector may be required to take on discrete values. When we have $\mathcal{X} \subset \mathbb{Z}^n$, we talk of *multi-objective integer programs* (MOIP). A particular case of a MOIP is a *multi-objective combinatorial optimization* (MOCO) problem. A MOCO problem can usually be modelled so that the components of its solution vector takes on binary values. These binary values typically models whether an option is chosen or not. The set of feasible solutions for a MOCO problem is thus $\mathcal{X} = \{0, 1\}^n$. In this thesis we will deal mainly with MOCO problems.

2.3 Solution Approaches

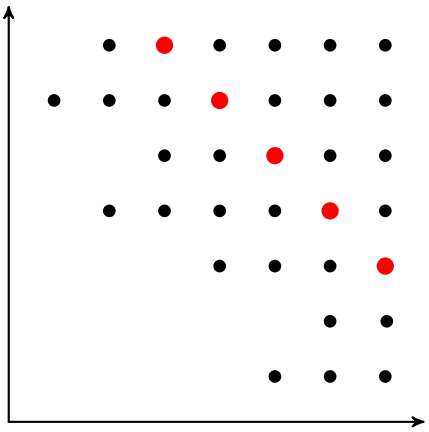
Although the solution of a MOP is typically a set, only one of these solutions is usually implemented in real life. For this reason, there is a decision maker whose work is to choose the solution to be implemented. A classification of solution approaches for MOPs may be



(a) Two goals



(b) Good intensification



(c) Good diversification

Figure 2.3: Two qualities of approximations.

made based on the stage of the solution process at which a decision maker is included.

2.3.1 A priori approaches

In an *a priori approach*, the decision maker defines his preference for each objective before the solution process starts and he doesn't interfere with the process after this. Only one solution reflecting the preference of the decision maker is returned after the solution process. This approach is interesting since we solve more or less a single-objective problem after the definition of preferences. A disadvantage of this approach is that it is not always easy to model the preferences of the decision maker as a single-objective problem.

2.3.2 Progressive approaches

In this approach the decision maker first defines his preferences before the solution process starts. Unlike in *a priori* approaches, he is allowed to modify his preferences during the solution process in order to guide the search. Just as for *a priori* approaches, only one solution is returned at the end of the process. Progressive approaches seem to have an advantage over a priori approaches since the initial preferences may be modified after having sufficient information on the problem. One disadvantage of progressive approaches, however, is that the decision maker should be present during the whole solution process in order to refine his preferences and guide the search. This is inconvenient in cases where the time needed to solve the problem is significantly long.

2.3.3 A posteriori approaches

These approaches seek to propose a set of solutions to the decision maker who will then be required to choose which one to implement. The problem is modeled and solved without the need for the decision maker to make preferences. He is only involved at the end of the process to have a final say on which solution he prefers. Unlike in the two previous approaches, the decision maker now has the advantage of knowing the options he has before making a decision. A clear disadvantage of this approach is the significant amount of time that is usually needed to search for the set of solutions.

The approach used to solve a particular MOP may not necessarily belong to only one of these three solution approaches. Indeed, the approach used in solving a MOP may consist of treating the objectives in a specific order of importance. If this order of importance is defined by a decision maker before an optimization method is applied, then the approach is *a priori*. However, if the order in which the objectives are treated is determined randomly, and the optimization method is applied several times by varying the order of treating the objectives then we have an *a posteriori* approach.

2.4 Approaches for Managing Objectives

Different strategies may be used to manage the objective functions of a MOP. These strategies may be classified into four groups namely *scalar*, *non-scalar*, *Pareto* and *indicator-based* approaches.

2.4.1 Scalar Approaches

Scalar approaches transform a MOP into a single-objective problem and treat it as such. These are the approaches employed by most traditional methods. The two most popular scalar approaches are the *weighted sum method* (Fishburn, 1967), and the *ε-constraint method* (Haimes et al., 1971). Other scalar approaches are *Benson's method* (Benson, 1978), the *elastic constraint method* (Ehrgott and Ryan, 2003) and other hybrid methods combining two or more of these techniques. Here, we only discuss the weighted sum method and the ε-constraint constraint method since they are the ones we will referred to later on. A detailed description of these two approaches as well as the others mentioned can be found in Ehrgott (2005).

The Weighted Sum Method (Fishburn, 1967)

Solving the MOP (2.1) by the weighted sum method consists in assigning a vector of weights $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r)$ to the vector of objective functions and optimizing a linear aggregation. This is given by:

$$\text{Minimize} \quad \sum_{i=1}^{r} \lambda_i f_i(x) \tag{2.2}$$

$$\text{subject to :} \quad x \in \mathcal{X} \,, \tag{2.3}$$

$$\sum_{i=1}^{r} \lambda_i = 1 \,, \tag{2.4}$$

$$\lambda_i \in [0,1] \quad \text{for } i = 1, 2, \ldots, r \,. \tag{2.5}$$

The values of λ_i may be seen as a way of indicating preferences among the different objectives. By varying these values, different efficient solutions to the original problem can be obtained. This is the most popular scalar approach used for MOPs Ehrgott (2005). One difficulty encountered in implementing this method is the definition of the weights when we don't have enough information about how the objectives functions interact with each other. Moreover, Das and Dennis (1997) showed that only supported solutions can be can be found by this method. No matter the choice of weights, nonsupported solutions can never be found by a weighted sum method (see Figure 2.4).

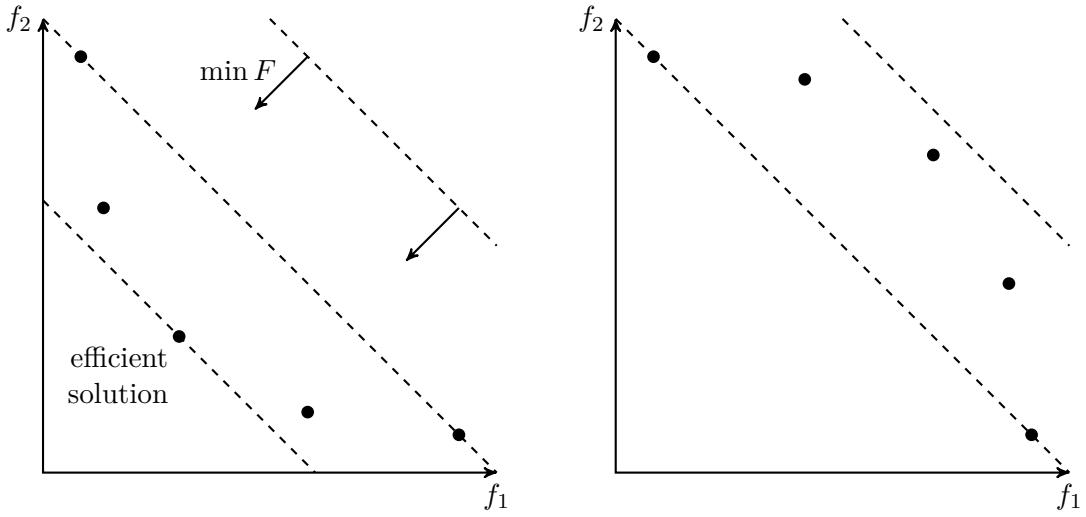


Figure 2.4: Solutions found by a weighted sum method for $\lambda_1 = \lambda_2 = 0.5$.
Notes: In the figure on the left, all nondominated points can be found by choosing the right values of λ_1 and λ_2. In the figure on the right, only the two extreme points can be found for all possible values of λ_1 and λ_2.

The ε-Constraint Method (Haimes et al., 1971)

The principle of the ε-constraint method is to turn a MOP into a single-objective problem by selecting only one of the objectives to optimize and converting all the other objectives into constraints. So for problem (2.1), this method will select an objective function f_k such that $1 \leq k \leq r$ and then solve the resulting problem. We get

$$\begin{aligned} & \text{Minimize} \quad f_k(x) && (2.6) \\ \text{subject to :} \quad & f_i(x) \leq \varepsilon_i \quad \text{for } i = 1, \ldots, r \text{ and } i \neq k \,, && (2.7) \\ & x \in \mathcal{X} \,. && (2.8) \end{aligned}$$

Unlike the weighted sum method, the ε-constraint method is capable of finding all the points that define the tradeoff surface (see Figure 2.5). A disadvantage of this method is that adding constraints on the objectives normally changes the structure of the original problem and the resulting problem may be more difficult to solve.

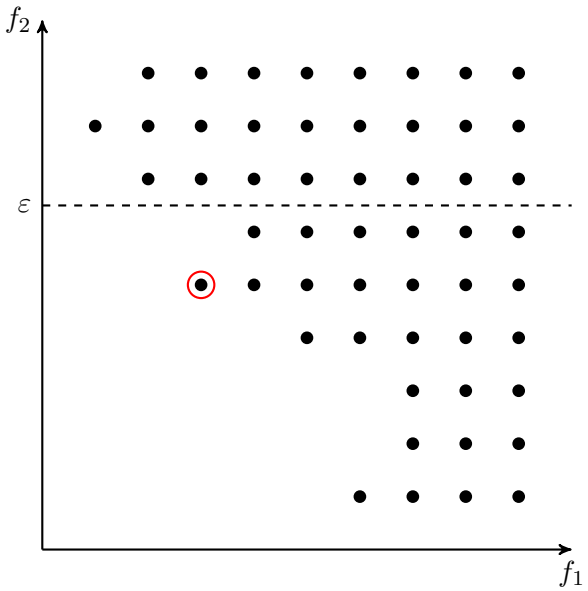


Figure 2.5: Solutions found by the ε-constraint method.
Notes: By chosing the right values of ε, all nondominated points can be found.

2.4.2 Non-Scalar Approaches

Unlike scalar approaches, non-scalar approaches don't transform a MOP into a single-objective problem but rather use different strategies to search for solutions by treating the objective functions separately. A popular example of such an approach is *lexicographic optimization* (Fourman, 1985) which is presented below.

Lexicographic Optimization

This method consists in ordering the components of the objective function vector based on their importance as proposed by a decision maker and then treating them one after the other in the given order. Suppose that the components of the objective function $F(x) = (f_1(x), f_2(x), \ldots, f_r(x))$ are ordered in such a way that $i < j$ implies the objective function $f_i(x)$ is more important than $f_j(x)$. In solving problem 2.1 by lexicographic method, we start by solving

$$\begin{aligned} \text{Minimize} \quad & f_1(x) \\ x \ & \in \ \mathcal{X} \, . \end{aligned}$$

Let the optimal solution of this problem be $\bar{x}^1$ with an associated value of $f_1^* = f_1(\bar{x}^1)$. The next problem is solved by using the of f_1^* as a constraint. We obtain

$$\begin{aligned} \text{Minimize} \quad & f_2(x) \\ x \ & \in \ \mathcal{X} \, , \\ f_1(x) \ & = \ f_1^* \, . \end{aligned}$$

Here too, we let $\bar{x}^2$ be the optimal solution and $f_2^* = f_2(\bar{x}^2)$ the associated value. In treating the next component of the objective function ($f_3(x)$), we use both f_1^* and f_2^* as constraints. This iterative process continues until all components of the objective function are treated. Thus for the i^{th} component of the objective function, we solve

$$\begin{aligned} \text{Minimize} \quad & f_i(x) \\ x \ & \in \ \mathcal{X} \, , \\ f_1(x) \ & = \ f_1^* \, , \\ f_2(x) \ & = \ f_2^* \, , \\ & \cdots \\ f_{i-1}(x) \ & = \ f_{i-1}^* \, , \end{aligned}$$

where $f_j^* = f_j(\bar{x}^j)$ is the value corresponding to the optimal value $\bar{x}^j$ obtained when treating the component $f_j(x)$. Figure 2.6 illustrates how lexicographic method works in optimizing a bi-objective problem. In this figure, we suppose that $f_1(x)$ is first minimized before minizing $f_2(x)$.

2.4.3 Pareto Approaches

A Pareto approach uses the notion of Pareto dominance to select preferred solutions among several candidates. Goldberg (1989) was one of the first reseachers to use this kind of approach which are found mainly in population based evolutionary algorithms. An example of such an approach is NSGA II which was proposed by Deb et al. (2002).

2.4.4 Indicator-Based Approaches

In contrast to Pateto approaches, indicator-based approaches use performance indicators to assign a score to a solution or set of solutions in order to guide the search for optimal

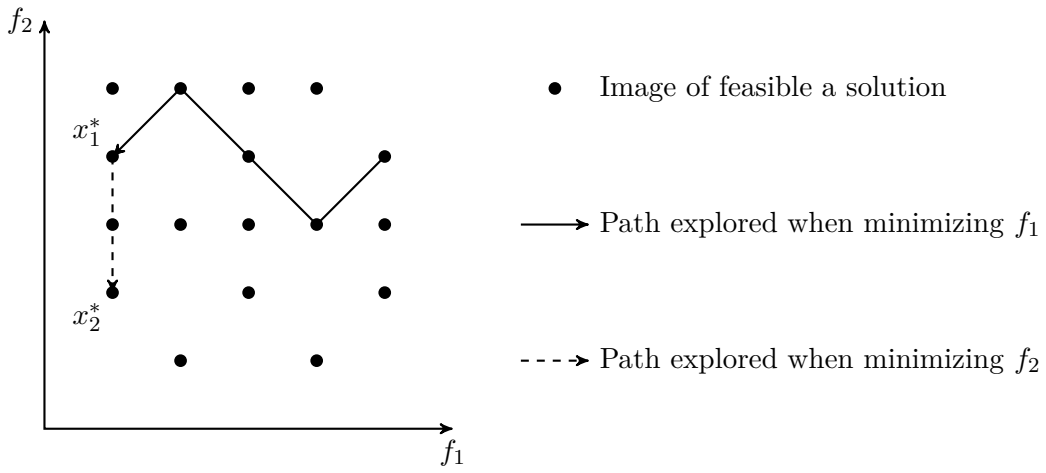


Figure 2.6: Solutions found by the lexicographic method.

solutions. The main idea used is to define the multi-objective problem as a problem which seeks to maximize the value of the performance indicator. A good performance indicator has to take both objectives of intensification and diversification into account. An example of an indicator-based method is the Indicator Based Evolutionary Algorithm (IBEA) proposed by Zitzler and Künzli (2004).

2.5 Solution Methods

Regardless of the approach used in managing the multiple objectives of a MOP, the method or algorithm used in solving the resulting problem can be classified as being either *exact* or *approximate* (heurisics and metaheuristics). Exact methods search for the complete set of nondominated points whereas approximate methods search for a set of points that tries to estimate the complete set of nondominated points. Due to the difficulty of MOPs, most of the literature concentrate on approximation methods rather than exact methods. In this section, we describe some exact methods based on the two most common scalarization approaches (weighted sum and ε-constraint method). We also give a general overview of some heuristics and metaheuristics. A detailed discussion of solution methods for MOPs can be found in Ehrgott and Gandibleux (2000) and Gandibleux (2002).

2.5.1 Lower and Upper Bounds

In single objective optimization, lower and upper bounds have been the backbone for solving many difficult problems. Bounds are also used to guarantee that a solution found by an approximate algorithm lies within no more than a given percentage from the optimal solution. Due to the importance of bounds in single objective optimization, it is natural to expect that they will be equally important in multi-objective optimization. The *ideal point* y^I and the *nadir point* y^N are widely accepted as a lower and an upper bound, respectively, of the nondominated set of a MOP. Given a vector of r objective functions $F = (f_1, f_2, \ldots, f_r)$, the i^{th} component (for $i = 1, 2, \ldots, r$) of the coordinates of the ideal

point is defined by

$$y_i^I := \min_{x \in \mathcal{X}} f_i(x)$$

whereas that of the nadir point is defined by

$$y_i^N := \max_{x \in \mathcal{X}_E} f_i(x) \ .$$

As it can be seen from Figure 2.7, these points are usually very far from most of the members of the nondominated set and so provide very little information. In addition, the problems needed to be solved during their computation may be very difficult. This is usually true for the ideal point and we may need to compute a relaxation for it rather than the true point. The relaxation of the ideal point is an even worse lower bound as it can be seen in Figure 2.7.

Motivated by the fact that the optimal solution of a MOP is a set of solutions rather than a single solution, Villarreal and Karwan (1981) proposed the use of sets to represent the lower and upper bounds of a MOIP. They define a lower bound of a MOIP as a set of points $\boldsymbol{L}$ such that every nondominated point of the problem is dominated by one of the members of $\boldsymbol{L}$. They also define an upper bound of a MOIP as a set of points $\boldsymbol{U}$ such that each of its members is either part of the nondominated set $\mathcal{Y}_N$ or is dominated by at least one member of $\mathcal{Y}_N$. We remark that the members of $\boldsymbol{L}$ may not correspond to images of feasible solutions.

This notion of *bound sets* has recently received attention from other researchers. Ehrgott and Gandibleux (2007) prove some general results on lower and upper bound sets for BOCO problems and propose the use of a so called *convex hull lower bound set*. In order to compute the convex hull lower bound set for a BOCO problem, they first search for efficient supported solutions by applying a weighted sum scalarization. The convex hull lower bound set is then defined as the curve joining these points. Sourd and Spanjaard (2008) use ideas similar to those presented by Ehrgott and Gandibleux (2007) to develop a multi-objective branch and bound framework. They define a lower bound as any function which separates the objective space into two halves in such a way that the complete set of feasible points are found on one halve. Delort and Spanjaard (2010) also present an efficient branch and bound algorithm for the bi-objective binary knapsack problem based on bound sets. A detailed discussion on bound sets for BOCO problems and how they can be constructed will be given in the next chapter.

2.5.2 Exact Methods

Most exact methods are limited to bi-objective problems rather than general MOPs. Some of these are the two phases method by Ulungu and Teghem (1995) and the exact ε-constraint method for bi-objective combinatorial optimization problems by Bérubé et al. (2009). One of the few exact methods that apply to problems with more than two objectives is the method proposed by Sylva and Crema (2004) (also by Jahanshahloo et al. (2005)). Two other popular exact methods for MOPs which apply to problems with more than two objectives are dynamic programming and branch and bound. These two methods are applied mainly to MOPs for which the methods are efficient for the corresponding single objective versions. For example, dynamic programming has been applied to multi-objective

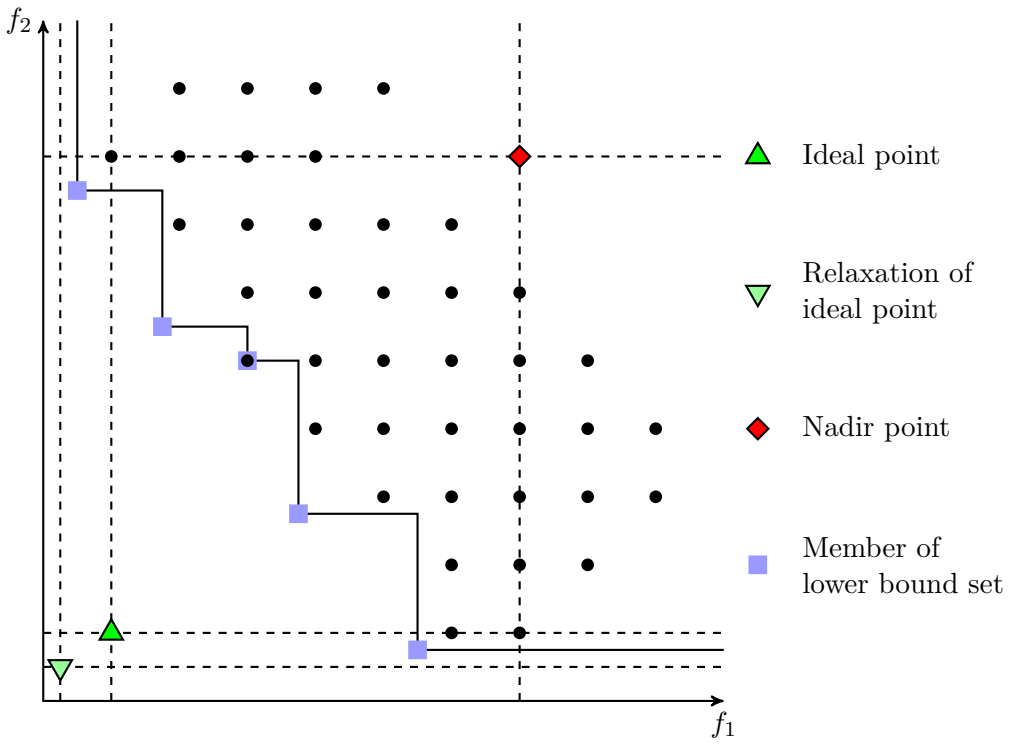


Figure 2.7: Bounds for a BOCO problem.

versions of the shortest path problem (Carraway et al., 1990), the traveling salesman problem (Fischer and Richter, 1982) and the transportation problem (Gallagher and Saleh, 1994). In the same way, multi-objective versions of the knapsack problem have been tackled both by dynamic programming (Cho and Kim, 1997; Bazgan et al., 2009; Delort and Spanjaard, 2010), and branch and bound (Ulungu and Teghem, 1997). Sourd and Spanjaard (2008) developed a multi-objective branch-and-bound framework and apply it to the bi-objective spanning tree problem. Madakat et al. (2013) also use a branch and bound approach to solve a bi-objective time dependent traveling salesman problem. Next, we describe some few exact methods for MOPs whose principles will be used in the chapters that follow.

A Method for Finding the Complete Set of Supported Nondominated Points

Aneja and Nair (1979) presented a method for finding the complete set of supported nondominated points for a bi-objective problem. That is, the method does not find the complete set of nondominated points but rather only the set of supported nondominated points. In addition, the method can also find at least one supported efficient solution corresponding to each of the supported nondominated points. It is often used in a framework called *two phases method* (Ulungu and Teghem, 1995). The method proposed by Aneja and Nair (1979) is different other exact methods (like Evans and Steuer (1973); Gal (1975); Yu and Zeleny (1974)) which search for the set of all supported efficient solutions. The method is based on a weighted sum scalarization and it is summarized in Algorithm 2.1.

After transforming a bi-objective problem into single objective, the method computes the lexicographic optimal points. It then procedes by using a dichotomy search to find the other extreme efficient points. This method terminates after a finite number of iterations when no more extreme points are found.

Algorithm 2.1 Aneja and Nair (1979)'s Method.

Output: the complete set of supported nondominated points $\mathcal{N}$.

Set $k \leftarrow 1$, $L \leftarrow \emptyset$, and $\mathcal{N} \leftarrow \emptyset$.
Compute $y_1^{(1)} = \min_{x \in \mathcal{X}} \{f_1(x)\}$ and $y_2^{(1)} = \min_{x \in \mathcal{X}} \{f_2(x) : f_1(x) = y_1^{(1)}\}$.
Set $\mathcal{N} \leftarrow \{(y_1^{(1)}, y_2^{(1)})\}$.
Compute $y_2^{(2)} = \min_{x \in \mathcal{X}} \{f_2(x)\}$ and $y_1^{(2)} = \min_{x \in \mathcal{X}} \{f_1(x) : f_2(x) = y_2^{(2)}\}$.
if $(y_1^{(2)}, y_2^{(2)}) \neq (y_1^{(1)}, y_2^{(1)})$ **then**
  Set $\mathcal{N} \leftarrow \mathcal{N} \cup \{(y_1^{(2)}, y_2^{(2)})\}$ and $L \leftarrow \{(1, 2)\}$.
  Set $k \leftarrow k + 1$.
  while $L \neq \emptyset$ **do**
    Choose an element $(r, s) \in L$.
    Set $\lambda_1 \leftarrow |y_2^{(s)} - y_2^{(r)}|$ and $\lambda_2 \leftarrow |y_1^{(s)} - y_1^{(r)}|$.
    Solve $\min_{x \in \mathcal{X}} \{\lambda_1 f_1(x) + \lambda_2 f_2(x)\}$ and let x^* be the optimal solution.
    Let $y_1^* = f_1(x^*)$ and $y_2^* = f_2(x^*)$.
    if $(y_1^*, y_2^*) \neq (y_1^{(r)}, y_2^{(r)})$ and $(y_1^*, y_2^*) \neq (y_2^{(s)}, y_2^{(s)})$ **then**
      Set $\mathcal{N} \leftarrow \mathcal{N} \cup \{(y_1^*, y_2^*)\}$ and $L \leftarrow L \cup \{(r, k), (k, s)\}$.
      Set $k \leftarrow k + 1$.
    end if
    Set $L \leftarrow L \backslash \{(r, s)\}$
  end while
end if

The Two Phases Method (Ulungu and Teghem, 1995)

The two phases method (Ulungu and Teghem, 1995) is a general framework for determining the efficient set of bi-objective combinatorial optimization problems. In the first phase of the method, supported solutions are found by solving a series of single objective problems obtained by a weighted sum scalarization. In the second phase, more specialized algorithms are used to search for nonsupported points in regions defined by the solutions found in the first phase. The series of diagrams in Figure 2.8 illustrate the different stages of this method. At any given point of the method and for any two adjacent supported solutions found for the algorithm, three different regions may be defined. There can be no nondominated points in the region shaded in blue, whereas feasible solutions in the region shaded in pink are dominated. It is only in the unshaded region between the two shaded regions where additional nondominated points may lie.

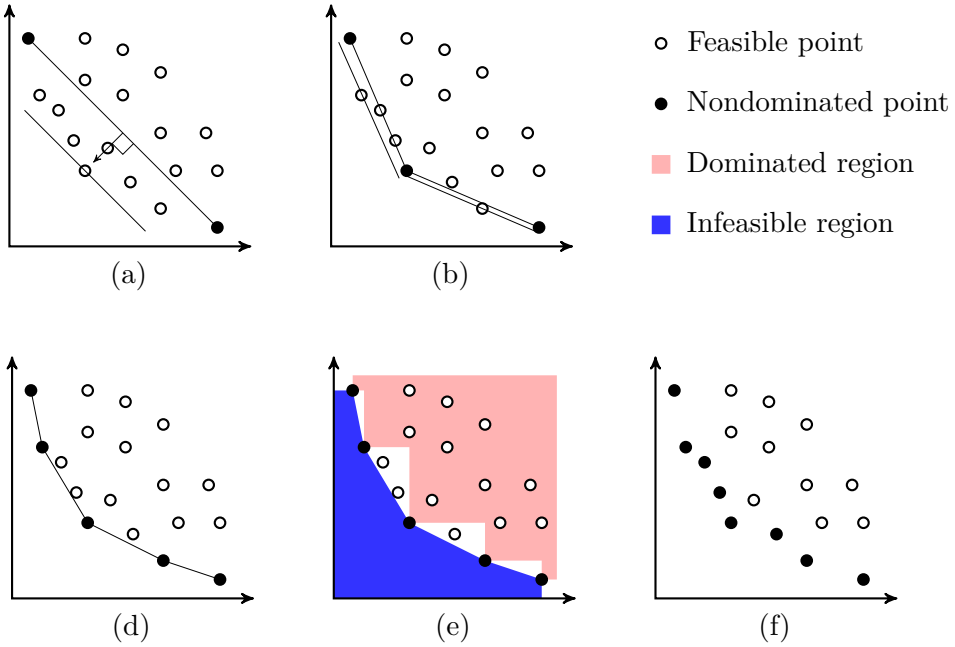


Figure 2.8: Different stages of the two phases method.
Notes: Phase I, (a) – (d), consist in finding supported solutions by using a method based on weighted sum. Unsupported solutions are found in Phase II, (e) – (f), by using problem specific algorithms.

An Exact ε-Constraint Method for BOCO problems (Bérubé et al., 2009)

As we have already noted, the ε-constraint is capable of finding the complete and exact nondominated set. The main challenge with the method is to know for which values of ε the single objective problem should be solved. Although it is very difficult in the general case to choose the correct values of ε, it can efficiently be done for certain classes of problems. The method presented here works for BOCO problems for which the objective coefficients are integers. This method was formally presented by Bérubé et al. (2009) after it had been used by several authors including Jozefowiez et al. (2007). The principle relies on the fact that the cardinality of the nondominated set of a BOCO problem is finite. We can therefore find the whole set by solving a series of ε-constraint problems provided we know exactly how to choose the right values of ε. The method starts by first deciding on which of the two objective functions should be converted into a constraint (by an ε-constraint method). Next, ε is set to a worse possible value of the constraint objective and the resulting problem is solved to optimality. The next value of ε is calculated by subtracting a predetermined step length, Δ, from the value corresponding to the constrained objective in the previously found point. Bérubé et al. (2009) proved that by choosing $\Delta = 1$, the complete nondominated set of a BOCO problem with integral objective coefficients can be found. The set found may contain some *weakly dominated* points and these can be simply deleted at the end of the process. The method is described in Algorithm 2.2.

Algorithm 2.2 An exact ε-constraint method for BOCO problems (Bérubé et al., 2009).

Choose a fixed step length Δ.
Compute the ideal point $y^I = (y_1^I, y_2^I)$, and nadir point $y^N = (y_1^N, y_2^N)$.
Set $\varepsilon \leftarrow y_2^N$ and the nondominated set $\mathcal{Y}_N \leftarrow \emptyset$.
repeat
    Solve the ε-constraint problem: min $\{f_1(x)$ subject to $x \in \mathcal{X}, f_2(x) \leq \varepsilon\}$.
    Let x^* be the optimal solution.
    Compute $f_1^* = f_1(x^*)$ and $f_2^* = f_2(x^*)$.
    Set $\mathcal{Y}_N \leftarrow \mathcal{Y}_N \cup \{(f_1^*, f_2^*)\}$.
    Set $\varepsilon \leftarrow f_2^* - \Delta$.
until $\varepsilon < y_2^I$.
Remove weakly dominated points from $\mathcal{Y}_N$.

Sylva and Crema's Method

The method described in this section was presented by both Sylva and Crema (2004) and Jahanshahloo et al. (2005). Its goal is to find the complete nondominated set of a multi-objective integer linear program (MOILP) by repeatedly solving a single objective problem obtained through a weighted sum with fixed weights. That is, the MOILP is transformed into a single objective problem by using a vector of weights $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r)$ such that $\lambda_i \in]0, 1[$ for all $i = 1, \ldots, r$ and $\sum \lambda_i = 1$. Once the weights are fixed, the resulting problem is solved repeatedly for as long as a new nondominated point is found. This is done by using the previously found points to define zones of the objective space where further nondominated points may lie. That is, constraints on the objective functions are added to the original formulation in order to limit the search space. This destroys the original problem structure and so it may make the problem more and more difficult at each iteration. A new nondominated point is found at each iteration of the method except the one when there are no more nondominated points. Figure 2.9 illustrates the stages of using this method.

The Parallel Partitioning Method

The Parallel Partitioning Method (PPM) was proposed by Lemesre et al. (2007) to solve bi-objective optimization problems. PPM combines ideas from the two phases method and the ε-constraint method. The main idea of PPM is to quickly determine some nondominated points and use them to define search regions where additional nondominated poitns may lie. The method consists of three stages and these are illustrated in Figure 2.10. In the first stage, (2.10a), the two lexicographic optimal points are computed. In the second stage, (2.10b – 2.10c), the objective space is partioned into equally spaced regions with respect to one objective function in order to find fairly distributed nondominated points. The method used here is actually the ε-constraint method and so both supported as well as nonsupported solutions may be found at this stage. This is one main difference between the PPM and the two phases method for which the complete set of nondominated points are searched for in a first phase. The final stage, (2.10b – 2.10c), consists in using the previously identified nondominated points in defining search regions where additional nondominated

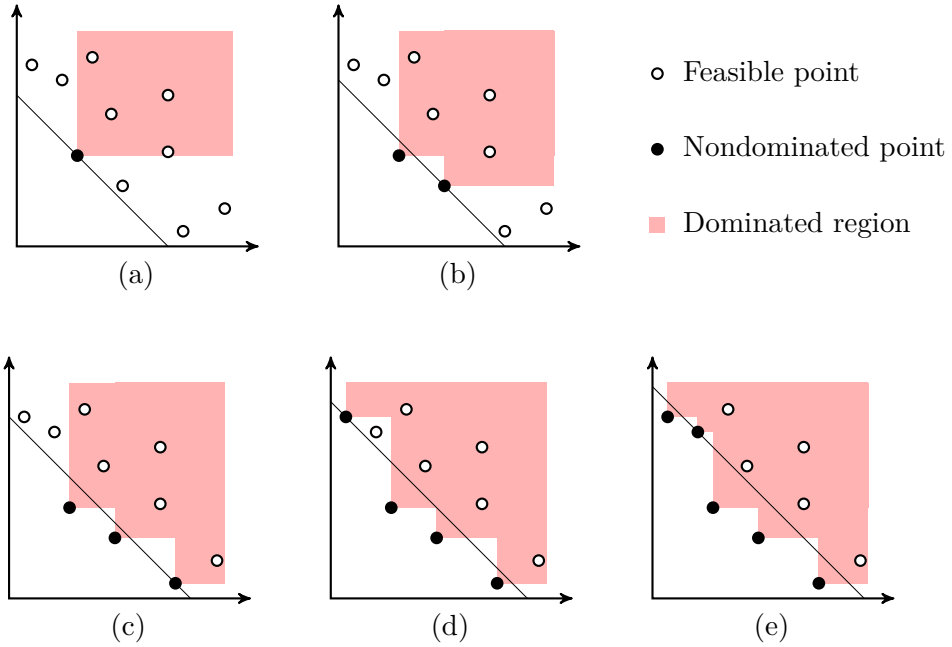


Figure 2.9: Illustration of Sylva and Crema's Method with $\lambda_1 = \lambda_2 = 0.5$.
Notes: A nondominated point is found at every iteration of the algorithm with the exception of the last one when there are no more nondominated points

points may lie. Just as in the two phases method, dedicated methods are used to search for nondominated points in these regions. A generalisation of PPM for problems with k-objectives ($k > 2$) has been proposed by Dhaenens et al. (2010).

2.5.3 Approximation Methods

Due to the diffuclty of MOPs, they are mainly solved with approximation methods (ie. heuristics and metaheuristics). Reeves (1993) defines a heuristic as a technique which seeks good (i.e. near optimal) solutions at a reasonable computational cost without any guarantee of feasibility or optimality. In general, a heuristic is specific to a particluar problem. That is, it cannot be easily adapted for other problems other than the one for which it was designed. A metaheuristic on the other hand is a more general technique that can be easily adapted for a large class of problems. It refers to an iterative strategy that combines different strategies for exploring and exploiting the search space. At each iteration, the method may manipulate one or several solutions. Some popular metaheuristics are genetic algorithms, simulated annealing and tabu search. We refer the reader to Ehrgott and Gandibleux (2004) for a detailed discussion.

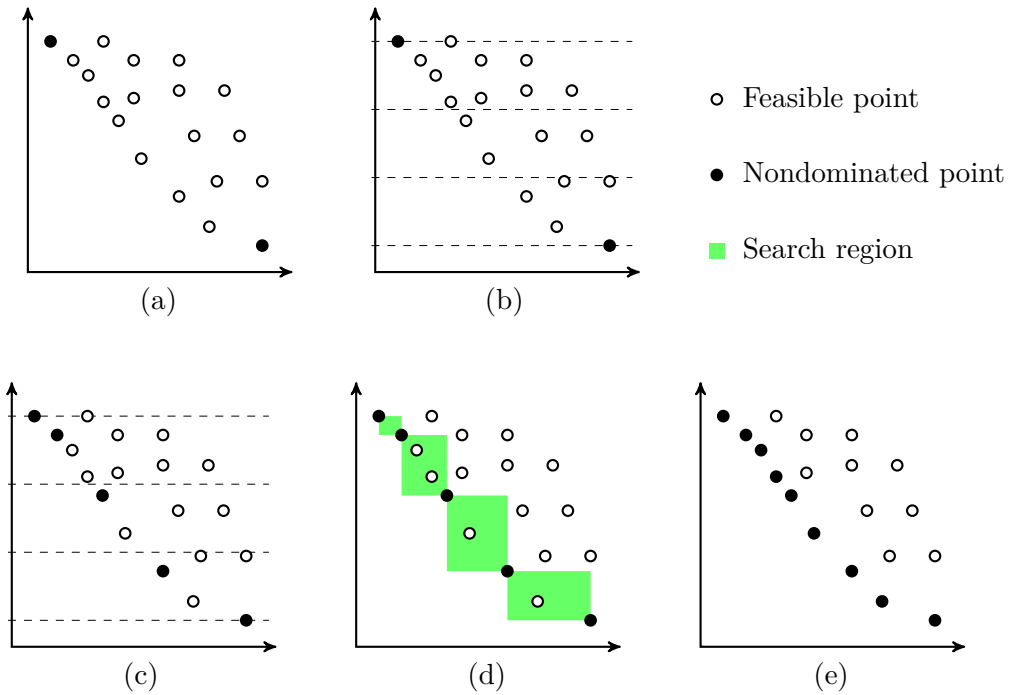


Figure 2.10: Illustration of the parallel partitioning method.
Notes: There are three stages. In stage I, (a), the two lexicographic optimal points are found. State II, (b) – (c), consist in partitioning the range of one objective space into equally spaced regions and solving ε-constraint problems. In stage III, (d) – (e), search regions are defined by the identified nondominated points and the remaining nondominated points are found.

2.6 Evaluating Approximation Methods and Solutions

One of the most challenging tasks in multi-objective optimization is the evaluation of solutions methods and approximate solutions. In order to compare two sets of approximate solutions, the concept of Pareto dominance may be used. Nevertheless, this is a difficult task for several reasons including the fact that some points in either set may be dominated by points in the other set while others may be incomparable. Figure 2.11 shows the possible situations that may arise in comparing two approximate solutions, $\mathcal{P}_A$ and $\mathcal{P}_B$, of a bi-objective problem. In spite of this difficulty, several methods which make use of so called *quality indicators* have been proposed to compare different algorithms and approximate solutions. These methods may, however, not be valid in all cases and interpreting their results may also be difficult (Zitzler et al., 2003). For a given approximation, a *quality indicator* may assign an absolute score, a score relative to a reference set (or point), or a score relative to another approximation. An indicator may also concentrate on aspects like convergence, diversification, or both of the solutions. The different indicators proposed in the literature can be classified according to the number of approximation sets they work on. The most common ones are those that assign a score to a single approximation set (*unary*

quality indicators) and those that assign a score based on two approximation sets (*binary quality indicators*). Below, we describe the *hypervolume indicator* or $\mathcal{S}$*-metric* (Zitzler and Thiele, 1998) and the *binary* ε *indicator* (Zitzler et al., 2003). An analysis and review on performance assessment in multi-objective optimization can be found in Zitzler et al. (2003).

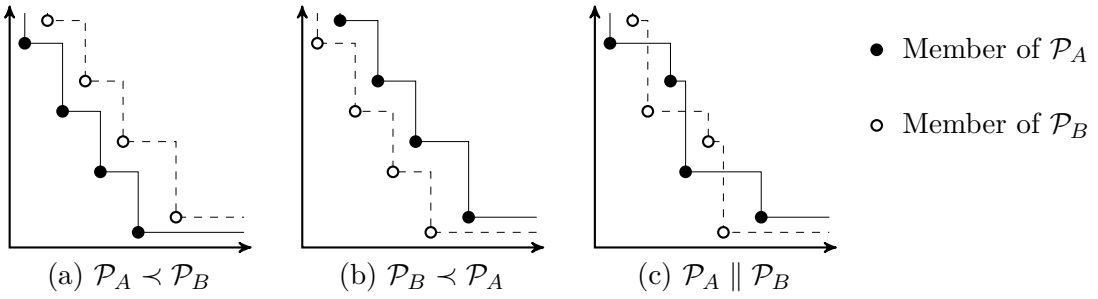


Figure 2.11: Difficulty in comparing approximation solution sets.
Notes: One set of solutions may dominate the other as in (a) and (b) or the two sets may be incomparable as in (c).

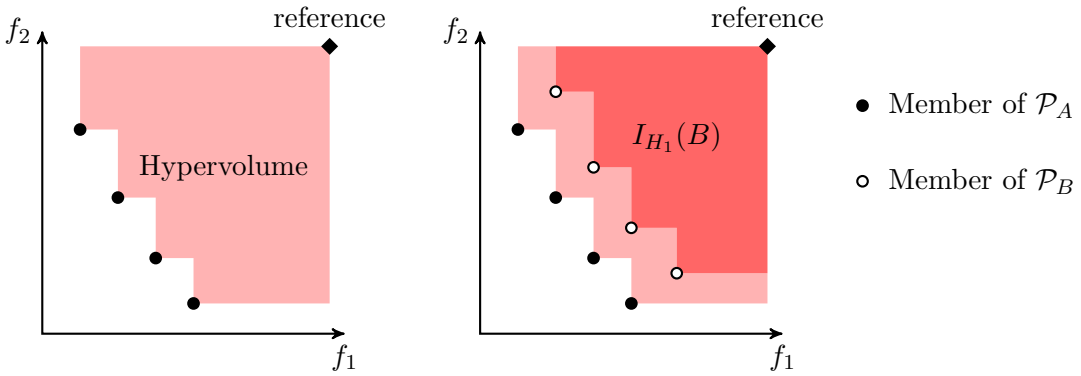
2.6.1 The Hypervolume Indicators

Given a potentially nondominated set $\mathcal{P}_A$, the unary hypervolume indicator (Zitzler and Thiele, 1998) assigns a score $I_{H_1}(A)$ based on the volume enclosed by the set with respect to a reference point (see Figure 2.12). The volume represents the region of the feasible space that is dominated by the set $\mathcal{P}_A$. In order to compare two potentially nondominated sets $\mathcal{P}_A$ and $\mathcal{P}_B$, the binary hypervolume indicator (Zitzler, 1999) assings scores $I_{H_2}(A, B)$ and $I_{H_2}(B, A)$ which represent the volume of the feasible region that is dominated by $\mathcal{P}_A$ (respectively $\mathcal{P}_B$) but not by $\mathcal{P}_B$ (respectively $\mathcal{P}_A$). From these values, we can conclude for example that $\mathcal{P}_A$ is preferred to $\mathcal{P}_B$ if $I_{H_2}(A, B) > 0$ and $I_{H_2}(B, A) = 0$. The use of these indicators requires the definition of a reference point that represents an upper bound on the feasible region. Given a vector of r objective functions, the time required by the unary hypervolume indicator to assign a score to a set $\mathcal{P}_A$ is given by $O(|\mathcal{P}_A|^{r+1})$. For this reason, Knowles and Corne (2002) recommends these indicators for cases where we have very few objective functions and the size of potentially nondominated sets are not so large.

2.6.2 The Binary ε-Indicator

Let $y' = (y'_0, \ldots, y'_r)$ and $y'' = (y''_0, \ldots, y''_r)$ be two arbitrary points in the objective space of a MOP. The point y' is said to *ε-dominate* y'', written as ($y' \preceq_\varepsilon y''$), if and only if there exists a real number $\varepsilon > 0$ such that $y'_i \leq \varepsilon \cdot y''_i$ for all $i \in \{1, 2, \ldots, r\}$. Given two potentially nondominated sets $\mathcal{P}_A$ and $\mathcal{P}_B$, the *binary ε-indicator* I_ε is defined as

$$I_\varepsilon(A, B) \;=\; \inf_{\varepsilon \in \mathbb{R}} \{\forall y'' \in B,\ \exists y' \in A : y' \preceq_\varepsilon y''\} \;,$$



(a) $I_{H_1}(A)$ (b) $I_{H_2}(A, B)$

Figure 2.12: The hypervolume indicators.
Notes: We deduce that $\mathcal{P}_A$ is preferred to $\mathcal{P}_B$ since $I_{H_2}(A, B) > 0$ and $I_{H_2}(B, A) = 0$.

where inf represents the *infimum* or greatest lower bound of a set. This indicator, thus, gives the minimum factor ε such that each point in $\mathcal{P}_B$ is ε-dominated by at least one point in $\mathcal{P}_A$. In other words, it is a representation of how worse $\mathcal{P}_B$ is with respect to $\mathcal{P}_A$ when all objectives are considered. The potentially nondominated set $\mathcal{P}_A$ is preferred to $\mathcal{P}_B$ if $I_\varepsilon(A, B) \leq 1$ and $I_\varepsilon(B, A) > 1$. An advantage of this indicator is that $I_\varepsilon(A, B)$ can be calcluated in a time of $O(n \cdot |A| \cdot |B|)$.

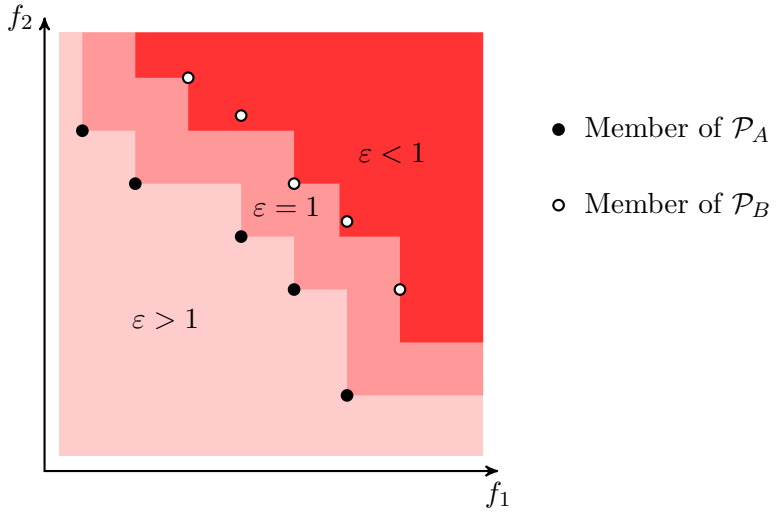


Figure 2.13: The Binary ε-indicator.
Notes: The shaded regions represent the areas that are ε-dominated by $\mathcal{P}_A$ for the indicated values of ε. The medium-shaded area includes the dark-shaded one while the light-shaded area includes both the medium-shaded and the dark-shaded ones.

2.7 Conclusion

This chapter has given a general overview of multi-objective optimization. In particular, we introduced the different terminologies and principles that will be useful for discussions in the chapters that follow. A detailled discussion of the notions introduced in the chapter can be found in a book like Ehrgott (2005).

Chapter 3

Column Generation for Bi-Objective Integer Programs

3.1 Introduction

In Chapter 1 we saw the application of column generation to vehicle routing problems. Chapter 2 introduced multi-objective optimization. In spite of their importance these two subjects have mainly been studied separately rather than together. To the best of the author's knowledge, very few published papers treat the application of column generation to multi-objective problems and even these are quite recent papers. A possible explanation for this is that the notion of lower and upper bounds for MOPs is not so well studied and understood as it is for single objective problems.

We saw in the preceding chapter that the ideal point and the nadir point are well known lower and upper bounds, respectively, for the nondominated set of a multi-objective problem. Nevertheless, these two points are most of the time not very indicative of where the members of the nondominated set lie. Since the solution of a multi-objective problem is a set of solutions rather than a single solution, a better way of defining lower and upper bounds is to use sets of points. Villarreal and Karwan (1981) was the first to propose bounds for a multi-objective problem based on sets of points. They define a lower bound for a MOIP as a set of points $\boldsymbol{L}$ such that each feasible point of the MOIP is dominated by at least one of the members of $\boldsymbol{L}$. The members of $\boldsymbol{L}$ may or may not correspond to feasible solutions. In a similar way, an upper bound may be defined as a set of feasible points $\boldsymbol{U}$ whose members do not dominate one another. Recently, this idea of using sets to define bounds for MOPs has been visited by authors like Ehrgott and Gandibleux (2007), Sourd and Spanjaard (2008) and Delort and Spanjaard (2010). Ehrgott and Gandibleux (2007) introduced the terminology *bound set* to describe a set used as a lower or upper bound of a MOP. They prove some general results on bound sets and discuss how they can be constructed for bi-objective problems by using a weighted sum approach. Sourd and Spanjaard (2008) and Delort and Spanjaard (2010) use ideas similar to those introduced by Ehrgott and Gandibleux (2007) to compute bound sets for the bi-objective spanning tree problem and the biobjective binary knapsack problem, respectively. The computed bounds are then used in a multi-objective branch-and-bound framework to develop exact algorithms for the considered problems. At the time of writing this thesis, we are not

aware of any other published work on bound sets for multi-objective problems apart from the ones cited here.

We could also not find much published work on the use of column generation in the context of multi-objective optimization. A few of the ones we found are worth citing. The first is the work of Ehrgott and Tind (2007) which treats the use of column generation in integer programming with applications in multi-objective optimization. The proposed approach consists in first converting a multi-objective problem into single objective through an ε-constraint method before combining column generation and cutting planes in solving the resulting problem. After finding an efficient solution together with the necessary dual information, a form of sensitivity analysis is used to search for neighbouring efficient solutions without changing the original master problem. Khanafer et al. (2012) also present a bi-objective extension of the bin packing problem in which the goal is to minimize the number of bins used as well as the total number of conflicts between items placed in the same bin. A column generation based metaheuristic is proposed for the problem. The master problem is a single objective set covering model and the subproblem is a variant of the knapsack problem in which the objective function is non-linear. Due to the difficulty of solving the subproblem exactly, heuristics are used.

An example of applying column generation to multi-objective problems that is similar to some strategies we propose later in this chapter is the work of Salari and Unkelbach (2013). The authors discuss an application of column generation for creating treatment plans in the field of radiotherapy. A bi-objective problem that arises in this field is to treat defective cells and also reduce the side effects resulting from the use of radio waves. A continuous convex model in which each column represents a certain treatment dose is proposed for the problem. A weekly plan (feasible point) consists of one or several doses (columns) and a complete treatement program (potentially efficient set) consists of several weekly treatment plans. The bi-objective problem is first converted into single objective through a weighted sum approach before applying column generation to obtain an approximation to the set of efficient solutions. In principle, each weekly plan can be obtained by fixing the weight vector and solving the problem to optimality by column generation. Using this approach, however, results in treatment programs in which completely different doses are to be administered each week. Such a treatment program is clinically impractical. In order to obtain a clinically practical treatement program, the authors use the following idea. At each iteration, a treatment program is constructed based on the currently available columns in the master problem. The subproblem to search for new columns is formulated in such a way that a set of columns that best improves the current treatment program (instead of just one weekly plan) is obtained. This approach helps in obtaining a treatment program in which similar weekly doses are administered. A similar column generation approach is proposed in Peng et al. (2012). Some other problems dealing with the application of column generation in multi-objective optimization can be seen in Bard and Purnomo (2005), Chauny et al. (2000), and Ogryczak (2000).

Contributions and Organization of Chapter. We discuss the application of column generation to bi-objective integer programs (BOIPs). We will concentrate more specifically on bi-objective problems with applications to vehicle routing problems. We treat how to compute lower bound sets for these problems by using a column method. In Section 3.2

we discuss how lower and upper bound sets for BOIPs can be constructed by considering different cases. We will start with the case where the number of variables in the given formulation is non-exponential and so there is no need for column generation. Given that a model which is well adapted for column generation usually has an exponential number of variables, we discuss how a lower bound set for a problem based on such a model can be computed. In computing lower bounds by column generation, we will show that we often need to solve a series of subproblems that are similar in form. For this reason we will seek possible ways to efficiently search for columns that are possibly useful for serveral subproblems at the same time without necessarily solving all these subproblems. The last section of the chapter is concerned with the evaluation of lower bound sets. We also give an example of the the bi-objective set covering problem (BOSCP) and show that the lower bound sets computed by the proposed methodology for this problem are good.

Related Publications. A large part of this chapter forms the core of an article that has been accepted for publication in the international journal RAIRO - Operations Research (Sarpong et al., 2013c). Some other parts are also used in a paper which is yet to be submitted to another international journal.

3.2 Constructing Bound Sets for BOIPs

We discuss how to construct lower and upper bound sets for BOIPs by considering the following problem, P, defined by :

$$\text{Minimize} \quad (c^1)^T x \tag{3.1}$$

$$\text{Minimize} \quad (c^2)^T x \tag{3.2}$$

$$\text{subject to :} \quad Ax \geq b\,, \tag{3.3}$$

$$x \geq 0 \text{ and integer.} \tag{3.4}$$

In this formulation, x is an n vector of decision variables and c^i is an n vector of integer coefficients in the ith objective function ($i = 1, 2$). The constraints are expressed by using an $m \times n$ matrix of coefficients, A, and an n vector of right hand side constants, b.

The main idea used in computing bound sets for a BOIP is to convert the bi-objective problem into single objective by using a scalarization method and then solve the resulting problem (or a relaxation of it) several times by varying the necessary parameters. In this thesis, we consider the weighted sum and the ε-constraint methods. We recall from Chapter 2 that the weighted sum method transforms Formulation (3.1 – 3.4) into single objective by using a vector of non-negative weights $\lambda = (\lambda_1, \lambda_2)$. The resulting problem $P(\lambda)$ is given by :

$$\text{Minimize} \quad \lambda_1 \cdot (c^1)^T x + \lambda_2 \cdot (c^2)^T x \tag{3.5}$$

$$\text{subject to :} \quad Ax \geq b\,, \tag{3.6}$$

$$x \geq 0 \text{ and integer.} \tag{3.7}$$

On the other hand, the ε-constraint approach obtains a single objective problem by restricting the worst possible value of one objective (say the second one) to a constant

$\varepsilon \in \mathbb{R}$. Thus, we obtain the single objective problem $P(\varepsilon)$:

$$\begin{aligned} & \text{Minimize} && (c^1)^T x && (3.8)\\ \text{subject to :} \quad & && Ax \geq b\,, && (3.9)\\ & && -(c^2)^T x \geq -\varepsilon\,, && (3.10)\\ & && x \geq 0 \text{ and integer.} && (3.11) \end{aligned}$$

We may distinguish two main cases for the single objective problems obtained by each scalarization method. The first case is when there is an efficient algorithm to solve $P(\lambda)$ or $P(\varepsilon)$. The other case is when both $P(\lambda)$ and $P(\varepsilon)$ are $\mathcal{NP}$-hard. Many practical problems, like VRPs, are $\mathcal{NP}$-hard even in their single objective form so their bi-objective extensions as well as both $P(\lambda)$ and $P(\varepsilon)$ are $\mathcal{NP}$-hard. Some other problems which are polynomial in their single objective form become $\mathcal{NP}$-hard in the bi-objective form. In this thesis, we will concentrate on the case where both $P(\lambda)$ and $P(\varepsilon)$ are $\mathcal{NP}$-hard. In this case, we have to resort to relaxations of problem $P(\lambda)$ or $P(\varepsilon)$ in order to compute a lower bound set for problem P. In general, an upper bound set for problem P is computed through heuristics or metaheuristics which may or may not rely on any scalarization method. In what follows, we describe the general algorithms that can be used in computing a lower bound set when we use each scalarization method. In a first instance, we suppose that P is manageable in the sense that we have a "small" number of variables in x (i.e. n is small). In such a situation, there is no need for column generation in computing a lower bound set for the problem.

3.2.1 Using the Weighted Sum Method

Using the weighted sum method to compute lower bound sets for BOIPs has been discussed in detail by Ehrgott and Gandibleux (2007). In this section, we only point out some few things that will be useful for our discussion later on. If there is an efficient algorithm to solve $P(\lambda)$, then we can compute the complete set of supported points of $P(\lambda)$ by using an algorithm proposed by Aneja and Nair (1979) (Algorithm 2.1). A lower bound set is defined by the lines connecting each supported solution to the closest one. The lower bound set obtained in this way is called the *convex hull lower bound set* of problem P and it is the best lower bound set that can be found by the weighted sum method (see Figure 3.1). If $P(\lambda)$ is $\mathcal{NP}$-hard, then we rather have to compute the convex hull lower bound set of a relaxation of P. In this manuscript we will only consider the linear relaxation P_L of P. In general, the members of a lower bound set may correspond to points that are infeasible for problem P. The definition of an upper bound set, however, requires that its members are feasible points of P. For this reason, heuristics and metaheuristics are used in computing upper bound sets. An example is the greedy heuristics used in computing upper bound sets for instances of the bi-objective set covering problem (see Ehrgott and Gandibleux (2007)). We stress that the heuristic or metaheuristic used in computing an upper bound set does not necessarily need to depend on the weighted sum method. Any known heurisitic or metaheuristic for problem P may be used.

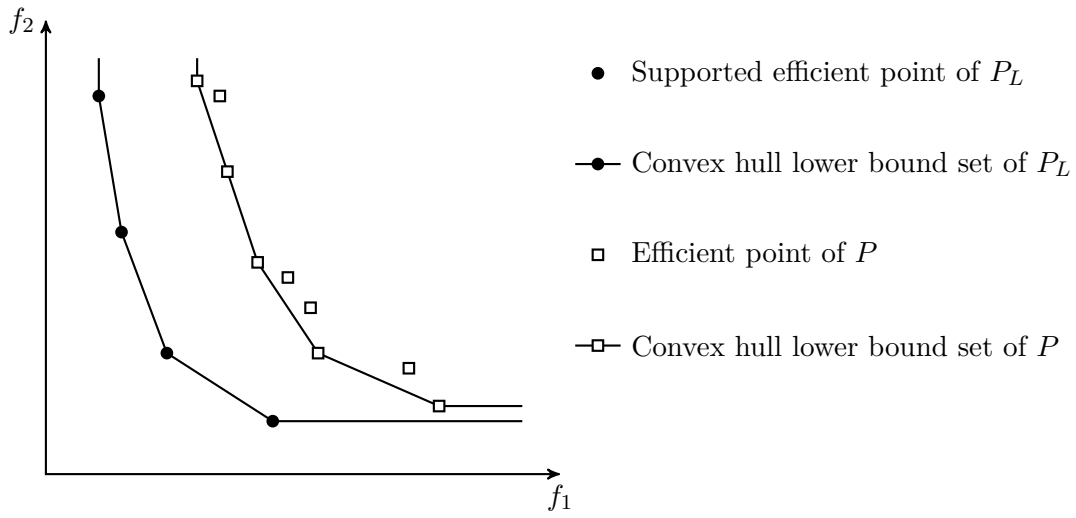


Figure 3.1: Constructing a lower bound set through a weighted sum method.

3.2.2 Using the ε-Constraint Method

The main idea in using an ε-constraint method to compute a lower bound is similar to the case where we use a weighted sum method. That is, depending on whether $P(\varepsilon)$ is $\mathcal{NP}$-hard or not, we will need to deal with $P(\varepsilon)$ directly or a relaxation of it. In general, problem $P(\varepsilon)$ is $\mathcal{NP}$-hard due to the addition of the constraint on an objective. The discussion in the section will therefore treat this case by considering the linear relaxation $P_L(\varepsilon)$ of $P(\varepsilon)$. The method consists in solving $P_L(\varepsilon)$ for different values of ε in such a way that each feasible point of $P(\varepsilon)$ is dominated by at least one of the computed points. The procedure is summarized in Algorithm 3.1 and we suppose that all feasible values for $(c^2)^T x$ lie in the range $[\min \varepsilon, \max \varepsilon]$. The output of Algorithm 3.1 is the set $\boldsymbol{L}$ and a sequence of values $\max \varepsilon = \varepsilon_1 > \varepsilon_2 > \cdots > \min \varepsilon$ corresponding to the elements of the set $\boldsymbol{L}$. Note that each step size δ_i is to be chosen in such a way that there can be no feasible point (f_1^*, f_2^*) of the original integer program satisfying the inequalities $f_2^i < f_2^* < \varepsilon_{i+1}$ (see Figure 3.2). This is a very difficult thing to do for a general bi-objective problem. For the considered problems in this document, however, we have integer objective coefficients so we can use an idea similar to the one used by Bérubé et al. (2009). At iteration i, we define the step size as $\delta_i = 1 - (\lceil f_2^i \rceil - f_2^i)$. As we will later see, it is possible to define even better step sizes for specific problems. We show in Proposition 3.1 that the set $\boldsymbol{L}$ returned by the algorithm is indeed a lower bound set for the considered problem.

Proposition 3.1. *Each feasible point (f_1^*, f_2^*) of problem (3.1 – 3.4) is either dominated by at least one point of the set $\boldsymbol{L}$ constructed by Algorithm 3.1 or is a point of $\boldsymbol{L}$.*

Proof. We know that the dominance relation ($\preceq$) is transitive (i.e. if $x^1 \preceq x^2$ and $x^2 \preceq x^3$, then $x^1 \preceq x^3$). We also know that each feasible point of a given multi-objective problem is either dominated by at least one element of the nondominated set $\mathcal{Y}_N$ or is itself a nondominated point. For these reasons, we show that each member $(f_1^*, f_2^*) \in \mathcal{Y}_N$ either belongs to $\boldsymbol{L}$ or is dominated by at least one member of $\boldsymbol{L}$. Let $(f_1^*, f_2^*) \in \mathcal{Y}_N$, then the

Algorithm 3.1 Using an ε-constraint method to compute a lower bound set

Output: A lower bound set $\boldsymbol{L}$ for the considered problem.

1: Set $\boldsymbol{L} \leftarrow \emptyset$.
2: Set $i \leftarrow 1$, and $\varepsilon_i \leftarrow \max \varepsilon$.
3: **while** $\varepsilon_i \geq \min \varepsilon$ **do**
4: Solve linear relaxation of Formulation (3.8 – 3.11) for $\varepsilon = \varepsilon_i$.
5: Let x^* be the optimal solution vector.
6: Compute the optimal values $f_1^i = (c^1)^T x^*$ and $f_2^i = (c^2)^T x^*$.
7: Set $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup (f_1^i, f_2^i)$.
8: Choose $\delta_i > 0$ such that there can be no feasible point (f_1^*, f_2^*) of the original integer program satisfying $f_2^i - \delta_i < f_2^* < f_2^i$.
9: Set $i \leftarrow i + 1$ and $\varepsilon_i \leftarrow f_2^{i-1} - \delta_{i-1}$.
10: **end while**

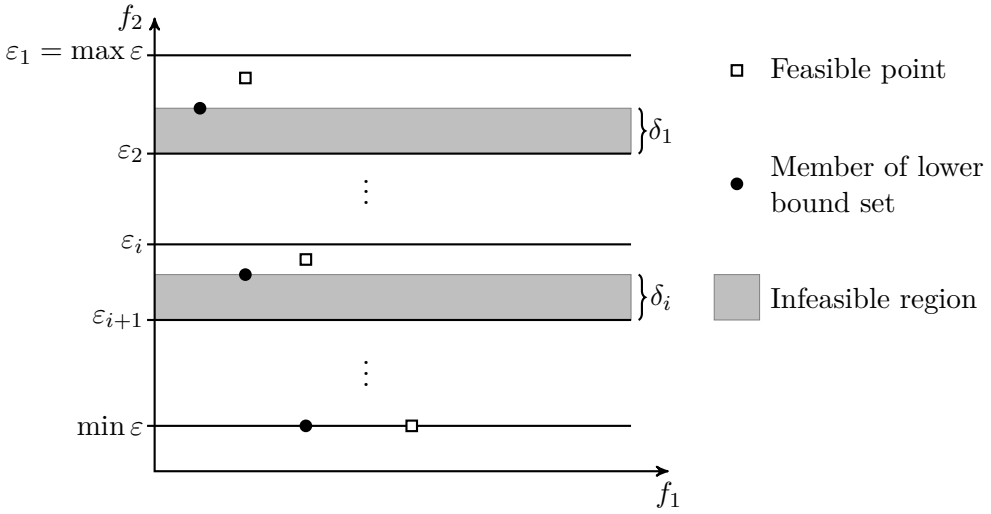


Figure 3.2: Constructing a lower bound set through an ε-constraint method.

theory behind the ε-constraint method assures us that there is a value of $\varepsilon = \varepsilon^*$ for which the solution of problem (3.8 – 3.11) maps onto the point (f_1^*, f_2^*). The exact value of ε^* is not unique so let us choose, without loss of generality, $\varepsilon^* = f_2^*$. By the way the set $\boldsymbol{L}$ is constructed, we know that $f_2^i \leq f_2^* = \varepsilon^* \leq \varepsilon_i$ at a certain iteration i of Algorithm 3.1. Moreover, since we solve linear relaxations when computing the members of $\boldsymbol{L}$, we have $f_1^i \leq f_1^*$. Hence, $(f_1^i, f_2^i) \leqq (f_1^*, f_2^*)$. □

We recall once again that an upper bound set can be computed through heuristics or metaheuristics that do not necessary depend on any scalarization method.

3.3 Constructing Lower Bound Sets for BOIPs by Column Generation

Many practical problems formulated in the form of (3.1 – 3.4) have an exponential number of decision variables. It is impractical and sometimes impossible to consider all the variables at once when treating such problems. As we saw in Chapter 1, the idea used by a column geneation method in solving such a problem is to start with only a reasonable number of variables (and the corresponding columns of matrix A) for which the problem is primal feasible. The other variables together with their corresponding columns of A are introduced when necessary by solving an auxillary problem. We recall that, the linear relaxation of the original integer program with an exponential number of variables (or columns) is called the linear programming master problem (LPM) and its restriction to a reasonable number of columns is called the restricted LPM (RLPM). The auxillary problem solved to propose new variables to be added to the RLPM or prove the convergence of the method is called the subproblem (S). Each iteration of column generation starts by solving the RLPM to obtain optimal primal and dual solutions. The subproblem uses the optimal dual values to search for new columns that can possibly improve the current objective value of the RLPM. One or several of such variables, if found, are added to the RLPM and the process repeats. The method converges when no new variables are proposed by the subproblem.

Next, we discuss how to use column generation in computing a lower bounds set for a BOIP when it is formulated with an exponential number of variables. We list the different models involved in a column generation method in the case of a weighted sum method and also those in the case of an ε-constraint method.

Using a Weighted Sum Method

Linear Programming Master Problem (LPM(λ)):

$$\begin{aligned} & \text{Minimize} && (\lambda_1 c^1 + \lambda_2 c^2)^T x && (3.12)\\ \text{subject to :} \quad & && Ax \;\geq\; b\,, && (3.13)\\ & && x \;\geq\; 0\,. && (3.14) \end{aligned}$$

Dual of LPM (DLPM(λ)): Let π be the vector of dual variables associated with Constraints (3.13). The dual formulation is

$$\begin{aligned} & \text{Maximize} && b^T\pi && (3.15)\\ \text{subject to :} \quad & && A^T\pi \;\leq\; \lambda_1 c^1 + \lambda_2 c^2\,, && (3.16)\\ & && \pi \;\geq\; 0\,. && (3.17) \end{aligned}$$

Subproblem (S(λ)): It is defined as finding variables that correspond columns of matrix A not already in the RLPM and which satisfy

$$\lambda_1 c^1 + \lambda_2 c^2 - A^T\pi \;<\; 0\,. \tag{3.18}$$

Using an ε-Constraint Method

Linear Programming Master Problem (LPM(ε)):

$$\begin{aligned} & \text{Minimize} \quad c^1 x && (3.19)\\ \text{subject to :} \quad & Ax \geq b\,, && (3.20)\\ & -c^2 x \geq -\varepsilon\,, && (3.21)\\ & x \geq 0\,. && (3.22) \end{aligned}$$

Dual of LPM (DLPM(ε)): Let π be the vector of dual variables associated with Constraints (3.20) and φ the dual variable associated with Constraint (3.21). The dual formulation is

$$\begin{aligned} & \text{Maximize} \quad b^T\pi - \varepsilon\varphi && (3.23)\\ \text{subject to :} \quad & A^T\pi - \varphi c^2 \leq c^1\,, && (3.24)\\ & \pi,\ \varphi \geq 0\,. && (3.25) \end{aligned}$$

Subproblem (S(ε)): It is defined as finding variables that correspond columns of matrix A not already in the RLPM and which satisfy

$$c^1 + \varphi c^2 - A^T\pi < 0\,. \tag{3.26}$$

General Form of the Subproblem

By comparing the inequalities (3.18) and (3.26), we can see that the subproblems encountered in both cases have a similar form. In addition, the general form of subproblem obtained by using a particular scalarization method does not change when we modify the parameters. Only the values of the dual variables and coefficients change. A similar result is obtained if we consider problems with more than two objectives. This means that strategies we describe in solving the subproblems obtained by one of these scalarization methods can easily be adapted to the other scalarization methods. Moreover, for any given scalarization method, it is possible to treat more than one subproblem at the same time when searching for columns. For example, it may be possible to easily modify a column found by solving the subproblem for a specific value of the parameter λ (or ε) in order to find another column for a different value of λ (or ε) without the need to solve the subproblem for this new value. We present different strategies that can be used to efficiently search for columns in next subsection.

3.3.1 Column Search Strategies

We discuss some approaches to search for relevant columns when computing a lower bound set by column generation. A very general column generation method that can be used in computing a lower bound set for a BOIP is summarized in Algorithm 3.2. The algorithm starts by first converting the BOIP into single objective through a scalarization method. We will limit ourselves to only the weighted sum method and the ε-constraint method. An RLPM is then formulated for the chosen scalarization method as discussed in the preceding

subsection before starting the actual column generation method. An iteration consists of first generating a set of points by solving the RLPM for one or more values of the necessary parameters based on the chosen scalarization method. For the weighted sum method, this means choosing one or more values of the vector λ and solving the RLPM for these values in order to find the corresponding supported points (based on the currently available columns in RLPM) together with the necessary dual values. In the case of the ε-constraint method, we need to solve LRPM for different values of ε to find the associated points and the corresponding dual values. After generating the points, we then need to solve the subproblem corresponding to one or several points and search for relevant columns. If it is proven that the RLPM has converged for any points, these points are saved. Any found columns are added to the RLPM and the process continues. The method terminates when the RLPM converges for all relevant values of the parameters for the chosen scalarization method.

Algorithm 3.2 A generalized column generation method for BOIPs

Output: A lower bound set $\boldsymbol{L}$ for the considered problem.

1: Choose a scalarization method and convert the BOIP into single objective.
2: Formulate RLPM for the chosen scalarization method.
3: Set $\boldsymbol{L} \leftarrow \emptyset$.
4: **repeat**
5: Generate a set of points by solving RLPM for different values of the parameter(s) for which RLPM has not converged.
6: Search for a set of columns that are relevant for one or several subproblems.
7: Add one or several found columns to RLPM.
8: Save any converged points in $\boldsymbol{L}$.
9: **until** RLPM converges for all *relevant* values of the parameters used.

Next, we present two main approaches for implementing Algorithm 3.2 namely the Point-by-Point Search (PPS) and the Sequential Search approaches. The main difference between these two is the order in which we search for the points in a lower bound set. Different variants of each of the two main approaches can also be defined based on the order in which we search for columns and the strategies used to accelerate the column generation method. In what follows, we describe the two main approaches for the cases in which we use a weighted sum method and an ε-constraint method. We demonstrate certain aspects only for the case where we use an ε-constraint method. These aspects can, however, be easily adapted to the case where a weighted sum method is used.

Point-by-Point Search (PPS)

A standard and very intuitive way of implementing Algorithm 3.2 involves solving the RLPM completely for any given value of λ (or ε) before continuing to a different value of λ (or ε). That is, for any given value of λ (or ε), RLPM(λ) or RLPM(ε) is solved by column generation until the subproblem proposes no new columns that can improve the current objective value of the RLPM. In the case of the weighted sum method this correponds to adapting the algorithm described by Aneja and Nair (1979) and which is

given in Algorithm 2.1. In the case of an ε-constraint method, the approach is similar to the one described in Algorithm 3.1 except that in Step 4 we need to solve RLPM(ε) to optimality by column generation. We call this approach the Point-by-Point Search (PPS) since the search is concentrated on finding a particular point of a lower bound set before moving on to another. The PPS identifies the points of a lower bound set following a predictable order. For the case of an ε-constraint method this order is indicated in Figure 3.3. The point corresponding to a smaller value of ε can only be found after all the points corresponding to greater values of ε have been found. Although the PPS is simple and quite easy to implement, it takes no advantage of the similarities of the subproblems solved for the different values of the parameter λ (or ε). The column generation method may also be slow to converge for a given value of the parameter but the PPS requires us to wait for it to converge before moving on to a different value of the parameter. This can result in a huge number of "irrelevant" columns being added to the RLPM and a long computational time. A variant of the PPS that aims at addressing some of these problems is the k-Step Point-by-Point Search (k-PPS) which is presented below.

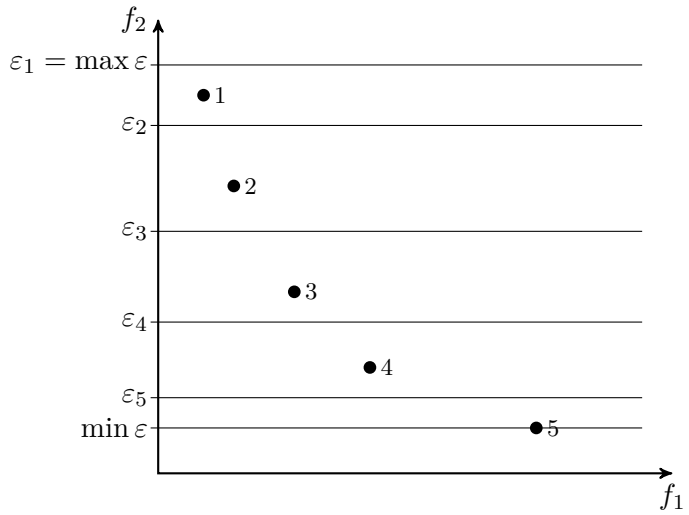


Figure 3.3: Order in which points of a lower bound set are identified by the PPS.

k-Step Point-by-Point Search (k-PPS)

In this approach, we have the freedom to leave a given value of ε before the RLPM has converged for it and switch to another value of ε. If it is necessary, the approach will return to a previously skipped value of ε later on. The k-PPS approach is summarized in Algorithm 3.3. The first step of this approach is to define the condition $\mathcal{O}$ under which a given value of ε for which the RLPM has not converged will be skipped. The condition can be as simple as setting a fixed number k of column generation iterations. Instead of using a fixed number of iterations, we may also decide to skip a given value of ε when the objective value of the RLPM has not improved significantly after k iterations. This can be a way of addressing some column generation problems like *the plateau effect* and *the tailing-off effect* that were described in Chapter 1. It is also possible to use any other

condition we find appropriate for a given problem. In Step 6 of Algorithm 3.3, we say that a value ε_i corresponds to a point (f_1^*, f_2^*) of $\boldsymbol{L}$ if $f_2^* \leq \varepsilon_i \leq \varepsilon_*$, where $\varepsilon_* \in \mathcal{E}$ is the value of ε for which (f_1^*, f_2^*) was computed. In such a case, there is no need to solve RLPM for ε_i since we are sure to end up with a solution that maps onto (f_1^*, f_2^*).

Algorithm 3.3 k-Step Point-by-Point Search (k-PPS)

1: Define a condition $\mathcal{O}$ under which a given value of ε will be skipped before RLPM(ε) converges for this value.
2: Set $\boldsymbol{L} \leftarrow \emptyset$ and $\mathcal{E} \leftarrow \emptyset$.
3: **repeat**
4: Set $i \leftarrow 1$, and $\varepsilon_i \leftarrow \max \varepsilon$.
5: **while** $\varepsilon_i \geq \min \varepsilon$ **do**
6: **if** ε_i does not correspond to a converged point of $\boldsymbol{L}$ **then**
7: Solve RLPM(ε) by column generation until the method converges or condition $\mathcal{O}$ is satisfied.
8: Let x^* be the current optimal solution vector.
9: Compute the current optimal values $f_1^i = (c^1)^T x^*$ and $f_2^i = (c^2)^T x^*$.
10: **if** the method converged in Step 7 **then**
11: Set $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup (f_1^i, f_2^i)$ and $\mathcal{E} \leftarrow \mathcal{E} \cup \varepsilon_i$.
12: **end if**
13: Choose an appropriate step size $\delta_i > 0$.
14: Set $i \leftarrow i + 1$ and $\varepsilon_i \leftarrow f_2^{i-1} - \delta_{i-1}$.
15: **end if**
16: **end while**
17: **until** RLPM(ε) converges for all values of ε in the inner loop.

Unlike in the case of the PPS, the points of a lower bound set are identified by the k-PPS in no particular order. That is, the RLPM can converge for a smaller value of ε before it converges for a greater value. Figure 3.4 illustrates a possible order in which the points of a lower bound set are identified by the k-PPS. This illustration is based on a fictitious example. We suppose that feasible values for the ε-constraint objective function are integers between $\min \varepsilon = 2$ and $\max \varepsilon = 20$. Each complete iteration of the k-PPS approach will thus start by solving RLPM(ε) for $\varepsilon = 20$ as in Figure 3.4a. A full description of the series of diagrams is given below. Note that we can obtain non-integral points since we are solving linear relaxations rather than integer programs. During the process, only points that are proved to have converged are saved from one iteration to another.

Figure 3.4a. There are no converged points at the start of the algorithm. Condition $\mathcal{O}$ is satisfied before RLPM(20) converges and the current unconverged point is $(13.9, 8.0)$. Similarly, condition $\mathcal{O}$ is satisfied before RLPM(7) converges. The corresponding unconverged point is $(21.5, 2.0)$.

Figure 3.4b. This iteration of k-PPS also starts with no converged points from the previous iteration. By solving RLPM(20) by column generation, condition $\mathcal{O}$ is satisfied before the method converges. The current unconverged point is $(12.2, 10.0)$. Next, we solve

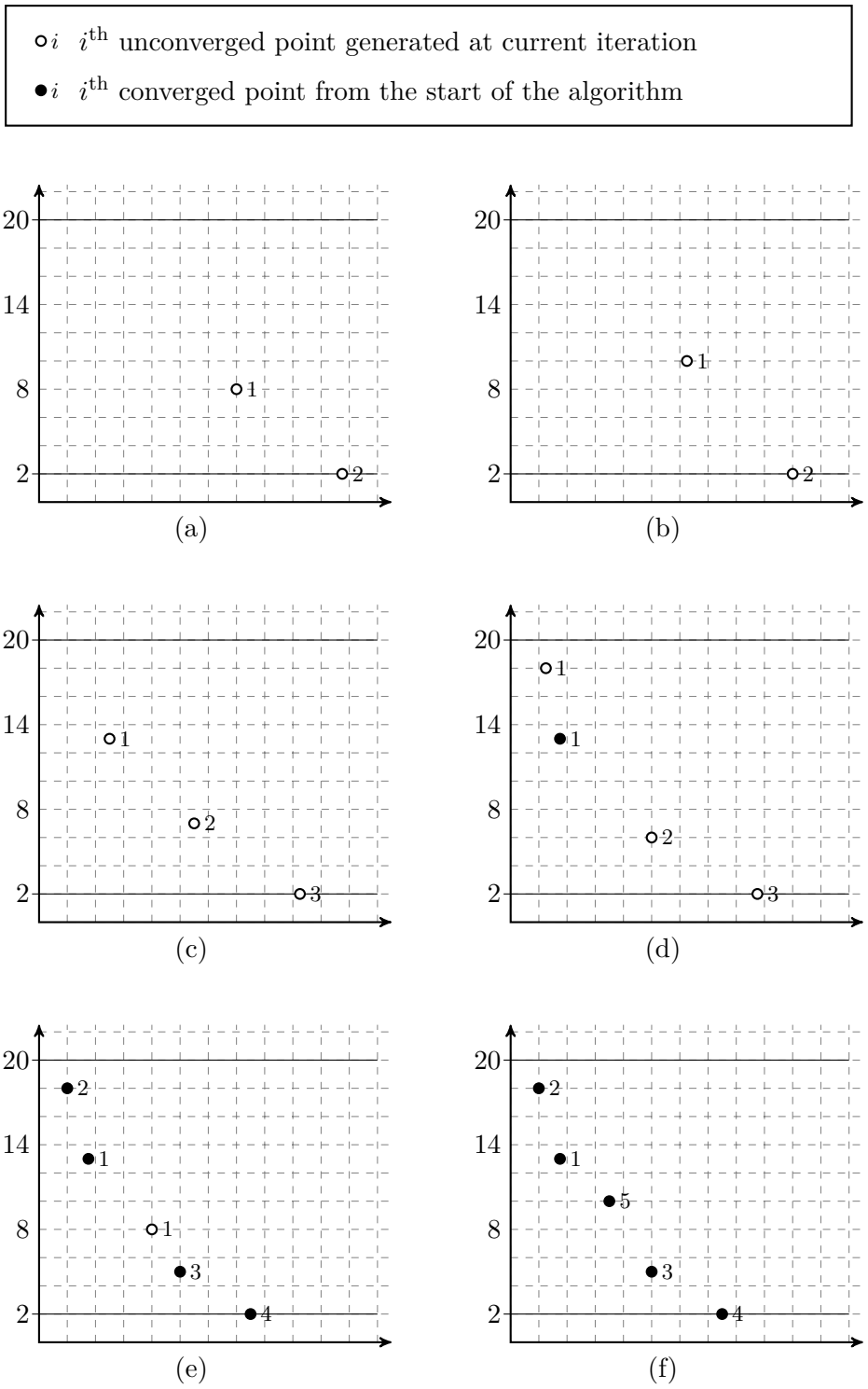


Figure 3.4: Possible order in which points of a lower bound set are identified by the k-PPS.

RLPM(9) by column generation and again condition $\mathcal{O}$ is satisfied before it converges. The corresponding unconverged point is $(20.0, 2.0)$.

Figure 3.4c. There are no converged points at the start of this iteration. Condition $\mathcal{O}$ is satisfied before RLPM(20) converges and the current unconverged point is $(4.9, 13.0)$. In a similar way, we obtain two additional unconverged points $(10.8, 7.0)$ and $(18.2, 2.0)$ corresponding to RLPM(12) and RLPM(6), respectively.

Figure 3.4d. There are still no converged points from the previous iterations. Solving RLPM(20) produces an unconverged point $(2.2, 18.0)$ since condition $\mathcal{O}$ is satisfied before the method converges. The column generation method converges for RLPM(17) before condition $\mathcal{O}$ is satisfied. The corresponding converged point is $(3.7, 13.0)$. Two other unconverged points $(18.2, 6.0)$ and $(17.5, 2.0)$ corresponding to RLPM(12) and RLPM(5), respectively, are produced.

Figure 3.4e. The point which converged in the preceding iteration is known at the start of this iteration. RLPM(20) converges at the point $(2.0, 18.0)$ before condition $\mathcal{O}$ is satisfied. There is no need to solve RLPM for values of ε between 13 and 17 since they correspond to the previously converged point $(3.7, 13.0)$. We thus solve RLPM(12) but this fails to converge before condition $\mathcal{O}$ is satisfied. The corresponding unconverged point is $(8.0, 8.0)$. Solving RLPM(7) produces another converged point $(10.0, 5.0)$ before condition $\mathcal{O}$ is satisfied. Another converged point $(14.9, 2.0)$ corresponding to RLPM(4) is also obtained.

Figure 3.4f. All previously converged points from preceding iterations are known. There is no need to solve RLPM for values of ε between 18 and 20 since they correspond to a converged point. The same is true for values between the interval 13 and 17. We move directly to RLPM(12) and we obtain a converged $(6.9, 10.0)$ before condition $\mathcal{O}$ is satisfied. There is no need to solve RLPM for values of ε between 2 and 9 since each of those values correspond to a converged point. The method k-PPS approach stops at this point.

It is important to note that the set returned by the k-PPS represents a lower bound for the considered BOIP. In other words, if a given value of ε is skipped, then the algorithm will either return to the skipped value later on or prove that the skipped value corresponds to an already converged point. Hence, no relevant points are missed. One main challenge of the k-PPS is that it is not easy to decide on a good condition $\mathcal{O}$ when we don't have enough information on how the RLPM behaves at the start of the algorithm. For this reason, it may be necessary to have an adaptive condition $\mathcal{O}$ that can be modified across the iterations when we have enough information.

Sequential Search

The main idea behind the sequential search approach is to work on a whole frontier rather than deal with individual points. This is achieved by seeking a set of columns that are relevant for several values of the parameter ε (or λ) for which the RLPM has not converged.

At the start of a sequential search approach, we first need to solve the RLPM for several values of the parameter (ε or λ) for which it has not converged without generating any columns. In the case where we use a weighted sum method, this can be achieved by Algorithm 3.4 which is adapted from the algorithm described by Aneja and Nair (1979). Algorithm 3.5 also shows how the first step of a sequential search iteration can be achieved in the case of an ε-constraint method. After generating a set of points, the second step in a sequential search approach is to search for a set of columns that is relevant for the whole set of the points generated in the first step. That is, for a large number of the points generated in the first step, we want to find at least one column which is of negative reduced cost. Different variants of the sequential search approach may be defined based on this second step. Two of such variants are sequential search 1 and sequential search 2 that are presented in Algorithms 3.6 and 3.7, respectively. These two algorithms are presented for the case where an ε-constraint method is used. The case for a weighted sum method is similar.

Algorithm 3.4 Generate a set of Points (weighted sum method)

Output: Set E of all extreme efficient points of RLPM.

Set $k \leftarrow 1$, $P \leftarrow \emptyset$, and $E \leftarrow \emptyset$.
Compute the first lexicographically optimal point $y^{(1)} = (y_1^{(1)}, y_2^{(1)})$ of RLPM.
Set $E \leftarrow \{y^{(1)}\}$.
Compute the second lexicographically optimal point $y^{(2)} = (y_1^{(2)}, y_2^{(2)})$ of RLPM.
if $y^{(2)} \neq y^{(1)}$ **then**
  Set $E \leftarrow E \cup \{y^{(2)}\}$ and $P \leftarrow \{(1,2)\}$.
  Set $k \leftarrow k + 1$.
  while $P \neq \emptyset$ **do**
    Choose an element $(i,j) \in P$.
    Set $\lambda_1 \leftarrow |y_2^{(j)} - y_2^{(i)}|$, $\lambda_2 \leftarrow |y_1^{(j)} - y_1^{(i)}|$ and $\lambda \leftarrow (\lambda_1, \lambda_2)$.
    Solve RLPM(λ) and let x^* be the optimal solution.
    Let $y^* = ((c^1)^T x^*, (c^2)^T x^*)$.
    if $y^* \neq y^{(i)}$ and $y^* \neq y^{(j)}$ **then**
      Set $E \leftarrow E \cup \{y^*\}$ and $P \leftarrow P \cup \{(i,k),(k,j)\}$.
      Set $k \leftarrow k + 1$.
    end if
    Set $P \leftarrow P \backslash \{(i,j)\}$
  end while
end if

In sequential search 1, the subproblem corresponding to each of the points generated in step I is solved in order to generate a set of columns that is representative of the set of points. A problem with this variant of the sequential search approach is that most of the columns that are found by solving the subproblem corresponding to a given point may have already been found for a previous point. This is because the vector of dual values associated to each of the points are computed in step I. Thus, the dual values corresponding to a given point does not take the columns found for another point in step II into account. Due to this, many subproblems may be solved without necessarily finding any new columns.

Algorithm 3.5 Generate a set of Points (ε-constraint method)

Input: A set $\boldsymbol{L}$ containing currently converged points of RLPM together with the set of values of ε for which each point was computed.

Output: A set of points F together with a corresponding set Π of dual vectors and a set $\mathcal{E}$ of the values of ε for which the points were computed.

Set $\varepsilon_i \leftarrow \max \varepsilon$, $F \leftarrow \emptyset$, $i \leftarrow 1$, $\Pi \leftarrow \emptyset$ and $\mathcal{E} \leftarrow \emptyset$.
while $\varepsilon_i \geq \min \varepsilon$ **do**
  if ε_i does not correspond to a point of $\boldsymbol{L}$ **then**
    Solve the RLPM(ε_i) once without generating any columns.
    Let x^* and π^i be the optimal solution and dual vectors, respectively.
    Compute the objective values $f_1^i = (c^1)^T x^*$ and $f_2^i = (c^2)^T x^*$.
    Set $F \leftarrow F \cup (f_1^i, f_2^i)$, $\Pi \leftarrow \Pi \cup \pi^i$ and $\mathcal{E} \leftarrow \mathcal{E} \cup \varepsilon_i$.
    Choose an appropriate step size $\delta_i > 0$.
    Set $i \leftarrow i + 1$ and $\varepsilon_i \leftarrow f_2^{i-1} - \delta_{i-1}$.
  end if
end while

Algorithm 3.6 Sequential Search 1

Output: A lower bound set $\boldsymbol{L}$.

Set $\boldsymbol{L} \leftarrow \emptyset$.
repeat
  Set $\Theta \leftarrow \emptyset$.
  Generate a set of points by Algorithm 3.5.
  Let F, Π and $\mathcal{E}$ be the sets returned by the preceding step.
  for each element π^i of Π **do**
    Solve the subproblem corresponding to π^i.
    Let Λ^i be the set of columns found and set $\Theta \leftarrow \Theta \cup \Lambda^i$.
    if $\Lambda^i = \emptyset$ **then**
      Set $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup (f_1^i, f_2^i)$ and $\mathcal{E} \leftarrow \mathcal{E} \cup \varepsilon_i$.
    end if
  end for
  Add all columns of Θ to the RLPM.
until $\Theta = \emptyset$.

Algorithm 3.7 Sequential Search 2

Output: A lower bound set $\boldsymbol{L}$.

Set $\boldsymbol{L} \leftarrow \emptyset$.
repeat
  Set $\Theta \leftarrow \emptyset$.
  Generate a set of points by Algorithm 3.5.
  Let F, Π and $\mathcal{E}$ be the sets returned by the preceding step.
  for each element π^i of Π **do**
    if no column of Θ is of negative reduced cost for π^i **then**
      Solve the subproblem corresponding to π^i.
      Let Λ^i be the set of columns found and set $\Theta \leftarrow \Theta \cup \Lambda^i$.
      if $\Lambda^i = \emptyset$ **then**
        Set $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup (f_1^i, f_2^i)$ and $\mathcal{E} \leftarrow \mathcal{E} \cup \varepsilon_i$.
      end if
    end if
  end for
  Add all columns of Θ to the RLPM.
until $\Theta = \emptyset$.

Sequential search 2 is designed to avoid the problem encountered in sequential search 1. Before solving the subproblem corresponding to a given point, we first need to verify if any of the columns already found for the other points is relevant (has a negative reduced cost) for this new point. If this is the case the new point is skipped without solving the subproblem corresponding to it since it is already represented by the set of generated columns. Thus, the subproblem corresponding to a new point is only solved if there are no previously found columns in the same iteration that are relevant for this point. By using this strategy, a fewer number of subproblems are solved in comparison to sequential search 1. For this reason, the overall running time of the sequential search 2 is likely to be better than that of the sequential search 1.

There is no particular order in which the points of a lower bound set are identified by a sequential search method. Note, however, that a point generated in step I of an iteration can be proven to have converged in step II only if the subproblem corresponding to it is solved directly by an exact algorithm and no relevant columns are found. A fictitious example to demonstrated a possible order in which the points of a lower bound set are identified by a sequential search approach is given in the series of diagrams in Figure 3.5. An explanation of each of the diagrams is given below.

Figure 3.5a. There are no converged points at the start of the approach. Step 1 produces two points $(13.9, 8.0)$ and $(21.5, 2.0)$ corresponding to RLPM(20) and RLPM(7), respectively. None of these points is proven to have converged in step II.

Figure 3.5b. There are no converged points at the start of this iteration. In step I, three points are generated. These are $(10.8, 13.0)$ corresponding to RLPM(20), $(13.9, 6.0)$ corresponding to RLPM(12) and $(20.0, 2.0)$ corresponding to RLPM(5). In step II, none of

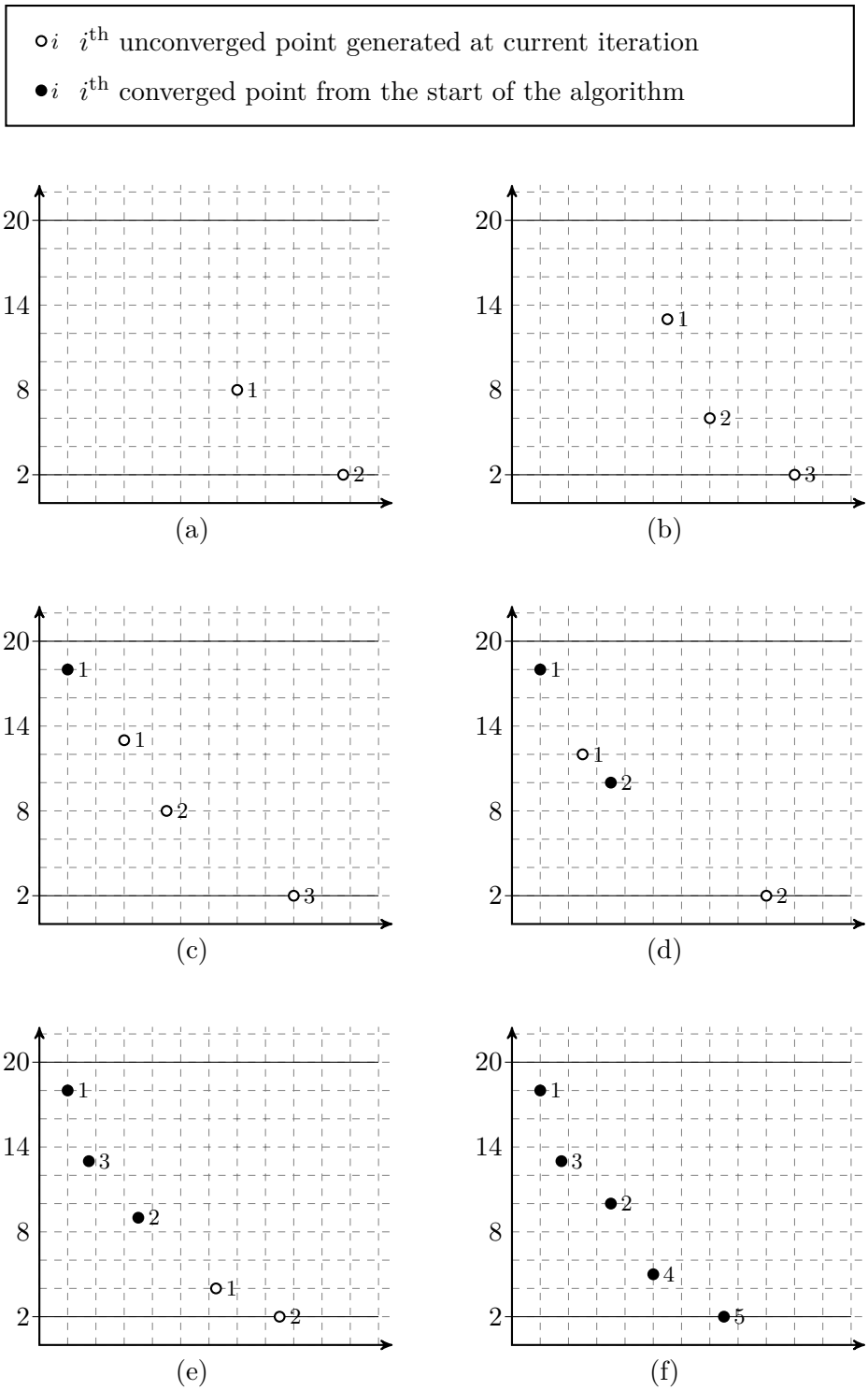


Figure 3.5: Possible order in which points of a lower bound set are identified by a sequential search approach.

these points is proven to have converged.

Figure 3.5c. There are still no converged points at the start of this iteration. Four points are generated during step I. In step II, the first point $(2.0, 18.0)$ corresponding to $\varepsilon = 20$ is proven to have converged. The remaining three points $(5.9, 13.0)$, $(8.8, 8.0)$, and $(18.0, 2.0)$ corresponding to RLPM(17), RLPM(12), and RLPM(7), respectively, have not yet converged.

Figure 3.5d. The point that converged in the preceding iteration is known at the begining of this iteration. There is no need to solve RLPM for values of ε between 18 and 20 when generating points in step I. Three points are generated in step I. The point $(4.9, 12.0)$ corresponding to RLPM(17) is not proven to have converged in step II. The point $(6.9, 10.0)$ corresponding to RLPM(11) converges. The point $(18.0, 2.0)$ corresponding to RLPM(9) does not converge.

Figure 3.5e. Some convereged points from the precedings iterations are known at the begining of this iteration. There is no need to solve RLPM for values of ε between 18 and 20 as well as for those betwwen 10 and 11. Three points are generated in step I. In step II, the point $(3.7, 13.0)$ corresponding to RLPM(17) converges. The other two points $(12.3, 4.0)$ and $(16.8, 2.0)$ corresponding to RLPM(12) and RLPM(3), respectively, do not converge.

Figure 3.5f. All previously converged points are known at the begining of this iteration. There is no need to solve RLPM for values of ε between 18 and 20, between 13 and 17, and 10 and 11. In step I, three points $(6.9, 10.0)$, $(10.0, 5.0)$, and $(14.9, 2.0)$ corresponding to RLPM(12), RLPM(9) and RLPM(4), respectively, are generated. The convergence of all these three points are proven in step II. Note that the point $(6.9, 10.0)$ corresponding to RLPM(12) had previously been found to have converged for RLPM(11). Nevertheless, we still needed to solve the subproblem corresponding to RLPM(12) since we could not justify that a previously converged point corresponds to RLPM(12). The algorithm ends since RLPM has converged for all integer values of ε between 2 and 20.

An advantage of a sequential search approach is that it ensures some uniformity in the convergence of the members of $\boldsymbol{L}$. This can be a very useful technique in the design of heuristics and metaheuristics that are based on column generation since we are sure to find a set points that is representative of the whole range of the Pareto frontier. Another advantage of a sequential search approach is that it encourages similar columns to appear in the solutions of the points making up a lower bound set. This is necessary in some applications (see for example Salari and Unkelbach (2013)).

3.3.2 Column Generation for BOIPs with a Min-Max Objective

We now consider how to use column generation to compute a lower bound set for a BOIP having one min-max objective function. We will refer to such a problem as a bi-objective

integer program with a min-max objective (BOIPMMO). More formally, we consider problems of the form :

$$\text{Minimize} \quad \sum_{k\in\Omega} c_k\theta_k \tag{3.27}$$

$$\text{Minimize} \quad \max_{k\in\Omega}\{\sigma_k\theta_k\} \tag{3.28}$$

$$\text{subject to :} \quad \sum_{k\in\Omega} a_{ik}\theta_k \geq b_i \qquad (i\in I)\,, \tag{3.29}$$

$$\theta_k \in \{0,1\} \qquad (k\in\Omega)\,, \tag{3.30}$$

where Ω is the set of all feasible columns and I is an index set for the constraints. For each column $k\in\Omega$, c_k and σ_k are two associated values. We need to select columns with minimum sum of c_k while the largest σ_k of a column in the selected set of columns is also minimized. Bi-objective generalizations of several VRPs satisfying this condition can be defined. In general, we want to minimize the "cost" of a set of routes such that the value of a property associated with the selected routes (eg. the maximum length of a route, the maximum capacity of a route, etc.) is minimized.

The first step in treating this problem is to linearize the min-max objective function given in 3.28. To do this, we introduce another variable $\Gamma_{\max}$, replace the min-max objective function with the minimization of this variable and then add the necessary constraints in order to ensure that the original min-max obective is not violated. We obtain the following linearized formulation :

$$\text{Minimize} \quad \sum_{k\in\Omega} c_k\theta_k \tag{3.31}$$

$$\text{Minimize} \quad \Gamma_{\max} \tag{3.32}$$

$$\text{subject to :} \quad \sum_{k\in\Omega} a_{ik}\theta_k \geq b_i \qquad (i\in I)\,, \tag{3.33}$$

$$\Gamma_{\max} \geq \sigma_k\theta_k \qquad (k\in\Omega)\,, \tag{3.34}$$

$$\theta_k \in \{0,1\} \qquad (k\in\Omega)\,. \tag{3.35}$$

Due to the introduction of Constraints 3.34, applying a weighted sum method to the above formulation may result in a problem that is more difficult to solve than expected. This is because, adding constraints on an objective function normally destroys the problem structure and so we lose desirable property of being able to use known methods of the original single objective problem also for the scalarized bi-objective problem. A possible solution is to relax Constraints 3.34 in a Lagrangean way while applying the weighted sum method. Nevertheless, solving the resulting Lagrangean dual problem after this relaxation may not be a straight forward task. In a situation like this, an ε-constraint method seems a better approach since it is possible to find unsupported solutions. A disadvantage of directly adding constraints of the form $\Gamma_{\max}\leq\varepsilon$ to a BOIPMMO is that solving linear relaxations of the resulting problem can significantly weaken the lower bound set. In order not to unecessarily weeken the lower bound set, we use a different variant of the ε-constraint method.

A close examination of Formulation (3.31 – 3.35) reveals that a BOIPMMO decomposes naturally into two problems. For any set of feasible columns, the associated value of $\Gamma_{\max}$

can be computed. We can therefore use a variant of the "standard" ε-constraint method with one main difference. Instead of explicitly adding a constraint of the form $\Gamma_{\max} \leq \varepsilon$ to the formulation, we rather drop Constraints 3.34 and use it to redefine the feasibility of a column. Thus, we define a new set of feasible columns $\overline{\Omega}$ where the feasibility of a column $k \in \overline{\Omega}$ now depends on its associated value σ_k. Depending whether or not a column $k \in \Omega$ may be associated with more than one value of σ_k, we may have a larger set of feasible columns after the redefinition. Note that, after droping this set of constraints, it is no longer necessary to keep the objective function 3.32 in the reformulation so we drop it too. The strength of the model is conserved at the expense of having a possibly more difficult subproblem to solve. The IPM becomes the following single-objective program:

$$\text{Minimize} \quad \sum_{k \in \overline{\Omega}} c_k \theta_k \tag{3.36}$$

$$\text{subject to :} \quad \sum_{k \in \overline{\Omega}} a_{ik} \theta_k \geq b_i \qquad (i \in I)\ , \tag{3.37}$$

$$\theta_k \in \{0,1\} \qquad (k \in \overline{\Omega})\ . \tag{3.38}$$

Before solving IPM for a given limit ε on the value of $\Gamma_{\max}$, we need to ensure that no column having $\sigma_k > \varepsilon$ is allowed in the solution.

We can still use all the ideas already discussed in using the ε-constraint when computing a lower bound set for this new model. We should however ensure that only columns that satisfy the limits of a current value of ε are allowed in the solution. As explained above, this is easily achieved in the RLPM by setting the domain of values for all unallowed columns to 0. In addition, we should ensure that no unallowed columns are generated by the subproblem. When the subproblem is a variant of the shortest path problem with resource constraints as it is often the case for VRPs, this can usually be achieved by introducing another resource constraint and modifying the dominance rule between labels accordingly.

3.3.3 Column Search Strategies for a BOIPMMO

As already stated, we can use all the ideas already presented to compute a lower bound set for a BOIPMMO. We present two other column search strategies in order to take advantage of the reformulation discussed for a BOIPPMO. The first strategy is a variant of the PPS whereas the second strategy is a variant of the sequential search approach. We describe these two strategies below.

Improved Point-by-Point Search (IPPS)

Using heuristics to generate columns can improve the performance of column generation (Desaulniers et al., 2002). These heuristics are used to cheaply generate other relevant columns from those found by a subproblem algorithm. In the bi-objective case, we are interested in heuristics that can take advantage of similarities in the different subproblems solved when computing each point of a lower bound. That is, once the cost of finding a first column has been paid we wish to quickly generate other relevant columns that are relevant for the current subproblem and may also be relevant for other subproblems. A

column which has negative reduced cost for a current subproblem, does not necessary have a negative reduced cost for another subproblem since the associated dual variables do not necessarily have the same values. Nevertheless, it can be expected that two subproblems that are close in terms of objectives, may also be close in terms of the solution of RLPM and therefore close in terms of dual variable values. For this reason, a column generated by a heuristic may also be of negative reduced cost for several other subproblems apart from the current one. In addition, standard algorithms used to solve a subproblem are most times only interested in finding a set of best columns that do not dominate one another. This means that many columns having negative reduced costs are left out because they are dominated by some other columns. This may be desirable in the single objective case. In the bi-objective case, however, a column which is dominated with respect to one subproblem may not be dominated with respect to another subproblem.

The main idea of the IPPS for a BOIPMMO is to use heuristics that can take advantage of the redefinition of a column and efficiently search for more columns by modifying the ones found by a subproblem algorithm. We acknowledge that such a heuristic can be used not just for a BOIPMMO but for any general BOIP. We, however, study IPPS for BOIPMMOs since it is usually easy to come up with a good heuristic that can take advantage of the reformulation. IPPS can be useful as a column generation based heuristic since at each iteration it tries to generate a set of columns that are relevant for several subproblems. For completeness, we summarize the IPPS is in Algorithm 3.8. The heuristic used in Step 9 obviously depends on the problem at hand and we will demonstrate it for a practical problem in the next chapter. The relevance of a column found by a heuristic is evaluated with respect to the same vector of dual values for which the original column was found. This is a distinctive feature of IPPS in contrast to the other strategy which we describe next.

Algorithm 3.8 Improved Point-by-Point Search (IPPS)

Output: A lower bound set $\boldsymbol{L}$.

1: Set $\varepsilon \leftarrow \infty$, and $\boldsymbol{L} \leftarrow \emptyset$.
2: **while** RLPM(ε) is feasible **do**
3: Solve RLPM(ε) once to obtain a vector of dual values.
4: Let c^* be the optimum and θ^* be the optimal vector.
5: Compute $\sigma^* = \max_{k \in \overline{\Omega}} \sigma_k \theta_k^*$.
6: Solve the subproblem $S(\varepsilon)$ and let Λ be the set of columns obtained.
7: **if** $\Lambda \neq \emptyset$ **then**
8: **for** each column in Λ **do**
9: Use heuristics to generate other relevant columns from it.
10: **end for**
11: **else**
12: $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup \{(c^*, \sigma^*)\}$.
13: Set $\theta_k = 0$ for all k such that $\sigma_k \geq \sigma^*$, and set $\theta_k \geq 0$ for all other columns.
14: **end if**
15: **end while**

Solve-Once-Generate-for-All (SOGA)

The Solve-Once-Generate-for-All (SOGA) approach is a variant of the sequential search that is adapted to take advantage of the reformulation of a BOIPMMO. A summary of the approach is presented in Algorithm 3.9. Just as for all the other variants of the sequential search, each iteration starts by generating a set of points based on the current columns in the RLPM without generating any columns. After generating a set of points, SOGA continues by solving the subproblem corresponding to a single point. If no relevant columns are returned by the subproblem, this point is proven to have converged and so it is saved in the lower bound set. Otherwise, from Step 8 to Step 13, each column found is modified several times by using dual information from the other points in order to generate more columns. Unlike in the case of IPPS, the relevance of a column found is evaluated with respect to another vector of dual values rather than the one for which the original unmodified column was found. This guarantees that at each iteration, a set of columns that are relevant for a very large number of points are returned to the RLPM. For this reason, SOGA as well as the other variants of the sequential search approach can be very useful in designing column generation based heuristics and metaheuristics. An advantage that SOGA has over the other variants is that, it solves only one subproblem but generates a set of columns that is guaranteed to be relevant for several subproblems. The main challenge of this approach is that it may not always be easy to combine information from an original column and a vector of dual values in heuristics to search for new columns. A SOGA heuristic will be presented for the application problem in the next chapter.

Algorithm 3.9 Solve-Once-Generate-for-All (SOGA)

Output: A lower bound set $\boldsymbol{L}$.

1: Set $\boldsymbol{L} \leftarrow \emptyset$.
2: **repeat**
3: Generate a set of points by following an idea similar to Algorithm 3.5.
4: Let $F = \{(c^i, \sigma^i)\}$ be the set of generated points, $\Pi = \{\pi^i\}$ be the corresponding set of dual vectors, and $\mathcal{E} = \{\varepsilon_i\}$ be the corresponding set of ε values.
5: Fix an index i and solve the subproblem $\mathrm{S}(\varepsilon_i)$ corresponding to π^i.
6: Let Λ^i be the set of columns found.
7: **if** $\Lambda^i \neq \emptyset$ **then**
8: **for** each column in Λ **do**
9: **for** each vector of dual values $\pi^j \in \Pi$ such that $i \neq j$ **do**
10: Use heuristics that combine information from the column in Λ and the vector π^j to generate some relevant columns corresponding to $\mathrm{S}(\varepsilon_j)$.
11: **end for**
12: Add all found columns to the RLPM.
13: **end for**
14: **else**
15: $\boldsymbol{L} \leftarrow \boldsymbol{L} \cup \{(c^i, \sigma^i)\}$.
16: Set $\theta_k = 0$ for all k such that $\sigma_k \geq \sigma^i$, and set $\theta_k \geq 0$ for all other columns.
17: **end if**
18: **until** $F = \emptyset$.

3.4 Evaluating the Quality of Bound Sets

In Chapter 2 we saw indicators that are used to evaluate the quality of approximations to the nondominated set $\mathcal{Y}_N$. These measures are mainly used to compare different upper bound sets. They may also be used in comparing different lower bound sets obtained by different methods or approaches. When it comes to comparing the quality of a lower bound set and a corresponding upper bound set, however, these measures are not very appropriate. This is because, they will only indicate that the upper bound set lies "above" the lower bound set which is a trivial thing. Ehrgott and Gandibleux (2007) discuss five measures which may be used in evaluating the quality of a lower bound set together with an upper bound set. In this thesis we will be using two of those measures in evaluating the quality of the bounds we compute. The first measure (μ_1) is distance based whereas the second measure (μ_2) is area based. Combining a distance based measure with an area based measure gives a good indication of the quality of the bounds. Roughly speaking, μ_1 represents the worst distance (with respect to the range of objective values) between a point of the upper bound set and a point of the lower bound set closest to it. Also, μ_2 represents the fraction of the area that is dominated by the lower bound set but not by the upper bound set. This is, the area where additional points of $\mathcal{Y}_N$ can be found.

Calculation and Interpretation of the measures

Given a lower bound set $\boldsymbol{L}$ and a corresponding upper bound set $\boldsymbol{U}$, we let $y_i^{\max}$ for $i \in \{1, 2\}$ be the maximum value of the i^{th} objective when we consider the union of $\boldsymbol{L}$ and $\boldsymbol{U}$. In the same way, we let $y_i^{\min}$ for $i \in \{1, 2\}$ be the minimum value of the i^{th} objective when we consider the union of $\boldsymbol{L}$ and $\boldsymbol{U}$. We define the points $y^{\max} := (y_1^{\max}, y_2^{\max})$ and $y^{\min} := (y_1^{\min}, y_2^{\min})$. The distance between $\boldsymbol{L}$ and $\boldsymbol{U}$ is defined as the maximum of the minimum distances by which we need to displace a point of $\boldsymbol{U}$ so that it is not strongly dominated by any point of $\boldsymbol{L}$. We denote this distance by $d(\boldsymbol{L}, \boldsymbol{U})$. The first measure is given by

$$\mu_1 := \frac{d(\boldsymbol{L}, \boldsymbol{U})}{\|y^{\max} - y^{\min}\|_2} \, .$$

Next, let $A_{\boldsymbol{L}}$ and $A_{\boldsymbol{U}}$ be the areas of the regions in the rectangle with $y^{\min}$ and $y^{\max}$ at opposite corners that are dominated by $\boldsymbol{L}$ and $\boldsymbol{U}$, respectively. The, second measure is given by

$$\mu_2 := \frac{A_{\boldsymbol{L}} - A_{\boldsymbol{U}}}{A_{\boldsymbol{L}}} \, .$$

The different components that are necessary for calculating μ_1 and μ_2 in the case where we use a weighted sum method are indicated in Figure 3.6. Figure 3.7 also indicates these components in the case where we use an ε-constraint method.

The two quality measures presented above complement each other and as explained by Ehrgott and Gandibleux (2007), they can be seen to play a role similar to the optimality gap in single objective optimization. If a lower bound set and a corresponding upper bound set are good, then we expect that both μ_1 and μ_2 will be small in value. The smaller both values are, the better the quality of the bounds.

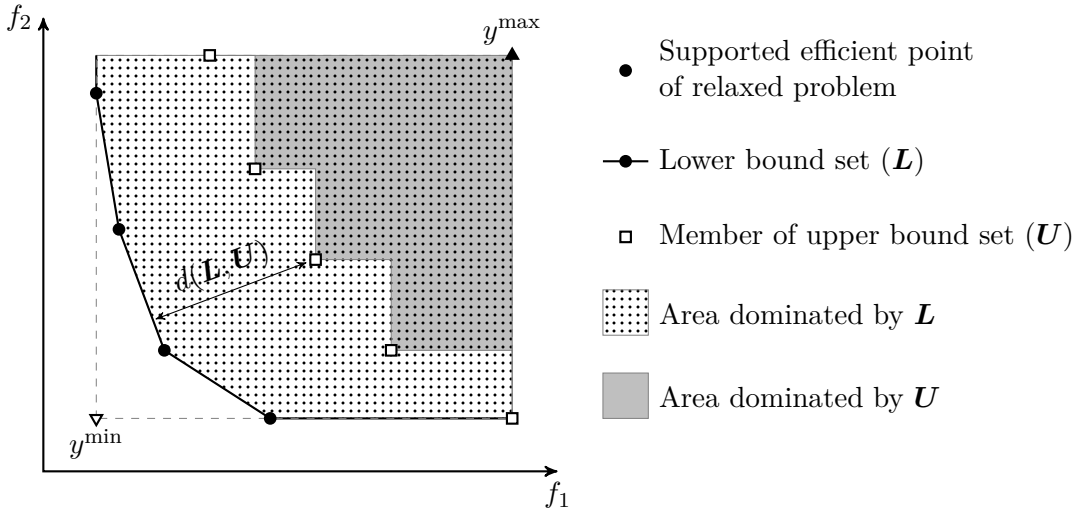


Figure 3.6: Calculation of quality measures in the case of a weighted sum method.

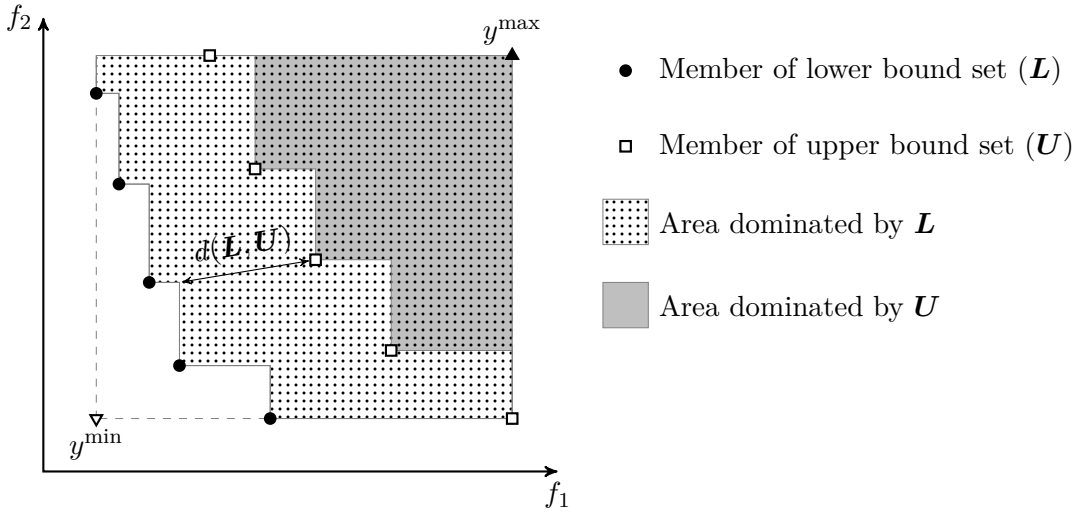


Figure 3.7: Calculation of quality measures in the case of an ε-constraint method.

3.4.1 Bound Sets for the Bi-Objective Set Covering Problem

We compare the quality of lower bound sets computed by the weighted sum and ε-constraint methods for the bi-objective set covering problem (BOSCP). Since the BOSCP serves as a base formulation for most vehicle routing problems solved by column generation, the aim of this section is to give a general idea of the quality of lower bounds computed by the two scalarization methods for this kind of problem and its extensions. The BOSCP is formulated as :

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j^1 x_j \tag{3.39}$$

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j^2 x_j \tag{3.40}$$

$$\text{subject to :} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1, \ldots, m \,, \tag{3.41}$$

$$x_j \in \{0,1\} \qquad j = 1, \ldots, m \,. \tag{3.42}$$

The integral objective coefficients are c_j^1 and c_j^2, and the matrix coefficients are $a_{ij} \in \{0,1\}$. We say that constraint i is *covered* by variable x_j if $a_{ij} = 1$.

Experiments and Summary of Results

The BOSCP instances used for the tests can be found at `http://xgandibleux.free.fr/MOCOlib/MOSCP.html`. They are the same instances that were used by Ehrgott and Gandibleux (2007). Since the goal of the tests is to evaluate the quality of lower bound sets, we used the exact nondominated sets computed by the algorithm proposed by Bérubé et al. (2009) as upper bound sets. For this reason, the values of the two quality measures obtained for the weighted sum method are different from those obtained by Ehrgott and Gandibleux (2007) who used greedy heuristics to compute upper bound sets. All computer codes were written in C/C++ and the LP/MIP optimizers of ILOG CPLEX 12.4 were used. The tests were run on an Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz computer with a 2 GiB RAM and a summary of the results are presented in Table 3.1. In this table, $|\boldsymbol{U}^*|$ represents the number of points in the exact nondominated set, *time* is the computational time in cpu seconds, $|ext|$ is the number of extreme points (or supported efficient points) computed when the weighted sum method is used, and $|\boldsymbol{L}|$ is the total number of points in a lower bound set when an ε-constraint method is used. Values of the two quality measures are expressed as percentages under the headings $\mu_1\%$ and $\mu_2\%$.

Discussion

The results show that, the lower bound sets obtained by both methods for almost all the intances are of good quality. The lower bounds obtained in the case of the ε-constraint method are slightly better than those obtained for the weighted sum method. The situation is quite different when we compare the computational times and the computational complexities of the two methods. In the case of the weighted sum method, relatively

Table 3.1: Comparison of lower bound sets for the BOSCP

		Weighted Sum				ε-Constraint			
Instance	$\lvert \boldsymbol{U}^* \rvert$	time	$\lvert ext \rvert$	$\mu_1\%$	$\mu_2\%$	time	$\lvert \boldsymbol{L} \rvert$	$\mu_1\%$	$\mu_2\%$
2scp11A	39	0.00	13	2.9	2.5	0.02	192	2.7	2.1
2scp11C	10	0.00	21	13.7	34.1	0.01	117	13.9	32.7
2scp41A	105	0.01	23	1.5	1.3	0.14	1028	1.4	1.2
2scp41C	24	0.02	17	2.6	4.5	0.04	280	2.7	5.7
2scp42A	206	0.02	37	0.6	0.5	0.45	1663	0.5	0.4
2scp42C	87	0.04	51	3.0	3.4	0.44	2245	2.9	3.4
2scp43A	46	0.04	63	3.6	4.3	0.12	473	3.5	4.1
2scp43C	12	0.02	38	6.3	11.1	0.04	225	6.3	10.6
2scp61A	254	0.06	58	1.1	0.8	1.39	2829	1.1	0.8
2scp61C	28	0.02	17	1.0	3.7	1.02	732	0.9	4.4
2scp62A	98	0.28	236	3.5	3.6	1.01	1240	3.5	3.7
2scp62C	6	0.23	180	38.9	46.6	0.00	2	36.6	58.5
2scp81A	424	0.07	47	0.5	0.4	3.49	5580	0.4	0.3
2scp81C	14	0.12	77	0.3	0.5	0.60	147	0.4	0.7

few extreme points need to be calculated and this is done in negligible time for the instances tested. The number of points that needs to be calculated when an ε-constraint method is used can be very large (possibly exponential) and this may require a very long computational time. For example, the ε-constraint method computed 5580 points in a time of 3.49 cpu seconds for instance 2scp81A. This is very large when we compare it with the 47 points computed in 0.07 seconds when a weighted sum method was used. This stresses the need to reduce the number of points calculated when using an ε-constraint method by determining good step sizes.

3.5 Conclusions

The discussion in this chapter has been concerned with the use of column generation in computation lower bound sets for BOIPs. The main idea used is to first convert the BOIP into a single objective problem by using either a weighted sum method or an ε-constraint method. The linear relaxation of the resulting single objective problem is solved by column generation for different values of the necessary parameters. Regardless of which of the two scalarization method is used, the subproblems required to be solved when computing a lower bound set have a similar form. Due to this, we investigate different strategies to effectively search for columns. These strategies will be evaluated and compared in the next chapter where a practical problem is considered. Although slightly better lower bound sets were obtained by the ε-constraint method on instances of the bi-objective binary set covering problem, more computational effort is required than when a weighted sum method is used. There is, hence, the need to improve the ε-constraint method when computing lower bound sets in order to reduce its computational complexity.

Chapter 4

The Bi-Objective Multi-Vehicle Covering Tour Problem

4.1 Introduction

In this chapter, we present an application problem to demonstrate the different ideas and approaches presented in Chapter 3. The problem considered is an extension of the covering tour problem (Gendreau et al., 1997) namely the bi-objective multi-vehicle covering tour problem (BOMCTP). The covering tour problem (CTP) consists in designing a single route over a subset of locations with the aim of minimizing the length of the route. In addition, each location not visited by the route should lie within a fixed radius from a visited location. The fixed radius is called the *cover distance*. An example of a solution to the CTP is indicated in Figure 4.1.

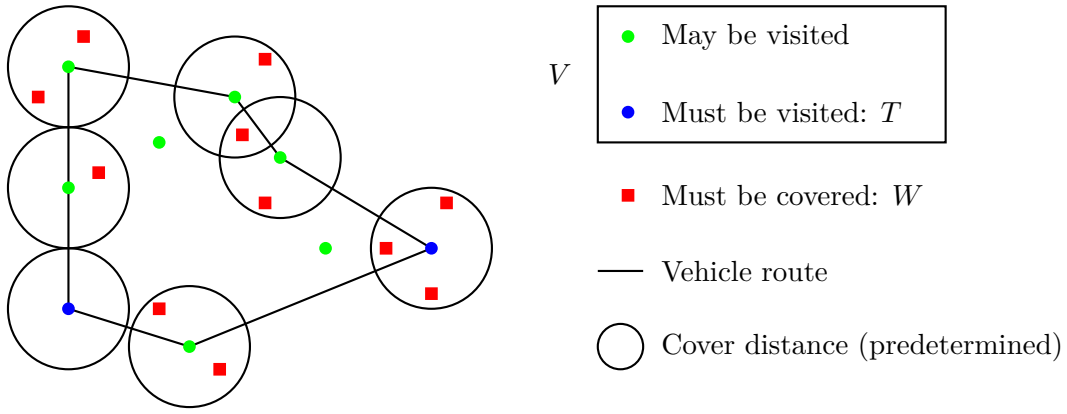


Figure 4.1: An example of a solution to the CTP (Gendreau et al., 1997).

The CTP has a generic application in the design of bi-level transportation networks (Current and Schilling, 1994). This kind of problems seeks to construct a primary route of minimum length on a subset of locations in such a way that all other locations that are not on the primary route can easily reach it. An example of the CTP arises in the problem of choosing where to locate post boxes among a set of candiate sites (Labbé and Laporte,

1986). The aim of this problem is to minimize the cost of a collection route through all post boxes and also ensure that every user is located within a reasonable distance from a post box. Several other examples of the CTP arise in the domain of humanitarian logistics. For example, in the planning of routes for visiting health care teams in developing countries where medical services can only be delivered to a subset of villages, but all users must be able to reach a visiting medical team (Current and Schilling, 1994; Hodgson et al., 1998).

A bi-objective generalization of the CTP has been proposed by Jozefowiez et al. (2007). The cover distance in the bi-objective CTP (BOCTP) is not fixed in advance but rather induced by the constructed route. It is computed by assigning each non-visited location to the closest visited location and calculating the maximum of the assigned distances. The objectives are to minimize the length of the route as well as the *induced cover distance*. The authors proposed a two-phase cooperative strategy to solve the problem. This strategy combines a multi-objective evolutionary algorithm with a branch-and-cut algorithm initially designed by Gendreau et al. (1997) to solve the CTP. Hachicha et al. (2000) present a multiple vehicle extension of the CTP namely the multi-vehicle covering tour problem (MCTP). In the MCTP, the combined length of a set of routes, all of which must start from a common location, is minimized for a fixed cover distance. In addition, the number of locations visited by a single route and the length of the route cannot exceed predetermined constants p and q, respectively. The authors proposed an integer linear formulation as well as three heuristic methods for the problem. An example of a solution to the MCTP is given in Figure 4.2. Quite recently, two exact methods have been proposed for the MCTP. The first is a branch-and-price algorithm proposed by Jozefowiez (2012) whereas the second is a branch-and-cut algorithm proposed by Hà et al. (2013). A metaheuristic for the MCTP was also proposed by Hà et al. (2013).

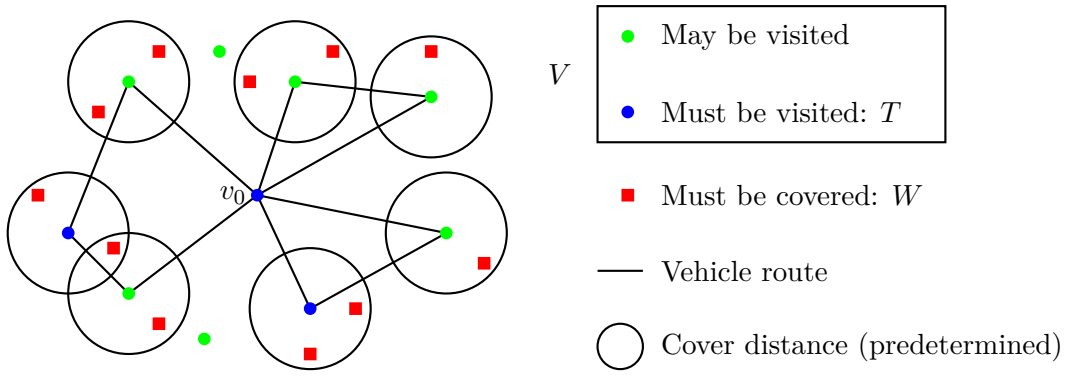


Figure 4.2: An example of a solution to the MCTP (Hachicha et al., 2000).

The BOMCTP discussed in this chapter can be seen as a combination of the BOCTP and the MCTP and it will be formerly presented later on. The rest of the chapter is organized as follows. In Section 4.2 we give a formal description of the BOMCTP and also introduce different notations that will be used throughout the chapter. A first set covering based formulation for the BOMCTP is given in Section 4.3 and a second one is given in Section 4.4. Each of these two sections will discuss the different models involved in a column generation methodology. Experiments and computational results based on each of the two formulations are given in Section 4.5. The chapter ends with concluding

remarks in Section 4.6.

Related Publications. Some parts of this chapter have been published in international conference proceedings and some other parts have been submitted to international journals. The paper presented at the ODYSSEUS 2012 international conference (Sarpong et al., 2012a) is based on the first formulation for the BOMCTP. Some results based on the second formulation formed part of the paper which was presented at the ATMOS 2013 conference (Sarpong et al., 2013a). An extended version of the paper based on the first formulation together with some elements of Chapter 3 (Sarpong et al., 2013c) has been accepted for publication in the journal RAIRO - Operations Research whereas an extended version of the one based on the second formulation is yet to be submitted to another international journal. The two technical reports (Sarpong et al., 2012b, 2013b) present some of the ideas in the submitted journal articles.

4.2 Description of the BOMCTP

The BOMCTP is defined on a graph $G = (V \cup W, E)$ where $V \cup W$ is a set of nodes and E is a set of edges. The nodes of V represent locations which may be visited by a route whereas the members of W are to be assigned to visited nodes of V. There is a subset of nodes $T \subseteq V$, which must be visited by at least one route. In real applications, the members of T represent important locations where we require at least one route to pass. In particular, $v_0 \in T$ is the depot where all routes must start and also end. Set E is made up of edges connecting all pairs of nodes in $V \cup W$ and a distance matrix $D = (d_{ij})$ satisfying the triangle inequality is defined on this set. The BOMCTP consists in designing a set of routes over a subset of V which should include all nodes of T. In addition, each route should visit not more than p nodes of $V\backslash\{v_0\}$ and its length must not exceed q. Both p and q are predetermined constants. The two objectives are to minimize the total length of the set of routes and the cover distance induced by the set.

4.2.1 Cover Distance Induced by a Set of Routes

The cover distance induced by a set of routes or induced cover distance (denoted by $\Gamma_{\max}$) is defined as the maximum distance from a node of W to the closest visited node of $V\backslash\{v_0\}$ (see Figure 4.3). By definition, the value of d_{ij} for every couple $(v_i, w_j) \in V\backslash\{v_0\} \times W$ is a candidate value for $\Gamma_{\max}$. Nevertheless, some of these candidate values do not correspond to feasible $\Gamma_{\max}$ values. For the BOMCTP, we can use an idea introduced by Jozefowiez et al. (2007) to determine the feasible values of $\Gamma_{\max}$ for any given instance. We state this idea in the following proposition.

Proposition 4.1. *Given $v_i \in V\backslash\{v_0\}$ and $w_j \in W$, d_{ij} is a feasible value for $\Gamma_{\max}$ if and only if the following two conditions are satisfied.*

1. *$\forall v_t \in T\backslash\{v_0\}$ such that $v_t \neq v_i$, $d_{ij} \leq d_{tj}$ and*
2. *$\forall w_l \in W$ such that $w_l \neq w_j$, $\exists v_h \in V\backslash\{v_0\}$ such that $d_{hl} \leq d_{ij} \leq d_{hj}$.*

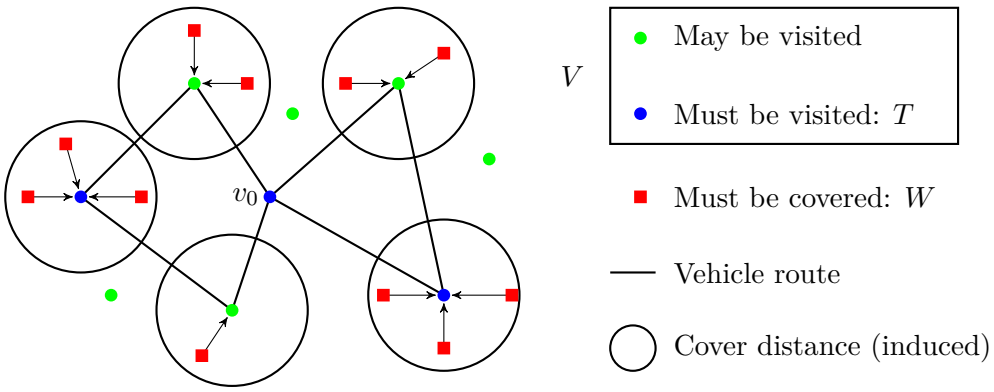


Figure 4.3: The cover distance induced by a set of routes.
Notes: After constructing the routes, each node of W must be assigned to a visited node of $V\backslash\{v_0\}$. The induced cover distance is defined as the maximum of the assigned distances.

Proof. The proof is given in two parts. Part I shows that if d_{ij} is a feasible value for $\Gamma_{\max}$ then conditions (1) and (2) are satisfied. Part II also shows that if conditions (1) and (2) are satisfied then d_{ij} is a feasible value for $\Gamma_{\max}$.

Part I. The necessary conditions for d_{ij} to be the value of $\Gamma_{\max}$ are that v_i is visited by a selected route and w_j is assigned to v_i. Yet, each node of W is assigned to a visited node of $V\backslash\{v_0\}$ that is closest to it. Since all nodes $v_t \in T\backslash\{v_0\}$ are visited in any feasible solution, w_j will be assigned to v_i only if $d_{ij} \leq d_{tj}$. This proves condition (1). Next, suppose that $\Gamma_{\max} = d_{ij}$ but condition (2) is false. That is, for a given $w_l \in W$ such that $w_l \neq w_j$, either $d_{ij} > d_{hj}$ or $d_{ij} < d_{hl}$ for all $v_h \in V\backslash\{v_0\}$. Then, either w_j will be assigned to v_h (since v_h is closer to it than v_i) or if w_j is assigned to v_i then d_{hl} is a better candidate for $\Gamma_{\max}$ (since it is greater than d_{ij} and $\Gamma_{\max}$ is determined by the maximum of the assigned distances). In both cases, d_{ij} cannot be the value for $\Gamma_{\max}$ and so condition (2) must be true if $\Gamma_{\max} = d_{ij}$.

Part II. If conditions (1) and (2) are satisfied then w_j will be assigned to v_i (if it is visited) rather than to a node $v_t \in T\backslash\{v_0\}$. If v_i is visited and w_j is assigned to v_i, then condition (2) implies that d_{ij} can possibly be the maximum value among all assigned distances and so a feasible value for $\Gamma_{\max}$.

□

Clearly, there is a finite number of values for $\Gamma_{\max}$ and by using the two conditions in Proposition 4.1, all the feasible values of $\Gamma_{\max}$ can be computed for any given instance of the BOMCTP.

4.3 Formulation 1

Let Ω represent the set of all feasible routes. A feasible route is defined as a Hamiltonian cycle over a subset of V which includes the depot, visits not more than p nodes and of length not exceeding q. The cost of a route $k \in \Omega$ is given by the sum of the cost of the

edges it uses and we denote it by c_k. Let variable $\theta_k = 1$ if route k is selected in the solution and $\theta_k = 0$, otherwise. Constant $a_{ik} = 1$ if route k visits node $v_i \in V\backslash\{v_0\}$ and 0 if this is not the case. Variable z_{ij} is used to indicate whether node $w_j \in W$ is assigned to node $v_i \in V\backslash\{v_0\}$ in the solution ($z_{ij} = 1$) or not ($z_{ij} = 0$). Let $\Gamma_{\max}$ be the cover distance induced by a given set of routes. Note that $\Gamma_{\max} = \max\{d_{ij}z_{ij} : v_i \in V\backslash\{v_0\} \text{ and } w_j \in W\}$. The BOMCTP can be described with the following set covering model.

$$\text{Minimize} \quad \sum_{k\in\Omega} c_k\theta_k \tag{4.1}$$

$$\text{Minimize} \quad \Gamma_{\max} \tag{4.2}$$

$$\text{subject to :} \quad \Gamma_{\max} - d_{ij}z_{ij} \geq 0 \qquad (v_i \in V\backslash\{v_0\}, w_j \in W)\ , \tag{4.3}$$

$$\sum_{v_i\in V\backslash\{v_0\}} z_{ij} \geq 1 \qquad (w_j \in W)\ , \tag{4.4}$$

$$\sum_{k\in\Omega} a_{ik}\theta_k - z_{ij} \geq 0 \qquad (v_i \in V\backslash\{v_0\}, w_j \in W)\ , \tag{4.5}$$

$$\sum_{k\in\Omega} a_{ik}\theta_k \geq 1 \qquad (v_i \in T\backslash\{v_0\})\ , \tag{4.6}$$

$$z_{ij} \in \{0,1\} \qquad (v_i \in V\backslash\{v_0\}, w_j \in W)\ , \tag{4.7}$$

$$\theta_k \in \mathbb{N} \qquad (k \in \Omega)\ . \tag{4.8}$$

In this formulation, the objectives of minimizing the total length of the set of routes and the induced cover distance are represented in (4.1) and (4.2), respectively. Constraints (4.3) indicate that the induced cover distance should be large enough to respect the distances of all assignments of a node $w_j \in W$ to a node $v_i \in V\backslash\{v_0\}$. Constraints (4.4) and (4.5) specify that each node of W should be assigned to at least one node of $V\backslash\{v_0\}$ which is visited by a route selected in the solution. The requirement that each node of $T\backslash\{v_0\}$ is visited by at least one selected route is represented in Constraints (4.6). Constraints 4.7 and 4.8 define the domains of the decision variables used. We define θ_k to be a non-negative integer instead of binary in order to prevent constraints of the form $\theta_k \leq 1$ in the linear relaxation. The optimal solution of the problem is not affected by this change.

4.3.1 Restricted LP Master Problem

We restrict the value of the cover distance induced by the set of routes to be at most $\varepsilon \in \mathbb{R}$ to obtain the constraint

$$-\Gamma_{\max} \geq -\varepsilon \tag{4.9}$$

For any given value of ε, the linear programming master problem (LPM(ε)) is defined by the linear relaxation of the formulation with objective function (4.1) and Constraints (4.3–4.9). The corresponding restricted LP master problem (RLPM(ε)) is obtained by replacing Ω in LPM(ε) with a subset Ω_1 for which the problem is primal feasible.

4.3.2 Dual of LPM(ε)

Let γ_{ij} ($v_i \in V\backslash\{v_0\}, w_j \in W$), β_j ($w_j \in W$), α_{ij} ($v_i \in V\backslash\{v_0\}, w_j \in W$), π_i ($v_i \in T\backslash\{v_0\}$) and λ be the non-negative dual variables associated with Constraints (4.3), (4.4), (4.5),

(4.6) and (4.9), respectively. The dual formulation DLPM(ε) corresponding to LPM(ε) is given by :

$$\text{Maximize} \quad \sum_{w_j \in W} \beta_j \; + \sum_{v_i \in T \backslash \{v_0\}} \pi_i \; - \; \varepsilon\lambda \tag{4.10}$$

subject to :

$$\sum_{\substack{v_i \in V \backslash \{v_0\} \\ w_j \in W}} a_{ik}\alpha_{ij} \; + \sum_{v_i \in T \backslash \{v_0\}} a_{ik}\pi_i \; \leq \; c_k \qquad (k \in \Omega) \; , \tag{4.11}$$

$$\beta_j - d_{ij}\gamma_{ij} \; - \; \alpha_{ij} \; \leq \; 0 \qquad (v_i \in V \backslash \{v_0\}, w_j \in W) \; , \tag{4.12}$$

$$\sum_{\substack{v_i \in V \backslash \{v_0\} \\ w_j \in W}} \gamma_{ij} - \lambda \; \leq \; 0 \; . \tag{4.13}$$

4.3.3 Sub-problem corresponding to RLPM(ε)

The sub-problem corresponding to RLPM(ε), denoted $S(\varepsilon)$, is to search for feasible routes $k \in \Omega \backslash \Omega_1$ such that

$$c_k \; - \sum_{\substack{v_i \in V \backslash \{v_0\} \\ w_j \in W}} a_{ik}\alpha_{ij} \; - \sum_{v_i \in T \backslash \{v_0\}} a_{ik}\pi_i \; < \; 0 \; . \tag{4.14}$$

We define $\pi_i^* = \pi_i$ if $v_i \in T \backslash \{v_0\}$ and $\alpha_{ij}^* = \alpha_{ij}$ if $v_i \in V \backslash \{v_0\}, w_j \in W$. In all other cases these variables are set to 0. Condition (4.14) can be simplified to

$$\sum_{(v_i, v_j) \in E} \Big(d_{ij} - \pi_i^* - \sum_{w_h \in W} \alpha_{ih}^* \Big) x_{ijk} \; < \; 0 \; . \tag{4.15}$$

where $x_{ijk} = 1$ if route k visits v_j immediately after visiting v_i and $x_{ijk} = 0$ if this is not the case. Feasible routes satisfying condition (4.15) are searched for by solving an elementary shortest path problem with resource constraints (ESPPRC) given by

$$\text{Minimize} \quad \sum_{(v_i, v_j) \in E} \Big(d_{ij} - \pi_i^* - \sum_{w_h \in W} \alpha_{ih}^* \Big) x_{ijk} \quad \text{subject to} \quad k \in \Omega \backslash \Omega_1 \; . \tag{4.16}$$

That is, we seek feasible routes with negative reduced costs where the reduced cost of $(v_i, v_j) \in V \times V$ is given by $d_{ij} - \pi_i^* - \sum_{w_h \in W} \alpha_{ih}^*$.

4.3.4 Solving $S(\varepsilon)$

As already seen in Chapter 1, the ESPPRC is very well studied since it appears as a subproblem in many vehicle routing problems solved by column generation. For this first formulation, there is nothing very special to do when solving $S(\varepsilon)$ with known dynamic programming algorithms. In particular, there is no need for any specialised dominance rule between labels. The subproblem is solved by the decremental state space algorithm (DSSR) (Righini and Salani, 2008; Boland et al., 2006). Two resources are considered in

implementing the DSSR algorithm for the subproblem. These are the number of nodes a route may visit which is limited to a maximum of p and its length which is limited to a maximum of q. We let $\Lambda_i = (\tilde{c}_i, \tilde{q}_i, \tilde{p}_i)$ represent a partial path coming from the depot v_0 to a node v_i. The components of the label are given by the reduced cost up to the current node $\tilde{c}_i$, the length of the partial path $\tilde{q}_i$, and the total number of nodes of $V\backslash\{v_0\}$ visited, $\tilde{p}_i$. In order to simplify the notation, some components of a label are not shown. For example, those that ensure that a constructed path is elementary. The validity of the discussions that follows is not affected by this omission. For any two labels Λ^i and Λ^j on the same node, we say that Λ^i dominates Λ^j if and only if $\Lambda_1^1 \leq \Lambda_1^2$. That is if and only if $\tilde{p}_i \leq \tilde{p}_j$, $\tilde{q}_i \leq \tilde{q}_j$, and $\tilde{c}^i \leq \tilde{c}^j$, and at least one of the inequalities is strict.

4.4 Formulation 2

In this section, we give a new set covering formulation for the BOMCTP based on the reformulation of a BOIPMMO discussed in Chapter 3. Although we use some of the notations introduced for the first formulation, we restate them here for completeness.

Let Ω represent the set of all feasible columns. A feasible column $k \in \Omega$ is defined as a route R_k which is a Hamiltonian cycle on a subset of V, includes the depot, visits not more than p nodes and of length not exceeding q. The length of R_k is denoted c_k. For each route R_k, we have to choose a subset $\Psi_k \subseteq W$ of nodes it may cover and then define σ_k as the maximum distance between a node of Ψ_k and the closest node of $R_k\backslash\{v_0\}$. Just as in the first formulation, we define $a_{ik} = 1$ if column $k \in \Omega$ is selected in the solution and $a_{ik} = 0$ if this is not the case. The constant $b_{jk} = 1$ if $w_j \in \Psi_k$ and $b_{jk} = 0$ otherwise. Let $\Gamma_{\max}$ represent the cover distance induced by a set of routes and θ_k be the binary variable that indicates whether column $k \in \Omega$ is selected ($\theta_k = 1$) or not ($\theta_k = 0$). A possible formulation for the BOMCTP is given by :

$$\text{Minimize} \quad \sum_{k\in\Omega} c_k\theta_k \tag{4.17}$$

$$\text{Minimize} \quad \Gamma_{\max} \tag{4.18}$$

$$\text{subject to :} \quad \sum_{k\in\Omega} a_{ik}\theta_k \geq 1 \qquad (v_i \in T\backslash\{v_0\}) , \tag{4.19}$$

$$\sum_{k\in\Omega} b_{jk}\theta_k \geq 1 \qquad (w_j \in W) , \tag{4.20}$$

$$\Gamma_{\max} - \sigma_k\theta_k \geq 0 \qquad (k \in \Omega) , \tag{4.21}$$

$$\theta_k \in \{0,1\} \qquad (k \in \Omega) . \tag{4.22}$$

The objectives of minimizing the length of the set of routes and the induced cover distance are given in (4.17) and (4.18), respectively. Constraints (4.19) ensure that each node of $T\backslash\{v_0\}$ is visited by at least one selected route. The fact that each node of W should be assigned to a visited node of $V\backslash\{v_0\}$ is indicated by Constraints 4.20. Finally, Constraints (4.21) ensure that the value of the induced cover distance conforms to its definition.

4.4.1 Restricted LP Master Problem

By following the idea of reformulating a BOIPMMO as presented in Chapter 3, we define a new set of feasible columns $\overline{\Omega}$ where the feasibility of a column $k \in \overline{\Omega}$ depends not just on R_k but also on σ_k. The resulting LP master problem (LPM) is given by :

$$\text{Minimize} \quad \sum_{k \in \overline{\Omega}} c_k \theta_k \tag{4.23}$$

$$\text{subject to :} \quad \sum_{k \in \overline{\Omega}} a_{ik} \theta_k \geq 1 \qquad (v_i \in T \backslash \{v_0\}) \,, \tag{4.24}$$

$$\sum_{k \in \overline{\Omega}} b_{jk} \theta_k \geq 1 \qquad (w_j \in W) \,, \tag{4.25}$$

$$\theta_k \geq 0 \qquad (k \in \overline{\Omega}) \,. \tag{4.26}$$

As it was explained in Chapter 3, the second objective to minimize $\Gamma_{\max}$ and the Constraints (4.21) do not appear in the above formulation. For any value of $\varepsilon \in R$ we define $\overline{\Omega}^{\varepsilon}$ to be the subset of columns $k \in \overline{\Omega}$ for which $\sigma_k \leq \varepsilon$. Thus, in order to find a solution to LPM that respects the constraint $\Gamma_{\max} \leq \varepsilon$, we solve LPM over the set of columns $\overline{\Omega}^{\varepsilon}$. We denote this problem by LPM(ε). The restricted LP master problem, RLPM(ε), is given by any restriction of LPM(ε) to a subset $\overline{\Omega}_1^{\varepsilon} \subseteq \overline{\Omega}^{\varepsilon}$ for which the problem is primal feasible.

4.4.2 Dual of LPM(ε)

Let π_i for $v_i \in T \backslash \{v_0\}$ and β_j for $w_j \in W$ be the vector of non-negative dual values associated with Constraints 4.24 and 4.25, respectively. The dual problem DLPM(ε) corresponding to LPM(ε) is given by

$$\text{Maximize} \quad \sum_{v_i \in T \backslash \{v_0\}} \pi_i + \sum_{w_j \in W} \beta_j \tag{4.27}$$

$$\text{subject to :} \quad \sum_{v_i \in T \backslash \{v_0\}} a_{ik} \pi_i + \sum_{w_j \in W} b_{ik} \beta_j \leq c_k \quad (k \in \overline{\Omega}^{\varepsilon}) \,, \tag{4.28}$$

$$\pi_i \geq 0 \quad (v_i \in T \backslash \{v_0\}) \,, \tag{4.29}$$

$$\beta_j \geq 0 \quad (w_j \in W) \,. \tag{4.30}$$

4.4.3 Subproblem corresponding to RLPM(ε)

The subproblem $S(\varepsilon)$ corresponding to RLPM(ε) consists in finding columns k in $\overline{\Omega}^{\varepsilon} \backslash \overline{\Omega}_1^{\varepsilon}$ that satisfy the condition

$$c_k - \sum_{v_i \in T \backslash \{v_0\}} \pi_i a_{ik} - \sum_{w_i \in W} \beta_j b_{ik} < 0 \,. \tag{4.31}$$

In order to simplify this condition, we define $\pi_i^* = \pi_i$ if $v_i \in T \backslash \{v_0\}$ and $\pi_i^* = 0$, otherwise. We also let $x_{ijk} = 1$ if route R_k visits v_j immediately after visiting v_i and

$x_{ijk} = 0$ if this is not the case. The subproblem $S(\varepsilon)$ is given by :

$$\text{Minimize} \sum_{(v_i,v_j)\in V\times V} (d_{ij} + \pi_i^*)x_{ijk} - \sum_{w_i \in W} \beta_j b_{ik} \text{ subject to } k \in \overline{\Omega}^{\varepsilon} \backslash \overline{\Omega}_1^{\varepsilon} . \tag{4.32}$$

Given that $R_k \subseteq V$ whereas $\Psi_k \subseteq W$, we need to construct a route on a subset of V with the aim of minimizing its cost $c_k - \sum_{v_i \in V} a_{ik}\pi_i^*$ and also choose a subset of nodes $w_j \in W$ with the aim of maximizing the profits (β_j) associated to its members. The profit associated to a node of $w_j \in W$ can be collected at most once on any single route even though different nodes of the route R_k may be able to cover it. Problem 4.32 is, thus, an ESPPRC having a special feature called non-additivity. Non-additivity in shortest path problems may be defined as the situation whereby the cost of a partial path is not necessarily the same as the sum of the costs of its subpaths. Figure 4.4 demonstrates the non-additive nature of Problem 4.32. In this figure, let us suppose that $d_{12} = 2$, $d_{23} = 1$, and $\beta_j = 1$ for $j = 1, 2, \ldots, 7$. Then, the reduced cost of the partial path $v_1 \rightarrow v_2 \rightarrow v_3$ is given by $3 - 7 = -4$. Similarly, the reduced costs of the partial paths $v_1 \rightarrow v_2$ and $v_2 \rightarrow v_3$ are $2 - 6 = -4$ and $1 - 5 = -4$, respectively. This means that the reduced cost of $v_1 \rightarrow v_2 \rightarrow v_3$ is not the same as the sum of the reduced costs of its constituent partial paths. This is a result of the fact that the nodes w_5 and w_6 can be covered by both v_2 and v_3. Thus, we need to modify the "usual" dominance rule in order to ensure that no labels that can lead to an optimal solution is eliminated. Reinhardt and Pisinger (2011) discuss non-additive shortest path problems and also present different dominance rules for specific problems but none of those rules presented perfectly fits the subproblem encountered here. Next, we briefly describe how we solve Problem 4.32 by dynamic programming.

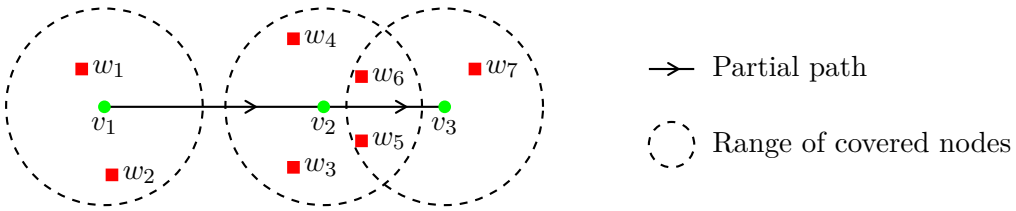


Figure 4.4: Non-additive nature of subproblem.

Notes: The reduced cost of the partial path $v_1 \rightarrow v_2 \rightarrow v_3$ is different from the sum of the reduced costs of the partial paths $v_1 \rightarrow v_2$ and $v_2 \rightarrow v_3$.

4.4.4 Solving $S(\varepsilon)$

Three resources are considered when implementing the decremental state space relaxation algorithm for Problem 4.32. The first two ressources are the number of nodes a route may visit which is limited to a maximum of p and its length which is limited to a maximum of q. The third resource constraint is that a route may only cover nodes of W that lie within a radius of ε from a node of $V\backslash\{v_0\}$ it visits. In what follows, let $\Lambda_i = (\tilde{c}_i, \tilde{q}_i, \tilde{p}_i)$ represent a partial path coming from the depot v_0 to node v_i. The components of the label are given by the reduced cost up to the current node $\tilde{c}_i$, the length of the partial path $\tilde{q}_i$, and the total number of nodes of $V\backslash\{v_0\}$ visited, $\tilde{p}_i$. In order to simplify the notation, some

components of a label are not shown. For example, those that ensure that a constructed path is elementary. The validity of the discussions that follows is not affected by this omission.

Label Extention. When extending a label from a node $v_i \in V$ to another node $v_j \in V$, nodes of W not yet covered by the label but which can be covered by v_j are identified and the resulting profit is subtracted from the current reduced cost of the label. Doing so ensures that we obtain the minimum possible reduced cost for each label without counting the profit associated to any node of W more than once. That is, even when the elementary condition is relaxed in the DSSR algorithm, there is no interest in visiting a node of $V \backslash T$ more than once since the reduced cost of the path resulting from the second visit will be worse than the one from the first visit.

Dominance Rule. Consider the case portrayed in Figure 4.5. The label Λ_1^1 represents a partial path from v_0 to v_1 that has already visited v_2 whereas Λ_1^2 is another partial path that has not yet visited v_2. That is, Λ_1^1 can no longer visit v_2 but it is possible to extend Λ_1^2 to v_2. The total profit that can be collected (from covering some nodes of W) by visiting node v_2 is 4. For simplicity, we suppose that no node of W can be covered by more than one node of V. The figure shows all the labels that are generated when no dominance rule is applied. Three labels arrive at v_d which is a copy of the depot. Out of these three labels, only Λ_d^1 and Λ_d^2 are actually of interest. If the usual dominance rule is applied, then we have $\Lambda_1^1 \leqq \Lambda_1^2$ and so Λ_1^2 is deleted and never extended from v_1. The result is that, the nondominated node Λ_d^3 is never generated. In order not to eliminate any label that can lead to a nondominated path, we define a new dominance rule. For any two labels Λ^i and Λ^j on the same node, we say that Λ^i dominates Λ^j if and only if the following two conditions are satisfied:

- $\Lambda_1^1 \leqq \Lambda_1^2$, and
- $\tilde{c}^i \leq \tilde{c}^j - F_{ij}$.

The factor F_{ij} represents the sum of the profits associated to nodes of W that are covered by Λ^i but not yet covered by Λ^j.

4.4.5 Implementation of Column Search Strategies

By following the discussions in the preceding sections, we know how to solve RLPM(ε) and $S(\varepsilon)$ for any given value of ε and so we can implement the Point-by-Point Search (PPS). In this section, we describe the Improved Point-by-Point Search (IPPS) and the Solve-Once-Generate-for-All (SOGA) approaches. In what follows, let $k' := (R'_k, \Psi'_k)$ be a column returned by the DSSR algorithm after solving the subproblem $S(\varepsilon')$. Let π' and β' be the vectors of dual values used by the DSSR algorithm in obtaining column k'.

Recall. A feasible column is defined by a route R_k together with a subset $\Psi_k \subseteq W$ of the choice of nodes it may cover. The length of R_k is denoted c_k. We denote σ_k as the maximum distance between a node of Ψ_k and the closest node of R_k. That is, $\sigma_k = \max\{d_{ij} : v_i \in R_k \text{ and } w_j \in \Psi_k\}$.

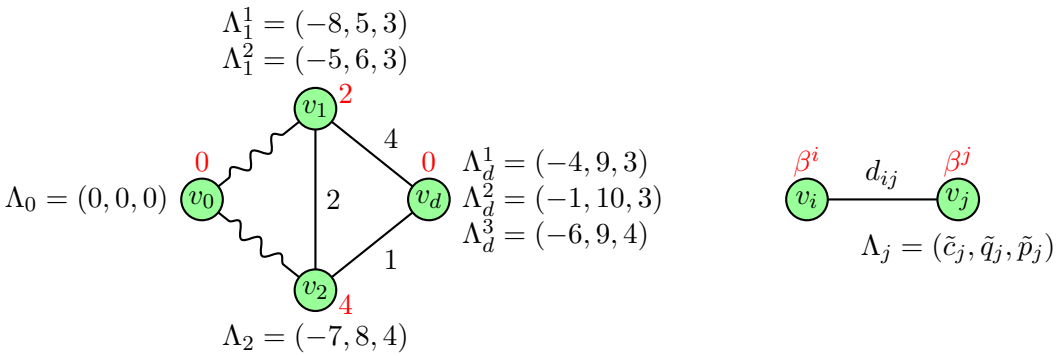


Figure 4.5: Dominance relationship between labels.

Notes: Node v_d is a copy of the depot, v_0. Λ_1^1 represents the partial path $v_0 \to v_3 \to v_2 \to v_1$, Λ_1^2 represents $v_0 \to v_3 \to v_4 \to v_1$, Λ_2 represents $v_0 \to v_3 \to v_4 \to v_1 \to 2$, Λ_d^1 represents $v_0 \to v_3 \to v_2 \to v_1 \to d$, Λ_d^2 represents $v_0 \to v_3 \to v_4 \to v_1 \to d$, Λ_d^3 represents $v_0 \to v_3 \to v_4 \to v_1 \to 2 \to d$.

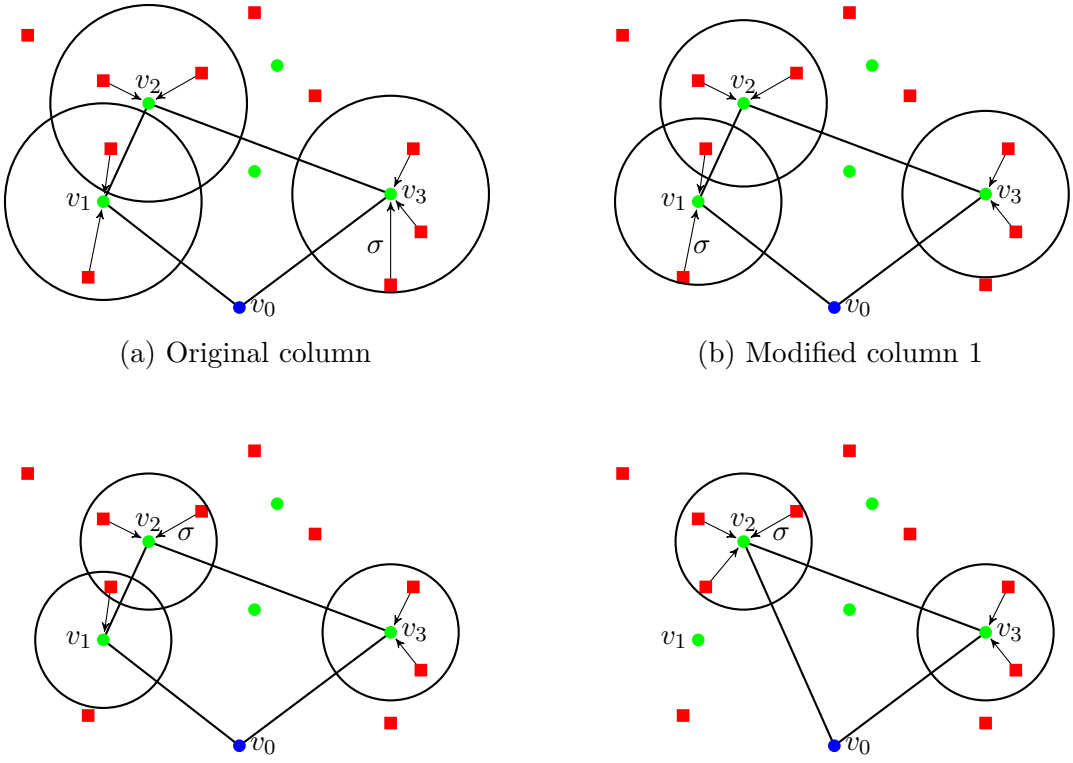
IPPS Heuristic for the BOMCTP

The DSSR algorithm for solving the $S(\varepsilon)$ constructs column $k' := (R'_k, \Psi'_k)$ by taking Ψ'_k to be all the nodes of W that lie within a radius of ε' from a node of R'_k. This helps with the goal of minimizing the reduced cost. We note, however, that Ψ'_k does not necessarily need to include all the nodes of W that can be covered by R_k. Indeed, Ψ'_k can be chosen to be any subset of W each of which lie within a radius of ε' from a node of R'_k and such that the sum of the profits associated with this subset exceeds the cost c_k. A column defined in this way is hardly returned by the DSSR algorithm for the current subproblem since it is dominated by another column defined by the same route, but covers some more nodes of W. The IPPS heuristic for the BOMCTP relies on this observation. Figure 4.6 depicts how the heuristic works and the details are explained below.

A column k' is modified by successively removing the node of Ψ'_k that induces the value of σ'_k (i.e. which is farthest from the closest node of R_k) in order to create another column $k'' := (R''_k, \Psi''_k)$ where $\sigma''_k < \sigma'_k$. The reduced cost of k'' is evaluated with respect to the same vectors of dual values π' and β' which was used by the DSSR algorithm in solving $S(\varepsilon')$. This means that if the reduced cost of k'' is negative, then it is guaranteed to be relevant for $S(\varepsilon')$. The relevance of k'' for another value of ε that is different from ε' is neither confirmed nor refuted by this heuristic. Initially, $c''_k = c'_k$ as in Figure 4.6c but if a node $v_i \in R''_k$ does not uniquely cover at least one node of Ψ''_k, then it is removed from R''_k in other to have $c''_k < c'_k$ and further minimize the reduced cost as in Figure 4.6d. The successive modifications end when no more columns having negative reduced costs can be obtained.

SOGA Heuristic for the BOMCTP

Suppose that π'' and β'' are the vectors of dual values corresponding to RLPM(ε'') where $\varepsilon'' \neq \varepsilon'$. Note that in general $\pi'' \neq \pi'$ and $\beta'' \neq \beta'$. The principle of a SOGA heuristic for the BOMCTP is to modify column $k' := (R'_k, \Psi'_k)$ to obtain another column $k'' := (R''_k, \Psi''_k)$



(a) Original column

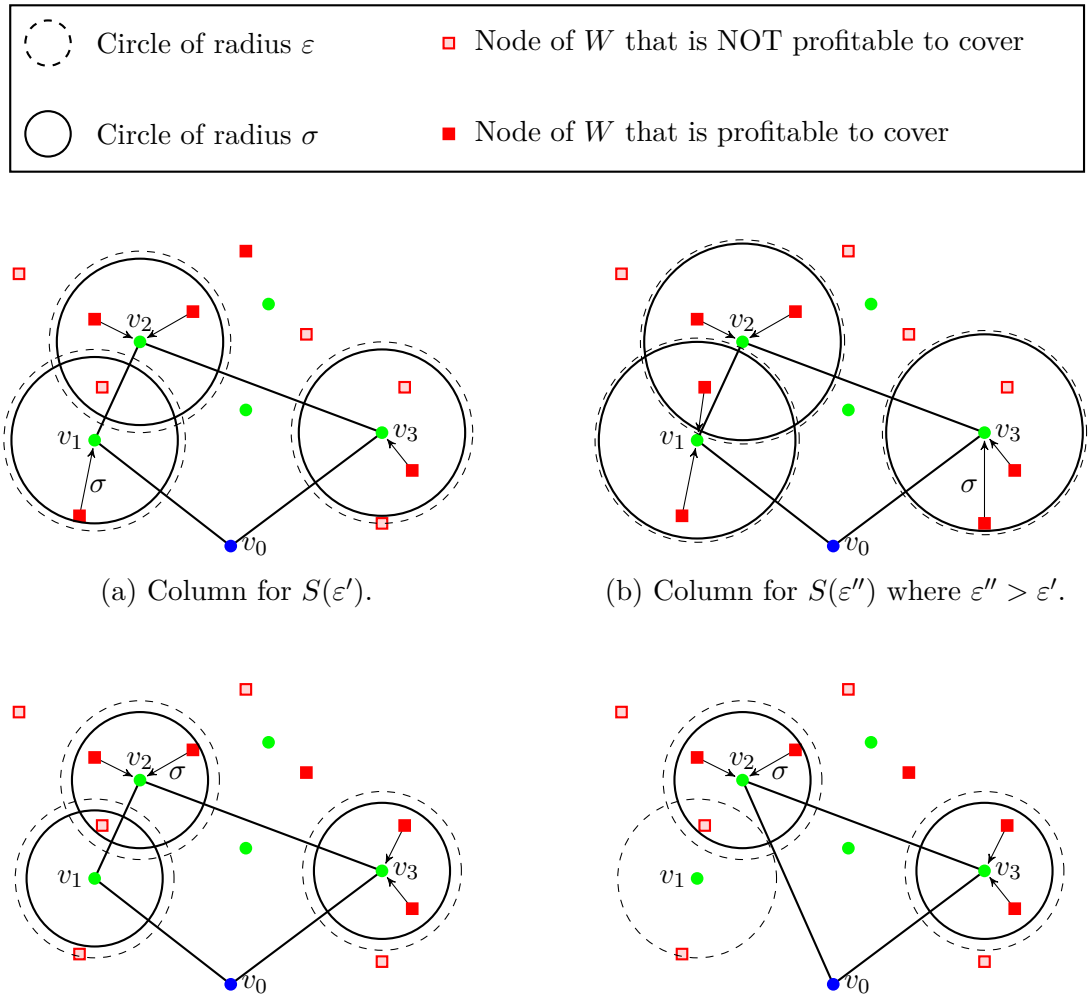
(b) Modified column 1

(c) Modified column 2

(d) Modified column 2 (improved)

Figure 4.6: IPPS heuristic for the BOMCTP.
Notes: A column is successively modified by removing the node of W that induces the value of σ in order to generate several other columns.

by completely reconstructing the set of nodes that may be covered (Ψ''_k). The principle of the heuristic is indicated in Figure 4.7. The set Ψ''_k is constructed by taking all profitable nodes within a radius of ε'' from a node of R''_k. The profit associated with covering a node of W depends on π'' and β'' rather than π' and and β'. In other words, the reduced cost of the modified column k'' is evaluated with respect to the dual vectors π'' and β''. This means that if the reduced cost of k'' is negative, then it is guaranteed to be relevant for $S(\varepsilon'')$ but possibly not for $S(\varepsilon')$. After constructing Ψ''_k, we compute $\sigma''_k = \max\{d_{ij} : v_i \in R''_k \text{ and } w_j \in \Psi''_k\}$. If a node $v_i \in R''_k$ does not uniquely cover any profitable node of Ψ''_k, then it is removed from R''_k in other to reduce the length of the route and further minimize the reduced cost (see Figure 4.7d). Finally, all the other non-profitable nodes of W that lie within a radius of σ''_k from a node of R''_k are added to Ψ''_k.



(a) Column for $S(\varepsilon')$.

(b) Column for $S(\varepsilon'')$ where $\varepsilon'' > \varepsilon'$.

(c) Column for $S(\varepsilon'')$ where $\varepsilon'' < \varepsilon'$.

(d) Improved column from (c).

Figure 4.7: SOGA heuristic for the BOMCTP.
Notes: A new column is constructed from an original column by incorporating dual values corresponding to another subproblem.

4.5 Computational Results

We now present results from experiments conducted to evaluate the quality of lower and upper bound sets computed for the BOMCTP. We also compare the relative perfomance of the different column search approaches. For each of the two formulations, lower bound sets are computed by the column search approaches discussed in Chapter 3. After computing a lower bound set, a corresponding upper bound set is computed by considering the set of columns currently present in the master problem.

4.5.1 Description of Instances and Experiments

The Mersenne Twister random number generator was used to generate instances similar to those described in the literature Gendreau et al. (1997); Hachicha et al. (2000); Jozefowiez et al. (2007) but which are not publicly available. The generator can be obtained at `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`. The node sets were obtained by generating $|V| + |W|$ points in the $[0, 100] \times [0, 100]$ square with the depot restricted to lie in $[25, 75] \times [25, 75]$. Set T (respectively, V) is taken to be the first $|T|$ (respectively, $|V|$) points and set W is taken as the remaining points. The distance between two points is calculated as the Euclidean distance between them. Five instances for every combination of $|V| \in \{30, 40, 50\}$ and $|W| \in \{2|V|, 3|V|\}$ were generated. Values of $|T|$ in $\{1, 0.25 \cdot \lceil |V| \rceil, 0.50 \cdot \lceil |V| \rceil\}$, p in $\{5, 8\}$, and $q = +\infty$ were tested. The instances used for our experiments can be found at `http://homepages.laas.fr/artigues/ctp_instances.zip`. All computer codes were written in C/C++ and the linear programs were solved with ILOG CPLEX 12.4. The tests were run on an Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz computer with a 2 GiB RAM. Summary of the results based on the first formulation are given in Section 4.5.2 and those based on the second formulation are given in Section 4.5.3. All the values in the tables presented in both sections are averages over the same group five generated instances as earlier described.

4.5.2 Summary of Results for Formulation 1

Lower bound sets for the BOMCTP based on the first formulation were computed by the PPS, the k-PPS, and the Sequential Search 2 approaches. As described in Chapter 3, the condition $\mathcal{O}$ used in implementing the k-PPS is to skip a current value of ε when the objective function of RLPM(ε) has not improved (i.e. decreased) by more than 1 after $k = 3$ iterations of column generation. A summary of the results are presents in Tables 4.1 and 4.2. Table 4.1 gives the quality of the computed bound sets whereas Table 4.2 gives the computational times and other characteristics of the column generation method. Given that column generation is an exact method for the LPM(ε) for any particular value of ε, the same lower bound set is obtained for a given instance no matter the column search approach used. For each instance, the number of elements in the lower bound set is given under the column with heading $|\boldsymbol{L}^*|$. The number of elements in an upper bound set is given under the columns with headings $|\boldsymbol{U}|$. For each instance, these values are different for the different column search approaches since different columns are generated by each approach when computing a lower bound set. The computational times (in cpu seconds) are given under the columns with heading *time*. The columns with headings *dssr* and

cols give the number of times the sub-problem was solved with the DSSR algorithm and the total number of columns generated, respectively, when computing a lower bound set. Values of the two quality indicators discussed in Chapter 3 are expressed as percentages under the headings $\mu_1\%$ and $\mu_2\%$.

The values in Table 4.1 show that the quality of the bound sets obtained are fairly good. This can be seen from the values under the columns with headings $\mu_1\%$ and $\mu_2\%$ for each of the search approaches. There seems to be no clear preference for one column search approach with respect to another in terms of the quality of the bound sets obtained. Nevertheless, the quality of the bound sets obtained by the sequential search approach are slightly better than those obtained by the PPS and k-PPS in most cases. For example, in the row corresponding to $p = 8$, $|T| = 1$, $|V| = 40$, $|W| = 120$ of Table 4.1. The value of the pair $(\mu_1\%, \mu_2\%)$ for the PPS, the k-PPS, and the sequential search approaches are (5.2, 25.7), (5.3, 25.4), and (5.1, 25.2), respectiveley.

In terms of computational times (see Table 4.2), the sequential search approach performs the best. The k-PPS also performs better than the PPS. For example, in the row corresponding to $p = 8$, $|T| = 13$, $|V| = 50$, $|W| = 150$ of Table 4.2, the computational time for the PPS is 110.41, the one for the k-PPs is 103.65 and that of the sequential search is 87.10. From the same row, we can also see that the sequential search approach solves a fewer number of subproblems (219) with respect to the PPS (294) and the k-PPS (272). This is a very good sign in favour of the sequential search approach since solving the subproblem is usually the most costly operation in a column generation algorithm.

4.5.3 Summary of Results for Formulation 2

Lower and upper bound sets based on the second formulation for the BOMCTP were computed by the PPS, the IPPS and the SOGA approaches. Tables 4.3 and 4.4 give a summary of the results obtained. These tables are the equivalent of those given in Tables 4.1 and 4.2 for the first formulation. Hence, the column headings have the same meanings as those presented in the earlier tables.

It can be seen from Table 4.3 that there is not much difference between the values of $\mu_1\%$ and $\mu_2\%$ for the different search approaches. No preference for a particular column search approach can be established from the values of $\mu_1\%$ and $\mu_2\%$ in this table. For example, in the row corresponding to $p = 5$, $|T| = 25$, $|V| = 50$, $|W| = 150$, the value of the pair $(\mu_1\%, \mu_2\%)$ for the PPS, the IPPS, and SOGA are (3.0, 57.7), (1.4, 41.1), and (1.8, 35.9), respectively. Similarly, the values for PPS, IPPS, and SOGA corresponding to $p = 8$, $|T| = 10$, $|V| = 40$, $|W| = 80$ Table 4.3 are (6.9, 19.6), (7.6, 26.8), and (7.3, 24.6), respectively. Just as it can be seen in Tables 4.1 for the first formulation, there is a general increase in the values of $\mu_1\%$ and $\mu_2\%$ when $|V|$ increases and also when $|T|$ increases. No matter the column search approach used, this trend is expected since increasing the values of $|V|$ and $|T|$ usually increases the difficulty of an instance.

In terms of computational times, there is a clear preference between the column search approaches. SOGA is clearly the best, followed by IPPS and then PPS. Moreover this preference becomes more evident for difficult instances. For example, the computational times for PPS, IPPS and SOGA for the instance $p = 8$, $|T| = 13$, $|V| = 50$, $|W| = 150$ of Table 4.4 are 2226.0, 1938.3, and 1033.4, respectively, whereas those for the instance $p = 8$,

Table 4.1: Quality of bound sets for Formulation 1

					PPS			k-PPS			Sequential 2		
p	$\|T\|$	$\|V\|$	$\|W\|$	$\|\boldsymbol{L}^*\|$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$
5	1	30	60	64	29	5.2	26.2	28	5.2	26.1	29	5.1	26.0
5	1	30	90	66	29	6.0	27.6	29	6.1	27.5	28	5.9	27.5
5	1	40	80	58	30	5.5	26.1	29	5.6	26.0	30	5.4	25.9
5	1	40	120	67	37	5.1	25.4	29	5.1	25.4	36	5.0	25.2
5	1	50	100	65	34	4.9	24.7	34	4.9	24.5	35	4.8	24.5
5	1	50	150	67	35	4.5	24.8	39	4.6	24.7	36	4.5	24.0
5	8	30	60	15	9	7.7	54.1	9	7.7	54.1	9	7.6	54.1
5	8	30	90	17	10	11.0	53.3	10	11.1	53.3	10	11.0	53.2
5	10	40	80	15	11	1.4	58.1	11	1.4	58.1	12	1.3	58.1
5	10	40	120	16	13	9.1	61.7	12	9.1	61.7	14	9.0	61.8
5	13	50	100	18	11	16.7	72.9	11	16.7	80.0	11	16.6	72.7
5	13	50	150	19	10	21.6	72.6	11	21.5	72.7	12	21.6	72.9
5	15	30	60	2	6	20.3	54.7	5	20.3	54.6	6	20.3	54.6
5	15	30	90	5	6	40.5	55.7	6	40.5	55.7	6	40.4	55.7
5	20	40	80	6	4	57.3	61.4	5	57.3	61.4	5	57.3	61.4
5	20	40	120	7	5	31.2	62.3	6	31.2	62.3	6	31.2	62.3
5	25	50	100	6	4	24.3	76.8	4	24.3	76.5	4	24.3	76.8
5	25	50	150	9	4	28.9	79.1	4	28.9	78.9	5	28.9	79.0
8	1	30	60	64	29	5.3	26.3	29	5.3	26.1	30	5.2	26.2
8	1	30	90	63	30	6.5	28.2	29	6.5	28.1	29	6.4	28.0
8	1	40	80	58	28	5.6	26.0	28	5.5	26.0	30	5.6	26.0
8	1	40	120	65	35	5.2	25.7	26	5.3	25.4	36	5.1	25.2
8	1	50	100	65	34	4.9	24.7	34	4.9	24.6	33	4.7	24.6
8	1	50	150	64	36	4.7	24.8	38	4.6	24.5	36	4.7	24.4
8	8	30	60	15	10	10.2	65.1	10	10.2	65.0	10	10.2	65.1
8	8	30	90	15	9	28.7	69.5	10	26.8	69.7	10	20.3	69.6
8	10	40	80	14	8	16.6	68.0	8	16.6	67.9	9	16.6	67.8
8	10	40	120	17	11	9.3	73.4	10	9.3	73.3	11	9.2	73.3
8	13	50	100	16	10	35.2	74.2	10	35.2	74.2	10	35.1	74.2
8	13	50	150	17	9	30.1	76.7	9	30.1	76.7	10	30.1	76.7
8	15	30	60	2	6	29.0	65.4	6	29.0	65.4	6	29.0	65.4
8	15	30	90	2	6	35.4	71.1	6	35.4	71.3	6	35.4	71.0
8	20	40	80	2	4	41.2	72.9	5	41.3	73.0	5	41.1	72.7
8	20	40	120	4	6	9.2	73.8	6	9.1	73.8	6	9.1	73.2
8	25	50	100	3	5	34.9	81.3	5	35.0	81.1	6	34.9	81.0
8	25	50	150	4	5	24.3	87.9	6	24.2	87.3	7	24.2	87.2

Table 4.2: Computational times for Formulation 1

				PPS			k-PPS			Sequential 2		
p	$\|T\|$	$\|V\|$	$\|W\|$	time	dssr	cols	time	dssr	cols	time	dssr	cols
5	1	30	60	7.8	160	358	6.7	126	388	7.0	98	372
5	1	30	90	12.8	93	332	10.8	95	364	9.1	73	351
5	1	40	80	15.0	72	416	11.4	70	479	7.8	59	461
5	1	40	120	16.0	129	435	12.5	112	521	7.8	90	478
5	1	50	100	26.0	152	570	13.3	147	574	8.8	135	583
5	1	50	150	24.7	137	591	15.3	116	573	11.5	106	619
5	8	30	60	6.3	165	808	8.7	147	823	7.9	123	747
5	8	30	90	15.3	143	837	13.1	139	891	14.1	104	865
5	10	40	80	18.6	157	969	19.4	142	1036	15.6	132	974
5	10	40	120	28.2	195	1048	28.2	176	986	23.9	155	1096
5	13	50	100	42.4	243	1160	39.5	198	1171	36.3	174	1235
5	13	50	150	51.7	286	1395	52.7	201	1343	42.1	194	1480
5	15	30	60	16.8	149	825	15.1	126	891	14.0	114	926
5	15	30	90	26.4	153	973	25.5	148	974	23.8	141	969
5	20	40	80	35.8	168	1232	36.8	160	1235	27.4	143	1198
5	20	40	120	53.5	192	1568	50.7	174	1549	42.8	158	1480
5	25	50	100	55.5	227	1913	49.0	195	1629	44.2	172	1670
5	25	50	150	68.2	285	2194	61.6	223	2160	53.3	206	2075
8	1	30	60	11.8	245	374	15.8	183	392	6.9	119	449
8	1	30	90	14.9	144	357	14.2	141	383	8.3	132	417
8	1	40	80	54.0	124	436	22.2	109	451	17.5	93	491
8	1	40	120	47.2	154	438	29.3	127	492	18.3	120	524
8	1	50	100	96.6	189	618	22.6	142	575	15.1	132	633
8	1	50	150	86.5	199	566	31.1	132	573	19.6	111	578
8	8	30	60	52.1	231	738	53.9	211	718	48.1	176	619
8	8	30	90	58.0	195	953	62.2	176	949	51.3	164	946
8	10	40	80	81.4	206	1087	76.1	184	1093	69.3	153	1074
8	10	40	120	69.5	237	988	61.9	201	1027	54.7	159	1080
8	13	50	100	123.7	273	1439	115.1	258	1335	79.9	213	1326
8	13	50	150	110.4	294	1712	103.7	272	1726	87.1	219	1766
8	15	30	60	42.1	259	952	42.9	209	968	38.2	182	825
8	15	30	90	60.3	198	1071	58.7	177	1108	46.5	169	1151
8	20	40	80	153.7	281	1209	129.9	275	1160	109.3	227	1080
8	20	40	120	118.7	296	1841	103.1	278	1833	98.4	254	1766
8	25	50	100	209.9	317	2058	196.5	297	2051	182.3	261	2049
8	25	50	150	264.9	334	2314	251.9	312	2217	201.5	259	2147

Table 4.3: Quality of bound sets for Formulation 2

					PPS			IPPS			SOGA		
p	$\|T\|$	$\|V\|$	$\|W\|$	$\|\boldsymbol{L}^*\|$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$
5	1	30	60	25	26	0.3	0.6	26	0.4	0.6	26	0.4	0.6
5	1	30	90	21	21	0.8	2.3	22	0.9	2.0	22	0.9	2.0
5	1	40	80	26	26	0.3	0.4	27	0.3	0.5	26	0.3	0.5
5	1	40	120	27	28	0.4	1.2	29	0.5	1.1	29	0.5	1.2
5	1	50	100	32	33	0.3	0.7	33	0.2	0.6	33	0.3	0.6
5	1	50	150	30	30	0.2	0.5	30	0.2	0.7	31	0.2	0.4
5	8	30	60	10	9	0.2	5.9	9	0.2	5.7	9	0.1	9.3
5	8	30	90	10	10	2.3	9.6	10	2.3	6.5	10	2.6	6.5
5	10	40	80	10	11	0.8	9.2	11	1.0	9.9	11	1.3	9.6
5	10	40	120	12	12	0.7	7.8	12	0.8	7.8	13	1.1	8.0
5	13	50	100	11	11	0.7	4.5	10	1.1	4.0	10	0.6	3.9
5	13	50	150	8	8	3.3	11.9	8	1.9	10.1	9	3.6	9.8
5	15	30	60	6	5	2.2	18.0	5	2.2	13.7	5	2.2	14.5
5	15	30	90	5	6	1.9	15.7	6	1.4	14.5	6	1.8	15.0
5	20	40	80	5	4	5.5	41.2	4	5.4	40.3	4	4.9	40.0
5	20	40	120	5	5	0.8	38.2	5	0.8	46.4	5	0.8	41.3
5	25	50	100	4	4	18.7	36.3	4	1.1	47.8	4	1.1	51.3
5	25	50	150	3	3	3.0	57.7	3	1.4	41.1	3	1.8	35.9
8	1	30	60	25	26	0.1	0.6	25	0.1	0.5	25	0.1	0.5
8	1	30	90	22	22	0.7	1.4	23	0.8	1.2	22	0.8	1.1
8	1	40	80	27	27	0.1	0.4	27	0.1	0.4	27	0.1	0.4
8	1	40	120	29	30	0.2	0.7	30	0.3	0.7	30	0.3	0.7
8	1	50	100	32	32	0.2	0.7	33	0.2	0.6	33	0.2	0.7
8	1	50	150	30	30	0.3	0.7	30	0.2	0.5	31	0.3	0.5
8	8	30	60	10	10	2.5	8.9	9	1.6	9.8	9	2.5	10.1
8	8	30	90	9	9	2.7	10.7	9	2.7	10.4	10	2.7	10.3
8	10	40	80	8	7	6.9	19.6	9	7.6	26.8	8	7.3	24.6
8	10	40	120	11	11	0.7	7.1	12	1.3	8.8	12	1.1	8.8
8	13	50	100	10	10	3.8	10.9	10	3.8	11.7	10	3.4	10.7
8	13	50	150	8	8	2.4	12.0	8	2.2	10.3	8	2.1	10.0
8	15	30	60	6	5	0.8	26.6	5	1.0	26.2	5	1.5	27.6
8	15	30	90	5	6	2.8	21.5	6	3.2	23.4	6	3.3	23.5
8	20	40	80	4	4	15.4	75.6	4	32.5	74.8	4	15.4	69.3
8	20	40	120	5	5	3.2	50.2	5	3.4	51.7	5	3.8	50.3
8	25	50	100	4	4	2.1	56.1	4	2.2	56.1	4	2.1	56.4
8	25	50	150	3	3	2.5	64.9	3	2.6	71.5	3	3.0	65.7

Table 4.4: Computational times for Formulation 2

				PPS			IPPS			SOGA		
p	$\|T\|$	$\|V\|$	$\|W\|$	time	dssr	cols	time	dssr	cols	time	dssr	cols
5	1	30	60	18.0	185	1198	13.9	124	1809	13.0	112	1229
5	1	30	90	15.4	163	1063	14.6	120	1569	12.3	106	1041
5	1	40	80	49.5	228	1597	40.5	155	2099	36.7	141	1610
5	1	40	120	126.8	330	2571	94.0	201	3388	86.4	174	2348
5	1	50	100	205.8	390	3035	153.9	226	3459	154.5	213	2871
5	1	50	150	392.5	486	4053	287.0	247	4054	268.1	224	3087
5	8	30	60	13.6	145	1155	10.9	90	1478	9.1	72	982
5	8	30	90	17.9	139	1118	14.3	100	1348	11.9	64	842
5	10	40	80	41.3	191	1600	36.0	138	2105	27.2	83	1229
5	10	40	120	104.2	182	2441	82.9	199	2441	58.2	101	1758
5	13	50	100	148.6	309	2728	123.6	231	3299	87.5	110	2207
5	13	50	150	186.8	264	2342	174.3	228	2755	102.8	87	1826
5	15	30	60	22.9	171	1463	17.6	113	2750	11.4	55	1444
5	15	30	90	29.5	141	1198	27.0	114	2067	17.7	51	1083
5	20	40	80	93.0	207	1832	75.0	156	3726	46.0	67	2143
5	20	40	120	143.3	236	2136	127.1	198	3369	67.7	67	2445
5	25	50	100	231.3	257	2363	216.0	219	4193	112.6	75	2886
5	25	50	150	264.4	185	1700	259.9	171	2902	146.9	65	2440
8	1	30	60	49.8	227	1627	29.7	152	2235	25.5	136	1657
8	1	30	90	31.3	215	1552	23.3	142	2010	18.7	122	1284
8	1	40	80	113.4	302	2283	103.6	218	2961	86.8	183	2253
8	1	40	120	511.2	481	3949	503.9	293	4663	326.6	243	3652
8	1	50	100	1343.7	522	4306	1012.5	335	5071	821.2	289	4393
8	1	50	150	1525.2	672	5799	1186.2	384	6005	1042.1	306	4782
8	8	30	60	53.6	231	2027	44.6	158	2466	31.9	123	1952
8	8	30	90	77.7	219	1873	65.2	154	2071	50.7	101	1558
8	10	40	80	213.4	298	2730	202.2	214	3560	149.6	119	2236
8	10	40	120	797.0	475	4427	623.5	320	4102	407.7	183	3837
8	13	50	100	1142.5	571	5379	973.4	413	6220	755.8	216	4749
8	13	50	150	2226.0	545	5119	1938.3	446	6060	1033.4	161	3939
8	15	30	60	182.9	333	3086	128.7	205	5493	87.5	103	3298
8	15	30	90	380.6	288	2691	283.4	209	4675	213.6	102	2451
8	20	40	80	1715.3	430	4129	1459.5	318	8396	978.6	120	5614
8	20	40	120	3202.5	564	5392	2973.4	455	8532	1568.9	149	6162
8	25	50	100	7861.5	627	6022	6963.5	538	11026	3917.6	165	8249
8	25	50	150	3143.1	374	3623	2941.0	342	6757	1731.5	130	5438

$|T| = 25$, $|V| = 50$, $|W| = 150$ in the same table are 3143.1, 2941.0, 1731.5, respectively. In addition, SOGA usually needs to solve relatively fewer number of subproblems than both PPS and IPPS when computing a lower bound sets. This can be seen from the columns with headings *dssr* in Table 4.4.

4.5.4 Comparison of Formulations 1 and 2

In order to better compare the quality of the bound sets obtained from the two formulations, we present Tables 4.5 and 4.6 which repeat part of the results presented in ealier tables.

Table 4.5 compares the quality of the bound sets obtained by PPS for each of the formulations. Although the principle of the PPS approach remains the same for both formulations, the actual implementation details depend on the specific formulation. That is, different lower and upper bound sets are obtained by PPS for each of the two formulations although the same test instances were used. A general trend that can be seen from Table 4.5 is that, the number of points in a bound set decreases when the size of T increases. This can be seen under the columns with headings $|\boldsymbol{L}|$ and $|\boldsymbol{U}|$. As we have already noted, the quality of the bound sets obtained from the second formulation are significantly better than those obtained from the first formulation. In most cases the values of $\mu_1\%$ for Formulation 1 are greater than 5 whereas those for for Formulation 2 are less than 5. By comparing the values of $\mu_2\%$ for the two formulations, we can also see that those corresponding to Formulation 1 are generally greater than 25 whereas the ones for Formulation 2 are generally less than 25.

We can see from Table 4.5 that Formulation 2 clearly obtains better values of $\mu_1\%$ and $\mu_2\%$. We can, however, not base on these values to conclude that the linear relaxation of Formulation 2 is stronger than the one for Formulation 1. This is because the values in Table 4.5 were computed with respect to the lower and upper bound sets that were computed independently by each of the formulations. Thus, the better values obtained by Formulation 2 may be because it obtained better lower bound sets, or better upper bound sets, or better lower and upper bound sets than Formulation 1. In order to verify whether Formulation 2 has a stronger linear relaxation than Formulation 1, we fixed an upper bound set for each instance and compared the lower bound sets obtained independently by each formulation with respect to the fixed upper bound set. For each instance, the fixed upper bound set was taken as the one obtained by Formulation 2. Table 4.6 presents the results obtained for these tests. In this table, the column with heading $|\boldsymbol{U}^*|$ represents the cardinality of the fixed upper bound set. All the other columns have the same meanings as before. It is seen from this table (Table 4.6) that the values of $\mu_1\%$ and $\mu_2\%$ obtained for Formulation 2 are significantly better (smaller) than those obtained by Formulation 1. This is a confirmation that the lower bound sets obtained by Formulation 2 were better than those obtained by Formulation 1. In other words, Formulation 2 has a stronger linear relaxation than Formulation 1. We can also conclude that the upper bound sets obtained by Formulation 2 were better than those obtained by Formulation 1. This is because by using the upper bound sets obtained Formulation 2 as the reference upper bound set, Formulation 1 obtained better values of $\mu_1\%$ and $\mu_2\%$ in Table 4.6 than in Table 4.5.

Table 4.5: Comparison of Formulations 1 and 2

				Formulation 1				Formulation 2			
p	$\|T\|$	$\|V\|$	$\|W\|$	$\|\boldsymbol{L}\|$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{L}\|$	$\|\boldsymbol{U}\|$	$\mu_1\%$	$\mu_2\%$
5	1	30	60	64	29	5.2	26.2	25	26	0.3	0.6
5	1	30	90	66	29	6.0	27.6	21	21	0.8	2.3
5	1	40	80	58	30	5.5	26.1	26	26	0.3	0.4
5	1	40	120	67	37	5.1	25.4	27	28	0.4	1.2
5	1	50	100	65	34	4.9	24.7	32	33	0.3	0.7
5	1	50	150	67	35	4.5	24.8	30	30	0.2	0.5
5	8	30	60	15	9	7.7	54.1	10	9	0.2	5.9
5	8	30	90	17	10	11.0	53.3	10	10	2.3	9.6
5	10	40	80	15	11	1.4	58.1	10	11	0.8	9.2
5	10	40	120	16	13	9.1	61.7	12	12	0.7	7.8
5	13	50	100	18	11	16.7	72.9	11	11	0.7	4.5
5	13	50	150	19	10	21.6	72.6	8	8	3.3	11.9
5	15	30	60	2	6	20.3	54.7	6	5	2.2	18.0
5	15	30	90	5	6	40.5	55.7	5	6	1.9	15.7
5	20	40	80	6	4	57.3	61.4	5	4	5.5	41.2
5	20	40	120	7	5	31.2	62.3	5	5	0.8	38.2
5	25	50	100	6	4	24.3	76.8	4	4	18.7	36.3
5	25	50	150	9	4	28.9	79.1	3	3	3.0	57.7
8	1	30	60	64	29	5.3	26.3	25	26	0.1	0.6
8	1	30	90	63	30	6.5	28.2	22	22	0.7	1.4
8	1	40	80	58	28	5.6	26.0	27	27	0.1	0.4
8	1	40	120	65	35	5.2	25.7	29	30	0.2	0.7
8	1	50	100	65	34	4.9	24.7	32	32	0.2	0.7
8	1	50	150	64	36	4.7	24.8	30	30	0.3	0.7
8	8	30	60	15	10	10.2	65.1	10	10	2.5	8.9
8	8	30	90	15	9	28.7	69.5	9	9	2.7	10.7
8	10	40	80	14	8	16.6	68.0	8	7	6.9	19.6
8	10	40	120	17	11	9.3	73.4	11	11	0.7	7.1
8	13	50	100	16	10	35.2	74.2	10	10	3.8	10.9
8	13	50	150	17	9	30.1	76.7	8	8	2.4	12.0
8	15	30	60	2	6	29.0	65.4	6	5	0.8	26.6
8	15	30	90	2	6	35.4	71.1	5	6	2.8	21.5
8	20	40	80	2	4	41.2	72.9	4	4	15.4	75.6
8	20	40	120	4	6	9.2	73.8	5	5	3.2	50.2
8	25	50	100	3	5	34.9	81.3	4	4	2.1	56.1
8	25	50	150	4	5	24.3	87.9	3	3	2.5	64.9

Table 4.6: Comparison of Lower Bound Sets for Formulations 1 and 2

					Formulation 1			Formulation 2		
p	$\|T\|$	$\|V\|$	$\|W\|$	$\|\boldsymbol{U^*}\|$	$\|\boldsymbol{L}\|$	$\mu_1\%$	$\mu_2\%$	$\|\boldsymbol{L}\|$	$\mu_1\%$	$\mu_2\%$
5	1	30	60	26	64	4.8	20.2	25	0.3	0.6
5	1	30	90	21	66	5.4	23.0	21	0.8	2.3
5	1	40	80	26	58	4.3	21.7	26	0.3	0.4
5	1	40	120	28	67	4.2	19.7	27	0.4	1.2
5	1	50	100	33	65	4.3	21.6	32	0.3	0.7
5	1	50	150	30	67	3.6	21.8	30	0.2	0.5
5	8	30	60	9	15	5.9	51.5	10	0.2	5.9
5	8	30	90	10	17	9.8	40.2	10	2.3	9.6
5	10	40	80	11	15	1.1	31.8	10	0.8	9.2
5	10	40	120	12	16	7.0	46.2	12	0.7	7.8
5	13	50	100	11	18	8.9	37.1	11	0.7	4.5
5	13	50	150	8	19	10.9	36.8	8	3.3	11.9
5	15	30	60	5	2	14.9	43.1	6	2.2	18.0
5	15	30	90	6	5	9.3	47.5	5	1.9	15.7
5	20	40	80	4	6	15.0	55.6	5	5.5	41.2
5	20	40	120	5	7	8.5	57.3	5	0.8	38.2
5	25	50	100	4	6	21.0	57.7	4	18.7	36.3
5	25	50	150	3	9	6.9	62.6	3	3.0	57.7
8	1	30	60	26	64	4.7	22.3	25	0.1	0.6
8	1	30	90	22	63	5.3	21.2	22	0.7	1.4
8	1	40	80	27	58	5.1	22.5	27	0.1	0.4
8	1	40	120	30	65	5.1	21.1	29	0.2	0.7
8	1	50	100	32	65	4.2	18.6	32	0.2	0.7
8	1	50	150	30	64	4.5	19.7	30	0.3	0.7
8	8	30	60	10	15	9.4	49.4	10	2.5	8.9
8	8	30	90	9	15	11.6	41.7	9	2.7	10.7
8	10	40	80	7	14	14.5	46.1	8	6.9	19.6
8	10	40	120	11	17	4.6	28.6	11	0.7	7.1
8	13	50	100	10	16	11.6	39.9	10	3.8	10.9
8	13	50	150	8	17	11.5	38.8	8	2.4	12.0
8	15	30	60	5	2	9.9	57.1	6	0.8	26.6
8	15	30	90	6	2	13.3	56.7	5	2.8	21.5
8	20	40	80	4	2	20.9	69.0	4	15.4	75.6
8	20	40	120	5	4	8.8	73.1	5	3.2	50.2
8	25	50	100	4	3	8.3	72.1	4	2.1	56.1
8	25	50	150	3	4	6.1	79.9	3	2.5	64.9

4.6 Conclusion

In this chapter, we have presented the bi-objective multi-vehicle covering tour problem (BOMCTP) which is an extension of the covering tour problem (CTP) (Gendreau et al., 1997). The BOMCTP, just as the other variants of the CTP, has several applications in the design of bi-level transportation networks. We presented two different formulations for the problems and used column generation to compute lower and upper bound sets based on each of them. The first formulation is based on the "standard" ε-constraint method whereas the second is based on a different variant of the ε-constraint method. Due to the bi-objective nature of the BOMCTP, different strategies to effectively search for columns were tested on some randomly generated istances similar to those used for the CTP. The results obtained show that significantly better quality bound sets are obtained from the second formulation (a variant of the ε-constraint method) than from the first formulation (a standard ε-constraint method). Morever, the different column search approaches tested show that significant improvement in computational times can be achieved when computing lower bound sets by column generation if the search for columns is intelligently managed.

Conclusions and Perspectives

Conclusions

The main interest of this thesis has been to study the application of column generation in computing lower bound sets for bi-objective integer linear programs. Just as in single objective integer programming, the main role of column generation in multi-objective integer programming is to compute dual bounds (that is, lower bounds for minimization problems and upper bounds for maximization problems) for the considered problems.

We started by reviewing the use of column generation in computing lower bounds for single objective vehicle routing problems (VRPs) in Chapter 1. In this chapter, we talked about the different types of formulations for VRPs and those that are used when we need to compute lower bounds by column generation. The main steps involved in applying column generation are demonstrated on the basic variant of the VRP. In the last section of this chapter, we present an original variant of the VRP namely the minimum and maximum distance-constrained capacitated VRP (MMDCVRP) and study how to compute lower bounds for this variant by column generation. Although the MMDCVRP is seemingly not so different from the DCVRP, we saw that the subproblem encountered was significantly more difficult to solve by well known dynamic programming algorithms. Indeed, we could only find one published article that deals with a similar subproblem but the dominance rule proposed in this article was not effective enough. Due to this, we propose another condition to strengthen the existing dominance rule. The effectiveness of the dominance rule we proposed is evident from the results obtained from experiments.

Chapter 2 was dedicated to reviewing different aspects of multi-objective optimization which were necessary for understanding the chapters that followed. In particular, we introduced the definitions and notations used throughout the manuscript. We also reviewed some popular exact methods for bi-objective optimization problems whose principles are used in Chapters 3 and 4.

Chapter 3 discusses how to use column generation to compute lower bound sets for bi-objective integer programs. Some recent published papers deal with the computation and use of bound sets for bi-objective problems but only one of them uses column generation. Moreover, all of these published papers used a weighted sum method in converting the bi-objective problems into single objective. To the best of our knowledge, we are the first to propose the use of a different scalarization method (an ε-constraint method) in computing bound sets. We propose a generic column generation algorithm to compute lower bound sets for bi-objective integer programs and discuss different approaches for implementing this algorithm. It is possible to use either a weighted sum method or an ε-

constraint method when implementing this generic algorithm. The main difference between the implementation approaches comes from the order in which the different subproblems encountered are treated. We propose two main implementation approaches namely the Point-by-Point Search (PPS) and the Sequential Search. In addition, we also propose two variants for each of these main approaches.

In Chapter 4, the relative performances of the column generation approaches proposed in Chapter 3 are evaluated on a generalization of the covering tour problem (CTP) namely the bi-objective multi-vehicle covering tour problem (BOMCTP). The BOMCTP has not been studied by any other authors and so no known formulation for the problem existed. We, thus, proposed two different set-covering based formulations the BOMCTP and used column generation to compute lower and upper bound sets based on each of these two formumlations. The first formulation uses the idea of a standard ε-constraint method. For this first formulation, the constraints involving ε are managed only in the master problem during a column generation algorithm. The second formulation is based on a variant of the ε-constraint method. In applying a column generation method to this second formulation, the constraints involving ε are managed both in the master problem and the subproblem. We performed experiments on randomly generated instances similar to the ones used in testing other variants of the CTP. The results obtained from our experiments show that the lower and upper bound sets obtained from the second formulation are significantly better than those obtained from the first formulation. Moreover, the efficiency of the proposed column search approaches is assessed from the computational times obtained.

Perspectives

Column generation and multi-objective optimization are two important domains in operations research. In spite of their importance, these two subjects are rarely studied together. The number of published articles that deal with the application of column generation to multi-objective problems are very few and even these were published not so long ago. Column generation is a complicated method and as such they are numerous possibilities of improving it for specific types of problems. In this thesis, we study its application and improvement for bi-objective integer linear programs. The work of this thesis is just a first step in a long journey of developing the column generation method for multi-objective optimization problems. There are several ways in which the work of this thesis can be extended and improved. We hope to achieve some of these extensions and improvements in the short term whereas some others are hoped to be achieved in the middle term and long term.

Our discussions on computing lower and upper bound sets in Chapter 3 rely on the fact that it is possible (and realistic) to compute the complete set of supported nondominated points of a relaxed problem (in the case of a weighted sum method) or chose the correct values of ε so that the computed bound sets satisfy the given definitions. It is difficult and sometimes impossible to compute bound sets for many practical problems if we rely on only these facts. For example, the number of supported nondominated points of a multi-objective problem can be exponential and so it can be impractical to compute all of them. It is also impossible to use the approaches based on the ε-constraint method if the objective coefficients are not integers. For these reasons, it is necessary to define

general methods that can be used to compute lower bound sets for bi-objective problems with non-integral objective coefficients or having an exponential number of supported nondominated points. This can be done in the short term and some investigations have already started. The main idea used in computing a lower bound set remains the same as before. That is, we first need to convert the original bi-objective integer program into single objective by using an appropriate scalarization method. Next, we solve the linear relaxation of the resulting single objective problems several times by varying the necessary parameters. The main difference now is that, we are no longer obliged to solve the single objective problem for specific values of the parameters (λ or ε) as it was before. Based on our needs and the time available, we only need to choose some few values of the parameters. A lower bound set can be defined by the series of lines linking the computed points as shown in Figure 4.8a. In this way, it is possible to combine more than one scalarization method or more than one type of formulation in computing a lower bound set for the same problem. Moreover, if we can prove that two consecutive points in a lower bound set satisfy the properties of the members of a lower bound set as defined in Chapter 3, then we can link these two points in the same way as it was done in that chapter. See Figure 4.8b for an indication of these idea. The interest of combining these pieces of ideas is to be able to define good lower bound sets in reasonably shorter computational times. It is important to note that all the strategies to search for relevant columns that were described in Chapter 3 remain valid. Some other goals we wish to achieve in the short term are to continue testing the developed column generation approaches on other problems and also develop some new approaches.

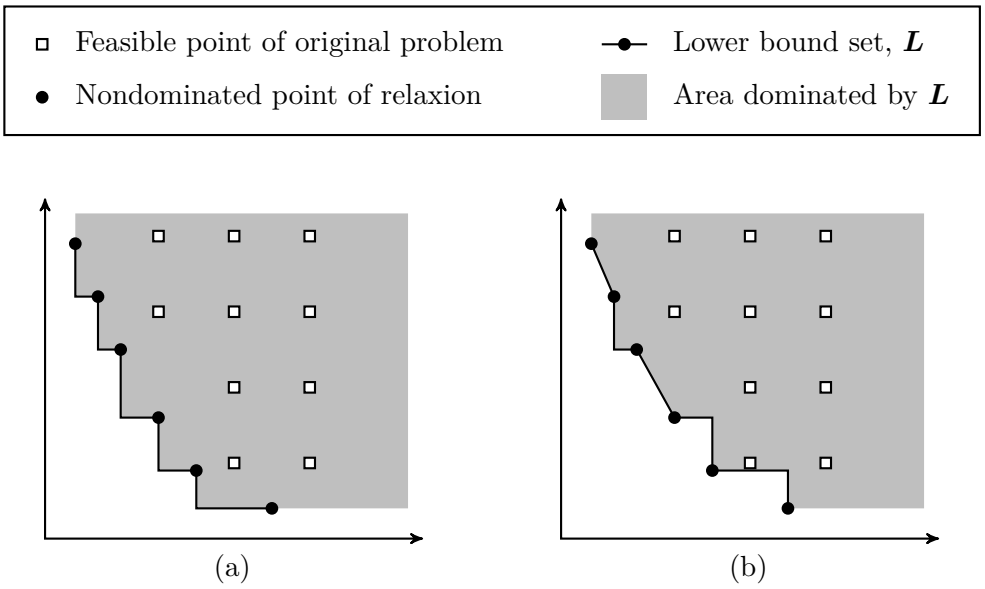


Figure 4.8: Ideas for defining more general lower bound sets.
Notes: We can compute just a few nondominated points of a relaxation and link them to define a lower bound set as in (a). We can also combine lower bound sets from different formulations as well as different scalarization methods in order to define a better and more general lower bound set as portrayed in (b).

Our main middle-term goal is to be able to incorporate the lower and upper bound sets computed by the proposed approaches in a multi-objective branch-and-bound framework in order to develop a bi-objective branch-and-price algorithm. A multi-objective branch-and-bound framework has already been developed by some researchers and so we can use it in our algorithm. Nevertheless, the design of branching rules in the context of a branch-and-price algorithm may not be very easy for a bi-objective problem. Several questions need to be answered. Can the same branching rules used for single objective problems be also used for bi-objective problems? Will they be as efficient in the bi-objective case as they are in the single objective case? How do we incorporate the constraints resulting from branching in the master problems at the nodes of the multi-objective branch-and-bound tree? We hope to be able to answer these questions and many others that will arise in the course of our work.

In the long term, we hope to extend the various ideas presented in this thesis to problems having more that two objectives. We also hope to develop and extend the mathematical theory governing the column generation method in single objective optimization to the multi-objective case. For example, it will be interesting to be able to prove when two subproblems are similar enough to have the same optimal solutions. We can also try to introduce a notion of "dominance" between subproblems by saying for example that a subproblem S_1 "dominates" another subproblem S_2 if every column that is of negative reduced cost for S_2 is also of negative reduced cost for S_1. This would be a very important result since we can then decide, for example, that there is no interest in solving S_2 directly when implementing a sequential search approach because we are sure that all relevant columns that can be found by solving S_2 can also be found by solving S_1. The work on column generation for multi-objective integer linear programs is still in its early stages and we intend to go as far as possible with our research.

Appendix A

Résumé étendu

A.1 Introduction

Les travaux de cette thèse visent à contribuer à la conception et l'étude d'algorithmes de génération de colonnes en programmation linéaire en nombres entiers multi-objectif. Pour cela nous étudions un problème de tournées de véhicules bi-objectif qui peut être considéré comme une généralisation de plusieurs autres problèmes de tournées de véhicules. Nous proposons des formulations mathématiques pour ce problème et des techniques pour accélérer le calcul des bornes inférieures par génération de colonnes. Les sous-problèmes qui doivent être résolus pour le calcul des bornes inférieures ont une structure similaire. Nous exploitons cette caractéristique pour traiter simultanément les sous-problèmes plutôt qu'indépendamment.

A.1.1 Principe de Génération de Colonnes

Afin d'obtenir de meilleures bornes qui peuvent être utilisées dans la conception de méthodes exactes, certains problèmes sont formulés avec un nombre exponentiel de variables de décision et ces problèmes sont résolus par la méthode de génération de colonnes. L'idée de la génération de colonnes pour résoudre un tel problème est de commencer par seulement un nombre raisonable des variables (et les colonnes de la matrice de contraintes correspondantes). Les autres variables sont ajouté quand c'est necéssaire.

Nous appelons un problème en nombres entiers le problème maître en nombres entiers (PME) et sa relaxation linéaire le problème maître linéaire (PML). Une restriction de PME (respectivement PML) à un sous-ensemble de colonnes est notée le PME restrient (PMER), respectivement, PMLR. Un problème auxiliaire résolu pour vérifier la convergence de la méthode est appelé un sous-problème. Une itération de la méthode commence par la résolution du PMLR afin d'obtenir un vecteur de variables duales. Le sous-problème utilise ce vecteur pour vérifier si la méthode a convergé ou non. Si la méthode n'a pas encore convergé, une ou plusieures colonnes sont proposé par le sous-problème et sont ajoutées au PMLR. Les itérations continuent jusqu'à la convergence de la méthode.

A.1.2 Optimisation Multi-Objectif

Un problème d'optimisation multi-objectif (PMO) concerne la minimisation d'un vecteur de $r \geq 2$ fonctions objectifs $F(x) = (f_1(x), \ldots, f_r(x))$ sur un domaine $\mathcal{X}$ de solutions réalisables. Le vecteur $x = (x_1, \ldots, x_n)$ est la variable de decision, $\mathcal{Y} = F(\mathcal{X})$ correspond aux images des solutions réalisables dans l'espace des objectifs, et $y = (y_1, \ldots, y_r)$, où $y_i = f_i(x)$, est un point dans l'espace objectif. Une solution x' *domine* une autre solution x'', (notée $x' \preceq x''$), si pour tous les indices $i \in \{1, \ldots, n\}$, $f_i(x') \leq f_i(x'')$ et il y a au moins un indice $i \in \{1, \ldots, n\}$ tel que $f_i(x') < f_i(x'')$. Une solution réalisable qui n'est dominée par aucune autre solution réalisable est appelée une *solution efficace* or *Pareto optimale* et son image dans l'espace objectif est appelé un *point nondominé*. L'ensemble de toutes les solutions efficaces est appelé *ensemble efficace* (noté $\mathcal{X}_E$) and l'ensemble de tous les points nondominés est appelé *ensemble nondominé* (noté $\mathcal{Y}_N$). En général, plus qu'une solution efficace peut correspondre au même point nondominé. Résoudre un PMO dans la practique correspond à trouver au moins une solution efficace correspondant à chaque point nondominé. L'ensemble nondominé décrit ce que l'on appel un *front Pareto*. Une solution efficace correspondant à un point nondominé qui se trouve sur la partie convexe du front Pareto et appelé une *solution supportée* et son image est aussi appelé un *point supporté*. Dans la pratique, les variables utilisées en optimisation multi-objectif représentent souvent des objets non fractionnables et on parle alors de problèmes multi-objectif en nombres entiers (PMOI).

Bornes d'un Problème d'Optimisation Multi-Objectif. Les bornes inférieures et supérieures d'un problème multi-objectif peuvent être décrites par des ensembles. Cette notion d'utilisation des ensembles pour définir des bornes a été introduite par Villarreal and Karwan (1981). D'autres auteurs (Delort and Spanjaard, 2010; Ehrgott and Gandibleux, 2007; Sourd and Spanjaard, 2008) ont aussi utilisés des notions de bornes basées sur des ensembles mais différentes de ce qui a été définit par Villarreal and Karwan (1981). Dans ce travail, nous utilisons une définition des bornes legèrement differente de ce qui a été proposé dans la littérature. Nous définissons une borne inférieure comme un ensemble de points (fini ou infini) $\boldsymbol{L}$ tel que l'image de tout point réalisable est soit un élément de $\boldsymbol{L}$ soit dominé par au moins un élément de $\boldsymbol{L}$. c'est-à-dire, un élément de $\boldsymbol{L}$ peut correspondre a un point réalisable. Une borne supérieure est aussi définie comme un ensemble fini de points réalisables $\boldsymbol{U}$ dont les éléments ne se dominent pas deux à deux. c'est-à-dire, un élément de $\boldsymbol{U}$ appartient nécessairement à $\mathcal{Y}$. L'idée générale utiliser dans la construction d'une borne inférieure est de convertir le problème PMO en un problème mono-objectif (par une méthode de scalarisation) et puis résoudre ce problème plusieurs fois en faisant varier les paramètres nécessaires. Pour tous les articles que nous avons trouvé dans la littérature, une scalarisation par la méthode de sommes ponderées a été utilisée. Pour construire un borne supérieure, nous utilisons en générale une heuristique ou une métaheuristique.

A.1.3 Contributions

Les contributions principaules de cette thèse sont listées ci-dessous :

- L'utilisation de la méthode ε-contraint pour construire des bornes.

- Proposition d'un algorithme général de génération de colonnes pour les problèmes linéaires en nombres entiers bi-objectif. Cet algorithme est basée soit sur une méthode de sommes pondérées soit sur une méthode ε-contraint.

- Proposition de differentes stratégies pour implementer l'algorithme de génération de colonnes général.

- Application des approches dévélopées à un problème de tournées de véhicules bi-objectif.

A.2 Génération de Colonnes pour les Problèmes Linéaires en Nombres Entiers Bi-Objectif

Considerons le problème bi-objectif P suivant :

$$\begin{aligned} & \text{Minimiser} && (c^1)^T x && \text{(A.1)} \\ & \text{Minimiser} && (c^2)^T x && \text{(A.2)} \\ \text{sous contraintes :} \quad & && Ax \geq b\,, && \text{(A.3)} \\ & && x \geq 0 \text{ et entier.} && \text{(A.4)} \end{aligned}$$

Dans cette formulation, x est un vecteur de n variables de décision et c^i est un vecteur de n coefficients entiers dans la $i^{\text{ème}}$ fonction objective ($i = 1, 2$). Les contraintes du problème sont exprimées en utilisant une matrice A de taille $m \times n$ et un vecteur de constantes b ayant m composants.

A.2.1 Construction de Bornes Inférieures

L'idée générale utilisée pour construire une borne inférieure est de convertir le problème bi-objectif en un problème mono-objectif en utilisant une méthode de scalarisation et puis résoudre ce problème (or une relaxation) plusieurs fois en faisant varier les paramètres nécessaires. Dans ce thèse nous considérons la méthode de sommes pondérées et la méthode ε-contrainte.

Le principe de la méthode de sommes pondérées est de définir un vecteur de poids non négatif $\lambda = (\lambda_1, \lambda_2)$ et de transformer le problème P en un problème mono-objectif $P(\lambda)$ donné par :

$$\begin{aligned} & \text{Minimiser} && \lambda_1 \cdot (c^1)^T x + \lambda_2 \cdot (c^2)^T x && \text{(A.5)} \\ \text{sous contraintes :} \quad & && Ax \geq b\,, && \text{(A.6)} \\ & && x \geq 0 \text{ et entier.} && \text{(A.7)} \end{aligned}$$

L'idée de la méthode ε-contrainte est de convertir une des deux fonctions objectifs en une contrainte en utilisant une constante $\varepsilon \in \mathbb{R}$. Le problème mono-objectif $P(\varepsilon)$ obtenu

après la transformation du problème P est donné par :

$$\text{Minimiser} \quad (c^1)^T x \tag{A.8}$$

$$\text{sous contraintes :} \quad Ax \geq b\,, \tag{A.9}$$

$$-(c^2)^T x \geq -\varepsilon\,, \tag{A.10}$$

$$x \geq 0 \text{ et entier.} \tag{A.11}$$

Si l'on suppose que le nombre de colonnes dans le problème A est assez petit (c'est-à-dire n est petit), alors nous n'avons pas besoin de génération de colonnes pour construire une borne inférieure pour ce problème. Ce cas a été traité par Ehrgott and Gandibleux (2007) qui ont utilisé la méthode de sommes pondérées pour la scalarisation de P. Étant donné que le problème $P(\lambda)$ est $\mathcal{NP}$-difficile dans le cas général, ils ont cherché l'ensemble de points nondominé de la relaxation linéaire de $P(\lambda)$ en utilisant une méthode proposée par Aneja and Nair (1979). Une borne inférieure est définie comme la ligne joignant les points nondominés. Pour utiliser la méthode ε-contrainte, nous devons aussi résoudre la relaxation linéaire de $P(\varepsilon)$ pour plusieurs valeurs de ε. En faisant cela, nous devons assurer que chaque point réalisable du problème P est soit généré ou dominé par au moins un des points généré (voir la figure A.1). C'est difficile à satisfaire cette condition pour un problème bi-objectif générale. Cependant si les coefficients des fonctions objectifs du problème sont entiers, nous pouvons utiliser une idée similaire à ce qui a été utilisé par Bérubé et al. (2009).

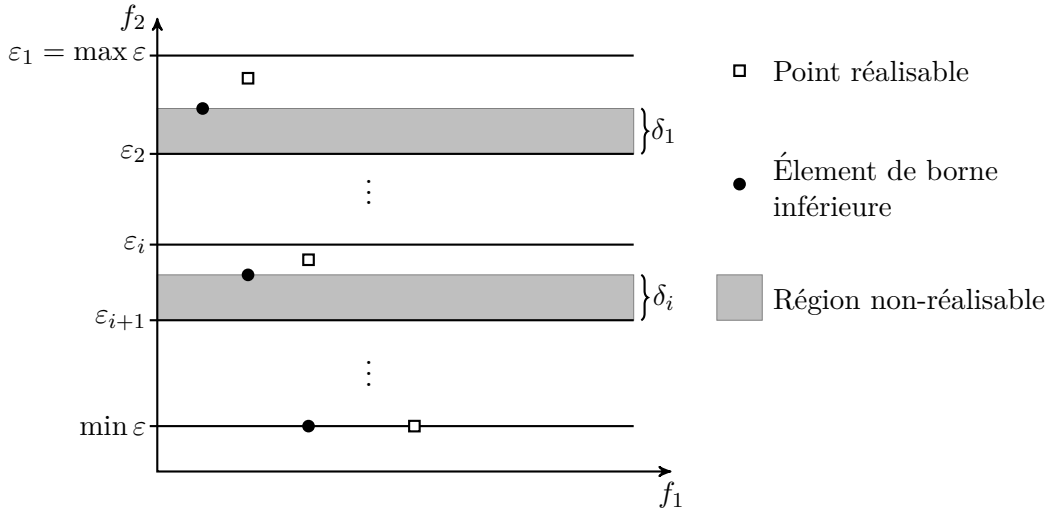


Figure A.1: Construction d'une borne inférieure par ε-contrainte.

A.2.2 Construction de Bornes Inférieures par Génération de Colonnes

Si le nombre de colonnes du problème P est trop important, nous devons utiliser une méthode de génération de colonnes pour la construction d'une borne inférieure. Nous listons maintenant les différents modèles impliqués dans l'utilisation de chacune des deux méthodes de scalarisation.

Cas de Sommes Pondérées

Problème maître Linéaire (PML(λ)):

$$\text{Minimiser} \quad (\lambda_1 c^1 + \lambda_2 c^2)^T x \tag{A.12}$$

$$\text{sous contraintes :} \quad Ax \geq b\,, \tag{A.13}$$

$$x \geq 0\,. \tag{A.14}$$

Problème dual du PML (DPML(λ)): Soit π le vecteur de variables duales associés aux contraintes (A.13). Le problème dual est donné par :

$$\text{Maximiser} \quad b^T \pi \tag{A.15}$$

$$\text{sous contraintes :} \quad A^T \pi \leq \lambda_1 c^1 + \lambda_2 c^2\,, \tag{A.16}$$

$$\pi \geq 0\,. \tag{A.17}$$

Sous-Problème (S(λ)): Il consiste à chercher au moins une variable (colonne de la matrice A) qui satisfait

$$\lambda_1 c^1 + \lambda_2 c^2 - A^T \pi < 0\,. \tag{A.18}$$

Cas de ε-contrainte

Problème maître Linéaire (PML(ε)):

$$\text{Minimiser} \quad c^1 x \tag{A.19}$$

$$\text{sous contraintes :} \quad Ax \geq b\,, \tag{A.20}$$

$$-c^2 x \geq -\varepsilon\,, \tag{A.21}$$

$$x \geq 0\,. \tag{A.22}$$

Problème dual du PML (DPML(ε)): Soit π le vecteur de variables duales associés aux contraintes (A.20) et φ la variable duale associée à la contrainte (A.21). Le problème dual est donné par :

$$\text{Maximiser} \quad b^T \pi - \varepsilon \varphi \tag{A.23}$$

$$\text{sous contraintes :} \quad A^T \pi - \varphi c^2 \leq c^1\,, \tag{A.24}$$

$$\pi,\ \varphi \geq 0\,. \tag{A.25}$$

Sous-Problème (S(ε)): Il consiste à chercher au moins une variable (colonne de la matrice A) qui satisfait

$$c^1 + \varphi c^2 - A^T \pi < 0\,. \tag{A.26}$$

Forme Générale du Sous-Problème

Une inspection des inéqualités (A.18) and (A.26) révèle que les sous-problèmes dans les deux cas ont une forme similaire. Ceci implique que des stratégies de résolution basée sur une des méthodes de scalarisation peuvent être adaptées pour l'autre méthode de scalarisation. Par ailleurs, c'est possible de traiter plus d'un sous-problème à la fois pendant la recherche de colonnes.

A.2.3 Un Algorithme Généralisé de Génération de Colonnes Pour les Problèmes Linéaires en Nombres Entiers Bi-Objectif

Grace aux similarités des sous-problèmes résolus pendant la construction d'une borne inférieure, nous proposons un algorithme général de génération de colonnes pour les problèmes linéaires en nombres entiers bi-objectif. Cet algorithme est illustré dans l'organigramme de la figure A.2 et pour implémenter, différentes possibilités se présentent. Nous présentons maintenant quelques une de ces possibilités que nous appelons les approches or stratégies de recherche de colonnes. Dans les descriptions de ces approches, nous supposons que la méthode ε-contrainte est utilisée pour la scalarisation du problème bi-objectif.

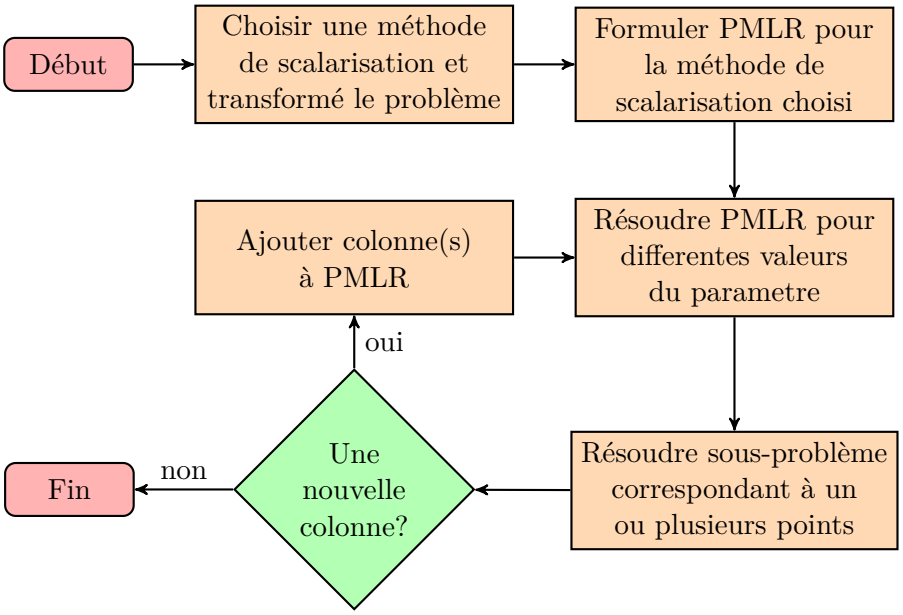


Figure A.2: Un algorithme généralisé de génération de colonnes.

Approche Point par Point (PPS)

Une implémentation très simple et intuitive de l'Algorithme A.2 consiste à résoudre le PMLR complètement pour un ε donné avant de changer la valeur de ε. c'est-à-dire, nous fixons une valeur de ε et puis résoudre PMLR(ε) par génération de colonnes jusqu'à convergence avant de changer la valeur de ε. Nous appelons cette approche Point par Point (PPS) puisque nous nous concentrons sur la recherche d'un point de la borne inférieure avant de chercher un autre point. Même si cette approche est simple et facile à implémenter, elle ne prend pas en compte les similarités des sous-problèmes associés au différentes valeurs de ε. En outre, la convergence de la méthode pour une valeur de ε peut être très lente mais cette approche ne nous donne pas la possibilité de changer le ε avant la convergence.

Approche k-Step PPS (k-PPS)

Cette approche que nous appelons k-Step PPS (k-PPS) est une variante de PPS qui donne la possibilité de changer la valeur de ε avant la convergence de PMLR(ε) pour cette valeur. Si nécessaire, l'approche va retourner aux valeurs de ε qui n'ont pas convergé avant. L'approche nécessite la définition d'une condition sous laquelle nous pouvons changer la valeur de ε avant la convergence de PMLR pour cette valeur. La condition peut être de changer la valeur de ε si PMLR ne converge pas après k itérations de la méthode. La condition que nous avons utilisé dans cette thèse est de changer la valeur de ε si la valeur de la fonction objective de RLPM ne s'améliore pas significativement après k itérations. La difficulté principale de cette approche est la définition d'une bonne condition en début de l'approche quand nous n'avons pas assez d'information sur le problème considéré.

Approche Improved PPS (IPPS)

Cette approche est basée sur l'idée que nous pouvons utiliser des heuristiques pour générer plus de colonnes après avoir trouvé quelques colonnes par un algorithme. Nous nous intéréssons plus particulièrement des heuristiques qui peuvent exploiter les similarités des sous-problèmes. C'est-à-dire, après avoir trouvé quelques colonnes par un algorithme conçu pour résoudre le sous-problème nous cherchons à modifier ces colonnes pour trouver d'autres. Une colonne trouvée par une telle heuristique est pertinente pour le sous-problème actuel et peut être aussi pertinente (mais sans aucune garantie) pour un autre sous-problème. Comme l'heuristique dépend du problème considéré, nous montrons l'idée de cette approche sur un problème spécifique dans la Section A.3.

Approche Séquentielle (Sequential)

L'idée d'une approche séquentielle est de travailler sur un ensemble de points plutôt que se concentrer sur un seul point à la fois. Pour cela, l'approche commence par résoudre PMLR(ε) pour différentes valeurs de ε sans générer aucune colonne. Ensuite, nous cherchons un ensemble de colonnes tel que chaqu'un des points générés précédemment a au moins une colonne pertinente dans l'ensemble. Nous résolvons les sous-problème correspondant au points un après l'autre. Avant de résoudre le sous-problème correspondant à un point, nous vérifions d'abord si nous avons déjà trouvé une colonne pertinente pour ce point. Si c'est le cas, nous sautons le point sans résoudre le sous-problème correspondant. Nous trouvons donc à chaque itération, un ensemble de colonnes qui est pertinent pour tous les points générés dans cette intération sans nécessairement résoudre pour tous les sous-problèmes correspondants.

Approche Solve-Once-Generate-for-All (SOGA)

Cette approche est une variante de l'approche séquentielle. Juste comme l'approche IPPS, SOGA reste sur l'idée que l'on peut utilisé des heuristiques dans la recherche des colonnes. La première étape consiste à résoudre PMLR(ε) pour différentes valeurs de ε sans générer des colonnes. Dans la deuxième étape, le sous-problème correspondant à un (seul) point est résolu pour chercher un ensemble de colonnes initial. Ensuite, des heuristiques sont utilisées pour modifier les colonnes trouvé afin de chercher d'autres colonnes pour les autres

points. La différence de cette approche par rapport à IPPS est que les valeurs duales des autres points sont utilisées dans les heuristiques. Cela nous donne la guarantie qu'une colonne trouvé par une heuristique est bien pertinente pour le point dont nous avons utilisé les valeurs duals. Nous montrons l'idée de cette approche sur un problème spécifique dans la Section A.3

A.2.4 Génération de Colonnes pour les PLNE Bi-Objectif ayant une Fonction Objectif Min-Max

Nous présentons un cas particulier d'un problème linéaire en nombres entiers (PLNE) pour lequel une des fonctions objectifs est une fonction min-max. Nous notons un tel problème PLNEBOMM. Nous considérons les problèmes de la forme :

$$\text{Minimiser} \quad \sum_{k\in\Omega} c_k\theta_k \tag{A.27}$$

$$\text{Minimiser} \quad \Gamma_{\max} \tag{A.28}$$

$$\text{sous contraintes :} \quad \sum_{k\in\Omega} a_{ik}\theta_k \;\geq\; b_i \qquad (i \in I)\,, \tag{A.29}$$

$$\Gamma_{\max} \;\geq\; \sigma_k\theta_k \qquad (k \in \Omega)\,, \tag{A.30}$$

$$\theta_k \;\in\; \{0,1\} \qquad (k \in \Omega)\,. \tag{A.31}$$

Dans cette formulation, l'ensemble de toutes les colonnes réalisables est notée Ω et I est l'ensemble des indices pour les contraintes. Deux valeurs c_k et σ_k sont associées à une colonne $k \in \Omega$. Le problème consiste à sélectionner un sous-ensemble des colonnes dans le but de minimiser la somme des c_k et aussi minimiser la valeur maximum de σ_k associée à une colonne sélectionnée.

reformulation de PLNEBOMM

Au lieu d'ajouter une contrainte de type $\Gamma_{\max} \leq \varepsilon$ à la formulation, nous proposons une reformulation du problème. Nous définissons une extension $\overline{\Omega}$ de Ω où la validité d'une colonne $k \in \overline{\Omega}$ dépend de la valeur de σ_k. Nous obtenons la formulation :

$$\text{Minimiser} \quad \sum_{k\in\overline{\Omega}} c_k\theta_k \tag{A.32}$$

$$\text{sous contraintes :} \quad \sum_{k\in\overline{\Omega}} a_{ik}\theta_k \;\geq\; b_i \qquad (i \in I)\,, \tag{A.33}$$

$$\theta_k \;\in\; \{0,1\} \qquad (k \in \overline{\Omega})\,. \tag{A.34}$$

Avant de résoudre le problème pour une valeur de ε, nous fixons $\theta_k = 0$ pour toutes colonnes $k \in \overline{\Omega}$ ayant $\sigma_k > \varepsilon$. Nous pouvons utiliser toutes les approches de recherches de colonnes déjà présentées pendant la construction d'une borne inférieure par génération de colonnes. Nous devons aussi assurer que les colonnes générés pendant la résolution de sous-problèmes respectent la contrainte liée à ε.

A.3 Problème de Tournée Couvrante Bi-Objectif à Plusieurs Véhicules

Dans cette section, nous appliquons les idées présentées dans les sections précedentes pour construire des bornes inférieures pour une extension bi-objectif du problème de tournées couvrantes (PTC) (Gendreau et al., 1997). Nous appelons cette extension le problème de tournée couvrante bi-objectif à plusieurs véhicules (PTCBOP).

A.3.1 Description du Problème

Le PTC consiste à construire une tournée unique sur un sous-ensemble des points dans le but de minimiser la longueur de la tournée. De plus, chaque point non visité par la tournée doit être dans un rayon prédéterminé (appelé la *distance de couverture*) d'un point visité. Des nombreuses applications du PTC sont citées dans la littérature (Current and Schilling, 1994; Labbé and Laporte, 1986; Current and Schilling, 1994; Hodgson et al., 1998). Une généralisation bi-objectif (Jozefowiez et al., 2007) et une extension à plusieurs véhicule (Hachicha et al., 2000) du PTC ont été proposées dans la littérature. Dans la version bi-objectif, la distance de couverture n'est pas prédéterminée mais plutôt engendrée par la tournée. Il est calculé par affecter chaque point non visité au point visité le plus proche et prendre le maximum des distance d'affectation. Les objectifs sont les minimisations de la longeur de la tournée et la distance de couverture engendrée par la tournée. Dans la version avec plusieurs véhicules, nous cherchons à minimiser la longeur totale d'un ensemble de tournées en vue d'une distance de couverture prédéterminée. Toutes les tournées commence et se terminent au même point. De plus, le nombre de points visité et la longueur d'une tournée sont limités à p et q, respectivement.

Le PTCBOP présenté ici combine les caractéristiques de la version bi-objectif et de la version à plusieurs véhicules. Le problème est définie sur un graphe $G = (V \cup W, E)$ où $V \cup W$ est l'ensemble des nœuds et E est l'ensemble des arcs. Les nœuds de V représentent des points qui peuvent être visités par une tournée et ceux de W doivent être affectés aux points de V visités par les tournées. Les nœuds $T \subseteq V$ doivent être visités par une tournée. En particulier, $v_0 \in T$ est le dépôt d'où une tournée doit commencer et se terminer. Une matrice de distance $D = (d_{ij})$ qui respecte l'inégalité triangulaire est définie sur l'ensemble E. Le PTCBOP consiste à construire un ensemble de tournées sur un sous-ensemble de V qui doit inclure tous les nœuds de T. Les deux objectifs sont la minimisation de la longeur total des tournées et la minimisation de la distance de couverture engendrée par ces tournées.

A.3.2 Formulation 1

Cette formulation est basée sur l'idée de ε-contrainte standard. Soit Ω l'ensemble de toutes colonnes réalisables. Une colonne réalisable est une tournée définie par un cycle Halmitonien sur un sous ensemble de V qui contient le dépôt v_0. De plus, le nombre de nœuds de $V \backslash \{v_0\}$ visités par la tournée et la longueur de la tournée ne doivent pas dépasser p et q, respectivement. Le coût d'une colonne $k \in \Omega$ (noté c_k) est la longueur de la tournée qu'elle représente. Nous utilisons une variable binaire θ_k pour determiner si une colonne est sélectionnée dans la solution ($\theta_k = 1$) ou non ($\theta_k = 0$). Nous notons $a_{ik} = 1$ si la tournée

de la colonne k visite le nœud $v_i \in V\backslash\{v_0\}$ et $a_{ik} = 0$ sinon. Une variable z_{ij} est utilisée pour indiquer si un nœud $w_j \in W$ est affecté à $v_i \in V\backslash\{v_0\}$ dans la solution ($z_{ij} = 1$) ou non ($z_{ij} = 0$). La distance de couverture engendrée par un ensemble de tournées est notée $\Gamma_{\max}$. Avec ces notation, le problème est défini par :

$$\text{Minimiser} \quad \sum_{k\in\Omega} c_k\theta_k \tag{A.35}$$

$$\text{Minimiser} \quad \Gamma_{\max} \tag{A.36}$$

sous contraintes :

$$\Gamma_{\max} - d_{ij}z_{ij} \geq 0 \quad (v_i \in V\backslash\{v_0\}, w_j \in W)\ , \tag{A.37}$$

$$\sum_{v_i\in V\backslash\{v_0\}} z_{ij} \geq 1 \quad (w_j \in W)\ , \tag{A.38}$$

$$\sum_{k\in\Omega} a_{ik}\theta_k - z_{ij} \geq 0 \quad (v_i \in V\backslash\{v_0\}, w_j \in W)\ , \tag{A.39}$$

$$\sum_{k\in\Omega} a_{ik}\theta_k \geq 1 \quad (v_i \in T\backslash\{v_0\})\ , \tag{A.40}$$

$$\Gamma_{\max} \geq 0\ , \tag{A.41}$$

$$z_{ij} \in \{0, 1\} \quad (v_i \in V\backslash\{v_0\}, w_j \in W)\ , \tag{A.42}$$

$$\theta_k \in \mathbb{N} \quad (k \in \Omega)\ . \tag{A.43}$$

Les objectifs pour minimiser la somme des longeurs des tournées et la distance de couverture engendrée par les tournées sont donnés dans (A.35) et (A.36), respectivement. Les contraintes (A.37) indiquent que la valeur de la distance de couverture engendrée par les tournées doit réspecter les distances des affectations. Les contraintes (A.38) et (A.39) indiquent que chaque nœud de W doit être affecté à au moins un nœud de $V\backslash\{v_0\}$ qui est visité par une tournée sélectionnée. Le fait que chaque nœud de $T\backslash\{v_0\}$ doit être visité par une tournée dans la solution est exprimé par les contraintes (A.40). Les contraintes (A.41 – A.43) expriment les domaines des variables de décision.

En suivant les discussions dans les sections précédentes, nous pouvons définir les différents modèles necéssaires pour appliquer la génération de colonnes basée sur cette formulation. Le sous-problème correspondant est donné par :

$$\text{Minimiser} \quad c_k \ - \sum_{\substack{v_i\in V\backslash\{v_0\}\\ w_j\in W}} a_{ik}\alpha_{ij} \ - \sum_{v_i\in T\backslash\{v_0\}} a_{ik}\pi_i \ < \ 0 \quad \text{pour} \quad k \in \Omega$$

où α_{ij} et π_i sont des variables duales. Ceci est un problème de plus court chemin élémentaire sous contraintes de ressources. Nous résolvons ce problème par un algorithme (DSSR) proposé dans la littérature (Righini and Salani, 2008; Boland et al., 2006).

A.3.3 Formulation 2

Cette formulation est basée sur l'idée de reformulation d'un problème PLNEBOMM. Nous définissons l'ensemble de colonnes réalisable $\overline{\Omega}$ comme toute combinaison d'une tournée réalisable R_k et d'un sous-ensemble $\Psi_k \subseteq W$. Pour une colonne $k \in \overline{\Omega}$, chaque nœud de Ψ_k doit être affecté au nœud de $R_k\backslash\{v_0\}$ le plus proche. La longueur d'une tournée R_k est notée c_k et le maximum des affectations est noté σ_k. Soit $a_{ik} = 1$ si la tournée R_k visite

le nœud $v_i \in V$. Nous notons $b_{jk} = 1$ si $w_j \in \Psi_k$ et $b_{jk} = 0$ sinon. Les variables $\Gamma_{\max}$ et θ_k ont les même significations que dans la première formulation. Le PTCBOP peut être décrit par la formulation suivante.

$$\text{Minimiser} \quad \sum_{k \in \overline{\Omega}} c_k \theta_k \tag{A.44}$$

$$\text{sous contraintes :} \quad \sum_{k \in \overline{\Omega}} a_{ik} \theta_k \ \geq \ 1 \qquad (v_i \in T \backslash \{v_0\}) \ , \tag{A.45}$$

$$\sum_{k \in \overline{\Omega}} b_{jk} \theta_k \ \geq \ 1 \qquad (w_j \in W) \ , \tag{A.46}$$

$$\theta_k \ \in \ \{0, 1\} \quad (k \in \overline{\Omega}) \ . \tag{A.47}$$

L'objectif pour minimiser la somme des longueurs des tournées est donné dans (A.44). Les contraintes (A.45) assurent que chaque nœud de $T \backslash \{v_0\}$ est visité par au moins une tournée dans la solution. Le fait qu'un nœud de W doit appartenir à un Ψ_k pour une colonne k dans la solution est exprimé par les contraintes (A.46). Le second objectif pour minimiser la distance de couverture engendrée par les colonnes sélectionnées n'apparaît pas dans cette formulation grace à la redéfinition d'une colonne.

En utilisant des variables duales π_i et β_j nous définissons le sous-problème correspondant à cette formulation par $S(\varepsilon)$:

$$\text{Minimiser} \quad c_k - \sum_{v_i \in T \backslash \{v_0\}} a_{ik} \pi_i - \sum_{w_i \in W} b_{ik} \beta_j \ < \ 0 \quad \text{pour} \quad k \in \overline{\Omega} \quad \text{et} \quad \sigma_k \leq \varepsilon \, .$$

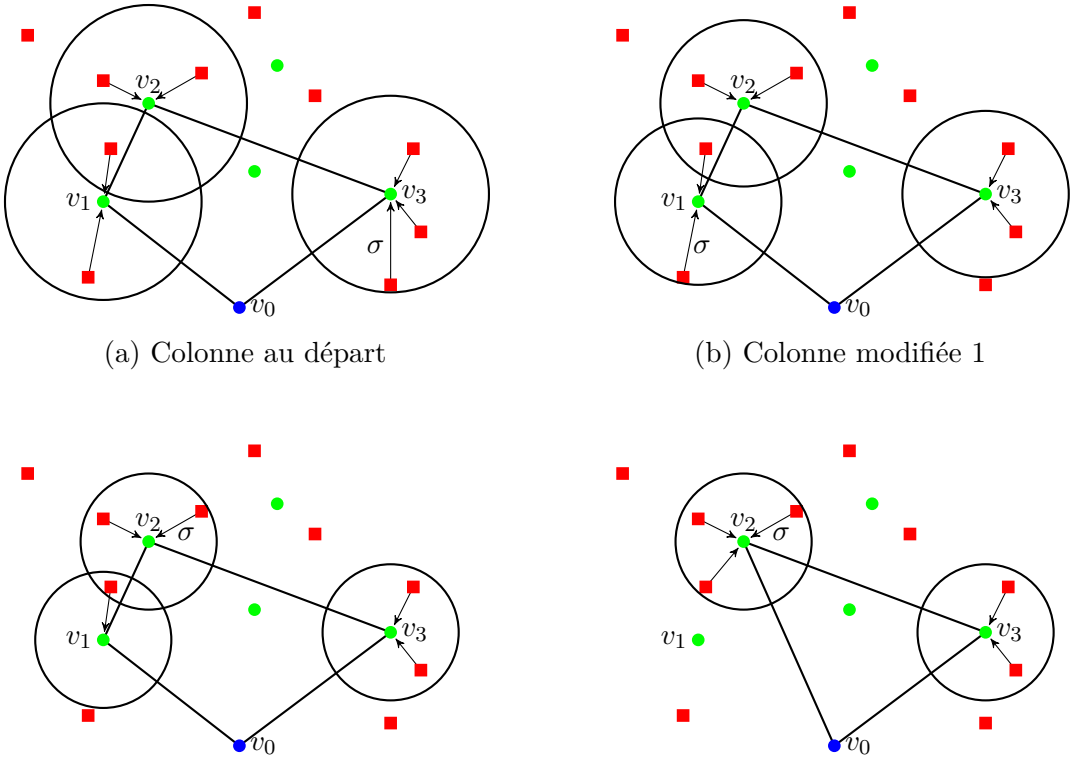
Ce problème est un plus court chemin élémentaire (non additif) sous contraintes de ressources. Nous résolvons ce problème par un algorithme modifié de DSSR (Righini and Salani, 2008; Boland et al., 2006). La modification concerne la redéfinition de la règle de dominance entre labels pour prendre en compte la caractéristique de non additivité.

Implementation de IPPS

Une colonne k' est modifiée successivement par l'enlèvement du nœud de Ψ'_k sur laquel la valeur de σ'_k est basée afin de creer une autre colonne $k'' := (R''_k, \Psi''_k)$ tel que $\sigma''_k < \sigma'_k$. C'est idée est illustrée dans la figure A.3. La pertinence de k'' est determinée par rapport aux mêmes vecteurs de valeurs duals que ceux utilisés pour trouver k'. Par conséquence, si la nouvelle colonne k'' est pertinente, alors elle l'est pour le sous-problème actuel $S(\varepsilon')$ mais pas nécessairement pour un autre sous-problème. En général, $c''_k = c'_k$ comme dans la figure A.3c mais dans certains cas un nœud de Ψ''_k peut être réaffecté à un autre nœud de $v_i \in R''_k$ afin de réduire la longueur de R''_k sans changer la valeur de σ''_k associé (voir la figure A.3d).

Implementation de SOGA

Supposons que π'' et β'' sont les vecteurs de valeurs duales associés à PMLR(ε'') où $\varepsilon'' \neq \varepsilon'$. En général, $\pi'' \neq \pi'$ et $\beta'' \neq \beta'$. Cette heuristique modifie une colonne au départ $k' := (R'_k, \Psi'_k)$ pour obtenir une autre colonne $k'' := (R''_k, \Psi''_k)$ par une reconstruction de l'ensemble Ψ''_k. Le principe de cette heurisque est illustrée dans la figure A.4. L'ensemble



(a) Colonne au départ

(b) Colonne modifiée 1

(c) Colonne modifiée 2

(d) Colonne modifiée 2 (améliorée)

Figure A.3: Heuristique IPPS pour PTCBOP.

Ψ''_k est reconstruit en prenant tous les nœuds de $w_j \in W$ dans un rayon ε d'un noeud $v_i \in R''_k \backslash \{v_0\}$ et pour lequel $\beta_j > 0$. De cette façon, si la nouvelle colonne k'' est pertinente alors elle l'est pour le sous-problème $S(\varepsilon'')$. Juste comme dans le cas de IPPS nous pouvons réaffecter un nœud de Ψ''_k à un autre nœud de $v_i \in R''_k$ dans le but de réduire la longueur de R''_k sans changer la valeur de σ''_k associée (voir la figure A.4d).

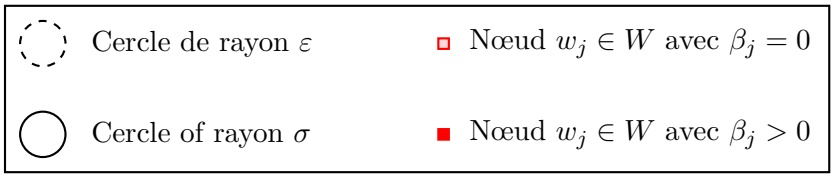
A.4 Résultats des expériences

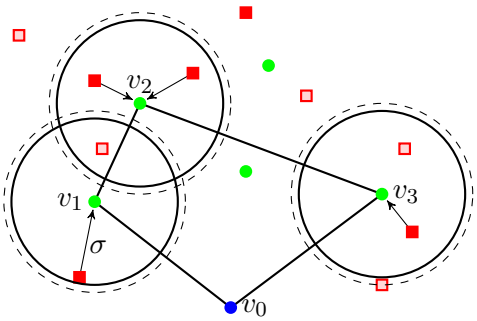
Afin d'évaluer l'efficacité des approches proposées, nous avons éffectué plusieures expériences et nous présentons les résultats dans cette section.

Nous avons utilisés deux métriques (μ_1 et μ_2) pour évaluer les qualités des bornes inférieures obtenues. Ces métriques ont été proposé dans Ehrgott and Gandibleux (2007). La figure A.5 montre comment ces deux valeurs sont calculées dans le cas d'une méthode de somme pondérée et d'une méthode ε-contrainte. Les formules utilisées sont :

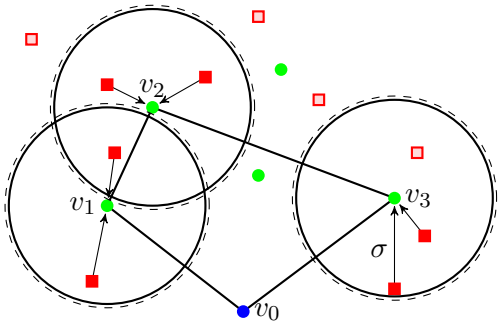
$$\mu_1 := \frac{d(\boldsymbol{L}, \boldsymbol{U})}{\|y^{\max} - y^{\min}\|_2} \qquad \text{et} \qquad \mu_2 := \frac{A_{\boldsymbol{L}} - A_{\boldsymbol{U}}}{A_{\boldsymbol{L}}},$$

où $d(\boldsymbol{L}, \boldsymbol{U})$ est la maximum des distances entre un élément de $\boldsymbol{U}$ et le point de $\boldsymbol{L}$ le plus proche. Les valuers de $\boldsymbol{L}$ et $\boldsymbol{U}$ representent les régions dominées par les ensembles $\boldsymbol{L}$ et $\boldsymbol{U}$,

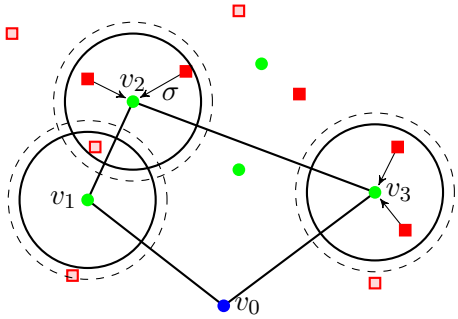




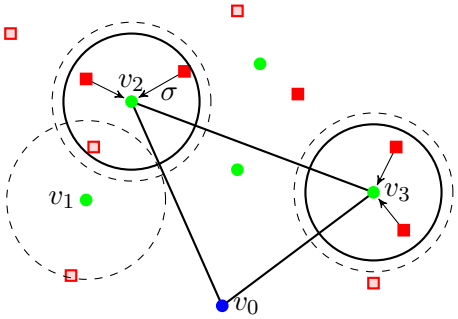
(a) Colonne pour $S(\varepsilon')$.



(b) Colonne pour $S(\varepsilon'')$ où $\varepsilon'' > \varepsilon'$.



(c) Colonne pour $S(\varepsilon'')$ où $\varepsilon'' < \varepsilon'$.



(d) Amélioration de la colonne dans (c).

Figure A.4: Heuristique SOGA pour PTCBOP.

respectivement. Si les valeurs de ces deux métriques sont très petites, alors les bornes dont on parle sont de bonne qualité.

A.4.1 Comparison de Deux Méthodes de Scalarisation

Nous avons effectué des expériences dans le but de comparer les qualités des bornes inférieures obtenues par chacune des deux méthodes de scalarisation. Le problème utilisé pour ces expériences est le problème d'ensemble couvrant bi-objectif (BOSCP) qui est défini par :

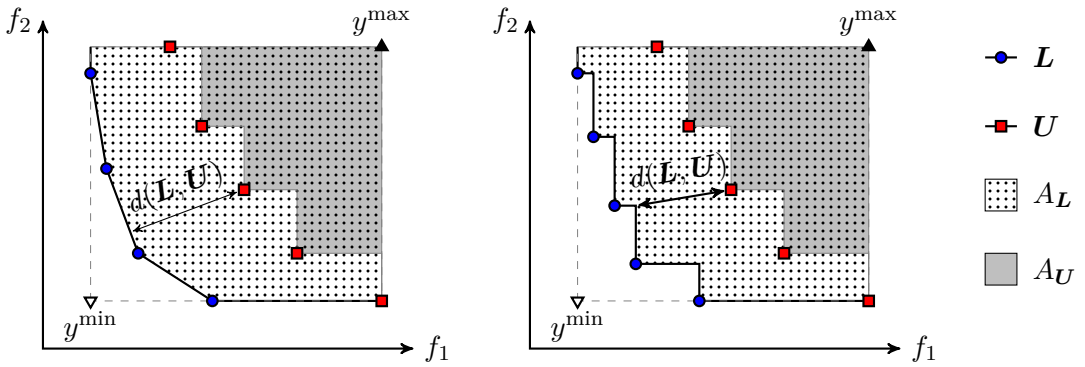


Figure A.5: Calculation des métriques d'évaluation

$$\text{Minimiser} \quad \sum_{j=1}^{n} c_j^1 x_j \tag{A.48}$$

$$\text{Minimiser} \quad \sum_{j=1}^{n} c_j^2 x_j \tag{A.49}$$

$$\text{sous contraintes :} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1, 2, \ldots, m \,, \tag{A.50}$$

$$x_j \in \{0, 1\} \quad j = 1, 2, \ldots, n \,. \tag{A.51}$$

Les instances utilisées sont les mêmes que celles utilisées dans Ehrgott and Gandibleux (2007). Pour chaque instance, une borne inférieure est construite par chacune des deux méhtodes de scalarisation et une borne supérieure est obtenue par un algorithme exact proposé par Bérubé et al. (2009). Puisque ces instances sont de tailles raisonables, nous n'avions pas besoin de génération de colonnes pour la construction des bornes.

Les résultats obtenus montrent que les bornes inférieures obtenues par les deux méthodes de scalarisations ont de qualités similaires et ces qualités étaient bonnes pour la plupart des instances. Le temps d'exécution dans le cas de ε-contrainte est plus long que celui dans le cas de sommes pondérée. Ceci est expliqué par le fait que le nombre de points necéssaires pour la définition d'une borne inférieure dans le cas de ε-contrainte est plus nombreux que dans le cas de sommes pondérées.

A.4.2 Résultats pour PTCBOP

Les instances utilisées pour ces expériences ont été générées aléatoirement en suivant les conseils dans la littérature. Les ensembles des nœuds sont obtenus en générant $|V| + |W|$ points dans $[0, 100] \times [0, 100]$. L'ensemble T (respectivement, V) est défini par les $|T|$ (respectivement, $|V|$) premiers points. Cinq instances pour différentes combinaisons de $|V| \in \{30, 40, 50\}$ and $|W| \in \{2|V|, 3|V|\}$ ont été générées. des valeurs de $|T|$ dans $\{1, \lceil 0.25|V| \rceil, \lceil 0.50|V| \rceil\}$ et p dans $\{5, 8\}$ ont été testées.

Résultats pour Formulation 1

Les approches PPS, k-PPS et Séquentielle ont été testées sur cette formulation. Les résultats obtenus ne montrent pas de différence en termes de qualité des bornes générées par les différentes approches. En termes de temps d'exécution (voir la figure A.6), l'approche séquentielle était la plus performante suivie par k-PPS et puis PPS. L'axe vertical de figure A.6 indique le pair de valeurs $|V|\_|T|$, et le temps d'exécution est indiqué sur l'axe horizontal.

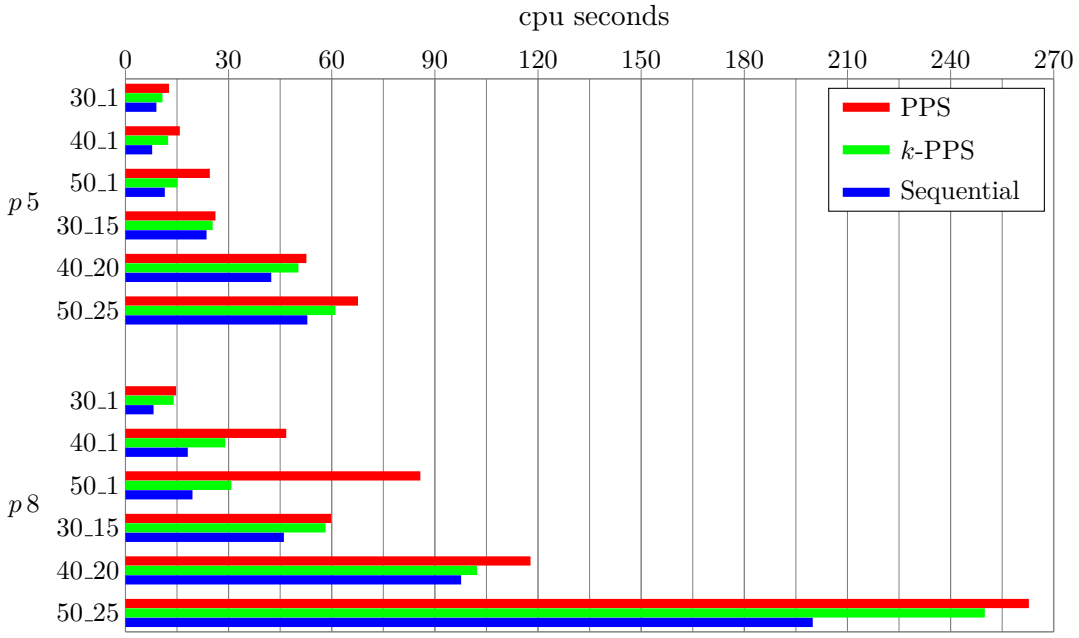


Figure A.6: Formulation 1 : Temps d'exécutions

Résultats pour Formulation 2

Les approches PPS, IPPS et SOGA ont été testées sur la deuxième formulation. Juste comme avant, les qualités des bornes obtenues par les différentes approches ne sont pas très différentes. La figure A.7 illustre le temps d'exécution pour les trois approches. Nous voyons clairement que l'approche SOGA était la plus performante suivie par IPPS et puis PPS.

Comparison des Deux Formulations

Afin de comparer les qualités de la relaxation linéaire des deux formulations nous avons évalué les bornes inférieures obtenues par PPS pour chacune des formulations vis-à-vis des mêmes bornes supérieures. La figure A.8 illustre les bornes obtenues par les deux formulations pour une instance. Nous avons eu des résultats similaires pour toutes les instances testées. Ces résultats montrent que la relaxation linéaire de la deuxième formulation est largement meilleure que celle de la première formulation. La formulation

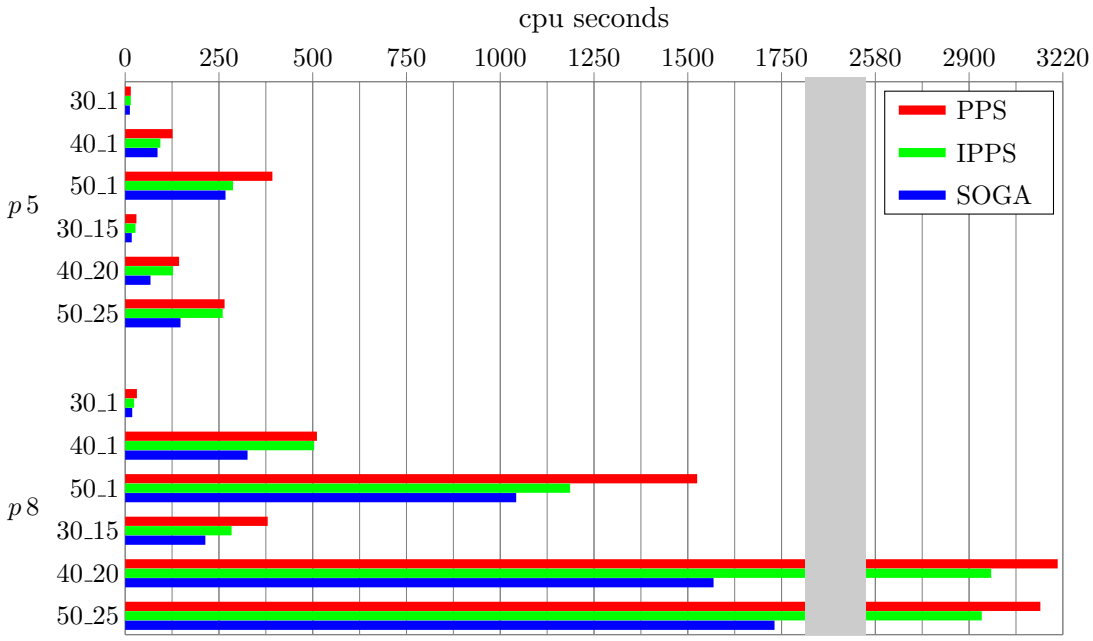


Figure A.7: Formulation 2 : Temps d'exécutions

2 a aussi obtenue des bornes supérieures qui sont meilleures que celles obtenue par la formulation 1.

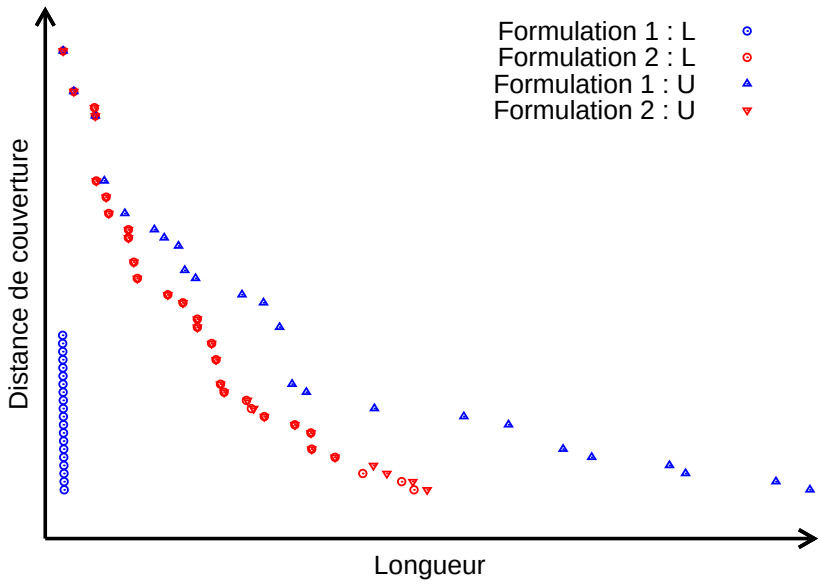


Figure A.8: Bornes pour une instance de type $|T| = 1$, $|V| = 50$, $|W| = 150$, $p = 5$, et $q = \infty$.

A.5 Conclusions et Perspectives

La génération de colonnes et l'optimisation multi-objectif sont deux domaines utiles. En dépit de ce fait, ces deux domaines sont rarement étudiés ensemble. Dans cette thèse nous avons étudié la conception d'algorithmes efficaces de génération de colonnes pour les problèmes linéaires en nombres entiers bi-objectif. Les résultats obtenus des expériences montrent l'éfficacité des méthodes et approches proposées. Une perspective à court terme consiste en la généralisation de l'approche de calcul de bornes pour des problèmes de structures moins contraintes. A moyen terme, nous visons d'incorporer les approches de calcul des bornes dans une méthode de Branch-and-Price. Une perspective à long terme consiste en l'extension de ces approches pour des problèmes à plus de deux objectifs.

Bibliography

Y. P. Aneja and K. Nair. Bicriteria transportation problem. *Management Science*, 25(1): 73–78, 1979.

L. H. Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3(1):53–68, 1969.

L. H. Appelgren. Integer programming methods for a vessel scheduling problem. *Transportation Science*, 5(1):64–78, 1971.

M. L. Balinski and R. E. Quandt. On an integer program for a delivery problem. *Operations Research*, 12(2):300–304, 1964.

J. F. Bard and H. W. Purnomo. Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164(2):510–534, 2005.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelbergh, and P. H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, 46(3), 1998.

C. Bazgan, H. Hugot, and D. Vanderpooten. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279, 2009.

J. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

T. Bektaş. Formulations and Benders decomposition algorithms for multidepot salesmen problems with load balancing. *European Journal of Operational Research*, 216(1):83–93, 2012.

H. Benson. Existence of efficient solutions for vector maximization problems. *Journal of Optimization Theory and Applications*, 26(4):569–580, 1978.

J.-F. Bérubé, M. Gendreau, and J.-Y. Potvin. An exact ϵ-constraint method for bi-objective combinatorial optimization problems: Application to the Traveling Salesman Problem with Profits. *European Journal of Operational Research*, 194(1):39–50, 2009.

N. Boland, J. Dethridge, and I. Dumitrescu. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34 (1):58–68, 2006.

I. Borgulya. An algorithm for the capacitated vehicle routing problem with route balancing. *Central European Journal of Operations Research*, 16(4):331–343, 2008.

O. Briant, C. Lemarechal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical Programming*, 113 (2):299–344, 2008.

R. L. Carraway, T. L. Morin, and H. Moskowitz. Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research*, 44(1):95–104, 1990.

A. Chabrier. Vehicle Routing Problem with Elementary Shortest Path Based Column Generatioin. *Computers & Operations Research*, 33:2972–2990, 2006.

F. Chauny, L. Ratsirahonana, and G. Savard. A model and column generation algorithm for the aircraft loading problem. *Cahiers du GERAD*, 2000.

K. I. Cho and S. H. Kim. An improved interactive hybrid method for the linear multi-objective knapsack problem. *Computers & Operations Research*, 24(11):991–1003, 1997.

G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

J. L. Cohon. *Multiobjective programming and planning*, volume 140. Courier Dover Publications, 2004.

Y. Collette and P. Siarry. *Multiobjective optimization: principles and case studies*. Springer, 2004.

J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany. *New heuristics for the vehicle routing problem*. Springer, 2005.

J. R. Current and D. A. Schilling. The median tour and maximal covering tour problems: Formulations and heuristics. *European Journal of Operational Research*, 73(1):114–126, 1994.

G. B. Dantzig and J. H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, 1959.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

I. Das and J. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. *Structural optimization*, 14(1):63–69, 1997.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

C. Delort and O. Spanjaard. Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem. In *Experimental Algorithms*, pages 253–265. Springer, 2010.

G. Desaulniers, J. Desrosiers, and M. M. Solomon. Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems. In *Essays and Surveys in Metaheuristics*, volume 15 of *Operations Research/Computer Science Interfaces Series*, pages 309–324. Springer US, 2002.

G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column generation*, volume 5. Springer-Verlag New York Incorporated, 2005.

M. Desrochers, J. Desrosiers, and M. Solomon. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, 40(2):342–354, 1992.

J. Desrosiers and M. E. Lübbecke. A primer in column generation. In *Column Generation*, pages 1–32. Springer, 2005.

C. Dhaenens, J. Lemesre, and E.-G. Talbi. K-PPM: A new exact method to solve multi-objective combinatorial optimization problems. *European Journal of Operational Research*, 200(1):45–53, 2010.

M. Dror. Note on the complexity of shortest path models for column generation in VRPTW. *Operations Research*, 42(5):977–978, 1994.

O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized Column Generation. *Discrete Mathematics*, 194:229–237, 1999.

G. Dueck. New optimization heuristics. *Journal of Computational physics*, 104(1):86–92, 1993.

M. Ehrgott. *Multicriteria optimization*. Springer, 2 edition, 2005.

M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR-Spektrum*, 22(4):425–460, 2000.

M. Ehrgott and X. Gandibleux. Approximative solution methods for multiobjective combinatorial optimization. *Top*, 12(1):1–63, 2004.

M. Ehrgott and X. Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, 34(9):2674–2694, 2007.

M. Ehrgott and D. M. Ryan. The method of elastic constraints for multiobjective combinatorial optimization and its application in airline crew scheduling. *Multi-Objective Programming and Goal Programming–Theory and Applications*, pages 117–122, 2003.

M. Ehrgott and J. Tind. Column generation in integer programming with applications in multicriteria optimization. Technical report, Faculty of Engineering, University of Auckland, New Zealand., 2007.

Ö. Ergun. *New neighborhood search algorithms based on exponentially large neighborhoods.* PhD thesis, Massachusetts Institute of Technology, 2001.

J. P. Evans and R. Steuer. A revised simplex method for linear multiple objective programs. *Mathematical Programming*, 5(1):54–72, 1973.

D. Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR: A Quarterly Journal of Operations Research*, 8(4):407–424, 2010.

D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the Elementary Shortest Path Problem with Resource Constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

R. Fischer and K. Richter. Solving a multiobjective traveling salesman problem by dynamic programming. *Mathematische Operationsforschung und Statistik. Series Optimization*, 13(2):247–252, 1982.

P. C. Fishburn. Additive Utilities with Incomplete Product Sets: Application to Priorities and Assignments. *Operations Research*, 15(3):537–542, 1967.

M. P. Fourman. Compaction of symbolic layout using genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 141–153. L. Erlbaum Associates Inc., 1985.

T. Gal. Rim multiparametric linear programming. *Management Science*, 21(5):567–575, 1975.

R. J. Gallagher and O. A. Saleh. Constructing the set of efficient objective values in linear multiple objective transportation problems. *European Journal of Operational Research*, 73(1):150–163, 1994.

M. Gamache, F. Soumis, D. Villeneuve, J. Desrosiers, and E. Gelinas. The preferential bidding system at Air Canada. *Transportation Science*, 32(3):246–255, 1998.

X. Gandibleux. *Multiple criteria optimization: state of the art annotated bibliographic surveys*, volume 52. Kluwer Academic Pub, 2002.

M. Gendreau, G. Laporte, and F. Semet. The Covering Tour Problem. *Operations Research*, 45(4):568–576, 1997.

E. H. Ghaziri. Solving routing problems by a self-organizing map. 1991.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem - Part II. *Operations research*, 11(6):863–888, 1963.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

D. E. Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.

B. L. Golden, A. A. Assad, and E. A. Wasil. Routing vehicles in the real world: applications in the solid waste, beverage, food, dairy, and newspaper industries. *The vehicle routing problem*, pages 245–286, 2002.

M. H. Hà, N. Bostel, A. Langevin, and L.-M. Rousseau. An exact algorithm and a metaheuristic for the multi-vehicle covering tour problem with a constraint on the number of vertices. *European Journal of Operational Research*, 226:211–220, 2013.

M. Hachicha, M. J. Hodgson, G. Laporte, and F. Semet. Heuristics for the multi-vehicle covering tour problem. *Computers & Operations Research*, 27(1):29–42, 2000.

Y. Y. Haimes, L. S. Lasdon, and D. A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(3):296–297, 1971.

M. J. Hodgson, G. Laporte, and F. Semet. A Covering Tour Model for Planning Mobile Health Care Facilities in SuhumDistrict, Ghama. *Journal of Regional Science*, 38(4): 621–638, 1998.

J. H. Holland. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. 1975.

G. Jahanshahloo, F. Hosseinzadeh, N. Shoja, and G. Tohidi. A method for generating all efficient solutions of 0-1 multi-objective linear programming problem. *Applied mathematics and computation*, 169(2):874–886, 2005.

N. Jozefowiez. A Branch-and-Price Algorithm for the Multi-Vehicle Covering Tour Problem. Rapport LAAS 12686, LAAS-CNRS, France, 2012.

N. Jozefowiez, F. Semet, and E.-G. Talbi. Parallel and hybrid models for multi-objective optimization: Application to the vehicle routing problem. In *Parallel Problem Solving from Nature—PPSN VII*, pages 271–280. Springer, 2002.

N. Jozefowiez, F. Semet, and E.-G. Talbi. The bi-objective covering tour problem. *Computers & Operations Research*, 34(7):1929–1942, 2007.

I. Kara and T. Bektaş. Minimal load constrained vehicle routing problems. In *Computational Science–ICCS 2005*, pages 188–195. Springer, 2005.

A. Khanafer, F. Clautiaux, S. Hanafi, and E.-G. Talbi. The min-conflict packing problem. *Computers & Operations Research*, 39(9):2122–2132, 2012.

S. Kim, K.-N. Chang, and J.-Y. Lee. A descent method with linear programming subproblems for nondifferentiable convex optimization. *Mathematical programming*, 71(1):17–28, 1995.

J. Knowles and D. Corne. On metrics for comparing nondominated sets. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 711–716. IEEE, 2002.

M. Labbé and G. Laporte. *Maximizing User Convenience and Postal Service Efficiency in Post Box Location*. Cahiers du GÉRAD. CIRRELT, 1986.

G. Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics (NRL)*, 54(8):811–819, 2007.

G. Laporte and Y. Nobert. Exact algorithms for the vehicle routing problem. *Surveys in Combinatorial Optimization*, 31:147–184, 1987.

T.-R. Lee and J.-H. Ueng. A study of vehicle routing problems with load-balancing. *International Journal of Physical Distribution & Logistics Management*, 29(10):646–657, 1999.

J. Lemesre, C. Dhaenens, and E.-G. Talbi. Parallel partitioning method (PPM): A new exact method to solve bi-objective problems. *Computers & Operations Research*, 34(8): 2450–2462, 2007.

M. E. Lübbecke and J. Desrosiers. Selected Topics in Column Generation. *Operations Research*, 53(6):1007–1023, 2005.

D. Madakat, J. Morio, and D. Vanderpooten. Biobjective planning of an active debris removal mission. *Acta Astronautica*, 84:182–188, 2013.

R. E. Marsten, W. W. Hogan, and J. W. Blankenship. The Boxstep Method for Large-Scale Optimization. *Operations Research*, 23(3):389–405, 1975.

C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

W. Ogryczak. Multiple criteria linear programming model for portfolio selection. *Annals of Operations Research*, 97(1-4):143–162, 2000.

V. Pareto. Cours d'économie politique. 1896.

J. M. Pasia, K. F. Doerner, R. F. Hartl, and M. Reimann. Solving a bi-objective vehicle routing problem by pareto-ant colony optimization. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, pages 187–191. Springer, 2007.

F. Peng, X. Jia, X. Gu, M. A. Epelman, H. E. Romeijn, and S. B. Jiang. A new column-generation-based algorithm for VMAT treatment plan optimization. *Physics in Medicine and Biology*, 57(14):4569–4588, 2012.

D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

C. R. Reeves. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., 1993.

L. B. Reinhardt and D. Pisinger. Multi-objective and multi-constrained non-additive shortest path problems. *Computers & Operations Research*, 38(3):605–616, 2011.

R. Ribeiro and H. Ramalhinho Dias Lourenço. A multi-objective model for a multi-period distribution management problem. 2001.

T. Rienthong, A. Walker, and T. Bektaş. Look, here comes the library van! Optimising the timetable of the mobile library service on the Isle of Wight. *OR Insight*, 24(1):49–62, 2011.

G. Righini and M. Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006.

G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170, 2008.

Y. Rochat and É. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, 1(1):147–167, 1995.

E. Salari and J. Unkelbach. A column-generation-based method for multi-criteria direct aperture optimization. *Physics in medicine and biology*, 58(3):621–639, 2013.

W. J. Samuels. Edgeworth's Mathematical Psychics: A Centennial Notice. *Eastern Economic Journal*, 7(3/4):193–198, 1981.

B. M. Sarpong, C. Artigues, and N. Jozefowiez. The Bi-Objective Multi-Vehicle Covering Tour Problem: Formulation and Lower Bound. In *5th International Workshop on Freight Transportation and Logistics*, ODYSSEUS 2012, pages 451–454, 2012a.

B. M. Sarpong, C. Artigues, and N. Jozefowiez. The bi-objective multi-vehicle covering tour problem: formulation and lower bound by column generation. Rapport LAAS 12562, LAAS-CNRS, France, 2012b.

B. M. Sarpong, C. Artigues, and N. Jozefowiez. Column Generation for Bi-Objective Vehicle Routing Problems with a Min-Max Objective. In *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 33 of *OASICS*, pages 137–149. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013a.

B. M. Sarpong, C. Artigues, and N. Jozefowiez. Column Generation for Bi-Objective Integer Programs: Application to Bi-Objective Vehicle Routing Problems. Rapport LAAS 13234, LAAS-CNRS, France, 2013b.

B. M. Sarpong, C. Artigues, and N. Jozefowiez. Using Column Generation to Compute Lower Bound Sets for Bi-Objective Combinatorial Optimization Problems. Rapport LAAS 13437, LAAS-CNRS, France, 2013c.

M. Schumann and R. Retzko. Self Organizing Maps for Vehicle Routing Problems-minimizing an explicit cost function. In F. Fogelman-Soulie, editor, *Proceedings of the International Conference on Artificial Neural Networks*, pages 401–406. EC2, 1995.

F. Sourd and O. Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3): 472–484, 2008.

J. Sylva and A. Crema. A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *European Journal of Operational Research*, 158(1): 46–55, 2004.

P. Toth and D. Vigo. Exact solution of the vehicle routing problem. In *Fleet management and logistics*, pages 1–31. Springer, 1998.

P. Toth and D. Vigo. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2002.

E. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.

E. Ulungu and J. Teghem. Solving multi-objective knapsack problem by a branch-and-bound procedure. In *Multicriteria analysis*, pages 269–278. Springer, 1997.

F. Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 331–358. Springer, 2005.

B. Villarreal and M. H. Karwan. Multicriteria integer programming: A (hybrid) dynamic programming recursive approach. *Mathematical Programming*, 21(1):204–223, 1981.

P. L. Yu and M. Zeleny. The techniques of linear multiobjective programming. *RAIRO - Operations Research - Recherche Opérationnelle*, 8(3):51–71, 1974.

E. Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*. PhD thesis, Swiss Federal Institute of Technology, Zurich, 1999.

E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 832–842. Springer, 2004.

E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms — A comparative case study. In A. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature — PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–301. Springer Berlin Heidelberg, 1998.

E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, 2003.