2016-11
Sune Lauth Gadegaard
PhD Dissertation

Discrete Location Problems – Theory, Algorithms, and Extensions to Multiple Objectives

DEPARTMENT OF ECONOMICS AND BUSINESS ECONOMICS
AARHUS BSS • AARHUS UNIVERSITY • DENMARK

Discrete Location Problems

– Theory, algorithms, and extensions to multiple objectives

A PhD dissertation

by

Sune Lauth Gadegaard

Aarhus BSS, Aarhus University
Department of Economics and Business Economics.

April 2016

Discrete Location Problems

– Theory, algorithms, and extensions to multiple objectives

Dissertation advisors

– Lars Relund Nielsen (main)

– Andreas Klose

– Kim Allan Andersen

Sune Lauth Gadegaard
Department of Economics and Business Economics
School of Business and Social Sciences
Aarhus University
Fuglesangs Allé 4
Building 2622
DK-8210
Denmark
sgadegaard@econ.au.dk
http://pure.au.dk/portal/en/sgadegaard@econ.au.dk

Copyright © 2013-2016 Sune Lauth Gadegaard
Rules of usage: None except the standard copyright rules. If any of the work, that being theory, computational results or figures, is used or reproduced in any way, cite the work and the author(s) in appropriate manners.

Layout and typography is chosen and implemented by the author
Font: Latin modern. 12pt scaled to 95 percent
Produced with pdfLaTeX and the *memoir*–class
All figures have been made using Ti*k*Z

To Irene who has the decency not to realize she is way out of my league
and to Selma – You are my sunshine

Table of contents

Resumé

Denne ph.d.–afhandling omhandler en række abstrakte lokaliseringsproblemer og foreslår forskellige løsningsmetoder. Litteraturen inden for operationsanalyse behandler lokaliseringsproblemer ved at opstille matematiske optimeringsmodeller for stiliserede problemstillinger, hvor en eller flere *faciliteter* skal placeres i forhold til en række kunder. Det kan for eksempel være placeringen af sygehuse, som skal ske under hensyntagen til kapacitetsbegrænsninger og størrelsen på omkringliggende byer.

Afhandlingen fokuserer især på et problem kaldet *single–source capacitated facility location problem* (SSCFLP). Dette problem består i at placere en række faciliteter og forsyne kunder fra disse åbne faciliteter på en sådan måde, at de samlede omkostninger ved at åbne faciliteterne og forsyne kunderne minimeres. En brugbar løsning er en løsning, hvor hver kunde forsynes fra netop én åben facilitet. Det antages, at kundernes efterspørgsel samt faciliteternes kapaciteter er kendte, og at faciliteternes kapacitet skal respekteres. Desuden antages, at både placeringen af en facilitet og allokeringen af en kunde til en facilitet resulterer i en fast (periodisk) omkostning.

Afhandlingen bidrager med nye metoder til løsning af dette problem når både et og to kriterier optimeres samtidigt. Afhandlingen beskæftiger sig hovedsageligt med eksakte metoder, som altid finder en optimal løsning. Når flere objektfunktioner optimeres samtidigt, betyder det, at alle ikke–dominerede løsningsvektorer skal genereres. Afhandlingen bidrager med nye løsningsprocedurer for såvel et–kriterie som for flerkriterie problemer.

Første kapitel giver en historisk introduktion til lokaliseringsproblemer samt en kort gennemgang af nogle af bidragene inden for diskret flerkriterie lokaliseringsplanlægning.

Andet kapitel foreslår en algoritme, der bruger cut–and–solve til at løse SSCFLP problemet. Algoritmen er effektiv og viser sig at være op til 40 gange hurtigere end de bedste kendte algoritmer fra litteraturen.

Tredje kapitel beskriver to algoritmer som integrer cut–and–solve med en såkaldt dual-ascent algoritme. Den første algoritme benytter en variation af cut–and–solve algoritmen fra kapitel 2 til at løse en række delproblemerne i dual ascent algoritmen. Den anden algoritme løser cut–and–solve algoritmes såkaldt *sparse* problemer ved hjælp af en dual

ascent algoritme.

I de to sidste kapitler (kapitel 4 og kapitel 5) af afhandlingen, optimeres to kriterier samtidigt. I kapitel 4 studeres et såkaldt bi–kriterie "omkostning–flaskehals" lokaliseringsproblem. Dette problem minimerer to kriterier, hvor det ene består af de samlede omkostninger ved at servicere kunder samt åbne faciliteter, mens det andet minimerer den maksimale transporttid fra en kunde til den nærmeste facilitet. Der foreslås en såkaldt ε–begrænsningsmetode, som bevarer problemets underliggende struktur, til at løse dette problem. Afhandlingen afsluttes med kapitel 5, hvor der udvikles en såkaldt branch–and–cut algoritme til at løse en bi–kriterie version af SSCFLP. Metoden finder alle rationelle kompromiser mellem omkostningen ved at åbne faciliter og omkostningerne ved at servicere kunderne.

Abstract

This PhD–dissertation proposes a number of solution procedures for discrete facility location problems. In the literature of operations research, location problems are mathematical models describing optimization problems where one or more *facilities* need to be placed in relation to a given set of customers or demand points. An example is the location of hospitals which needs to be performed in such a way as to take into account capacity limits and the sizes of nearby towns and cities.

The dissertation particularly focuses on the so–called *single–source capacitated facility location problem* (SSCFLP). The problem consists in opening a set of facilities and allocating the customers' demands to these open facilities in such a way that the capacity of each facility is respected and such that each customer is serviced from exactly one open facility. An optimal solution to the SSCFLP is a solution in which the total cost of opening facilities and servicing customers is minimized. In this problem, it is assumed that all costs and the customers' demand and the capacity of the facilities are fixed and known.

The dissertation proposes new solution procedures for both single objective as well as bi–objective versions of the SSCFLP. The dissertation considers exact methods that guarantee an optimal solution. When two objective functions are optimized simultaneously, an optimal solution consists of a *set* of solutions which maps into the entire set of *non–dominated outcome vectors*.

The first chapter of the dissertation provides a historic overview of location problems and a short survey of the literature on discrete multi–objective facility location.

The second chapter proposes an improved cut–and–solve based algorithm for the SSCFLP. The algorithm is effective and turns out to be up to 40 times faster compared to state–of–the–art algorithms from the literature.

The third chapter describes two algorithms that integrates the cut–and–solve framework with a semi–Lagrangean dual ascent algorithm in order to solve large instances of the SSCFLP. The first algorithm uses a variant of the cut–and–solve algorithm proposed in Chapter 2 to solve subproblems arising in the dual ascent algorithm whereas the second solves the sparse problems of the cut–and–solve algorithm using a semi–Lagrangean based

dual ascent algorithm.

In the last two chapters (Chapter 4 and Chapter 5) of the dissertation, bi–objective location problems are considered. In Chapter 4, a so–called bi–objective "cost–bottleneck" location problems is studied. The problem minimizes two objective functions simultaneously. The first minimizes the total cost while the second minimizes the maximal transportation time from a customer to a nearest open facility. An ε–constrained algorithm is proposed which preserves the structure of the underlying location problem. In the last chapter of the dissertation, Chapter 5, branch-and-cut algorithms for bi-objective optimization are developed. The proposed algorithms rely on explicitly and implicitly given *lower bound sets* and compute all rational compromises between the cost of opening facilities and the cost incurred by servicing the customers.

Acknowledgments

These pages are intended to be the place where I have the opportunity of thanking the people who helped me during the process of completing this dissertation. This will, of course, not do justice to the fantastic people mentioned here, but I hope they know how much I appreciated being with them.

First of all, I wish to thank my main supervisor Lars Relund Nielsen for his guidance, encouragement, contributions, and suggestions during the process. He has not only helped me complete this dissertation, but has also taught me how to be an academic with all that this entails: writing a paper, using different software, attending conferences, creating courses, and navigating in the sometimes inert systems at a university. You have kept a tight rein on me, but never so tight than I could not pursue fruitful research ideas. I have, indeed, been very pleased with our collaboration.

Second, I would like to thank professor Kim Allan Andersen, my co–supervisor. Although a busy man, he never evaded any of my requests for proofreading, literature suggestions or good advice in general.

Last, but definitely not least, among my horde of supervisors is Andreas Klose, who created the spark that ignited my interest in Operations Research. Andreas has been a resourceful, inventive and extremely helpful co–author on several projects and I owe him great thanks! During the past five years Andreas has become a true mentor.

During the fall of 2014 I visited Professor Matthias Ehrgott at Lancaster University, England. For four months, I had the privilege to draw on his seemingly inexhaustible knowledge of the field of multiple objective optimization. I am very grateful for our meetings where almost all aspects of bi–objective branch–and–bound were discussed.

I would also like to thank all the members of our research group, CORAL, for many interesting and enlightening discussions, courses and (pizza) seminars. It has been a pleasure to work with all of you during the last three–and–a–half years.

Moreover, I would like to thank both present and previous colleagues here at the Department of Economics and Business Economics for always creating a pleasant working atmosphere. Especially, I would like to thank Reza Pourmoayed for sharing an office with

me through the entire journey, and for never getting annoyed with my visitors, phone calls and my perpetual talk about cake. Furthermore, I would like to thank Jeanne Andersen, Lukas Bach, Tue Christensen, Maria Elbek, Marie Herly, Lene Gilje Justesen, David Sloth Pedersen, and Camilla Pisani: These three and some years would never have been so much fun had it not been for Friday beers, summer house stays, the lunch box club, the "cake rules" decree, the (endless) amount of journeys to the mail room, and the millions of liters of coffee we have been drinking together.

Last, but definitely not least, I have to thank my family, Irene and Selma, for being a source of eternal joy and comfort. Particularly, my wonderful wife Irene has been an invaluable source of support and encouragement and deserves a standing ovation for putting up with my very absent–minded behavior when I drift into "the world of mathematics".

Sune Lauth Gadegaard
April 2016

Notation

Throughout this dissertation I have aimed at keeping a consistent use of symbols. In general, calligraphic capitals denote sets, and lower case Latin and Greek letters denote elements of sets, variables, functions, parameters, or indices. Table 1 summarizes the standardized notation for the dissertation.

Table 1: Notation

Notation	Explanation of notation
$\mathcal{X}$	Feasible set of an optimization problem.
$\mathcal{Z}$	Feasible set in objective space of a multi–objective optimization problem.
$\mathcal{I}$	Index set of potential facility sites.
$\mathcal{J}$	Index set of demand points.
y	Vector of binary location variables indicating if a facility is open at location $i \in \mathcal{I}$ ($y_i = 1$) or not ($y_i = 0$).
x	Matrix of assignment variables. x_{ij} denotes the proportion of demand point j's demand covered by facility i.
z	Vector of objective function values.
$\lambda \in \mathbb{R}^2$	Vector of objective function weights.
f_i	Fixed cost of opening a facility.
c_{ij}	Cost of assigning all of customer j's demand to facility i.
$z^1 < z^2$	$z^1_l < z^2_l$, for $l = 1, 2$.
$z^1 \leqq z^2$	$z^1_l \leq z^2_l$, for $l = 1, 2$.
$z^1 \leq z^2$	$z^1 \leqq z^2$ and $z^1 \neq z^2$.
$\mathbb{R}^2_>$	$\{y \in \mathbb{R}^2 \,:\, y > 0\}$.
$\mathbb{R}^2_{\leqq}$	$\{y \in \mathbb{R}^2 \,:\, y \leqq 0\}$.
$\mathbb{R}^2_{\geqq}$	$\{y \in \mathbb{R}^2 \,:\, y \geqq 0\}$.
S_N	$\{s \in S \,:\, (\{s\} - \mathbb{R}^2_{\leqq}) \cap S = \{s\}\}$. The non–dominated set of S.
$\mathcal{Z}_N$	Set of non–dominated outcomes of a bi–objective optimization problem.

Introduction

This dissertation considers the field of discrete facility location where the set of potential facility sites and the set of demand points or customers are finite. The main focus of the dissertation is on the solution processes: though simple to explain, many facility location problems are surprisingly hard to solve to optimality for larger instances. Therefore, a number of exact solution approaches is developed for single objective facility location problems, but also bi–objective facility location problems are considered.

I will in this chapter introduce the field of facility location problems (FLPs) via a historic overview of the literature and characterize FLPs by three underlying "topologies" and three kinds of objectives to optimize. The three most popular objectives in the literature are *cost/minisum*, *bottleneck/minimax*, and *covering*. The *cost–objective* aims at minimizing total cost (or a surrogate hereof, e.g. total distance to customers). In some problems a fixed opening cost is incurred when opening a facility and in others a fixed number of facilities with identical cost needs to be opened. Furthermore, the service cost for supplying a customer from a given open facility is often included in the total costs as well. The cost–objective focuses on efficiency. The *bottleneck–objective* seeks to minimize the maximum (weighted) distance from customers to the nearest open facility. This type of objective function is especially appropriate when locating public facilities such as fire stations and hospitals. The focus of this objective function is equity. The *covering–objective* has two variants. In both cases, a customer can be considered covered if a facility is opened within a certain exogenously given distance. In the first variant, a fixed number of facilities to open is given and the objective then maximizes the number of (weighted) customers it is possible to cover. In the second variant it is the objective to minimize the (weighted) number of facilities needed to cover the customers within that maximum distance.

The three topologies that will be used are *planar*, *network* and *discrete*. The *planar* location problems assume that facilities can be placed anywhere on the plane. Conversely, the *network* location problems only allow facilities to be placed on a network which is represented

as a graph with a set of demand nodes and a set of edges. The facilities can then be placed in the nodes as well as on the edges of the graph. Finally, *discrete* location problems assume that a finite set of potential facility sites is given, and a decision is to be made as to which of these potential facility sites is to be used. As this dissertation concentrates on the *discrete* topology, the overview in Section 1.1 will have these problems as its focus.

The rest of this introductory chapter is organized as follows: Section 1.1 provides a short overview of the history of facility location. After this historic overview, an overview of the discrete multi–objective facility location literature is given in Section 1.2 and the contributions of the present dissertation are then described in Section 1.3.

1.1 Facility location problems

Facility location problems are core problems within operations research. As the name suggests, these problems consist of determining a best location for one or more facilities in such a way that a certain set of demand points, or customers, are serviced in a satisfactory way. In order to evaluate a certain constellation of facilities and determining the best one, we require objectives or criteria and constraints on the system to be modelled. Numerous models have been proposed for a variety of different problems in the literature. We refer the reader to the survey paper by Klose and Drexl (2005) which includes a detailed overview of facility location problems used in distribution systems and states many mathematical programming formulations for a large variety of location problems.

In the following a historic overview of the field of facility location is given. The focus of the first part is on the seminal contributions in the field and the second part focuses on contributions in discrete multi–objective facility location.

1.1.1 A historic overview of facility location

The continuous location problems laid the foundation for the academic field of location science. The problem can be stated simply as follows: given a set of demand points in the plane that need to be served, determine a position in the plane for one or more facilities, such that an objective is optimized. Pierre de Fermat was probably the first person to present this problem, and in his book Methodus ad disquirendam maximam et minima (see de Fermat (1679)) it is stated as follows: given three points in the plane A, B and C, locate a fourth point D such that the sum of distances from points A, B and C to D is minimal. The point D is now known as a *Fermat point*. This problem is equivalent to calculating the geometric median of the three points. The problem was extended to a weighted version by Carl Friedrich Launhardt in Launhardt (1900). He determined the optimal location of an

industrial facility combining iron ore and coal in order to produce pig–iron. The objective was to minimize the transportation cost arising from transporting iron ore and coal to the production facility plus the transportation cost incurred by transporting the pig-iron to the final customer.

Weber (1922) popularized the problem for an arbitrary number of demand points, and this specification is now known as the Weber problem (a generalization of Launhardt's problem). It consists in finding a location for a facility in such a way that the sum of weighted distances from all demand points to the location of the facility is minimized. A fixed point method for solving the Weber problem was proposed in Weiszfeld (1937) (recently translated in Weiszfeld and Plastria (2009)).

The first natural extension of the Weber problem is the so–called multi–source Weber problem where an optimal location of $p > 1$ facilities in the plane has to be determined. A complicating feature of the multi–source Weber problem is that an optimal *allocation* of customers to facilities needs to be determined as well as the location of the facilities. This makes the multi–source Weber problem much harder than the single–Weber problem, and in fact, despite its simple nature, the problem is strictly $\mathcal{NP}$–hard (Megiddo and Supowit, 1984). Although some exact methods for the multi–Weber problem have been proposed (see for example the column generation based approach for the multi–Weber with $p = 2$ problem by Drezner (1984)), the probably most well–known algorithm is the so–called Cooper's location–allocation heuristic proposed by Cooper (1964). Some extensions of Weber problems have been proposed in the literature: Hamacher and Nickel (1994) proposed algorithms for the Weber problem where some areas of the plane are forbidden; Klamroth (2001) proposed a model where the demand points can be separated by line barriers; and Melachrinoudis (1988) solved an *obnoxious* location problem in the plane.

In many situations, it might not be possible to cover all demand points. Mehrez and Stulman (1982) therefore suggested the so–called maximal covering problem in the plane, where a fixed number of possible facilities is given as input, and a solution consists of a set of facilities covering as many (weighted) demand points as possible. A demand point is considered covered if it lies within a given radius of an open facility.

In 1964, Hakimi published the seminal paper on the problem of locating a facility on a network such that either the sum of distances from the nodes of the network to the facility is minimized (minisum/cost objective) or such that the maximum distance from all nodes to the facility is minimized (mini–max/bottleneck objective). These two problems have been known as the median and center problems on networks, respectively. Hakimi (1965) showed that an optimal solution to the p–median problem, where p facilities should be placed on a network, can be found among the nodes of the graph (the property of the p–median problem is known as the Hakimi–property or *nodal optimality*–property). Later, Minieka (1970) showed that

an optimal solution to the p–center problem can be found among the nodes of the graph and a finite number of so–called intersection points on edges of the network. Both of these classical problems have received enormous attention since then, and overviews of the scientific contributions for these problems can be found in Reese (2006) for the p–median problem and in Revelle, Eiselt, and Daskin (2008) for the p–center problem. Many extensions have been proposed including capacity constraints (see for example Mulvey and Beck (1984) and Bar-Ilan, Kortsarz, and Peleg (1993)), combinations of the two objectives (see e.g. Halpern (1978), Hansen, Labbé, and Thisse (1991), and Ogryczak (1997a)), and in the paper by Nickel and Puerto (1999) the *ordered median problem* is introduced which includes the p–median, the p–center, and the p–centdian problems as special cases.

For location problems arising in the public sector such as the location of hospitals or fire stations, it often happens that (political) constraints stating a maximal travel– or service time are added to the model. This means that a demand point is considered *uncovered* if the time it takes to travel from this location to the nearest open facility exceeds a specified threshold. This gives rise to the so–called *covering location problems*. On networks, the standard covering problem is the *maximal covering problem* which was introduced in Church and ReVelle (1974). This problem is similar to the p–center problem of Hakimi (1965), but it differs in that a fixed *covering radius* is given as a parameter and then the number of demand points which can be covered by a set of p facilities within the given radius is maximized.

Presumably independently of each other and concurrently with the development of network location problems, Balinski and Wolfe (1963) and Manne (1964) published the first mixed integer linear programming (MILP) formulations of a location problem. This problem has later become known as the *uncapacitated facility location problem* (UFLP). The UFLP is the prototypical *discrete* facility location problem: a finite set of potential facility sites and a finite set of demand points are given, and the objective is to minimize the cost incurred by opening facilities and supplying customers from the open facilities. Contrary to the standard network location problems, the number of open facilities is not given as a problem parameter; rather it is a result of the tradeoff between the cost of opening facilities and the cost of supplying the demand points. Interestingly, even though the p–median problem was originally stated as a network location problem, most of the research on solution algorithms for this problem has been addressed from the discrete facility location perspective: due to the Hakimi–property, the p–median problem can be stated as a discrete facility location problem where the finite set of potential facility sites and the set of demand points are both given by the set of nodes on the network. The first formulation of the p–median problem as an MILP was given by ReVelle and Swain (1970). Despite the simple structure of both the p–median problem and the UFLP, much contemporary research is still devoted to these problems. Among recent contributions we find a new formulation of the p-median problem that can

easily be adapted to the UFLP proposed by Elloumi (2010), a simultaneous row–and–column generation procedure for the p–median problem proposed by Avella, Sassano, and Vasilév (2007), an aggressive reduction approach for the UFLP proposed by Letchford and Miller (2014), and a semi–Lagrangean based dual ascent algorithm proposed by Jörnsten and Klose (2015) for large scale instances of the UFLP. For a fixed set of open facilities, the remaining *semi–assignment problem* of assigning demand points to open facilities is trivial for both the p–median problem and the UFLP.

Another widely studied problem is the *capacitated facility location problem* (CFLP) which extends the UFLP to include capacity constraints on the facilities. The objective is to find a set of open facilities and then to allocate the demands to these open facilities in such a way that total opening and allocation costs are minimized. In the CFLP a demand point may be serviced from several open facilities. A vast amount of literature considers this problem from both an algorithmic (heuristic and exact) and a polyhedral perspective. One of the first exact methods for the CFLP was proposed by Davis and Ray (1969), and the strengths of several bounds obtained from *Lagrangean relaxations* are compared in Cornuejols, Sridharan, and Thizy (1991). Many variants of this problem have been proposed in the literature including problems with staircase costs (Holmberg, 1994), increasing production costs (Harkness and ReVelle, 2003), and many more. If splitting demand is not feasible the CFLP becomes a so–called *single–source* CFLP (SSCFLP). This problem is surprisingly much harder to solve than the CFLP. Neebe and Rao (1983) applied a column-generation approach to this problem, and Holmberg, Rönnqvist, and Yuan (1999) developed a branch-and-bound algorithm relying on a Lagrangean relaxation.

The UFLP, the p–median problem, and the CFLP all minimize a cost/minisum objective. Church and ReVelle (1974) suggested the maximization of the (weighted) number of covered demand points instead. Like for the planar and the network location problems, a demand point is covered if it is within a certain distance of an open facility. Another covering problem, arises when the number of open facilities is not given a priori. Toregas, Swain, ReVelle, and Bergman (1971) formulated the problem of finding the minimum number of facilities needed to satisfy a certain coverage radius and computed the set of rational comprises between the coverage radius and the number of facilities needed to obtain that radius.

Regarding the network location problems a variant of the p–center problem is known in the field of discrete facility location as well: the *vertex–p–center* problem. This problem places facilities on the nodes of the graph only and can therefore be stated as *non–linear* discrete location problem. It can, however, easily be stated as an MILP by linearizing the objective function. Unfortunately, the resulting MIP is not of much use as the program is large and produces a rather weak lower bound. However, Elloumi, Labbé, and Pochet (2004) recently proposed a new formulation of the problem and developed an exact algorithm based

Table 1.1: Characterization of some of the most well–known facility location problems by topology and objective.

	Topology		
	Planar	Network	Discrete
Cost	Weber multi–Weber	1–median p–median	UFLP CFLP p–median
Bottleneck	planar center	center p–center	vertex–p–center bottleneck CFLP
Covering	maximal covering (in the plane)	maximal covering	set covering maximal covering

on binary search. Others have used the strong relation between the vertex–p–center problem and set covering in order to devise algorithms for this problem as well.

1.1.2 Characterizing facility location by topology and objective

As was mentioned in the beginning of the chapter, location problems can be roughly partitioned on the basis of the underlying topology and the objective function. Table 1.1 shows a three–by–three matrix containing some of the location problems mentioned. A column in Table 1.1 corresponds to a topology (planar, network, or discrete) and a row corresponds to an objective to be optimized. For example, the p–center problem on a network has a network topology and a bottleneck objective and is therefore placed in entry (Bottleneck, Network). From the discussion above, it was clear that the p–median problem could be viewed as a network location problem as well as a discrete location problem. Therefore, the p–median problem turns up in both (Cost, Network) and in (Cost, Discrete).

In this dissertation, the focus will be on discrete facility location, and on the bottleneck and the cost objectives. Particularly, the focus will be on the single–source capacitated facility location problem. As already stated, this problem is very similar to the widely studied CFLP. However, due to the increased computational complexity of the SSCFLP compared to the CFLP, the SSCFLP has not received the same attention as the CFLP. This dissertation studies the SSCFLP from both a single objective as well as from a bi–objective perspective.

1.2 Bi–objective facility location

Real world optimization problems often consist of problems with conflicting objectives. This means that more often than not a single *ideal* solution that is optimal for all objectives does

not exist. Facility location problems are not an exception and are consequently often of a multi–objective nature. Deciding on where to locate facilities is usually a strategic decision whereas the allocation or assignment of customers' demands to open facilities is a more tactical decision. Furthermore, the cost of opening facilities might be negatively correlated with the cost of allocating customer's demands to the open facilities. Consider, for example, the decision of where to locate (public) hospitals. A natural objective is to both minimize the cost of building the hospitals while at the same time minimizing the driving distance the citizens have to travel in order to reach the nearest hospital. The cost of building hospitals and the distance a citizen has to drive can both be expressed in monetary terms, but the costs are usually negatively correlated: the driving distance decreases as more hospitals are opened whereas the cost associated with opening these hospitals will naturally increase. *Multiple–objective location problems* are therefore a natural extension of the classical single objective facility location problems.

As this dissertation considers discrete facility location problems, we provide a short overview of the literature on multi-objective discrete facility location. That is, problems combining objectives, in a multi–objective fashion (third column of Table 1.1). As can be seen in for example Nickel, Puerto, and Rodríguez-Chía (2015), planar and network multi–objective location problems have been studied from a methodological point of view resulting in important structural results and algorithms known to generate efficient solutions, but little attention has been devoted to the structural analysis of discrete multi–objective location problems. Instead, the focus has been on modeling complex problems with often complex objective functions.

The paper by Ross and Soland (1980) is one of two papers found that take a methodological perspective on multi-objective discrete location problems. Here, the authors characterize the set of Pareto optimal solutions by rewriting the location problem as a generalized assignment problem with an additional constraint. However, the authors argue against generating all efficient solutions, and they propose an *interactive* approach instead. Fernández and Puerto (2003) released the second methodological paper considering a multi–objective versions of the uncapacitated facility location problem (UFLP). A dynamic programming approach was proposed based on a decomposition of the UFLP into a facility selection problem and a demand allocation problem. The dynamic programming algorithm was further improved by introducing lower and upper bounds allowing for implicit enumeration of some of the states.

Another bi–objective variation of the UFLP is proposed by Myung, Kim, and Tcha (1997). In the proposed model, a fixed cost is incurred when facilities are opened whereas a profit is gained when a customer is serviced. The model aims to maximize net profit and the rate of return at the same time. The authors use a parametric approach capable

of generating all supported efficient solutions, but only some of the unsupported efficient solutions. Bi–objective models for reverse logistics network design have also received some attention during the last decade. Du and Evans (2008) propose a model for a manufacturer outsourcing its post-sale services to a third–party logistics provider. The model decides on the capacity of repair stations to be installed at the third–party logistics provider.

The combination of cost and bottleneck objectives in multi–objective settings does not seem to be that well studied in the literature. In Drezner, Drezner, and Salhi (2006), however, five objectives are considered: cost (p-median), bottleneck (p–center), two covering objectives, and the minimization of the variance of distances between demand points and their closest open facility. A minimax–regret multi–objective approach is taken, where the maximum relative deviation of each objective to its ideal value is minimized.

Current, ReVelle, and Cohon (1985) introduce the so–called maximum covering/shortest path problem. The problem consists in finding a least cost path between a predetermined source and sink node pair, while at the same time maximizing the covered demand. The model considers the demand of a node covered if the node lies on the path or if the distance from the node to a node on the shortest path is below a given threshold. A branch–and–bound algorithm is used to generate the efficient frontier of an example network with 15 nodes and 34 edges. Bhaskaran and Turnquist (1990) investigate the combination of objectives minimizing total transportation costs and maximizing demand coverage for a car producing firm in North America. The authors note that an optimal solution for one objective might be very inferior for the other. The study showed that for that particular area and data, good compromises between the objectives could be found using bi–objective optimization techniques. Brimberg and ReVelle (1998) propose a slight variation of the UFLP where only some customers need to be served and propose a parametric solution approach for generating the subset of the efficient solutions lying on the convex envelope of the non–dominated frontier (the supported non–dominated outcomes). A computational study shows that the proposed integer program has an integer feasible LP-relaxation in many cases. Villegas, Palacios, and Medaglia (2006) also study an uncapacitated facility location problem where an additional covering objective is added. The model is used to solve the problem of redesigning a Colombian coffee supply network.

Ogryczak (1997b) takes a lexicographic approach to the p–center location problem: The standard bottleneck approach considers only the minimization of the largest distance from demand points to open facilities. Ogryczak also minimizes the second largest outcome (provided that the largest one remains as small as possible), minimizes the third largest (provided that the two largest remain as small as possible), and so on.

The location of emergency medical services (EMS) is considered in Chanta, Mayorga, and McLay (2011) where bi–objective models combining covering objectives and bottleneck

objectives are proposed. It is noted that in many previous approaches to EMS location, only the covering aspect is considered, leading to solutions where rural areas are neglected. Therefore a bottleneck objective minimizing the maximum distance from an uncovered area to its nearest open EMS is considered. A standard ε–constraint approach is taken to generate the efficient frontier of the problems.

To summarize, it seems that the combination of cost and covering objectives is particularly popular in the field of discrete multi–objective location problems. Furthermore, as many of the problems are $\mathcal{NP}$–hard even in the single objective case, it has not been the scope of many contributions to generate the entire set of non–dominated outcomes. Therefore, combinations of other types of objective functions and methods for generating all non–dominated outcomes seem to be promising research areas where interesting insights can be gained.

1.3 Contribution and outline of the dissertation

This dissertation has as its main focus the single–source capacitated facility location problem which is studied both as a single objective as well as a bi–objective problem. The dissertation consists of four self–contained papers which can be read independently. For the single objective problem (Chapter 2 and Chapter 3), the contribution of the dissertation is primarily algorithmic in the sense that the best known algorithms for the problem are significantly improved.

Section 1.2 showed that most research in the field of discrete multi–objective facility location is focused on the modeling aspect. In Chapter 4 and Chapter 5 of this dissertation, however, the focus is on the methodology. A bi-objective version of the SSCFLP is studied in which the bottleneck objective is considered as the second objective, and to the best of the author's knowledge no prior research has studied this problem in the discrete case. A new direct solution method based on a branch–and–cut framework for bi–objective combinatorial problems is also suggested and used to solve a bi–objective version of the SSCFLP where facility establishing costs and allocation costs are treated as separate objectives. Previous literature has not presented an exact solution of this problem.

In Chapter 2, an effective three–phased algorithm is proposed for solving the single objective single–source capacitated facility location problem (SSCFLP). The chapter starts by reviewing the literature on the SSCFLP and proceeds by explaining the algorithmic approach. The facets of the knapsack constraints obtained from the capacity constraints are characterized and the characterization is used to reduce the effort in separation algorithms subsequently proposed and employed to strengthen the LP–bound. A simple local branching heuristic is then proposed which takes the structure of the problem into account, and finally an accelerated cut–and–solve algorithm is used to close the gap between the LP relaxation

and the heuristic upper bound. The paper contains experimental results clearly showing that the methodology is superior to a state of the art algorithm for the SSCFLP.

Chapter 3 proposes two algorithms for the single objective SSCFLP based on a semi–Lagrangean relaxation. The two algorithms integrate the cut–and–solve framework with a semi–Lagrangean based dual ascent algorithm. The first algorithm solves the semi–Lagrangean subproblems by use of a modified version of the algorithm proposed in the paper of Chapter 2. The second algorithm uses a dual ascent algorithm to solve the sparse problems of a cut–and–solve algorithm.

Chapter 4 considers a family of bi–objective discrete facility location problems that combine, in a bi–objective manner, a minisum or cost objective with a minimax or bottleneck objective. It is shown that bi-objective facility location problems of this type can be solved efficiently by means of an ε–constraint method that solves at most a polynomial number of minisum problems. Additionally, the algorithm is compared to a two–phase method. Extensive computational results obtained from several classes of facility location problems are reported. The proposed algorithm is shown to compare very favorably to other bi–objective algorithms.

Finally, in Chapter 5, a novel direct branch–and–cut algorithm for bi–objective combinatorial optimization is developed. The algorithm develops methods for comparing lower and upper bound sets by both implicitly and explicitly stated lower bounds. Furthermore, a simple lower bound updating rule is devised for explicitly stated lower bound sets which reduce the number of bi–objective LPs the algorithm needs to solve. In addition, we introduce and test the concept of extended Pareto branching. The branch–and–cut algorithm compares favorably to a two–phase method.

An improved cut–and–solve algorithm for the single–source capacitated facility location problem

This chapter is based on the paper Gadegaard, Klose, and Nielsen (2016d).
submitted to EURO Journal on Computational Optimization,
June 2016

An improved cut–and–solve algorithm for the single–source capacitated facility location problem

Sune Lauth Gadegaard[†], Andreas Klose[*] and Lars Relund Nielsen[†]

[†]Department of Economics and Business Economics, Aarhus University, Denmark,
{sgadegaard, larsrn}@econ.au.dk
[*]Department of Mathematics, Aarhus University, Denmark, aklose@imf.au.dk

Abstract

In this paper we present an improved cut–and–solve algorithm for the single–source capacitated facility location problem in the form of a three-phase-algorithm. The first phase strengthens the integer program by a cutting plane algorithm to obtain a tight lower bound. The second phase uses a two level local branching heuristic to find an upper bound and, if optimality has not yet been established, the third phase uses an accelerated cut–and–solve algorithm to close the optimality gap. Extensive computational results are reported, showing that the proposed algorithm runs 10 to 80 times faster on average compared to state of the art problem specific algorithms.

Keywords: Facility location; capacitated facility location; single–sourcing; cut–and–solve; cutting planes; local branching.

2.1 Introduction

The *single–source capacitated facility location problem* (*SSCFLP*) is the problem of opening a set of facilities and assigning each customer to exactly one open facility while respecting the capacity of each facility. An optimal solution to the SSCFLP is an allocation which minimizes fixed opening and allocation costs. The difference between the SSCFLP and the more widely studied *capacitated facility location problem* (*CFLP*) is that each customer must be serviced by only one facility. The SSCFLP is a significant problem in the area of location science as it exhibits many of the challenging aspects and features of general location problems. Furthermore, the SSCFLP often arises as a subproblem in more complex problems

such as hierarchical location problems with capacities and capacitated facility problems with piecewise linear costs.

It is well known that the SSCFLP can be formulated as an *integer linear program* (ILP) with a set of demand constraints and a set of capacity constraints. Since the SSCFLP is a strongly $\mathcal{NP}$-hard problem (the SSCFLP polynomially reduces to the node cover problem, which was one of the original 21 $\mathcal{NP}$–complete problems listed in Karp (1972)), most research has been focused on heuristic solution approaches.

Since the *linear programming* (*LP*) lower bound of the SSCFLP as stated above is known to be weak, most of the existing literature focuses on a Lagrangean relaxation of one or more of the constraints of the SSCFLP. These Lagrangean based heuristics primarily differ in the way constraints are relaxed and in the way a primal solution is obtained from the Lagrangean subproblem. Klincewicz and Luss (1986) relax the capacity constraints in a Lagrangean manner where the Lagrangean subproblem becomes an uncapacitated facility location problem. A heuristic primal solution is obtained by an add heuristic and a refinement heuristic that improves the primal feasible solutions to the Lagrangean subproblem. If the demand constraints are relaxed in a Lagrangean manner instead, Bitran, Chandru, Sempolinski, and Shapiro (1981) show that the Lagrangean subproblem decomposes into an independent knapsack problem for each facility site. Sridharan (1993) uses this fact to derive a tight lower bound for the SSCFLP by maximizing the Lagrangean dual function using subgradient optimization. In order to obtain good primal solutions, a generalized assignment problem is solved over the open facilities in each iteration of the subgradient procedure; the approach can therefore hardly be applied to large problem instances. The capacitated concentrator location problem, which is equivalent to the SSCFLP, is studied in Pirkul (1987), where a lower bound is obtained by maximizing the Lagrangean dual function that results from a relaxation of the demand constraints. In Beasley (1993) both the capacity constraints and the demand constraints are relaxed and different Lagrangean heuristics are compared. Barceló and Casanovas (1984) consider a slight variation of the SSCFLP where the number of open facilities is limited by a given constant K. They propose a Lagrangean heuristic that decomposes into two subproblems; a plant selection and an allocation phase. The heuristic is based on relaxing the demand constraints in a Lagrangean manner. Properties of the dual of the LP-relaxation are used to guide a heuristic computing the Lagrangean multipliers. Barceló, Fernández, and Jörnsten (1991) propose a Lagrangean decomposition approach that separates the demand constraints from all other constraints. The Lagrangean subproblem thus decomposes into two subproblems; one being identical to the one obtained when relaxing the demand constraints and one being a simple semi-assignment problem. As the latter shows the integrality property, the Lagrangean decomposition bound is no stronger than the Lagrangean bound based on relaxing the demand constraints. Taking a different

approach Barceló, Hallefjord, Fernández, and Jörnsten (1990) suggest, to relax the capacity constraints, including a total demand constraint (stating that the open facilities' capacity needs to cover the total demand). In addition, they generate simple cover inequalities from the total demand constraint on the fly during subgradient optimization. The added cover inequalities are then relaxed and therefore they do not alter the Lagrangean subproblem but may nevertheless contribute to an improved bound. A more recent Lagrangean relaxation based heuristic is proposed in Chen and Ting (2008) where the Lagrangean dual function is approximately maximized using subgradient optimization, and a multiple ant colony heuristic is used in each iteration in order to improve the best upper bound.

Other types of heuristics have been developed as well. Some of the more recent approaches count a repeated matching heuristic proposed in Rönnqvist, Tragantalerngsak, and Holt (1999), a scatter search strategy developed by Contreras and Díaz (2008), and a multi–exchange heuristic presented in Ahuja, Orlin, Pallottino, Scaparra, and Scutella (2004). The latter is based on a very large-scale neighborhood search first presented in Ahuja, Orlin, and Sharma (2000) and originally applied to the capacitated minimum spanning tree problem.

Less research has been devoted to exact solution methods primarily due to the large number of integer variables. Neebe and Rao (1983) reformulate the SSCFLP as a set partitioning problem and column generation is used to solve the resulting model. Holmberg, Rönnqvist, and Yuan (1999) relax the demand constraints in a Lagrangean manner in order to obtain lower bounds. Upper bounds are generated using a repeated matching heuristic. These bounds are used in a branch–and–bound algorithm to find a proven optimal solution to the SSCFLP. A branch–and–price algorithm is developed by Díaz and Fernández (2002). Ceselli and Righini (2005) consider the capacitated p-median problem, which is closely related to the SSCFLP; instead of including fixed facility costs, the number of open facilities may not exceed a number p. They also propose a branch–and–price algorithm based on a set–covering like formulation of the problem. The lower bound obtained from the LP-relaxation of the reformulated problem is easily shown to be the same as the one obtainable from the Lagrangean relaxation of the demand constraints.

Over the years, various methods have been successfully used to strengthen the lower bound on general ILPs, e.g. knapsack separation using lifted cover inequalities (Balas, 1975; Balas and Zemel, 1978; Crowder, Johnson, and Padberg, 1983; Gu, Nemhauser, and Savelsbergh, 1998, 1999; Kaparis and Letchford, 2010), weight inequalities (Weismantel, 1997) and exact separation (Boyd, 1993). Avella, Boccia, and Salerno (2011) proposed a reformulation of the SSCFLP based on dicut inequalities. Here, exact knapsack separation is used to strengthen the formulation and the problem is solved using branch–and–cut. Furthermore, in Yang, Chu, and Chen (2012) a cutting plane algorithm based on exact knapsack separation is used to strengthen the formulation of the SSCFLP and a cut–and–solve framework is used to

solve the strengthened formulation.

In this paper, we propose a new efficient three–phase algorithm improving the running time on known test beds compared to state of the art problem specific software. Inspired by the computational results obtained using knapsack separation reported in Avella et al. (2011) and Yang et al. (2012), we characterize the facets of the knapsack structures arising from the capacity constraints. The first phase of the algorithm consists of an efficient and effective cutting plane algorithm that takes advantage of this characterization. A heuristic based on local branching is employed in the second phase in order to obtain a strong upper bound. In fact, the computational studies suggest that the heuristic could be used as a stand alone heuristic. Finally, in case the lower bound found in the first phase does not reach the upper bound found in the second phase, a third phase is considered and an accelerated cut–and–solve algorithm is used to solve the problem to optimality. In this paper, we propose a new way of defining the subproblems which tends to generate smaller subproblems and prove termination of the resulting algorithm. The speed of the algorithm is further substantially improved by simple, but usually effective, variable fixing in conjunction with premature pruning of subproblems. The four main contributions of this paper are:

1. We propose a three–phase algorithm which improves the solution times on known test beds considerably.
2. We characterize the facets of the knapsack structures arising from the capacity constraints and exploits this result when strengthening the lower bound of the problem.
3. We propose a new local branching heuristic that takes advantage of the two decision levels in the SSCFLP.
4. We propose and successfully test an accelerated cut–and–solve algorithm.

The paper is organized as follows: In Section 2.2, a mathematical programming formulation of the SSCFLP is given. Section 2.3 describes the three phases of the algorithm. In Section 2.4 extensive computational results comparing the algorithm to commercial as well as specialized algorithms are reported. Finally, Section 2.5 gives conclusions and directions for further research.

2.2 Problem formulation

Let $\mathcal{I}$, $|\mathcal{I}| = n$, be a set of potential facility sites and $\mathcal{J}$, $|\mathcal{J}| = m$, be a set of customers. Each facility, $i \in \mathcal{I}$, has a fixed capacity, $s_i > 0$, and each customer has a fixed and known demand, $d_j > 0$. Opening a facility i results in a fixed cost $f_i \geq 0$ and allocating a customer j to a facility i involves a cost of $c_{ij} \geq 0$. Let y_i be a binary variable equaling one only if a facility is opened at site i and similarly let x_{ij} be a binary variable that equals one only if

customer j is allocated to facility i. The SSCFLP can then be stated as the following ILP problem:

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i \tag{O}$$

$$\text{s.t.:} \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \tag{D}$$

$$\sum_{j\in\mathcal{J}} d_j x_{ij} \le s_i y_i \quad \forall i \in \mathcal{I}, \tag{C}$$

$$\sum_{i\in\mathcal{I}} s_i y_i \ge D, \tag{T}$$

$$0 \le x_{ij}, y_i \le 1, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}, \tag{N}$$

$$y_i,\ x_{ij} \in \{0,1\}, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}. \tag{I}$$

The objective function (O) minimizes the total cost. Constraints (D), referred to as the *demand constraints*, make sure that each customer is allocated to exactly one facility. The *capacity constraints* (C) ensure that the capacities of the facilities are respected. In the total demand constraint (T), D represents the total demand. This constraint is in fact redundant, but it will be used later to tighten the LP relaxation of the problem. Constraints (N) state that all variables should be non-negative and less or equal to one. Finally, constraints (I) state that all variables should be binary. It will be assumed throughout the paper that all parameters are integral.

2.3 Solution methodology

In order to solve the SSCFLP to optimality, we propose an algorithm running in three phases (see Algorithm 2.1). The first phase consists of a cutting plane algorithm that separates fractional solutions from the knapsack polytopes defined by the capacity constraints (C) and the total demand constraint (T). In the second phase, we implement a two level local branching strategy that exploits the two levels of decisions (location and allocation) in order to produce an initial near optimal solution. And finally, a third phase based on the cut–and–solve framework is used to solve the problem to proven optimality.

Following the notation used in Cornuejols, Sridharan, and Thizy (1991), we let $P(R)$ denote the set of solutions satisfying a set of constraints R and $\text{conv}(R)$ denote the convex hull of these solutions. For simplicity, $P(RS) \coloneqq P(R) \cap P(S)$ and $\text{conv}(RS) \coloneqq \text{conv}(P(RS))$. Furthermore, by $\mathcal{P}(R)$ we define the optimization problem

$$\mathcal{P}(R): \quad \min\{cx + fy \ :\ (x,y) \in P(R)\}.$$

Using this notation, the SSCFLP can be written as $\mathcal{P}_{\text{IP}} \coloneqq \mathcal{P}(DCTI)$ and its (weak) linear relaxation $\mathcal{P}_{\text{LP}} \coloneqq \mathcal{P}(DCTN)$. Furthermore, the conventional notation $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$

Phase 1:

1.1: Apply the cutting plane algorithm described in Algorithm 2.2.

1.2: If the solution returned is integral, stop as it is an optimal solution. Else go to Phase 2.

Phase 2:

2.1: Apply the local branching heuristic described in Section 2.3.2.

2.2: If the solution value of the returned solution equals the lower bound from Phase 1, stop as the solution is optimal. Else go to Phase 3.

Phase 3:

3.1: Apply the cut–and–solve algorithm described in Algorithm 2.5.

3.2: Stop as the solution returned by the cut–and–solve algorithm is optimal.

Algorithm 2.1: Summary of the complete three–phase algorithm

is used. Solutions corresponding to a lower bound on the optimal solution will be denoted using underlining as $(\underline{x}, \underline{y})$ and solutions corresponding to upper bounds will be denoted using overlining, that is $(\overline{x}, \overline{y})$.

2.3.1 Phase 1 – Cutting planes

It is well known that the lower bound obtained by solving $\mathcal{P}_{\text{LP}}$ is usually very weak. Therefore, *implied variable upper bounds*

$$x_{ij} - y_i \leq 0, \quad \forall i \in I \; j \in \mathcal{J}, \tag{2.1}$$

are often added to $\mathcal{P}_{\text{LP}}$. The resulting LP is, however, very large and consequently hard to solve, which is why we add these constraints on the fly as needed.

In order to further strengthen the LP bound, we ideally want to solve

$$\min\{cx + fy \; : \; (x, y) \in P(D) \cap \text{conv}(CTI)\}.$$

One particular way to do this is to relax the assignment constraints (D) in a Lagrangean manner and to solve the resulting Lagrangean dual. From Geoffrion (1974) we know that solving the Lagrangean dual implicitly corresponds to building the convex hull of solutions to the kept constraints. In this paper, however, we want to approximate the convex hull of integer solutions to the capacity constraints (C) and the total demand constraint (T) by cutting planes. By doing so, we avoid separating from the entire polytope defined by $\text{conv}(CTI)$ and instead we need only consider one constraint at a time.

The following observations will be used to further reduce the effort in separating fractional points from conv(CTI). Let

$$\mathcal{X} = \{x \in \{0,1\}^m \ : \ a^T x \leq a_0\},$$
$$\mathcal{X}_y = \{(x,y) \in \{0,1\}^{m+1} \ : \ a^T x \leq a_0 y\},$$

where $a_k > 0$ for $k = 0, 1, \ldots, m$. Moreover, assume that $a_j \leq a_0$ for all $j = 1, \ldots, m$ as if this was not the case, the corresponding variables could be removed. Now observe that if $\pi^T x \leq \pi_0$ is a valid inequality for $\mathcal{X}$, then $\pi^T x \leq \pi_0 y$ is valid for $\mathcal{X}_y$. Furthermore, it is easily seen that if $\pi^T x \leq \pi_0$ is facet defining for conv($\mathcal{X}$), then $\pi^T x \leq \pi_0 y$ will be facet defining for conv($\mathcal{X}_y$), implying that strong cutting planes derived from conv($\mathcal{X}$) will be strong for conv($\mathcal{X}_y$). The converse is also true but before proving it, the following well known result from polyhedral theory is needed (see (Nemhauser and Wolsey, 1988, Proposition 6.6, p.108) for a proof).

Proposition 2.1. *Let $S = \{x \in \mathbb{R}^n_{\geqq} \ : \ Ax \leq b\} \cap \mathbb{Z}^n$, where A and b are matrices of appropriate sizes. If $\pi^T x \leq \pi_0$ defines a face of dimension $k-1$ of* conv(S)*, there are k affinely independent points $x^1, \ldots, x^k \in S$ such that $\pi^T x^l = \pi_0$ for $l = 1, \ldots, k$.*

With this result established, we are ready to prove Proposition 2.2.

Proposition 2.2. *If $\pi^T x \leq \pi_0 y$ is facet defining for* conv($\mathcal{X}_y$) *then $\pi^T x \leq \pi_0$ is facet defining for* conv($\mathcal{X}$).

Proof. Let $(x,y)^l$ $l = 1, \ldots, m+1$ be $m+1$ affinely independent points from the facet

$$\{(x,y) \in \text{conv}(\mathcal{X}_y) \ : \ \pi x = \pi_0 y\}.$$

According to Proposition 2.1 we can assume that these $m+1$ points are from $\mathcal{X}_y$. One of the $m+1$ affinely independent points $(x,y)^l$ $(l = 0, 1, \ldots, m)$ on the facet is the point $(x,y) = (0,0)$. Let this be point $(x,y)^0$. For all other points we have $y = 1$. Accordingly the points x^l $(l = 1, \ldots, m)$ give m affinely independent points on the face $\{x \in \text{conv}(\mathcal{X}) \ : \ \pi x = \pi_0\}$. □

Combining Proposition 2.2 and the previous observation we obtain

Corollary 2.1. *$\pi^T x \leq \pi_0 y$ is facet-defining for* conv($\mathcal{X}_y$) *if and only if $\pi^T x \leq \pi_0$ is facet-defining for* conv($\mathcal{X}$).

We now examine the form of a facet, $\pi^T x - \pi_y y \leq \pi_0$, of conv($\mathcal{X}_y$).

Lemma 2.1. *Any inequality of the form $\pi x - \pi_y y \leq \pi_0$ that is valid for $\mathcal{X}_y$ is equivalent to or dominated by the inequality $\pi x \leq (\pi_0 + \pi_y) y$.*

Proof. For the case $y = 1$, both inequalities are the same. For the case $y = 0$, the first inequality gives $0 \leq \pi_0$, which can be strengthened to $0 \leq \pi_0 y = 0$. □

From Lemma 2.1 we immediately get Corollary 2.2.

Corollary 2.2. *Any facet defining inequality of* $\text{conv}(\mathcal{X}_y)$ *different from* $y \leq 1$ *can be written as* $\pi x \leq \pi_y y$.

Proof. From Lemma 2.1 we know that any facet-defining inequality $\pi x - \tilde{\pi}_y y \leq \pi_0$ needs to be equivalent to $\pi x \leq (\tilde{\pi}_y + \pi_0)y$. Letting $\pi_y = \tilde{\pi}_y + \pi_0$, the result follows. □

We now determine the sign of the coefficients π and π_y.

Lemma 2.2. *Let* $\pi x \leq \pi_y y$ *be a facet-defining inequality for* $\text{conv}(\mathcal{X}_y)$ *different from* $y \leq 1$ *and* $-x_j \leq 0$. *Then* $\pi_y > 0$.

Proof. From $(0, 1) \in \mathcal{X}_y$, we get $0 = 0\pi \leq 1\pi_y = \pi_y$. Assume $\pi_y = 0$, so that the inequality becomes $\pi x \leq 0$. As, by assumption, the points (x, y) with $x = e_j$ and $y = 1$ are from $\mathcal{X}_y$, we get $e_j \pi = \pi_j \leq 0$ for all j. But then $\pi x \leq 0$ is dominated by the set of inequalities $-x_j \leq 0$ for each j. Hence, if $\pi x \leq \pi_y y$ is facet defining and different from $-x_j \leq 0$, then $\pi_y > 0$. □

Lemma 2.3. *Let* $\pi^T x \leq \pi_y y$ *be a facet defining inequality of* $\text{conv}(\mathcal{X}_y)$ *different from* $-x_j \leq 0$ *for all* $j \in \{1, \ldots, m\}$. *Then* $\pi_j \geq 0$ *for all* $j \in \{1, \ldots, m\}$.

Proof. Suppose there exists an index $\tilde{j} \in \{1, \ldots, m\}$ such that $\pi_{\tilde{j}} < 0$. Note that $x_{\tilde{j}} = 0$ in any optimal solution to the program

$$\begin{aligned} &\max \ \pi^T x - \pi_y y \\ &\text{s.t.: } (x, y) \in \mathcal{X}_y. \end{aligned}$$

Now define the vector $\tilde{\pi}$ as follows

$$\tilde{\pi}_j = \begin{cases} \pi_j, & \text{if } j \neq \tilde{j} \\ 0, & \text{if } j = \tilde{j} \end{cases}$$

This leads to the fact that $x_{\tilde{j}} = 0$ is an optimal value of $x_{\tilde{j}}$ in the program

$$\begin{aligned} &\max \ \tilde{\pi}^T x - \pi_y y \\ &\text{s.t.: } (x, y) \in \mathcal{X}_y, \end{aligned}$$

stating that

$$0 \geq \max\{\pi^T x - \pi_y y \ : \ (x, y) \in \mathcal{X}_y\} = \max\{\tilde{\pi}^T x - \pi_y y \ : \ (x, y) \in \mathcal{X}_y\}$$

whereby $\tilde{\pi}^T x \leq \pi_y y$ is valid for $\text{conv}(\mathcal{X}_y)$. Furthermore, as $\tilde{\pi}_{\tilde{j}} > \pi_{\tilde{j}}$ the inequality $\tilde{\pi}^T x \leq \pi_y y$ dominates $\pi^T x \leq \pi_y y$, a contradiction. Therefore, $\pi \geq 0$. □

Step 0: Set $Z^0 = -\infty$ and iteration counter $k = 0$.

Step 1: Set $k = k + 1$. Solve the program $\mathcal{P}_{\text{LP}}$ and let $(\underline{x}, \underline{y})$ be an optimal solution and Z^k the solution value.

Step 2: If $(\underline{x}, \underline{y})$ is integral, return $(\underline{x}, \underline{y})$ as it is optimal.

Step 3: If $Z^k - Z^{k-1} < \varepsilon$, return $(\underline{x}, \underline{y})$.

Step 4: For each $i \in \mathcal{I}$ do the following
Step 4.2: Add all violated implied bounds of the form (2.1) to $\mathcal{P}_{\text{LP}}$.
Step 4.2: Separate $(\underline{x}, \underline{y})$ by a General Lifted Cover Inequality (GLCI) (see Algorithm 2.3); if possible, add it to $\mathcal{P}_{\text{LP}}$. If not, try to generate a Fenchel cutting plane (see Algorithm 2.4); if possible, add this to $\mathcal{P}_{\text{LP}}$.

Step 5: If $k < K$ go to Step 1, else return $(\underline{x}, \underline{y})$.

Algorithm 2.2: Summary of the cutting plane algorithm. The parameter $\varepsilon > 0$ is a suitable number used to check if the improvement in the lower bound is below a predefined limit.

Finally, by combining the above results, Corollary 2.3 is obtained.

Corollary 2.3. *All facet defining inequalities of* $\text{conv}(\mathcal{X}_y)$*, different from the trivial facets* $-x_j \leq 0$ *and* $y \leq 1$*, are of the form* $\pi^T x \leq \pi_y y$*, where* $\pi \geq 0$*,* $\pi_y > 0$*, and* $\pi^T x \leq \pi_y$ *is facet defining for* $\text{conv}(\mathcal{X})$*.*

In the following we will therefore omit the location variable when generating cutting planes from the capacity constraints (C) and then translate the resulting inequality by multiplying the right–hand side by y. The cutting plane algorithm developed in this paper is described in Algorithm 2.2. Two types of cutting planes are used, namely lifted cover inequalities and Fenchel cutting planes. All cutting planes are described for the knapsack polytope defined by the capacity constraints, but it should be obvious that they apply to the total demand constraint (T) as well (simply complement location variables by $z_i = 1 - y_i$ and an ordinary knapsack constraint is obtained).

Lifted cover inequalities

A *cover* of the i'th capacity constraint is a set of customers whose total demand exceeds the capacity of facility i. That is, a cover is a set $\tilde{\mathcal{J}} \subseteq \mathcal{J}$ such that $\sum_{j \in \tilde{\mathcal{J}}} d_j > s_i$. A *minimal cover* of a capacity constraint is a cover that additionally satisfies $\sum_{j \in \tilde{\mathcal{J}} \setminus \{\bar{j}\}} d_j \leq s_i$ for all $\bar{j} \in \tilde{\mathcal{J}}$. Let $\tilde{\mathcal{J}}$ be a minimal cover of capacity constraint i, then obviously $\sum_{j \in \tilde{\mathcal{J}}} x_{ij} \leq |\tilde{\mathcal{J}}| - 1$ is a valid inequality. If the minimal cover $\tilde{\mathcal{J}}$ is partitioned into two disjoint sets $\mathcal{J}^0$ and $\mathcal{J}^1$, the inequality $\sum_{j \in \mathcal{J}^0} x_{ij} \leq |\mathcal{J}^0| - 1$ is only valid if $x_{ij} = 1$ for all $j \in \mathcal{J}^1$. Obtaining a valid

Step 0: Input a fractional solution $\underline{\mathbf{x}}_i$ which satisfies capacity constraint i.

Step 1: Let variables $z_j = 0$ for all variables where $\underline{x}_{ij} = 0$. Sort the rest of the allocation variables in non-increasing order of $\underline{x}_{ij}$. Start from the beginning and set $z_j = 1$ until $\sum_{j:z_j=1} d_j > s$. Then $\tilde{\mathcal{J}} = \{j \in \mathcal{J} \ : \ z_j = 1\}$ defines a cover of capacity constraint i.

Step 2: Partition the cover, $\tilde{\mathcal{J}}$, into two disjoint sets $\mathcal{J}^1 = \{j \in \tilde{\mathcal{J}} \ : \ \underline{x}_{ij} = 1\}$ and $\mathcal{J}^0 = \tilde{\mathcal{J}} \backslash \mathcal{J}^1$. If $\tilde{\mathcal{J}}$ is not a minimal cover, delete items from $\mathcal{J}^0$ until it is.

Step 3: Set $\mathcal{F} = \{j \in \mathcal{J} \backslash \tilde{\mathcal{J}} \ : \ \underline{x}_{ij} > 0\}$ and $\mathcal{N} = \{j \in \mathcal{J} \backslash \tilde{\mathcal{J}} \ : \ \underline{x}_{ij} = 0\}$. Starting from the inequality $\sum_{j \in \mathcal{J}^0} x_{ij} \leq |\mathcal{J}^0| - 1$ do the following

1. Up-lift variables in $\mathcal{F}$ in a greedy manner. That is, lift the variable x_{ij} with the greatest value of $\alpha_j \underline{x}_{ij}$ (this requires finding all the (remaining) coefficients for each variable in $\mathcal{F}$).
2. If $\sum_{j \in \mathcal{J}^0} \underline{x}_{ij} + \sum_{j \in \mathcal{F}} \alpha_j \underline{x}_{ij} \leq |\mathcal{J}^0| - 1$, the inequality cannot be made violated by lifting more coefficients. Therefore stop.
3. Down-lift all variables in $\mathcal{J}^1$ in order of non-decreasing magnitude of reduced cost.
4. Up-lift variables in $\mathcal{N}$ in order of non-decreasing magnitude of reduced cost.

Step 4: Multiply the right–hand side of the resulting GLCI by y_i and output the violated cutting plane

$$\sum_{j \in \mathcal{J} \backslash \tilde{\mathcal{J}}} \alpha_j x_{ij} + \sum_{j \in \mathcal{J}^1} \gamma_j x_{ij} + \sum_{j \in \mathcal{J}^0} x_{ij} \leq (|\mathcal{J}^0| - 1 + \sum_{j \in \mathcal{J}^1} \gamma_j) y_i$$

Algorithm 2.3: Separation procedure for GLCI for a single capacity constraint.

inequality for the polytope $\text{conv}(\{x_{ij} \in \{0,1\}^m \ : \ \sum_{j \in \mathcal{J}} d_j x_{ij} \leq s_i\})$ can then be achieved by down-lifting the variables in $\mathcal{J}^1$. Furthermore, the inequality can be strengthened by up-lifting variables not in the initial cover (that is, $j \in \mathcal{J} \setminus \tilde{\mathcal{J}}$). The resulting inequality is referred to as a *general lifted cover inequality* (*GLCI*) and is of the form

$$\sum_{j \in \mathcal{J} \backslash \tilde{\mathcal{J}}} \alpha_j x_{ij} + \sum_{j \in \mathcal{J}^1} \gamma_j x_{ij} + \sum_{j \in \mathcal{J}^0} x_{ij} \leq |\mathcal{J}^0| - 1 + \sum_{j \in \mathcal{J}^1} \gamma_j.$$

Furthermore, $\alpha_j \geq 0$ for all $j \in \mathcal{J} \setminus \tilde{\mathcal{J}}$ and $\gamma_j \geq 0$ for all $j \in \mathcal{J}^1$ (see Nemhauser and Wolsey, 1988, Part II.1.2). The coefficients α_j and γ_j are up-lift and down-lift coefficients, respectively, and determining each α_j and γ_j requires the solution of a 0-1 knapsack problem, and the derivation of a GLCI is therefore $\mathcal{NP}$-hard (Gu et al., 1999).

We use the method proposed by Gu et al. (1998) to generate the initial minimal cover and the lifting sequence. The method is summarized in Algorithm 2.3 (Step 0–Step 3). For a detailed survey on cover inequalities and their extensions, the reader is referred to Gu et al.

(1998, 1999). Using the observations that if $\pi^T x \leq \pi_0$ is valid for $\mathcal{X}$, then $\pi^T x \leq \pi_0 y$ is valid for $\mathcal{X}_y$, we translate the resulting GLCI by multiplying the right–hand side by y_i and return the resulting cutting plane in Step 4 of Algorithm 2.3.

Fenchel cutting planes

Let $\mathcal{X}'_{iy} = \{(\mathbf{x}_i, y_i) \in \{0,1\}^{|\mathcal{J}|+1} \ : \ \sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i\}$ and $\text{proj}_x(\mathcal{X}'_{iy})$ be the projection of $\text{conv}(\mathcal{X}'_{iy})$ on the space of the x variables. As the set of all integer solutions to $\text{proj}_x(X'_{iy})$ is an independence system, all non-dominated valid inequalities for $\text{proj}_x(\mathcal{X}'_{iy})$ different from $-x_{ij} \leq 0$ are of the form $\lambda^T \mathbf{x}_i \leq 1$ (Nemhauser and Wolsey, 1988, p.237). A Fenchel cutting plane is in this case a cutting plane of the form $\lambda^T \mathbf{x}_i \leq 1$ that cuts as deep as possible into the relaxed polytope. Generating a Fenchel cutting plane for $\text{proj}_x(\mathcal{X}'_{iy})$ can be seen as a proof of the existence of a hyperplane that separates a fractional solution $\underline{\mathbf{x}}_i$ from $\text{proj}_x(\mathcal{X}'_{iy})$. Boyd (1993) states the separation problem for the knapsack polytope $\text{proj}_x(\mathcal{X}'_{iy})$ as

$$\begin{aligned} \nu = \max\ & \underline{\mathbf{x}}_i^T \lambda \\ \text{s.t.:}\ & \mathbf{x}_i^T \lambda \leq 1, \quad \forall \mathbf{x}_i \in \text{proj}_x(\mathcal{X}'_{iy}), \\ & 0 \leq \lambda_j \leq 1, \quad \forall j \in \mathcal{J}. \end{aligned} \tag{2.2}$$

If $\nu > 1$, then there exists a hyperplane that separates $\underline{\mathbf{x}}_i$ from $\text{proj}_x(\mathcal{X}'_{iy})$, and this hyperplane is given by $\sum_{j\in\mathcal{J}} \lambda^*_j x_{ij} \leq 1$, where λ^* is an optimal solution to (2.2). In fact, Boyd (1993) proves a reduction result stating that there exists a hyperplane separating $\underline{\mathbf{x}}_i$ from $\text{proj}_x(\mathcal{X}'_{iy})$ if and only if there exists a hyperplane separating $\underline{\mathbf{x}}_i$ from

$$\begin{aligned} \text{proj}_x(\mathcal{X}'_{iy}) \cap \{\mathbf{x}_i \in \mathbb{R}^{|J|} \ & : x_{ij} = 0 \ \forall j \in \mathcal{J}^0 \\ & \ \ x_{ij} = 1 \ \forall j \in \mathcal{J}^1\}, \end{aligned}$$

where $\mathcal{J}^0 = \{j \in \mathcal{J} \ : \ \underline{x}_{ij} = 0\}$ and $\mathcal{J}^1 = \{j \in \mathcal{J} \ : \ \underline{x}_{ij} = 1\}$. We can therefore exclude these variables from the separation problem and thereby reduce the effort quite substantially.

The separation problem (2.2) obviously has too many constraints to be taken directly into account. For that reason, Boccia, Sforza, Sterle, and Vasilyev (2008) propose a generic row-generation procedure. We do, however, propose an equivalent column generation procedure which is summarized in Algorithm 2.4 (Step 0–Step 8).

We have chosen to implement the method as a column generation procedure instead of a row generation procedure as the basis matrix of the problem (2.3) only has the dimension of the number of variables included in the separation problem, whereas the basis matrix of the equivalent row generation procedure has the dimension of H (or the dimension of linear independent rows in H). As rows are generated, the dimension of the basis matrix increases, and the simplex iterations become computationally more demanding. This is avoided when solving problem (2.2) by column generation. When solving the binary knapsack problems

Step 0: Input a capacity constraint $\sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i$ and a fractional solution $\underline{\mathbf{x}}_i$.

Step 1: Let $\mathcal{J}^1$, $\mathcal{J}^0$ and $\mathcal{J}^f$ be the set of indices where $\underline{x}_{ij} = 1$, $\underline{x}_{ij} = 0$ and $0 < \underline{x}_{ij} < 1$, respectively. Define the residual capacity, $\overline{s}$, as $\overline{s} = s_i - \sum_{j\in\mathcal{J}^1} d_j$. Set the iteration counter $l = 0$ and initialize the set of solutions to the knapsack problem $H = \{e_j\}_{j\in\mathcal{J}^f}$, where e_j is the j'th column og the $|\mathcal{J}^f| \times |\mathcal{J}^f|$ identity matrix.

Step 2: Set up the linear column generation master problem

$$\begin{aligned} \min \ & \sum_{h\in H} \gamma_h \\ \text{s.t.:} \ & \sum_{h\in H} x_i^h \gamma_h \geq \underline{x}_{ij}, \quad \forall j \in \mathcal{J}^f \\ & \gamma_h \geq 0 \end{aligned} \tag{2.3}$$

where H is an index set of solutions satisfying the knapsack constraint $\sum_{j\in\mathcal{J}^f} d_j x_{ij} \leq \overline{s}$ generated in step 5. If $l = 0$, then $H = \emptyset$.

Step 3: Solve the linear column generation master problem and denote an optimal primal–dual pair (γ^l, λ^l).

Step 4: If $\sum_{j\in\mathcal{J}^f} \underline{x}_{ij}\lambda_j^l \leq 1$, no violated cutting plane exists. Therefore stop.

Step 5: Solve the binary knapsack problem

$$Z_{kp} = \max\{ \sum_{j\in\mathcal{J}^f} \lambda_j^l x_j \ : \ \sum_{j\in\mathcal{J}^f} d_j x_j \leq \overline{s}, \quad x_j \in \{0,1\}\}$$

and denote an optimal solution x^l.

Step 6: If $Z_{kp} \leq 1$ go to step 7. Otherwise, the inequality $\sum_{j\in\mathcal{J}^f} \lambda_j^l x_{ij} \leq 1$ is not valid for

$$\text{proj}_x\Big(\mathcal{X}'_{iy}\Big) \cap \{\mathbf{x}_i \in \mathbb{R}^{|\mathcal{J}|} \ : \ x_{ij} = 1, \ \forall j \in \mathcal{J}^1 \text{ and } x_{ij} = 0, \ \forall j \in \mathcal{J}^0\}$$

Therefore add the index l of the solution x^l to H, set $l \coloneqq l + 1$ and go to step 2.

Step 7: Down–lift all variables in $\mathcal{J}^1$ and up–lift all variables in $\mathcal{J}^0$.

Step 8: Output the violated cutting plane $\pi\mathbf{x}_i \leq \pi_0 y_i$.

Algorithm 2.4: Column generation procedure for generating a Fenchel cutting plane for capacity constraint i.

in Step 5 of Algorithm 2.4, one should note that the cost coefficients λ^k are fractional, and therefore numerical problems can arise. In order to overcome this, we simply multiply the coefficients with a large scalar and then round the coefficients to the nearest integer, which allows to use the highly efficient COMBO algorithm (Martello, Pisinger, and Toth, 1999) for solving the binary knapsack problems. When it is no longer possible to find violated inequalities this way, we submit the generated cutting plane to the rounding procedure proposed by Kaparis and Letchford (2010) and check the result for validity. If this is the case, the cutting plane is returned, otherwise a weakened inequality is returned. The cutting plane generated by the row generation procedure described in Algorithm 2.4 is, after lifting the fixed variables, of the form $\pi^T x \leq \pi_0$. Again, we translate the resulting cutting plane by multiplying the right–hand side by y_i and obtain $\pi^T x \leq \pi_0 y_i$ in Step 8.

2.3.2 Phase 2 – Local branching

Local branching (LB) is in principle a technique for finding exact solutions to *mixed integer programs* (MIP). It can, however, easily be converted into a heuristic, e.g. by setting a limit on the time used for solving a given instance or a limit on the number of improving solutions. The technique uses a general purpose MIP–solver to solve restricted subproblems of the original problem instance defined by linear *local branching constraints*, which in turn are derived from a feasible solution. For a detailed introduction to LB, the reader is referred to Fischetti and Lodi (2003) and Fischetti, Polo, and Scantamburlo (2004). The two level LB heuristic developed in this paper is summarized as follows; First, an initial feasible solution is found by exploiting the LP-solution, second, the locational decisions are refined by local branching, and finally the allocation of customers is determined, likewise by local branching.

Initial feasible solution

First, an initial feasible solution is found. Fischetti and Lodi (2003) argue to let the general purpose MIP–solver find an (almost) random initial solution. In our case, however, it seems more appropriate to guide the search from the start. We utilize the information available from the cutting plane algorithm to this end. Let $(\underline{x}, \underline{y})$ be the fractional solution from the last iteration of the cutting plane algorithm. Assuming that the probability of facility i being open in an optimal solution to the SSCFLP is higher when $y_i > 0$ than when $y_i = 0$, we add a constraint of the form

$$\sum_{i:\underline{y}_i=0} y_i \leq h_1,$$

for a small integer value of h_1. This constraint simply states that at most h_1 of the facilities not used in the fractional solution can be used in a heuristic solution to SSCFLP. The general

purpose MIP–solver is then used to solve the resulting problem. The MIP-solver is stopped if a heuristic stop criterion is met (see Section 2.3.2). The solution returned is denoted (x^0, y^0).

Refining the locational decision

Next, based on the initial solution (x^0, y^0), we add a local branching constraint for the location variables of the form

$$\sum_{i\in\mathcal{I}} |y_i^0 - y_i| \leq h_2 \Leftrightarrow \sum_{i:y_i^0=0} y_i + \sum_{i:y_i^0=1} (1 - y_i) \leq h_2,$$

to the formulation of SSCFLP, where h_2 is a small integer. The problem is now solved with the additional requirement that only improving solutions are accepted. Again, a heuristic stop criterion is used to end the optimization (see Section 2.3.2). The resulting solution is denoted (x^1, y^1). After this refinement of the locational decision, all location variables where $y_i^1 = 0$ are fixed to zero. Note that only a single branch is created, meaning that a heuristic fixation of non-promising location variables is done rather than an actual local branching.

Refining the allocation decision

Having almost fixed the location variables (we are still allowed to close open facilities), we consider the allocation variables. Summing up the assignment constraints over $j \in \mathcal{J}$ we have for any feasible solution to SSCFLP that

$$\sum_{j\in\mathcal{J}} \sum_{i\in\mathcal{I}} x_{ij} = |\mathcal{J}|.$$

We can therefore define a k-opt neighborhood around the allocation corresponding to x^1 by means of the local branching constraint

$$N(x^1, k): \quad \sum_{(i,j):x_{ij}^1=1} x_{ij} \geq |\mathcal{J}| - k, \tag{2.4}$$

for a given positive integer k. Adding this constraint will significantly reduce the solution space, but the problem can still be too hard to solve to optimality. Therefore the search is stopped prematurely if a heuristic stop criterion is met. If an improving solution is found, say (x^2, y^2), the constraint (2.4) is removed and a reversed local branching constraint of the form

$$\sum_{(i,j):x_{ij}^1=1} x_{ij} \leq |\mathcal{J}| - k - 1,$$

is added to the problem. This constraint removes the neighbourhood $N(x^1, k)$ from further consideration. The procedure is repeated and the problem solved in iteration n is given by

$$\mathcal{P}([DCTI \cap N(x^n, k)] \setminus \bigcap_{l=1}^{n-1} N(x^l, k)).$$

If an iteration does not lead to any improvement, the best solution found so far is returned.

Heuristic stop criteria

Although the local branching constraints significantly reduce the size of the problem, solving every subproblem to optimality is usually too time–consuming. Instead we set a limit on the time and an upper bound for the number of improving solutions for the MIP-solver. Ideally, these limits should depend on the problem data, but fixed limits seem to work well in practice.

We do, however, distinguish between local branching on the location variables and on the allocation variables. As the neighborhoods searched when refining the location decisions are much larger than those examined for the allocation variables, we set a time limit of t_y seconds for those subproblems and a time limit of $0 < t_x < t_y$ for the allocation refinement.

When refining the locational decisions, we do not impose any upper bound on the number of improved solutions, because the MIP-solver usually finds a large number of improving solutions for the locational decisions in short computation times. For the allocation variables, we set a limit of $1 < k_x < \infty$ improving solutions, because in most cases only a few improving solutions exist in the much smaller neighborhoods of the current allocation.

2.3.3 Phase 3 – Cut–and–solve

The cut–and–solve framework is to some extent very similar to LB as it also adds invalid linear constraints to the problem in order to reduce its size. It is essentially a branch–and–bound algorithm which branches on a set of variables instead of branching on single fractional variables. At each level in the search tree (see Figure 2.1) there are only two nodes. The left node is associated with the linear constraint stating that the sum of the variables in a set Ω is less than or equal to an integer γ. This subproblem is called the *sparse problem* (SP). The right node is associated with the sum being larger than or equal to $\gamma + 1$ and this problem is called the *dense problem* (DP). The constraint associated with the DP is called a *piercing cut* and these cuts are accumulated. An obvious choice for binary programs is to set the parameter γ equal to zero as this completely fixes the variables in the set Ω to zero in the SP. If that choice is made, it is often possible to use a general purpose MIP–solver to solve the sparse problem. The optimal solution to the SP provides an upper bound on the optimal solution value to SSCFLP (if the SP shows a feasible solution). If this upper bound improves the incumbent, it is updated. Concurrently, a lower bound, $\underline{Z}$, for the DP is obtained from a relaxation of the DP. If the lower bound for the DP is not smaller than the value of the incumbent, we know that no improving solutions can exist in the yet unexplored solution space defined by the DP, and the incumbent is proven optimal. This procedure is

Step 0 Obtain a lower bound of the dense problem.

Step 1 Select a piercing cut.

Step 2 Find optimal solution in space removed by the piercing cut (sparse problem).
Update the incumbent if necessary.

Step 3 If lower bound $\geq$ incumbent, return the incumbent.

Step 4 Add piercing cut to dense problem and go to Step 0.

Algorithm 2.5: Generic cut–and–solve algorithm.

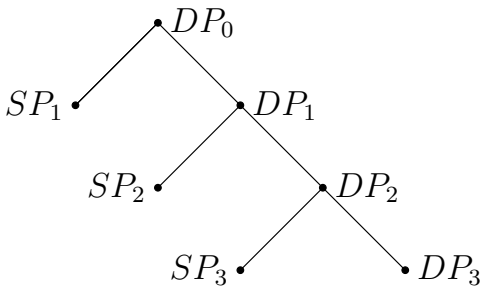


Figure 2.1: Cut–and–solve search tree.

repeated until an optimal solution is found (using the convention that a lower bound on an infeasible DP is equal to infinity, this procedure will return an optimal solution). For a more detailed introduction to the cut–and–solve framework, the reader is referred to Climer and Zhang (2006).

Relaxation and piercing cuts

The way the DP is relaxed and piercing cuts are derived is of utmost importance for the efficiency of the cut–and–solve approach. First of all, we want to use a relaxation that gives strong lower bounds. The lower bound should also increase rapidly as piercing cuts are added. Climer and Zhang (2006) suggest to use the linear programming relaxation and to use reduced costs to define the piercing cuts. The problem is, however, that the linear programming relaxation of SSCFLP is often highly degenerate and adding piercing cuts does not improve the lower bound fast enough to obtain an efficient algorithm. We therefore adopt the *partial integrality strategy* proposed by Yang et al. (2012). This strategy means that we preserve the integrality of the location variables and relax the allocation variables, leading to a CFLP as a relaxation of the SSCFLP. Defining the set Ω can therefore be done in a straightforward way by letting

$$\Omega = \{i \in \mathcal{I} \,:\, \underline{y}_i = 0\}, \tag{2.5}$$

where $\underline{y}$ is an integer solution to the location variables in the DP and the piercing cut becomes $\sum_{i\in\Omega} y_i \geq 1$. This definition is mainly motivated by the assumption that an optimal solution

to the CFLP should have a number of attributes in common with an optimal solution to the SSCFLP. This assumption is supported by the computational results reported in Section 2.4. Furthermore, choosing Ω as in (2.5), many variables can be fixed in the SP as $\sum_{i\in\Omega} y_i = 0$ implies that $x_{ij} = 0$ for all $i \in \Omega$ and $j \in \mathcal{J}$.

Termination

When proving finiteness of the cut–and–solve algorithm, Climer and Zhang (2006) assume finiteness of the problem's feasible region and that all SPs have at least one feasible solution. The latter is, however, not necessarily true with the choice of Ω given in (2.5) as it might be impossible to create a single–source solution using the set of facilities used in the CFLP–relaxation. Termination is, however, easily proven in Proposition 2.3.

Proposition 2.3. *Assume that the piercing cuts are defined by the set Ω stated in* (2.5)*. Then the algorithm terminates if the algorithms for solving the dense and the sparse problems are guaranteed to terminate.*

Proof. First of all, the set of feasible facility constellations for the CFLP version of the SSCFLP is finite. There are $|\{\iota \subseteq \mathcal{I} : \sum_{i\in\iota} s_i y_i \geq D\}|$ such solutions. The piercing cut

$$\sum_{i:y_i=0} y_i \geq 1,$$

removes the current solution to the modified CFLP from further considerations (in fact it also removes all solutions which consist of a subset of the facilities used in the CFLP solution). As there is a finite number of solutions to the locational variables of the CFLP, the cut–and–solve algorithm terminates if the algorithms used to solve the dense and the sparse problems terminate. □

Variable fixing and pruning of nodes

As the open facilities in the solution to the DP often constitute a minimal cover of the total demand, it is usually possible to fix many of the location variables in the SP to a value of one in the following way. If

$$\sum_{i\in(\mathcal{I}\setminus\Omega)\setminus\{\tilde{i}\}} s_i < D,$$

for some $\tilde{i} \in \mathcal{I} \setminus \Omega$, the location variable $y_{\tilde{i}}$ can be fixed to one as the total demand cannot be covered otherwise. If all variables in $\mathcal{I} \setminus \Omega$ can be fixed to one, the SP reduces to a *generalized assignment problem* (*GAP*). Though GAP is also $\mathcal{NP}$-hard, the only remaining decision is to decide on the allocation of customers, whereby the effort for solving the SP is reduced.

Note that it is not necessary to find an optimal solution to the SP in each iteration. It suffices to prove that no improving solution exists in the space of solutions to the SP. The optimization of the SP can then be prematurely terminated if a lower bound of this SP equals or exceeds the value of the incumbent. For the same reason, the last DP can be stopped prematurely as soon as it can be proven that its optimal solution does not fall below that of the incumbent.

Cutting planes for the SP

For solving the DPs and SPs efficiently, the cutting planes generated in phase one of the algorithm are all appended to these programs. Although the cuts generated for the capacity constraints are not valid for the CFLP version of the SSCFLP, they are valid for the SSCFLP. The modified CFLP is therefore still a relaxation of the SSCFLP. The variable fixing (2.5) defined by the solution to the DP may, of course, remove the part of the solution space for which these cuts were generated. If this is the case, the cuts are only of limited use in the SP. We thus include additional cutting planes at the root node of each SP so that these problems can be solved efficiently. The cutting planes are generated in the same way as in the cutting plane algorithm shown in Algorithm 2.2. The implied variable bounds are not used as they seem to contribute little to improvements in the lower bound at this stage. The reason for this lack of effect is simply that many location variables can be fixed to one a priori, making the implied bound $x_{ij} \leq y_i$ redundant.

2.4 Computational experiments

In this section, the computational efficiency of the cut–and–solve approach is first tested by comparing it to a commercial state–of–the–art software. Secondly, we test our algorithm against the cut–and–solve algorithm developed by Yang et al. (2012), which to the best of our knowledge is one of the most efficient algorithms for solving the SSCFLP. Finally, our algorithm is tested on a set of new instances such that some insight can be obtained into which parameter settings constitute difficult instances.

2.4.1 Implementation detail

The algorithm developed in this paper uses the linear programming solver and the branch–and–cut framework provided by ILOG CPLEX Concert Technology 12.3 (*CPLEX*). All programs have been coded in C++ and C languages and compiled using g++ and gcc, respectively, with optimization option O2. All experiments were carried out on a Dell Vostro

3450 laptop with 4 GB RAM and a 2.5 GHz Intel® Core i5-2450M processor running a 32 bit version of Ubuntu 12.04.

In CPLEX, the ParallelMode switch is set to deterministic such that the running times for different instances can be compared. Moreover, in Phase 1, the default settings are used to solve the linear programming problems in each iteration of the cutting plane algorithm. This means that CPLEX is allowed to preprocess the problem instance before the separation routines are used. The separation routines are called for each knapsack constraint (capacity or total demand) if the slack is below a certain threshold. A rather large slack is permitted, as good results were obtained by including separating hyperplanes even for capacity constraints showing large values of the slack variables at the current LP solution.

CPLEX is also used as MIP solver for solving the restricted problems in Phase 2. As mentioned in Section 2.3.2, the parameters h_1, h_2 and k that determine the neighborhoods's sizes are fixed independently of the problem data and attain the values 0, 1 and 5, respectively.

In order to solve the dense problem in Phase 3, CPLEX is applied with the MIPSearch set to "Traditional Branch-and-Cut". The relative and the absolute optimality gaps are both set to zero. All other settings are at their default values. All cuts generated in Phase 1 are appended to the dense problem. CPLEX is also used for solving the sparse problem. Settings are as for the dense problem, except that the absolute optimality gap is set to 0.99 due to integral data. Furthermore, we branch on the location variables before branching on the assignment variables. As the SPs often require memory exceeding the RAM capacity, the NodeFileInd switch is set to 3 to allow for the branching tree to be written to a file on the disk. All cuts generated in Phase 1 are added to this program as well, and furthermore, we use a cut callback to separate GLCIs and Fenchel cutting planes in order to strengthen the lower bound of the sparse problem. The cut generation is limited to the root node of the SP as experience has shown that cutting planes are more effective at nodes in the top of the branching tree.

2.4.2 Test instances

The solution approach has been evaluated on four different test beds, denoted TBi, $i = 1, \ldots, 4$. TB1 consists of 57 instances from Díaz and Fernández (2002) ranging from small instances with 10 potential facility sites and 20 customers to larger instances with 30 potential facility sites and 90 customers. These instances are publically available at `http://www-eio.upc.es/~elena/sscplp/index.html`. TB2 consists of 71 instances used by Holmberg et al. (1999) with problems consisting of 10 facility sites and 50 customers up to problems with 30 facility sites and 200 customers. These instances are available at `http://www.mai.liu.se/~kahol/problemdata/cloc/`. The third test bed, TB3, consists of relatively large problem instances reported in Yang et al. (2012). The sizes of these problems range from 30 facility

sites and 200 customers to 80 facility sites and 400 customers.

TB4 is a new set of problem instances generated for this paper. It tries to mimic the situation where large fixed opening costs lead to small production costs (e.g. more efficient and therefore expensive machinery leads to lower production costs). The problem instances have been generated so as to be similar to instances in the literature on the CFLP and the SSCFLP (see for example Cornuejols et al. (1991), Klose and Görtz (2007) and Yang et al. (2012)). That is, demands and capacities are uniformly distributed in the intervals $[5, 35]$ and $[10, 160]$, respectively. Then the capacities are scaled so as to obtain a specific ratio between total capacity and total demand. Fixed opening costs are generated using the formula

$$f_i = U(0, 90) + \sqrt{s_i} U(100, 110)$$

where $U(a, b)$ denotes the uniform distribution of the set $\{a, a + 1, \ldots, b - 1, b\}$. Finally, the assignment costs c_{ij} are usually determined as follows: generate facility sites and customers so as to be uniformly distributed points in the unit square and set $c_{ij} = \lfloor\ 10\ \delta(i, j)\ \rfloor$, where $\delta(i, j)$ denotes the Euclidean distance between facility point i and customer point j and $\lfloor \delta(i, j) \rfloor$ denotes the largest integer no greater than $\delta(i, j)$. For the problems in TB4, however, we will let this number denote the transportation cost of delivering customer j's demand from facility i. That is $t_{ij} = \lfloor\ 10\ \delta(i, j)\ \rfloor$. The production cost of producing d_j units of the desired product at facility i is then calculated as

$$p_{ij} = \left\lfloor \frac{2 \max_{i \in I} f_i}{f_i} \right\rceil d_j$$

where $\lfloor \cdot \rceil$ means rounded to nearest integer. This specification has been made in order to reflect the idea that items may be produced at a lower unit price in an expensive facility. Another implication is that low opening costs for a facility will lead to high supply costs, thus making it harder to determine the "right" set of facilities. The assignment cost is then set to be $c_{ij} = t_{ij} + p_{ij}$, that is transportation plus production costs. In order to make the percentage optimality gap, given by

$$gap = \frac{\text{best integer} - \text{best lower bound}}{\text{best lower bound}} 100$$

more "honest", the assignment costs are scaled as follows

$$c_{ij} = c_{ij} - \min_{i \in \mathcal{I}} c_{ij}$$

which will decrease the magnitude of the solution values. The assignment cost c_{ij} can then be interpreted as the cost that should be paid if customer j is assigned to facility i instead of the cheapest alternative.

These four test beds range from small over medium–sized to large–sized problems. In the instances of TB2, the assignment costs dominate the fixed opening costs whereby an optimal

Table 2.1: Abbreviations used in the column headings of the result tables

Heading	Description
ID	The ID of the problem under consideration.
$\|\mathcal{I}\| \times \|\mathcal{J}\|$	Number of facilities and customers in the instances.
r	Ratio between total capacity and total demand.
%-time in phase	Displays the percentage of the total cpu time used in Phase 1, 2, and 3.
$\underline{Z}$, $\overline{Z}$, Z^*	Denotes the lower bound obtained from the first dense problem, best upper bound found using the local branching heuristic and the optimal solution value, respectively.
LB_{gap}	$\frac{(Z^*-\underline{Z})}{\underline{Z}}100$. That is, a measure of the quality of the lower bound.
UB_{gap}	$\frac{\overline{Z}-\underline{Z}}{\underline{Z}}100$. That is, a measure of the quality of the upper bound.
cpu	The average cpu time used by our algorithm to solve the instances in seconds.
CPU	The cpu time, in seconds, used to solve a single instance using our algorithm.
ρ	The ratio between the cpu time used by another algorithm and the cpu time used by the improved cut–and–solve algorithm of this paper.

solution tends to include more facilities. In the rest of the test beds, the fixed opening costs dominate the assignment costs, and the optimal solution will therefore often open as few facilities as possible. Furthermore, a new cost structure for the allocation costs is used in TB4. As a result, the instances in TB1-TB4 cover many different scenarios and it should be possible to evaluate the effectiveness of our three–phase algorithm.

Table 2.1 simplifies the reading of the tables by summarizing the column headings.

2.4.3 The efficiency of cut–and–solve

In order to test the efficiency of the cut–and–solve algorithm in Phase 3, we have implemented a solution procedure that first uses our cutting plane algorithm and then the traditional branch–and–cut algorithm provided by CPLEX. Furthermore, the optimal solution value is used as an upper cut-off value in CPLEX in order to mimic a very strong and fast heuristic. All other settings are at their default values allowing CPLEX to generate cuts and preprocess the instance.

Table 2.2 presents the aggregated results obtained using the combination of our cutting plane algorithm and the branch–and–cut algorithm. It is evident that the branch–and–cut algorithm performs well for the small test instances of TB1 and TB2. The ρ-value shows that for $TB2$ the branch–and—cut algorithm generally outperforms the cut–and–solve algorithm

Table 2.2: Results obtained using branch–and–cut instead of cut–and–solve.

				ρ		
TB	$\lvert\mathcal{I}\rvert \times \lvert\mathcal{J}\rvert$	r	cpu	min	mean	max
1	$10 \times 20 - 30 \times 90$	1.27-5.17	10.04	0.29	5.28	133.20
2	$10 \times 50 - 30 \times 200$	1.37-8.28	1.92	0.23	0.78	2.44
3	$30 \times 200 - 80 \times 400$	1.73-7.30	401.62	1.38	136.47	1652.70
4	$50 \times 100 - 60 \times 300$	2.00-5.00	6,442.88	0.56	10.52	140.42

over the entire test bed. This is due to the small size of these problems; it is less time consuming to solve one integer program than it is to alternate between solving a MIP corresponding to the DP and the restricted MIP defined by the SP. The reason why the branch–and–cut algorithm performs worse relative to our algorithm in TB1 than in TB2 is that the integrality gap between the value of the LP relaxation and the optimal solution value to SSCFLP is larger in TB1 than in TB2. As the cutting plane in conjunction with the partial integrality strategy provides better lower bounds than the LP relaxation, the gap is closed faster.

When we look at TB3 and TB4, it is apparent that the size of the problems in TB3 and the tight capacity to demand ratios of TB4 make the branch–and–cut algorithm much more time consuming than the cut–and–solve. On average, the branch–and–cut algorithm requires about 135 times more cpu time than the cut–and–solve to solve the instances in TB3. Even though the instances in TB4 exhibit a relatively small capacity to demand ratio, implying larger SPs, our algorithm runs 10 times faster than the branch–and–cut algorithm provided by CPLEX on average. Furthermore, the branch–and–cut algorithm was not able to solve a number of instances (instances n16, n19, n20 and n35) within a time limit of 50,000 cpu seconds. These instances were solved by the three phase algorithm within half that time.

We therefore conclude that for small instances with a small integrality gap the cut–and–solve approach is comparable to CPLEX' branch–and–cut algorithm and for large instances our cut–and–solve algorithm clearly outperforms CPLEX. Furthermore, the authors would like to note that for some of the instances in TB4 the memory requirement for the branch–and–cut algorithm exceeded 80 GB before the time limit was exceeded; for the cut–and–solve algorithm it never exceeded 7 GB.

2.4.4 Computational results of TB1–TB4

This section compares our improved cut–and–solve algorithm to the one proposed by Yang et al. (2012). To distinguish the two algorithms, we denote the improved cut–and–solve

algorithm developed in this paper `Imp-CS` and the cut–and–solve algorithm developed in Yang et al. (2012) is denoted `Y-CS`. As Mr. Zhen Yang kindly provided the implementation of the algorithm proposed in Yang et al. (2012), and as the two algorithms have been run on the same machine, the CPU-times are truly comparable. We should mention, that `Y-CS` originally considered the implied bounds $x_{ij} - y_i \leq 0$ as a part of the formulation of the SSCFLP. However, we noticed that this slowed down the `Y-CS` significantly. Therefore, we changed the code slightly so that the implied variable bounds were added on the fly like by the `Imp-CS`. The results obtained for TB1 to TB4 are displayed in Table 2.3. It is apparent that the lower and upper bounds obtained before solving the first sparse problem are very close to the optimal solution in almost all cases. Except for the first subsets of test instances in TB1, the average deviation of the lower bound obtained from the first dense problem from the optimal solution never exceeds 0.5 percent. The larger gaps in the two first subsets of TB1 mainly stem from two instances, namely d1 and d7, respectively. The absolute gaps in these two instances are relatively small, but as the objective function values are not that large for these instances either, the percentage gap becomes larger. In 4 and 22 instances of TB1 and TB2, respectively, the cutting plane algorithm was able to close the integrality gap and found an integer solution whereby Phases 2 and 3 were never entered. This never happened for any instances in TB3 and TB4.

The simple local branching heuristic seems to work well; for each test bed it produces solutions deviating less than 1.09 percent from the lower bound on average. In fact, for test beds TB1, TB2, and TB3, the deviation is less than one percent in each test bed (see Table 2.3, the column denoted UB_{gap}). This indicates that even though the heuristic is relatively simple, it may be used as a stand–alone heuristic with a known quality of the computed solution. In fact, in 86 of the 193 instances tested here, the local branching heuristic found the optimal solution. It should be mentioned, however, that in five instances of TB4 the heuristic failed to find a feasible solution (see Section 2.4.5 for an elaboration of the results on TB4).

Regarding the percentage of the time spent in the three phases, for the smaller instances of TB1 and TB2 most of the time is spent in the first two phases, while in TB3 and TB4 the majority of the time is spent in Phase 3. There are two reasons for this result. First, even though the lower bound percentage deviation from the optimal solution is quite small in TB3, the absolute gaps are relatively large. This is due to the larger objective function values in TB3 than in TB1 and TB2. In fact, the average optimal objective function values in TB1 and TB2 are almost 40 and 60 percent smaller than those in TB3. For this reason, large absolute gaps can be hidden in the "percentage deviation"-measure. Second, as the problem size grows, so does the time-consuming SPs which need to be solved in Phase 3, leading to more time spent in this phase. Note also that the time spent in Phase 3 in TB4

Table 2.3: Aggregated results on the four test beds when our algorithm is compared to the algorithm proposed by Yang et al. (2012)

						%-time in phase				ρ		
TB	ID	$\lvert\mathcal{I}\rvert \times \lvert\mathcal{J}\rvert$	r	LB_{gap}	UB_{gap}	1	2	3	cpu	min	mean	max
1	d1-d6	10×20	$1,32 - 1,54$	1.15	1.53	19.0	35.6	45.4	0.90	1.72	4.68	8.53
	d7-d17	15×30	$1.33 - 3.15$	0.51	0.72	45.0	36.7	18.3	1.47	2.00	8.64	37.94
	d18-d25	20×40	$1.30 - 3.93$	0.13	0.46	46.6	27.3	26.1	3.31	1.65	12.27	38.25
	d26-d33	20×50	$1.27 - 4.06$	0.16	0.55	41.9	37.9	20.2	36.58	0.67	10.20	30.21
	d34-d41	30×60	$1.64 - 5.17$	0.16	0.28	43.2	26.8	30.0	5.42	4.00	12.46	35.17
	d42-d49	30×75	$1.43 - 3.01$	0.03	0.13	40.9	28.7	30.3	14.29	2.02	15.25	30.08
	d50-d57	30×90	$1.49 - 3.47$	0.08	0.20	52.8	25.7	21.5	8.88	3.26	25.94	74.42
			Avg.	0.30	0.53	40.6	28.5	30.9	9.99		12.84	
2	h1-h12	10×50	$1.37 - 2.06$	0.04	0.05	55.4	35.0	9.6	0.20	2.19	4.60	13.66
	h13-h24	20×50	$2.77 - 3.50$	0.01	0.08	69.7	16.6	13.7	0.34	0.65	2.10	8.06
	h25-h40	30×150	$3.03 - 6.06$	0.11	0.23	58.2	24.5	17.3	2.61	1.23	33.40	122.73
	h41-h55	30×100	$1.52 - 8.28$	0.10	0.20	48.1	31.5	20.4	0.67	0.46	24.40	67.20
	h56-h71	30×200	$1.97 - 3.95$	0.08	0.22	60.8	16.5	22.7	5.29	1.52	21.41	116.44
			Avg.	0.07	0.17	60.6	24.7	14.7	2.01		18.64	
3	y1-y5	30×200	$1.73 - 1.98$	0.10	0.78	20.1	40.8	39.1	51.00	4.81	28.11	48.60
	y6-y10	60×200	$2.88 - 3.49$	0.16	0.78	8.5	8.2	83.3	1261.82	0.87	29.54	52.00
	y11-y15	60×300	$3.42 - 5.78$	0.07	1.02	32.0	17.9	50.1	65.63	42.17	114.50	212.97
	y16-y20	80×400	$3.50 - 7.30$	0.05	0.68	27.4	15.0	57.6	228.01	27.49	169.62	370.80
			Avg.	0.10	0.82	22.0	20.5	57.5	401.62		85.44	
4	n1-n5	50×100	2	0.02	—	5,8	26,4	67,8	1078.94	0.84	6.12	18.72
	n6-n10		3	0.02	0.97	7,2	17,6	75,2	931.25	2.34	6.33	9.69
	n11-n15		5	0.02	0.76	45,2	15,2	39,6	16.83	11.77	59.93	127.41
	n16-n20	50×200	2	0.08	—	6,3	7,3	86,4	7871.77	9.72	14.03	18.33
	n21-n25		3	0.01	—	12,6	32,8	54,6	4221.90	5.48	12.17	23.74
	n26-n30		5	0.01	0.13	73,6	13,0	13,4	14.32	42.88	101.34	150.92
	n31-n35	60×300	2	—	—	0,1	0,5	99,4	42938,86	—	—	—
	n36-n40		3	0.01	—	12,4	34,7	52,9	863,94	0.58	20.28	44.14
	n41-n45		5	0.02	—	60,8	22,7	16,5	48.11	79.57	138.38	203.69
			Avg.	0.03	1.09	24,9	18,9	56,2	6442,88		43.11	

decreases as the capacity to demand ratio increases. The rather obvious explanation is that as the ratio increases, fewer facilities are needed to cover the demand, and the SPs become smaller and thereby easier to solve.

Comparing the running times obtained using `Imp-CS` to the ones obtained using `Y-CS`, we observe that in all four test beds the average running time of `Imp-CS` is just a small fraction of that obtained by `Y-CS`. In TB1 and TB2, `Y-CS` algorithm runs faster than `Imp-CS` in one out of 57 and in seven out of 71 instances, respectively. `Y-CS` usually only outperforms `Imp-CS` in cases where the cutting plane algorithm is capable of closing the integrality gap, thus eliminating the need for evoking phases 2 and 3. This is probably due to the fact that the Fenchel cuts generated by Yang et al. (2012) are deeper than those generated here. Even though these deeper cuts come at a price in terms of longer computation times, this does not make that large an impact on these small instances, and the gap can be closed faster. Considering TB3 and TB4, the difference between the two algorithm becomes more apparent. On average, `Y-CS` uses about 85 and almost 45 times more computation time compared to `Imp-CS` on the instances in TB3 and TB4, respectively. Thus, the improved cut-and-solve algorithm `Imp-CS` achieves a very significant decrease in the running times.

We ascribe the large differences in running times to five primary improvements: 1) the way in which cutting planes are generated and the relaxation is strengthened; 2) the place where the cutting planes are added in the cut-and-solve tree; 3) the definition of the piercing cuts; 4) the pruning of sparse problems before an optimal solution has been established; and 5) the strong upper bounds generated in Phase 2. In Phase 1 of the `Imp-CS` algorithm, we generate the Fenchel cuts in a different way than is done by Yang et al. (2012); First of all, we only consider the fractional support of the LP solution in the separation problem and we use Corollary 2.3 to exclude the y-variables from the separation problems. This significantly reduces the size of the linear master problem in Algorithm 2.4. In addition to this, we solve the separation problem for the Fenchel inequalities using column generation instead of the traditional row generation procedure, which reduces the size of the basic matrix of the linear master problem. We also noticed that the main bottleneck of the cut–and–solve algorithm was the solution of the sparse problems and not the number of cut–and–solve nodes created. For that reason, we focused on reducing the effort for solving the sparse problems by means of additional cutting planes, instead of adding cuts to the dense problem in order to increase the lower bound. Furthermore, the way we define the sparse problems generally results in smaller sparse problems compared to those generated by `Y-CS`. Yang et al. (2012) use the solution from the dense problem to guide a heuristic to generate a feasible solution and let this feasible solution define the sparse problem. In this way, they can use the proof of termination given in Climer and Zhang (2006) as each sparse problem contains at least one feasible solution. We, however, use Proposition 2.3 and do this way not need each sparse

problem to contain a feasible solution. Furthermore, we use the observation that if a lower bound of the sparse problem exceeds the best known solution, no improving solution can be found in that cut–and–solve node, and we therefore stop the optimization of the sparse. This saves a significant amount of CPU time. Finally, as the sparse problem defined by `Imp-CS` are small, it is often hard to find a feasible solution to these problems. Without a good upper bound, solving the sparse problems would often be impossible as none of the branching nodes can be pruned before a feasible solution is known. The good upper bounds generated in Phase 2 are thus very helpful for solving the sparse problems. Actually, in the 86 cases where the local branching heuristic produced the optimal solution, no sparse problems needed to be solved to optimality due to the pruning rule.

2.4.5 Elaborated results on TB4

As mentioned previously, TB4 consists of problems where an extra "production cost" is added to the distance matrix. Three different problem sizes are considered, namely 50×100, 50×200 and 60×400. For each problem size, three sets of five instances showing a capacity to demand ratio of 2, 3, and 5 were generated. The results are presented in Table 2.4.

For these instances an extremely small gap between the lower bound and the optimal solution is observed; in none of the cases does the lower bound deviate more than one third of a percent. This is a very convincing result as the assignment costs have been scaled down in order to make the percentage-error measure more independent from the magnitude of the cost data. The two level local branching does, however, show some weaknesses. In five of the 45 instances, the heuristic is not able to generate a feasible solution, and for five other instances, a deviation of more than two percent from the lower bound is observed. Both the failure of finding a feasible solution and the large deviations are encountered in problems with a small capacity to demand ratio ($r = 2$ and 3). The main reason is that we use a general MIP-solver to find an initial solution, and when the capacity to demand ratio is small, a branching tree of a certain depth is often needed to find a feasible solution. On average, however, the local branching heuristic performs well showing an average deviation between lower bound and heuristic upper bound over the entire bed of about one percent, only.

In four of the 45 instances (instances n31-n34), the algorithm failed to find the optimal solution as the set maximum of 50,000 cpu seconds was reached and computation discontinued. The tendency is that the smaller the r-value, the more computation time is required for solving the instance. If the capacity to demand ratio gets smaller, more facilities are needed to cover the demand, implying that the size of the sparse problems increases as fewer facilities can be closed. It seems we have reached the limit of the problem size for the case $r = 2$ solvable by our algorithm. Note that the larger problem instances solved to optimality in

Table 2.4: Results on test bed TB4

			LB		UB			%-time in phase				
ID	$\lvert\mathcal{I}\rvert \times \lvert\mathcal{J}\rvert$	r	$\underline{Z}$	LB_{gap}	$\overline{Z}$	UB_{gap}	Z^*	1	2	3	CPU	ρ
n1	50×100	2	18,291.3	0.01	18,319	0.15	18,294	2.2	4.8	93.0	616.74	0.88
n2			19,684.3	0.06	20,590	4.40	19,688	0.8	4.5	94.7	3705.52	5.32
n3			19,073.2	0.01	—	—	19,075	7.2	43.8	49.0	192.32	4.34
n4			18,618.7	0.01	18,620	0.01	18,620	16.0	77.3	6.8	92.10	18.72
n5			18,498.0	0.02	18,871	1.98	18,502	2.5	1.7	95.8	788.01	1.40
n6		3	16,942.6	0.03	17,215	1.61	16,948	0.2	0.2	99.6	3887.94	2.87
n7			15,061.7	0.01	15,065	0.02	15,063	29.8	54.9	15.3	32.37	8.91
n8			15,103.5	0.02	15,372	1.78	15,107	2.8	16.9	80.3	170.56	7.87
n9			14,344.1	0.02	14,452	0.75	14,347	1.8	3.8	94.4	89.11	9.69
n10			14,808.8	0.03	14,912	0.70	14,813	1.1	12.3	86.6	476.28	2.34
n11		5	12,071.0	0.01	12,078	0.06	12,072	60.0	11.6	28.4	10.36	63.09
n12			11,893.9	0.03	12,014	1.01	11,898	7.1	5.6	87.3	46.46	11.77
n13			11,124.5	0.00	11,125	0.00	11,125	47.7	23.3	29.0	8.58	127.41
n14			11,815.8	0.01	12,000	1.56	11,817	69.6	15.8	14.6	10.05	36.67
n15			11,486.3	0.02	11,622	1.18	11,489	41.6	19.4	39.0	8.68	60.68
n16	50×200	2	25,989.1	0.01	27,272	4.94	25,992	2.6	8.9	88.5	2267.62	22.05
n17			25,866.5	0.01	26,132	1.03	25,868	22.5	17.8	59.7	122.87	9.72
n18			26,928.2	0.01	—	—	26,930	6.4	9.2	84.4	869.55	18.33
n19			25,950.4	0.11	26,120	0.65	25,980	0.1	0.2	99.7	14,790.36	3.38
n20			25,323.7	0.26	—	—	25,390	0.0	0.4	99.6	21,308.44	2.35
n21		3	20,697.0	0.02	20,787	0.43	20,701	1.3	0.8	97.9	1397.86	5.48
n22			22,018.8	0.01	—	—	22,021	0.1	0.4	99.5	19,588.60	2.55
n23			20,036.8	0.01	20,038	0.01	20,038	36.0	58.5	5.5	7.42	23.74
n24			20,594.3	0.00	20,941	1.68	20,595	15.6	14.3	70.1	28.08	8.44
n25			21,167.3	0.00	21,168	0.00	21,168	10.0	90.0	0.0	87.55	11.00
n26		5	16,658.7	0.00	16,659	0.00	16,659	69.8	30.2	0.0	11.54	42.88
n27			16,136.7	0.01	16,140	0.02	16,138	75.0	8.6	16.4	18.92	105.77
n28			17,754.6	0.00	17,804	0.28	17,755	70.1	7.9	22.0	15.37	124.45
n29			15,855.7	0.01	15,914	0.37	15,858	56.0	15.8	28.2	12.08	145.81
n30			16,883.7	0.00	16,884	0.00	16,884	97.3	2.7	0.0	13.68	82.68
n31	60×300	2	34,858.5	—	34,861	0.01	—	0.1	0.2	99.7	50,000	1
n32			36,543.5	—	36,742	0.54	—	0.1	0.3	99.6	50,000	1
n33			34,876.2	—	34,884	0,02	—	0.1	0.1	99.8	50,000	1
n34			34,817.6	—	36,057	3.27	—	0.1	0.3	99.6	50,000	1
n35			37,136.3	0.16	38,603	3.80	37,196	0.0	1.5	98.5	14,694.31	3.40
n36		3	27,901.2	0.01	—	—	27,903	0.6	2.4	97.0	3,492.47	0.58
n37			27,593.5	0.00	27,729	0.49	27,594	30.6	36.7	32.7	83,20	44.14
n38			29,229.5	0.01	31,101	6.02	29,231	5.6	52.3	42.1	393,35	5.02
n39			27,437.6	0.01	27,440	0.01	27,439	20.5	66.6	12.9	62.34	13.58
n40			28,030.3	0.01	28,190	0.57	28,033	4.8	15.4	79.8	288.35	38.10
n41		5	21,043.2	0.01	21,045	0.01	21,045	71.5	22.5	6.0	30.57	174.09
n42			22,587.6	0.01	22,924	1.47	22,589	46.0	7.3	46.7	29.51	203.69
n43			21,447.5	0.01	21,822	1.72	21,449	47.0	49.0	4.0	87.32	102.83
n44			21,464.6	0.01	21,466	0.01	21,466	72.8	21.8	5.4	34.17	79.57
n45			21,859.0	0.00	21,879	0.09	21,860	66.9	12.8	20.3	58.97	131.73
			Avg.	0.02		1.17		24.9	18.9	56.2	644.9	39.36

TB3 have a significantly larger ratio r than the unsolved instances in TB4. Considering the distribution of computation time across the three phases, it is evident that most of the time is spent in Phase 3. This is mainly due to the large time consumption in Phase 3 for the instances showing a small r-value. It is also worth noting that our way of generating the allocation costs c_{ij} differs from the way traditionally used in existing literature, but this seems to have almost no impact on the performance and thus our algorithm proves robust in this respect.

Comparing `Imp-CS` to `Y-CS`, we see that in only two out of 45 cases (instances n1 and n36) our algorithm does not perform as well as that of Yang et al. (2012). In the four unsolved instances, the time consumption exceeded 50,000 seconds and both `Imp-CS` and `Y-CS` were stopped. One should note that as the ratio r increases, so does the value of ρ, meaning that `Imp-CS` becomes more efficient relative to `Y-CS` as the cut–and–solve algorithm in Phase 3 becomes more relevant. This underlines the results obtained in Section 2.4.3, namely that our improved cut–and–solve algorithm is indeed very fast. Over the entire test bed, `Imp-CS` runs about 40 times faster than `Y-CS` on average, suggesting an efficient solution procedure.

2.5 Conclusions

We have proposed an algorithm to solve the single–source capacitated facility location problem to optimality. The proposed algorithm works in three phases: The first phase consists of a cutting plane algorithms based on knapsack separation used to strengthen the SSCFLP. Secondly, strong upper bounds are generated in Phase 2 by a local branching heuristic that starts by heuristically fixing location variables and then improves the allocation of customers. Finally, in Phase 3, an accelerated cut–and–solve algorithm is proposed to search for an optimal solution if one has not been found in the first two phases.

The computational results show that the lower bounds produced by the cutting plane algorithm in Phase 1 combined with the partial integrality strategy produces very strong lower bounds in relatively short computation times. Furthermore, the characterization of the facets of the integer hull of the capacity constraints facilitates a more efficient generation of very strong Fenchel cutting planes.

When we combine the strong lower bound with the simple LB heuristic, we get a provably very good solution in most cases. Although the heuristic is relatively simple by nature, combined with Phase 1 it could in fact work as a stand alone heuristic.

The accelerated cut–and–solve algorithm in Phase 3 also seems to be efficient in searching for an optimal solution. The lower bound of the dense problem increases fast and only a few time consuming sparse problems need to be solved. Adding cuts in the root node of every sparse problem also seems to have a positive effect on the performance of our algorithm.

Compared to the algorithm presented in Yang et al. (2012) our algorithm solves the problems significantly faster in almost all instances. Our new algorithm far outperforms their algorithm with running times 10 to 80 times faster, on average. As the algorithm by Yang et al. (2012) is considered state-of-the-art, this suggests a very efficient algorithm.

Directions for further research include the examination of the effect of generating cutting planes, not only for the knapsack-like structures, but also for substructures including the demand constraints. Furthermore, the proposed framework seems appropriate for other types of location problems as well, for example hierarchical location models with capacity constraints. Finally, the model for the capacitated facility location problems with modular distribution costs proposed in Correia, Gouveia, and Saldanha-da Gama (2010) exhibits many of the same features as the SSCFLP, and therefore an adaptation of the algorithm presented here might serve as a good solution method.

Acknowledgment

The authors are grateful to Mr. Zhen Yang for providing the code, enabling us to compare the two algorithms, and to Professor Kim Allan Andersen for insightful comments and suggestions.

Integrating cut–and–solve and semi–Lagrangean based dual ascent for the single–source capacitated facility location problem

This chapter is based on the paper Gadegaard (2016).
The research was conducted during the spring of 2016 and submitted to optimization-online.org in April 2016

Integrating cut–and–solve and semi–Lagrangean dual ascent for the single–source capacitated facility location problem

Sune Lauth Gadegaard[†]

[†]Department of Economics and Business Economics, Aarhus University, Denmark,
sgadegaard@econ.au.dk

Abstract

This paper describes how the cut–and–solve framework and semi–Lagrangean based dual ascent algorithms can be integrated in two natural ways in order to solve the single–source capacitated facility location problem. The first uses the cut–and–solve framework both as a heuristic and as an exact solver for the semi–Lagrangean subproblems. The other uses a semi–Lagrangean based dual ascent algorithm to solve the sparse problems arising in the cut–and–solve algorithm. Furthermore, we developed a simple way to separate a special type of cutting planes from what we denote the effective capacity polytope with generalized upper bounds. From our computational study, we show that the semi–Lagrangean relaxation approach has its merits when the instances are tightly constrained with regards to the capacity of the system, but that it is very hard to compete with a standalone implementation of the cut–and–solve algorithm. We were, however, able to increase the size of the instances solvable by almost 25 percent compared to methodologies proposed in the literature.

Keywords: capacitated location problem; single–source; semi–Lagrangean; cut–and–solve; dual ascent

3.1 Introduction

The *single–source capacitated facility location problem* (SSCFLP) is the problem of opening facilities and allocating each customer's demand to one single open facility, while respecting the capacity of the facilities. An optimal solution is a solution that minimizes the total cost incurred by opening facilities and assigning the demand of the customers to the open facilities. Despite the fact that the problem has a simple verbal description, it is strongly

$\mathcal{NP}$–hard and most research has consequently been devoted to heuristics. It is beyond the scope of this paper to review the vast literature on heuristics for the SSCFLP, and we will therefore limit the discussion of heuristics for the SSCFLP to papers dealing with Lagrangean or semi–Lagrangean heuristics.

Klincewicz and Luss (1986) relax the so–called capacity constraints in a Lagrangean manner where the Lagrangean subproblem becomes an uncapacitated facility location problem. Primal solutions are obtained by an add heuristic and a refinement heuristic improving the primal feasible solutions to the Lagrangean subproblem. If the assignment constraints are relaxed in a Lagrangean manner instead, Bitran, Chandru, Sempolinski, and Shapiro (1981) showed that the Lagrangean subproblem decomposes into an independent knapsack problem for each facility site. Sridharan (1993) uses this fact to derive a tight lower bound for the SSCFLP by maximizing the Lagrangean dual function using subgradient optimization. In order to obtain good primal solutions, a generalized assignment problem is solved over the open facilities in each iteration of the subgradient procedure which makes this approach unsuited for large problem instances. The capacitated concentrator location problem, which is equivalent to the SSCFLP, is studied in Pirkul (1987), where a lower bound is obtained by maximizing the Lagrangean dual function resulting when relaxing the demand constraints. In Beasley (1993) both the capacity constraints and the demand constraints are relaxed and different Lagrangean heuristics are compared. Barceló and Casanovas (1984) consider a slight variant of the SSCFLP where the number of open facilities is limited by a given constant K. They propose a Lagrangean heuristic that decomposes into two subproblems; a plant selection and an allocation problem. The heuristic relaxes the demand constraints in a Lagrangean manner and properties of the dual of the LP-relaxation are used to guide a heuristic computing the Lagrangean multipliers. Barceló, Fernández, and Jörnsten (1991) propose a Lagrangean decomposition approach that separates the demand constraints from all other constraints. The Lagrangean subproblem thus decomposes into two subproblems; one being identical to the one obtained when relaxing the demand constraints and one being a simple semi-assignment problem. Due to the integrality property of the latter subproblem, the Lagrangean decomposition bound is no stronger than the Lagrangean bound based on relaxation of the demand constraints. Barceló, Hallefjord, Fernández, and Jörnsten (1990) suggest, to relax the capacity constraints, including a total demand constraint (stating that the total capacity of the open facilities needs to cover the total demand). In addition, they generate simple cover inequalities from the total demand constraint on the fly during subgradient optimization. The added cover inequalities are then relaxed and consequently do not alter the Lagrangean subproblem, but may nevertheless contribute to an improved bound. The assignment constraints are also dualized by Cortinhal and Captivo (2003) and the solutions to the Lagrangean subproblem are transformed into a feasible solution by a

repair heuristics based on local search. A more recent Lagrangean relaxation based heuristic was proposed in Chen and Ting (2008), where the Lagrangean dual bound is approximated using subgradient optimization techniques and a multiple ant colony heuristic is used in each iteration in order to improve the best upper bound. We would also like to mention the heuristic dual ascent method based on a semi–Lagrangean relaxation for the *uncapacitated facility location problem* proposed in Monabbati (2014).

In addition to the computational complexity of the SSCFLP, the traditional arc–based formulation of the problem suffers from a weak LP–bound when the cost of opening facilities is large compared to the assignment costs. This is often the case for instances proposed in the literature and exact approaches have therefore traditionally relied on stronger lower bounds. Neebe and Rao (1983) and Díaz and Fernández (2002) therefore reformulate the problem in a Dantzig–Wolfe fashion and use a branch–and–price algorithm for solving the SSCFLP. Holmberg, Rönnqvist, and Yuan (1999) imbed a Lagrangean relaxation in a branch–and–bound framework and use a repeated matching heuristic to generate upper bounds. Recently, Yang, Chu, and Chen (2012) proposed a cut–and–solve method based on relaxing only assignment variables, effectively using the, also $\mathcal{NP}$–hard, capacitated facility location problem (CFLP) as a relaxation of the SSCFLP. In Gadegaard, Klose, and Nielsen (2016d) an improved cut–and–solve algorithm running in three phases was developed.

Recently, several papers have dealt with the so–called semi–Lagrangean relaxation approach for facility location problems. The approach was proposed in Beltran, Tadonki, and Vial (2006) for the p–median problem, and applied again to the uncapacitated facility location problem (UFLP) in Beltran-Royo, Vial, and Alonso-Ayuso (2012). Beltran et al. (2006) show that the semi–Lagrangean relaxation shows no duality gap, and that it therefore constitute an exact solution procedure. Both Monabbati (2014) and Jörnsten and Klose (2015) propose a surrogate relaxation of a set of constraints before dualizing them in a Lagrangean manner. This *condensed semi–Lagrangean relaxation* does not show any duality gap either, and the resulting Lagrangean dual problem becomes a one–dimensional optimization problem. Based on this observation Monabbati (2014) proposes a heuristic solution procedure from the (UFLP) and Jörnsten and Klose (2015) develop an exact dual ascent algorithm guided by primal feasible solutions.

The semi–Lagrangean approaches proposed in the literature have only been applied to location problems where primal feasible solutions can easily be recovered from the solutions to the semi–Lagrangean dual subproblem. These primal feasible solutions have then been used to guide the search for optimal multipliers. For the SSCFLP it is, however, an $\mathcal{NP}$–hard problem just to find a feasible solution, implying that guiding a dual ascent by feasible solutions can be prohibitive. In addition, the solution approaches for the SSCFLP proposed in the literature often use knapsack separation exclusively from the capacity constraints,

thereby ignoring the so called assignment constraints when generating cutting planes.

We wish to close the gap in existing literature by overcoming the above-mentioned obstacles and we contribute as follows:

1. We propose to separate a special kind of cutting planes from what we denote the *effective capacity polytope with generalized upper bounds* in order to strengthen the programs before applying the algorithms.
2. We develop a semi–Lagrangean based dual ascent algorithm which integrates the cut–and–solve framework, both as a heuristic and as an exact solver.
3. We propose an enhanced cut–and–solve algorithm that integrates the semi–Lagrangean based dual ascent algorithm as a solution procedure for the otherwise time consuming subproblems.
4. We test three different algorithms and three different schemes for initializing the semi-Lagrangean multiplier. We show empirically that the semi–Lagrangean based dual ascent algorithms performs very well on instances with a small ratio between the total capacity and the total demand.

The remainder of the paper is organized as follows: Section 3.2 introduces the semi–Lagrangean based dual ascent and the cut–and–solve algorithms for general integer linear programs. In Section 3.3 we show how the two aforementioned solutions procedures can be integrated while Section 3.4 adapts the solution methodologies to the single–source capacitated facility location problem. Section 3.5 reports on extensive computational experiments, and finally we conclude on our findings in Section 3.6.

3.2 Preliminaries

In this section we introduce the preliminaries for the remainder of the paper. In Section 3.2.1 we introduce the concept of semi–Lagrangean relaxation and describe two simple dual ascent algorithms based on this relaxation. Section 3.2.2 introduces the cut–and–solve framework. To that end, consider the generic integer linear program

$$\begin{aligned} Z^* = \min\ & c^T x \\ \text{s.t.: } & Ax = b \\ & x \in P \cap \{0,1\}^n \end{aligned} \tag{3.1}$$

where $P \subseteq \mathbb{R}^n_+$ is a polyhedron, A is a rational $m \times n$ matrix, and b and c are rational vectors of dimensions m and n, respectively. Unless otherwise stated, it will be assumed throughout the paper that (3.1) has an optimal solution.

3.2.1 Semi–Lagrangean relaxation

In traditional Lagrangean relaxation approaches one would introduce *Lagrangean multipliers* λ and relax the constraints $Ax = b$ in a Lagrangean manner in order to obtain a lower bound of (3.1) given by $L(\lambda) = \lambda^T b + \min\{(c - \lambda^T A)x \ : \ x \in P \cap \{0,1\}^n\}$. Although the traditional approach in many cases improves the bound compared to the LP relaxation bound, the duality gap is usually not closed by maximizing $L(\lambda)$. However, another approach called semi–Lagrangean relaxation was recently proposed in Beltran et al. (2006). By stating the equations $Ax = b$ as the two sets of inequalities $Ax \leq b$ and $Ax \geq b$ we obtain an equivalent formulation of (3.1). Relaxing the latter set of inequalities with multipliers $\lambda \in \mathbb{R}^m_+$ in a Lagrangean manner results in the *semi–Lagrangean relaxation*

$$\begin{aligned} L(\lambda) = \lambda^T b + \min \ & (c - \lambda^T A)^T x \\ \text{s.t.: } & x \in \mathcal{X}, \end{aligned} \tag{3.2}$$

where $\mathcal{X} = \{x \in P \cap \{0,1\}^n \ : \ Ax \leq b\}$. The Lagrangean dual problem then becomes the problem of finding multipliers $\lambda^* \in \mathbb{R}^m_+$ such that

$$\lambda^* \in \arg\max\{L(\lambda) \ : \ \lambda \in \mathbb{R}^m_+\}. \tag{3.3}$$

As the variables are positive, the system $Ax = b$ can also be described by the inequalities $Ax \leq b$ and the surrogate relaxation $e^T Ax \geq e^T b$ of the constraints $Ax \geq b$, where the vector $e \in \mathbb{R}^m$ is given by $e = (1, \ldots, 1)$. Dualizing the single constraint $e^T Ax \geq e^T b$ with multiplier $\mu \geq 0$ leads to the so–called *condensed semi–Lagrangean relaxation* given by

$$\begin{aligned} \mathcal{L}(\mu) = \mu e^T b + \min \ & (c - \mu e^T A)^T x \\ \text{s.t.: } & x \in \mathcal{X}. \end{aligned} \tag{3.4}$$

The condensed semi–Lagrangean relaxation was suggested by Monabbati (2014) and Jörnsten and Klose (2015). Compared to the semi–Lagrangean dual problem (3.3) the dual problem

$$\max\{\mathcal{L}(\mu) \ : \ \mu \geq 0\}, \tag{3.5}$$

has the clear advantage that it is one–dimensional. Preliminary tests confirmed the findings for the uncapacitated facility location problem reported in Jörnsten and Klose (2015), namely that the condensed semi–Lagrangean relaxation outperforms the ordinary semi–Lagrangean relaxation by an order of magnitude. Therefore this paper is limited to the study of the condensed semi–Lagrangean relaxation only. For that reason, we will henceforth use the term *semi–Lagrangean relaxation* for the relaxation (3.4) instead of the rather cumbersome *condensed semi–Lagrangean relaxation.*

In Theorem 3.1 we have stated some properties of the semi–Lagrangean function and the semi–Lagrangean dual problem which were originally proven in Monabbati (2014).

Input: Vectors c and b, matrix A, and feasible set $\mathcal{X}$.
Output: An optimal solution x^* to (3.1) and optimal multiplier $\mu^* \in \mathcal{M}$.

Step 1: (Initialization) Set $k = 1$ and initialize multiplier, μ^k, to a suitably small value.

Step 2: (Solve subproblem) Obtain an optimal solution x^k to the semi–Lagrangean subproblem

$$\mathcal{L}(\mu) = \min\{(c - \mu^k e^T A)^T x \ : \ x \in \mathcal{X}\}.$$

Step 3: (Termination check) If $Ax^k = b$, stop with (x^k, μ^k) as an optimal primal–dual pair.

Step 4: (Update) Let $\mu^{k+1} = \mu^k + \delta^k$, $\delta^k > 0$, and return to *Step 2*. Go to *Step 2*.

Algorithm 3.1: A dual ascent algorithm for a semi–Lagrangean relaxation of a general ILP.

Theorem 3.1 (Monabbati (2014))**.** *Let* $\mathcal{X}(\mu) = \arg\min\{(c - \mu e^T A)x \ : \ x \in \mathcal{X}\}$ *and let* $\mathcal{M} = \arg\max\{\mathcal{L}(\mu) \ : \ \mu \geq 0\}$. *Then the following holds true*

1. $\mathcal{L}(\mu)$ *is monotone and* $\mathcal{L}(\mu') \geq \mathcal{L}(\mu)$ *if* $\mu' \geq \mu$. *The inequality is strict if* $\mu' > \mu$ *and* $\mu' \notin \mathcal{M}$.
2. *If* $x^* \in \mathcal{X}(\mu)$ *and* $Ax^* = b$, *then* $\mu \in \mathcal{M}$ *and* x^* *is optimal for* (3.1).
3. *Conversely, if* $\mu^* \in int(\mathcal{M})$, *then every* $x \in \mathcal{X}(\mu^*)$ *is optimal for* (3.1) *(where* $int(\mathcal{M})$ *denotes the* interior *of the set* $\mathcal{M}$*).*
4. $\max\{\mathcal{L}(\mu) \ : \ \mu \geq 0\} = Z^*$, *that is, the semi–Lagrangean relaxation shows no duality gap.*

From Theorem 3.1 point 1 we see that the lower bound produced by $\mathcal{L}(\mu)$ is monotone and increasing in μ, and in point 4 it is stated that the dual problem (3.5) closes the duality gap. This obviously implies that if μ^* is an optimal dual multiplier, then every $\mu \geq \mu^*$ will also be optimal. Furthermore, point 2 states that if an optimal solution to the dual problem is primal feasible, then it is primal optimal and the corresponding dual multiplier is optimal for the semi–Lagrangean dual problem. On the other hand, if $\mu \in int(\mathcal{M})$, then an optimal solution to the semi–Lagrangean subproblem will be a primal optimal solution as well. Theorem 3.1 naturally leads to the generic dual ascent algorithm described in Algorithm 3.1. In Step 1, the multiplier μ is initialized. Next, the semi–Lagrangean subproblem is solved to optimality in Step 2. The solution obtained in Step 2 is tested for primal feasibility in Step 3. According to point 2 of Theorem 3.1 the solution x^k to the semi–Lagrangean subproblem is optimal to (3.1) if x^k is primal feasible, and the procedure can therefore be terminated. But if $Ax^k \neq b$, the multiplier μ is incremented in Step 4 and the algorithm returns to Step 2.

Input: Vectors c and b, matrix A, and feasible set $\mathcal{X}$.
Output: An optimal solution x^* to (3.1) and optimal multiplier $\mu^* \in \mathcal{M}$.

Step 1: (Initialization) Set $k = 1$ and initialize multiplier, μ^k, to a suitably small value.

Step 2: (Solve subproblem)

Step 2.1: Use a heuristic to obtain a good feasible solution $\bar{x}^k$ to

$$\min\{(c - \mu^k e^T A)^T x \ : \ x \in \mathcal{X}\}. \tag{3.6}$$

If $A\bar{x} \neq b$, set $x^k = \bar{x}^k$ and go to Step 3. Otherwise, the subproblem must be solved to optimality, therefore go to Step 2.2.

Step 2.2: Determine $x^k \in \mathcal{X}(\mu^k)$, that is an optimal solution to (3.6), and proceed to Step 3.

Step 3: (Termination check) If $Ax^k = b$, stop with (x^k, μ^k) as an optimal primal–dual pair.

Step 4: (Update) Let $\mu^{k+1} = \mu^k + \delta^k$, $\delta^k > 0$, and return to *Step 2*. Go to *Step 2*.

Algorithm 3.2: A dual ascent algorithm with heuristically solved subproblems for a semi–Lagrangean relaxation of a general ILP.

The obvious drawback of this methodology is that the semi–Lagrangean subproblem has the same computational complexity as the original problem, meaning that the subproblem might be just as hard to solve as the original problem. One should, however, note that if the vector x can be decomposed in such a way that if $x = (x^1, x^2) \in \mathcal{X}$ and $\tilde{x}^1 \leq x^1$ then $(\tilde{x}^1, x^2) \in \mathcal{X}$, then the semi–Lagrangean subproblem (3.2) can be reduced in size by eliminating all variables x_j^1 where $c_j^1 - \mu e^T a_j^1 \geq 0$. Here a_j^1 is the column corresponding to the variable x_j^1. Also note that the larger the value of μ, the fewer variables can be eliminated in this way. On the other hand, it should be obvious that for a sufficiently large value of μ the solution to the semi–Lagrangean subproblem will be optimal to (3.1). Thus, one should find a $\mu \in int(\mathcal{M})$ which is as small as possible and if possible, find such a multiplier using a method solving the semi–Lagrangean subproblem as a few times as possible.

Just like pricing problems can be solved heuristically in a column generation procedure, the semi–Lagrangean subproblem can also be solved heuristically to indicate optimality of the multiplier μ. This leads to Algorithm 3.2, which is a version of Algorithm 3.1 where Step 2 is expanded into two sub–steps that allow the subproblem to be solved heuristically. In Algorithm 3.2 the semi–Lagrangean subproblem is only solved to optimality if the heuristic suggests that the multiplier μ^k is optimal ($A\bar{x}^k = b$). Otherwise the subproblem is simply solved by a heuristic that is assumed to produced good (near optimal) solutions in reasonable time. This can potentially speed ud the search for optimal dual multipliers by reducing the

time spent on solving the subproblems. On the other hand, this might increase the value of μ more than necessary, implying that the final subproblem cannot be reduced as much as otherwise possible.

Updating the multiplier

In Step 4 of Algorithms 3.1 and 3.2 the semi–Lagrangean multiplier is updated by adding a strictly positive number to the current multiplier. One possible way of updating the multiplier is to use subgradient optimization. Let x^k be an *optimal* solution to the semi–Lagrangean subproblem at multiplier μ^k. Then

$$g^k = e^T(b - Ax^k)$$

is easily seen to be a subgradient at μ^k. The multiplier of iteration $k+1$ can then be determined as $\mu^{k+1} = \mu^k + \phi^k g^k$, where $0 < \phi^k$ is a positive step size. A step size often suggested in the literature on subgradient optimization is

$$\phi^k = \frac{UB - \mathcal{L}(\mu^k)}{(e^T(b - Ax^k))},$$

where UB is an upper bound on the optimal solution value of the original problem. This leads to the updating strategy $\mu^{k+1} = \mu^k + UB - \mathcal{L}(\mu^k)$. One might, however, also regularize the step size by the length of the gradient whereby the new iterate is determined by $\mu^{k+1} = \mu^k + \frac{UB - \mathcal{L}(\mu^k)}{e^T(b - Ax^k)}$. Both of these strategies close the duality gap as the increment is strictly positive until the duality gap is closed. The main drawback of this method is that the semi–Lagrangean subproblem needs to be solved to optimality for g^k to be a subgradient. Another and more straightforward procedure is to simply use a strictly positive $\delta^k = \delta > 0$ and increment the multiplier μ^k by a fixed amount in each iteration. In Section 3.4.2 we propose an updating scheme which is based on dual feasible solutions.

3.2.2 Cut–and–solve

The cut–and–solve approach was first proposed by Climer and Zhang (2006) for the asymmetric traveling salesman problem. The methodology is essentially a branch–and–bound algorithm which instead of branching on a single variable branches on a *set* of variables. At each level in the search tree there are only two nodes (see Figure 3.1). The left node is associated with a linear constraint stating that the sum of the variables in a set Ω is less than or equal to an integer γ, while the right node is associated with the sum being larger than or equal to $\gamma + 1$. The MIP corresponding to a left node is called a *sparse problem* (SP) as the search space is considerably reduced compared to the original problem. The problems solved at the right nodes are called dense problems and the cut $\sum_{i \in \Omega} x_i \geq \gamma + 1$ is called a

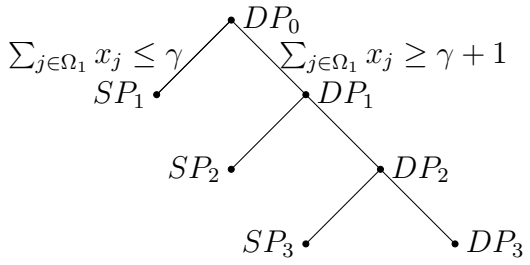


Figure 3.1: The cut–and–solve search tree.

Input: A mixed inter programming problem.
Output: An optimal solution x^* to (3.1).

Step 1: (Initialization) Set $L = -\infty$, $UB = \infty$, and $H = \emptyset$, where H is the set of piercing cuts.

Step 2: (Dense problem) Solve a relaxation of (3.1) with all piercing cuts in H added. Let the solution value be L.

Step 3: (Termination check) If $L \geq UB$, return the incumbent.

Step 4: (Piercing cut selection) Select an index set, Ω, of variables for the piercing cut.

Step 5: (Sparse problem) Solve the problem (3.1) with the constraint $\sum_{j\in\Omega} x_j \leq \gamma$ added.

Step 6: (Incumbent update) If the solution found in Step 4 improves the incumbent, then update UB. If $L \geq UB$, return the incumbent.

Step 7: (Adding piercing cut) Add piercing cut $\sum_{j\in\Omega} x_j \geq \gamma + 1$ to H and go to Step 1.

Algorithm 3.3: A description of a generic cut–and–solve algorithm.

piercing cut. For MIPs where the integer constrained variables are binary, it is natural to select the constant $\gamma = 0$. In this way, all variables in the set Ω will be fixed to zero in the sparse problem, thus reducing the problem significantly.

A generic cut–and–solve algorithm for the MIP (3.1) is given in Algorithm 3.3. In Step 1, the lower and upper bounds are initialized and the set of piercing cuts is initialized to the empty set. In the second step, Step 2, of Algorithm 3.3 a lower bound of the dense problem is obtained. Note that this lower bound is a lower bound on the optimal solution in the space *not* removed by the piercing cuts. In Step 3, the lower bound is compared to the current best solution and if the lower bound exceeds the value of the incumbent, the incumbent is optimal. In Step 4, a piercing cut is selected and in Step 5, the SP associated with the piercing cut is solved. If the solution to SP improves the current incumbent, it is updated in Step 6 and the value of the incumbent is compared to the lower bound obtained in Step 1.

If the lower bound exceeds the new incumbent, an optimal solution has been found. Finally, in Step 7, the search space investigated in the sparse problem in Step 5 is excluded by the addition of the piercing cut and the procedure is repeated from Step 2.

Climer and Zhang (2006) propose to use the LP–relaxation of the dense problem and to define the set Ω as those variables having a reduced cost above a specified (positive) threshold. Climer and Zhang (2006) prove termination of the cut–and–solve algorithm, Algorithm 3.3, by assuming that the search space removed by the piercing cuts removed at least one feasible solution from the problem. It was noted in Gadegaard et al. (2016d) that an optimal solution to the original problem was often found at the first sparse problem or in nodes close to the top of the search tree when the sparse problems are chosen carefully. Thus, if the cut–and–solve algorithm is stopped before optimality is proven, it becomes a heuristic with a known quality as both a lower and an upper bound of the problem are provided by the algorithm. If the sparse problems are chosen appropriately, an optimal solution might be found very close to the top of the search tree, yielding a potentially powerful heuristic.

3.3 Integrating cut–and–solve and semi–Lagrangean based dual ascent

In this section we describe two ways of integrating the two frameworks of cut–and–solve and semi–Lagrangean based dual ascent. There are two natural ways to integrate these two frameworks; the first approach is to solve the subproblems arising in the semi–Lagrangean dual ascent algorithm using the cut–and–solve framework. Section 3.3.1 proposes a way to do this in which the cut–and–solve framework is used as a heuristic and only as an exact solver when the heuristic solution to the semi–Lagrangean subproblems is primal feasible. The second natural way is to solve the sparse problems arising in the cut–and–solve algorithm using a semi–Lagrangean dual ascent algorithm. Such an approach is developed in Section 3.3.2.

3.3.1 Integrating cut–and–solve in a semi–Lagrangean based dual ascent algorithm

As mentioned in Section 3.2.2, the cut–and–solve framework can be used both as a heuristic and as an exact solution method. This makes the cut–and–solve framework ideal as a method to perform both Steps 2.1 and 2.2 in Algorithm 3.2. If the cut–and–solve algorithm described in Algorithm 3.3 is terminated before optimality has been proven, the framework becomes a heuristic. This means that Step 2.1 of Algorithm 3.2 can be handled by a truncated version of the cut–and–solve algorithm, while Step 2.2, where an optimal solution is found

Input: Vectors c and b, matrix A, and feasible set $\mathcal{X}$.
Output: An optimal solution x^* to (3.1) and optimal multiplier $\mu^* \in \mathcal{M}$.

Step 1: (Initialization) Set $k = 1$ and initialize the multiplier, μ^k, to a suitably small value.

Step 2: (Solving subproblem) Use the cut–and–solve framework to solve the semi–Lagrangean subproblem

Step 2.1: (Initialization of cut–and–solve) Set $best = \infty$ and initialize the incumbent $x^k = null$.

Step 2.2: (Solution of dense problem) Solve a relaxation of

$$\min\{(c - \mu^k e^T A)^T x \ : \ x \in \mathcal{X}\},$$

and let Z^{DP} be the objective function value of the relaxation. If $Z^{DP} \geq best$, go to Step 3 with x^k as an optimal solution to the semi–Lagrangean subproblem. Otherwise, proceed to Step 2.3.

Step 2.3: (Piercing cut selection) Determine a subset of variables, Ω, defining the piercing cut.

Step 2.4: (Solution of sparse problem) If the sparse problem is feasible, set

$$Z^{SP} = \min\{(c - \mu^k e^T A)^T x \ : \ x \in \mathcal{X}, \ \sum_{j \in \Omega} x_j \leq \gamma\}.$$

Else set $Z^{SP} = \infty$. Go to Step 2.5.

Step 2.5 (Incumbent update) If $Z^{SP} \leq best$, update the subproblem incumbent x^k, set $best = Z^{SP}$ and go to Step 2.6. Else, go to Step 2.7.

Step 2.6 (Feasibility check) If $Ax^k \neq b$, go to Step 4. Else, go to Step 2.7.

Step 2.7 (Adding piercing cut) Add the piercing cut $\sum_{j \in \Omega} x_j \geq \gamma + 1$ to $\mathcal{X}$ and go to Step 2.2.

Step 3: (Termination check) If $Ax^k = b$, stop with (x^k, μ^k) as an optimal primal–dual pair.

Step 4: (Multiplier update) Let $\mu^{k+1} = \mu^k + \delta^k$, $\delta^k > 0$, and return to *Step 1.* Go to *Step 1.*

Algorithm 3.4: A semi–Lagrangean based dual ascent algorithm integrating the cut–and–solve framework for solving the semi–Lagrangean subproblems.

to the semi–Lagrangean subproblem, can be handled by *continuing* from the cut–and–solve branching node where the heuristic stopped. Such a framework is proposed in Algorithm 3.4.

In this setting, the cut–and–solve framework (Steps 2.1 to 2.7 of Algorithm 3.4) is used as a heuristic. The search is terminated as soon as a primal infeasible solution that improves

the incumbent of the subproblem is found. Only if no such solution to the sparse problems is found will the semi–Lagrangean subproblem be solved to optimality. Note that the choice of piercing cuts may make the sparse problems infeasible. The proof of termination of the cut–and–solve algorithm provided by Climer and Zhang (2006) rely on the sparse problems containing at least one feasible solution, meaning that the piercing cuts have to be chosen accordingly or that another termination proof must be provided. If the piercing cuts are chosen such that the sparse problems always exhibit a feasible solution, the modified cut–and–solve algorithm in Steps 2.1 to 2.7 of Algorithm 3.4 only performs one iteration unless the solution found is primal feasible. Therefore, in the first iterations of the dual ascent algorithm, little computational effort is put into solving the subproblems. Only when the top nodes of the cut–and–solve search tree start to indicate that the multiplier μ^k might be optimal (by finding primal feasible solutions), the effort is increased.

3.3.2 Integrating semi–Lagrangean dual ascent in a cut–and–solve algorithm

When employing the cut–and–solve framework for solving integer programming problems one of the obstacles is to be able to solve the sparse problems in Step 5 of Algorithm 3.3 efficiently. Even though the piercing cuts reduce the size of the sparse problem considerably compared to the original problem, it might still be very time consuming to solve these reduced problems. The dual ascent algorithm described in Algorithm 3.2 might offer a way to further reduce the sparse problem and thereby speed up the solution of difficult problems. Furthermore, when the feasibility version of the problem (3.1) is $\mathcal{NP}$–hard, it can be very hard to find a feasible, and specifically an optimal, solution to the sparse problems. Therefore, relaxing the equation system $Ax = b$ to the inequality system, $Ax \leq b$, can make it easier to find feasible solutions that can be utilized by heuristics in the solver employed to solve the subproblems. For completeness, Algorithm 3.5 describes the cut–and–solve algorithm using a dual ascent algorithm to solve the sparse problems. Note that Algorithm 3.5 is identical to Algorithm 3.3 except for Step 5, where Algorithm 3.5 uses the semi–Lagrangean based dual ascent algorithm to solve the sparse problems.

3.4 Applications to the single–source capacitated facility location problem

In this section, we start by formulating the single–source capacitated facility location problem (SSCFLP). Next, we apply the semi–Lagrangean relaxation to the single–source capacitated facility location problem (SSCFLP) and suggest procedures for initializing and updating the

Input: Vectors c and b, matrix A, and feasible set $\mathcal{X}$.
Output: An optimal solution x^* to (3.1).

Step 1: (Initialization) Set $L = -\infty$, $UB = \infty$, and set the set of piecing cuts $H = \emptyset$. Initialize the incumbent to an empty solution $x^* = null$.

Step 2: (Dense problem) Solve a relaxation of (3.1) with all piercing cuts in H added. Let the solution value be L.

Step 3: (Termination check) If $L \geq UB$, return the incumbent.

Step 4: (Piercing cut selection) Select a set of variables Ω for the piercing cut.

Step 5: Use the dual ascent algorithm described in Algorithm 3.2 to find an optimal solution in the space removed by the piercing cut (sparse problem).

Step 5.1: (Initialization) Set $k = 1$ and initialize multiplier, μ^k, to a suitably small value.

Step 5.2: (Solve subproblem)

Step 5.2.1: Use a heuristic to obtain a good feasible solution $\bar{x}^k$ to

$$\min\{(c - \mu^k e^T A)^T x \ : \ x \in \mathcal{X}, \sum_{j \in \Omega} x_j \leq \gamma\}.$$

If $A\bar{x} \neq b$, set $x^k = \bar{x}^k$ and go to Step 5.4. Otherwise the subproblem needs to be solved to optimality, therefore go to Step 5.2.2.

Step 5.2.2: Determine $x^k \in \arg\min\{(c - \mu^k e^T A)^T x \ : \ x \in \mathcal{X}, \sum_{j \in \Omega} x_j \leq \gamma\}$, and proceed to Step 3.

Step 5.3: (Termination check) If $Ax^k = b$, go to Step 6 as x^k is an optimal solution to the sparse problem. Else go to Step 5.4.

Step 5.4: (Update) Let $\mu^{k+1} = \mu^k + \delta^k$, $\delta^k > 0$, and return to *Step 2*. Go to *Step 2*.

Step 6: (Incumbent update) If the solution found in Step 4 improves the incumbent, then update UB. If $L \geq UB$, return the incumbent.

Step 7: (Adding piercing cut) Add piercing cut $\sum_{j \in \Omega} x_j \geq \gamma + 1$ to H and go to Step 1.

Algorithm 3.5: A cut–and–solve algorithm integrating a semi–Lagrangean dual ascent algorithm for solving the sparse problems.

semi–Lagrangean dual multiplier, μ. We continue by proposing a new way to generate cutting planes that can be used to strengthen the lower bound of the problems. Finally, we propose a dual ascent algorithm integrating the cut–and–solve framework and a cut–and–solve algorithm integrating the dual ascent algorithm.

The SSCFLP can be formulated as a pure integer linear programming (ILP) problem in the following way: Let $\mathcal{I}$, $|\mathcal{I}| = n$, be a set of potential facility sites and $\mathcal{J}$, $|\mathcal{J}| = m$, be a set of customers. Each facility, $i \in \mathcal{I}$, has a fixed capacity, $s_i > 0$, and each customer has a fixed and known demand, $d_j > 0$. Opening a facility at site $i \in \mathcal{I}$ results in a fixed opening cost $f_i > 0$ and allocation of customer j to facility i yields a cost of $c_{ij} \geq 0$. Let y_i be a binary variable equaling one if and only if a facility is opened at site i and similarly let x_{ij} be a binary variable that equals one if and only if customer j is allocated to facility i. The SSCFLP can then be stated as the following ILP problem:

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i \tag{3.7}$$

$$\text{s.t.:} \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \tag{3.8}$$

$$\sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i y_i, \quad \forall i \in \mathcal{I}, \tag{3.9}$$

$$0 \leq y_i,\ x_{ij} \leq 1, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}, \tag{3.10}$$

$$y_i,\ x_{ij} \in \{0,1\} \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}. \tag{3.11}$$

The objective function (3.7) minimizes the total cost (assignment plus opening costs). Constraints (3.8) state that each customer needs to be assigned to exactly one facility while constraints (3.9) make sure that each facility's capacity is respected. Finally, the constraints (3.10) impose lower and upper bounds on each variable and the constraints (3.11) state that all variables should be binary.

3.4.1 Semi–Lagrangean relaxation applied to the SSCFLP

The equations (3.8) can be replaced by the two sets of inequalities $\sum_{i\in\mathcal{I}} x_{ij} \leq 1$ for all $j \in J$ and $\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}} x_{ij} \geq m$. The latter is then dualized with multiplier $\mu \geq 0$, resulting in the

Lagrangean dual function

$$\begin{aligned}\mathcal{L}(\mu) = \mu m + \min\ & \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}(c_{ij}-\mu)x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i \\ \text{s.t.:}\ & \sum_{i\in\mathcal{I}} x_{ij} \le 1, \quad \forall j \in J, \\ & \sum_{j\in\mathcal{J}} d_j x_{ij} \le s_i y_i, \quad \forall i \in I, \\ & 0 \le x_{ij},\ y_i \le 1, \quad \forall i \in I, j \in J, \\ & x_{ij}, y_i \in \{0,1\}, \quad \forall i \in I, j \in J.\end{aligned}$$

If $(\bar{x}, \bar{y})$ is feasible for the Lagrangean subproblem, then $(\tilde{x}, \bar{y})$ is feasible too if $\tilde{x}_{ij} \le \bar{x}_{ij}$, $\forall i \in \mathcal{I}, j \in \mathcal{J}$. This implies that we can exclude all variables showing a Lagrangean reduced cost $c_{ij} - \mu \ge 0$ from the subproblem. For small values of μ this may lead to a substantial reduction in the number of x_{ij}–variables.

We can also present the semi–Lagrangean subproblem in the same form as the original problem as follows: Introduce a "dummy" facility, $i = 0$, with $c_{0j} = \mu$ for all $j \in \mathcal{J}$, $f_0 = 0$ and $s_i = D$. Setting $\mathcal{I}_0 = \mathcal{I} \cup \{0\}$ the Lagrangean subproblem can be posed as the following SSCFLP

$$\begin{aligned}\mathcal{L}(\mu) = \min\ & \sum_{i\in I_0}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in I_0} f_i y_i \\ \text{s.t.:}\ & \sum_{i\in\mathcal{I}_0} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \\ & \sum_{j\in\mathcal{J}} d_j x_{ij} \le s_i y_i, \quad \forall i \in \mathcal{I}_0, \\ & x_{ij} = 0, \quad \forall i \in \mathcal{I}, j \in \mathcal{J} : c_{ij} \ge \mu, \\ & y_0 = 1, \\ & x_{ij},\ y_i \in \{0,1\}, \quad \forall i \in \mathcal{I}_0, j \in \mathcal{J}.\end{aligned} \tag{3.12}$$

As customer j can be assigned to the "dummy" facility at a cost of μ, all assignment variables x_{ij} with $c_{ij} \ge \mu$ can be removed from the program. This leads to the interpretation of the semi–Lagrangean multiplier as a fixed prize obtainable by servicing a customer. Alternatively, μ can be thought of as a threshold which defines a "core" problem, that gradually has to be enlarged to facilitate the eventual proof of optimality. Core algorithms have proved very effective for problems like the binary knapsack problem (see Pisinger (1997)), but core–like algorithms have also been successfully used as heuristics for facility location problems (see e.g. Avella, Boccia, Sforza, and Vasilév (2008) and Guastaroba and Speranza (2012)).

The lower bound $\mathcal{L}(\mu)$ can be considerably improved by adding the constraint $\sum_{i\in\mathcal{I}} s_i y_i \ge \sum_{j\in\mathcal{J}} d_j \coloneqq D$ to the semi–Lagrangean subproblem (3.12). Note that this constraint is

redundant for the original problems as it is implied by constraints (3.8) and (3.9). However, when the constraints (3.8) are relaxed in a semi–Lagrangean manner, this is no longer the case. The added constraint ensures that the lower bound $\mathcal{L}(\mu)$ is never worse than $\min\{\sum_{i\in\mathcal{I}} f_i y_i : \sum_{i\in\mathcal{I}} s_i y_i \geq D\} + \sum_{j\in\mathcal{J}} \min_{i\in\mathcal{I}_0} c_{ij}$. Since the semi–Lagrangean subproblem can be cast as an SSCFLP, it can be solved by any algorithm customized for the SSCFLP.

Even though the optimization problem (3.12) is just as hard as the original SSCFLP (3.7)–(3.11) seen from a computational complexity viewpoint, the feasibility version is significantly easier. It is well known that the feasibility problem "does the SSCFLP (3.7)–(3.11) contain a feasible solution" is $\mathcal{NP}$-complete (this can for example be seen by reduction from the generalized assignment problem). However, the problem "does the semi–Lagrangean of SSCFLP given by (3.12) contain a feasible solution" is polynomially solvable, and the answer is always "yes". This means that a feasible solution for (3.12) is easy to obtain, and that such a feasible solution can be used as a starting solution for a heuristic in order to find a feasible solution to the original problem.

3.4.2 Initializing and updating the semi–Lagrangean multiplier

It is crucial to have a good initial value for the dual variables for Algorithm 3.1 to be efficient. For that reason, we start by setting a lower bound on the optimal multiplier.

Proposition 3.1. *For all $\mu \in \mathcal{M}$ we have $\mu \geq \mu^{\min} := \max_{j\in\mathcal{J}} \min_{i\in\mathcal{I}} c_{ij}$.*

Proof. If there exists a $j \in \mathcal{J}$ such that $\mu < \min_{i\in\mathcal{I}} c_{ij}$, then that customer will be serviced from the dummy facility in any optimal solution to the semi–Lagrangean subproblem. □

For the *uncapacitated* facility location problem Jörnsten and Klose (2015) show that there exists an optimal multiplier with $\mu \leq \max_{i\in\mathcal{I}j\in\mathcal{J}} c_{ij}$. This is, however, not the case for the SSCFLP due to the presence of the capacity constraints. However, a trivial upper bound is readily available by $\mu^{\max} := \sum_{i\in\mathcal{I}} f_i + \max_{i\in\mathcal{I},j\in\mathcal{J}}\{c_{ij}\}$. This upper bound is obvious, as it is cheaper to open all facilities and assign each customer to its most expensive facility than to assign it to the dummy facility. However, one should note, that if $\mu^k \geq \max\{c_{ij} : i \in \mathcal{I}, j \in \mathcal{J}\}$, then no variables can be eliminated from the subproblem. Below we list four different choices of starting values for the semi–Lagrangean multipliers which all satisfy Proposition 3.1:

1. Compute (near) optimal Lagrangean multipliers λ^* by for example subgradient methods to the relaxation obtained by relaxing constraints (3.8) in a Lagrangean manner. Then set $\mu = \min\{\mu^{\max}, \max_{j\in\mathcal{J}} \lceil \lambda_j^* \rceil\}$.
2. Solve the linear relaxation of (3.7)–(3.11) and let δ^* be an optimal dual solution corresponding to the assignment constraints (3.8). Set $\mu = \max_{j\in\mathcal{J}} \lceil \delta_j^* \rceil$.

3. Specify a minimum number of possible assignments for each customer, say T. Then set $\mu = \min\{n \in \mathbb{N} \,:\, |\{i \in \mathcal{I} \,:\, c_{ij} \leq n\}| \geq T, \forall j \in \mathcal{J}\}$. T should be large enough to yield an "interesting" subproblem.
4. Let $(\bar{x}, \bar{y})$ be a (good) feasible solution for the SSCFLP. Then set $\mu = \min\{c_{ij} \,:\, c_{ij} > c(\bar{x}), i \in \mathcal{I}, j \in \mathcal{J}\}$, where $c(\bar{x}) = \max\{c_{ij} \,:\, \bar{x}_{ij} > 0\}$.

An obvious disadvantage of the first suggestion for initializing the multipliers is that it requires potentially many subgradient iterations and that a series of knapsack problems needs to be solved in each iteration. The other three suggestions each has their own merits: suggestion 2 is easy to obtain. If in addition cutting planes are used to approximate the convex hull of integer solutions to the knapsack constraints defined by the capacity constraints (3.9), the optimal dual variables approximate optimal Lagrangean variables as well. Specifying a certain number of allowed assignments in suggestion 3 makes it possible to have complete control over the size of the first subproblem. And finally 4 provides the possibility that improved primal feasible solutions can be found quickly.

The next crucial component regarding the multiplier is the updating scheme. To that end, let (x^k, y^k) be the solution to the semi–Lagrangean subproblem computed in Step 2 of Algorithm 3.2 at multiplier μ^k. Furthermore, let $\mathcal{J}^0 = \{j \in \mathcal{J} \,:\, \sum_{i \in \mathcal{I}} x_{ij}^k = 0\}$. That is, $\mathcal{J}^0$ is the subset of customers which is assigned to the dummy facility. The fact that these customers are effectively not serviced indicates that the prize μ^k is not sufficiently high for these customers to be profitable. Hence, a natural updating rule is

$$\mu^{t+1} = \min\{c_{ij} \,:\, c_{ij} > \mu^k, i \in \mathcal{I},\ j \in \mathcal{J}^0\}.$$

However, it is not guaranteed that any $c_{ij} > \mu^k$ for $i \in \mathcal{I}$ and $j \in \mathcal{J}^0$ exists. To overcome this issue, one might try to make some of the already serviced customers even more profitable by updating the multiplier as follows

$$\mu^{k+1} = \min\{c_{ij} \,:\, c_{ij} > \mu^k, i \in \mathcal{I},\ j \in \mathcal{J}\}.$$

Even this updating strategy may fail if no assignment cost exceed the value of the current multiplier. This means that no x_{ij}–variables can be removed based on the rule $c_{ij} \geq \mu^k \Rightarrow x_{ij} = 0$.The obvious choice is therefore to increase the multiplier to its maximum value $\mu^{k+1} = \mu^{\max}$. For many instances of the SSCFLP proposed in the literature, the updating rules above often lead to very small increments in the multiplier. Therefore a more aggressive updating strategy is taken where the above rules are used and then the multiplier is updated according to the rule

$$\mu^{k+1} = \max\{\mu^{k+1}, \mu^k + K\}.$$

That is, we do not allow an increase that is smaller than a constant $K > 0$. This strategy clearly makes the subproblems larger than necessary, but it does also decrease the number

of subproblems we need to solve. The above strategies can be summarized by

$$\mu^{k+1} = \begin{cases} \max\{\min_{i\in\mathcal{I},\ j\in\mathcal{J}^0}\{c_{ij} \ : \ c_{ij} > \mu^k\}, \mu^k + K\}, & \text{if } \max\{c_{ij} \ : \ i \in \mathcal{I}, j \in \mathcal{J}^0\} > \mu^k \\ \max\{\min_{i\in\mathcal{I},\ j\in\mathcal{J}}\{c_{ij} \ : \ c_{ij} > \mu^k\}, \mu^k + K\}, & \text{if } \max\{c_{ij} \ : \ i \in \mathcal{I}, j \in \mathcal{J}\} > \mu^k \\ \mu^{\max}, & \text{otherwise} \end{cases}$$

3.4.3 Cut–and–solve applied to the SSCFLP

The cut–and–solve framework was successfully used for solving the SSCFLP in Gadegaard et al. (2016d) and we therefore use the recommendations given in this paper and use a capacitated facility location problem (CFLP) as a relaxation of the dense problem. Although the CFLP itself is an $\mathcal{NP}$–hard problem, it can be effectively handled by a good MIP solver such as CPLEX or Gurobi. Using the CFLP as a relaxation means that only the x_{ij}–variables are continuously relaxed. This gives a very straightforward way of defining the set of variables Ω defining the piercing cuts. Let (x^{DP}, y^{DP}) be an optimal solution to the CFLP solved as a relaxation of the dense problem, then $\Omega = \{i \in \mathcal{I} \ : \ y_i = 0\}$. The choice of the CFLP as a relaxation is based on the fact that it shares many of the characteristics of the SSCFLP, implying that an optimal solution is often found in one of the first cut–and–solve branching nodes. However, it should be noted that this choice of Ω does not guarantee a feasible solution to each sparse problem. This has implications for the dual ascent algorithm used to solve the sparse problem in the adaptation of Algorithm 3.5 as we have assumed that the problems solved using the dual ascent algorithm were all feasible. We discuss this shortly in Section 3.4.6. Furthermore, as the SSCFLP is a binary integer programming problem, we choose to set $\gamma = 0$. In conjunction with the choice of Ω this ensures that all variables y_i and x_{ij}, $i \in \Omega, j \in \mathcal{J}$ are fixed to zero in the sparse problem, reducing these problems considerably.

3.4.4 Cutting planes for the SSCFLP

It is well established that separating LP solutions from the convex hull of integer solutions to the capacity constraints (3.9) generally works very well for the SSCFLP (see e.g. Gadegaard et al. (2016d)). However, separating from the convex hull of integer solutions to the capacity constraints alone, means that the assignment constraints (3.8) are completely ignored. To remedy this, we first reformulate the SSCFLP. To that end, let $\mathcal{I}_1, \mathcal{I}_2 \subseteq \mathcal{I}$ with $\mathcal{I}_1 \neq \emptyset$, $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$, and $\mathcal{I}_1 \cup \mathcal{I}_2 = \mathcal{I}$. Furthermore, introduce new binary variables z_j^l, $l = 1, 2$, defined by

$$z_j^l = \begin{cases} 1, & \text{if } \exists i \in \mathcal{I}_l : \ x_{ij} = 1 \\ 0, & \text{otherwise} \end{cases}$$

With these additional binary variables we get the new formulation of the feasible set of the SSCFLP

$$P = \{(x,y,z) \in \{0,1\}^{n\times m+n+2\times m} \ : \ \sum_{i\in\mathcal{I}_1} x_{ij} = z_j^1, \ \sum_{i\in\mathcal{I}_2} x_{ij} = z_j^2, \quad \forall j \in \mathcal{J}, \\ z_j^1 + z_j^2 = 1, \quad \forall j \in \mathcal{J}, \\ \sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i y_i, \quad \forall i \in \mathcal{I}\}$$

It should be clear, that for any solution $(x,y,z) \in P$ we have that (x,y) is a solution to the SSCFLP. The goal is now to generate valid inequalities for $\mathcal{P} = \text{conv}(P)$ which are violated by an LP solution. To do so, we sum all capacity constraints over $i \in \mathcal{I}_1$ and obtain the valid inequality

$$\sum_{i\in\mathcal{I}_1}\sum_{j\in\mathcal{J}} d_j x_{ij} \leq \sum_{i\in\mathcal{I}_1} s_i y_i,$$

which we denote an *effective capacity constraint*. Rearranging terms and using $\sum_{i\in\mathcal{I}} x_{ij} = z_j^1$ yields the inequality

$$\sum_{j\in\mathcal{J}} d_j z_j^1 \leq \sum_{i\in\mathcal{I}_1} s_i y_i,$$

which can be turned into a standard knapsack constraints by complementing the y–variables, $\gamma_i = 1 - y_i$, thereby yielding:

$$\sum_{j\in\mathcal{J}} d_j z_j^1 + \sum_{i\in\mathcal{I}_1} s_i \gamma_i \leq \sum_{i\in\mathcal{I}_1} s_i \tag{3.13}$$

Since (3.13) is a valid inequality for $\mathcal{P}$, we have that

$$\mathcal{P} \subseteq \text{conv}(\{(z^1,\gamma) \in \{0,1\}^{m+|\mathcal{I}_1|} \ : \ \sum_{j\in\mathcal{J}} d_j z_j^1 + \sum_{i\in\mathcal{I}_1} s_i\gamma_i \leq \sum_{i\in\mathcal{I}_1} s_i\}) =: \mathcal{P}^{z\gamma}(\mathcal{I}_1). \tag{3.14}$$

This implies that given a solution $(\underline{x},\underline{y})$ to the LP relaxation of the SSCFLP, we can set $\underline{z}_j^1 = \sum_{i\in\mathcal{I}_1} \underline{x}_{ij}$ for all $j \in \mathcal{J}$ and $\underline{\gamma}_i = 1 - \underline{y}_i$ for all $i \in \mathcal{I}_1$ and then separate $(\underline{z},\underline{\gamma})$ from $\mathcal{P}^{z\gamma}(\mathcal{I}_1)$. Given a hyperplane $\sum_{j\in\mathcal{J}} \pi_j z_j^1 + \sum_{i\in\mathcal{I}_1} \psi\gamma_i \leq \pi_0$ that separates $(\underline{z}^1,\underline{\gamma})$ from $\mathcal{P}^{z\gamma}(\mathcal{I}_1)$ we can simply reinsert the original variables using the definitions of z_j^1 and γ_i and obtain the cutting plane

$$\sum_{j\in\mathcal{J}} \pi_j(\sum_{i\in\mathcal{I}_1} x_{ij}) \leq (\pi_0 - \sum_{i\in\mathcal{I}_1}\psi_i) + \sum_{i\in\mathcal{I}_1}\psi_i y_i.$$

This means that we can use all the classical inequalities such as cover inequalities, lifted cover inequalities, and Fencel inequalities known for the 0-1 knapsack polytope to separate the current LP solution from the polytope $\mathcal{P}^{z\gamma}(\mathcal{I}_1)$.

Note that using exact knapsack separation to separate $(\underline{z}^1,\underline{\gamma})$ from $\mathcal{P}^{z\gamma}(\mathcal{I}_1)$ is equivalent to solving the following separation problem: Find an inequality of the form $\sum_{j\in\mathcal{J}} \pi_j(\sum_{i\in\mathcal{I}_1} x_{ij}) + \sum_{i\in\mathcal{I}_1}\psi_i y_i \leq \pi_0$ that separates $(\underline{x},\underline{y})$ from

$$\text{conv}(\{(x,y) \in \{0,1\}^{n\times m+n} \ : \ \sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}_1} d_j x_{ij} \leq \sum_{i\in\mathcal{I}_2} s_i y_i, \ \sum_{i\in\mathcal{I}_1} x_{ij} \leq 1, \ \forall j \in \mathcal{J}\}), \tag{3.15}$$

or prove that no such inequality exists.

The procedure thus provides a way to exactly separate inequalities of the form

$$\sum_{j\in\mathcal{J}} \pi_j \Big(\sum_{i\in\mathcal{I}_1} x_{ij} \Big) + \sum_{i\in\mathcal{I}_1} \psi_i y_i \leq \pi_0$$

from the polytope (3.15), which is denoted an *effective capacity polytope with generalized upper bounds*.

The reader might note that for $\mathcal{I}_1 = \mathcal{I}$ and $\mathcal{I}_2 = \emptyset$ we get the well known *total demand* constraint $\sum_{i\in\mathcal{I}} s_i y_i \geq \sum_{j\in\mathcal{J}} d_j$ and for $\mathcal{I}_1 = \{i\}$ we simply get the original capacity constraint $\sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i y_i$. In this paper we consider pairs of facilities, that is $\mathcal{I}_1 = \{i_1, i_2\}$ with $i_1, i_2 \in \mathcal{I}$ and $i_1 \neq i_2$. This yields $n(n-1)/2$ new knapsack structures from which cutting planes can be separated. We have summarized the cutting plane algorithm used to strengthen the program in Algorithm 3.6.

In order to decrease the computational effort when separating cutting planes we only separate from $\mathcal{P}^{z\gamma}$ if we cannot generate any violated inequalities from the knapsack polytopes defined by single capacity constraints. In Steps 4.2 and 6.2 we follow the recommendations from Kaparis and Letchford (2010)and first attempt to separate a general lifted cover inequality and in case we fail to do so, we resort to Fenchel cutting planes.

3.4.5 A semi–Lagrangean based dual ascent algorithm using cut–and–solve

An outline of the complete dual ascent algorithm using the cut–and–solve framework for the SSCFLP is given in Algorithm 3.7. In Step 1 the algorithm is initialized. First, we run the cutting algorithm in order to strengthen the formulation of the SSCFLP and to get a dual solution that can be used to initialize the semi–Lagrangean multiplier. In Steps 2.1 to 2.6 we exploit the fact that the semi–Lagrangean subproblem can be rewritten as a SSCFLP and that we thereby can use a modified version of the cut–and–solve algorithm proposed in Gadegaard et al. (2016d) to solve the semi–Lagrangean subproblem. In Step 2.1 we initialize the upper bound on the subproblem to a large number, and the current best solution of the subproblem is set to the empty solution. In Step 2.2 the dense problem is solved and the solution value is first compared to the best primal feasible solution. As the dense problem provides a lower bound on the optimal solution to the semi–Lagrangean subproblem it is a valid lower bound on the optimal solution, which means that if the value of the dense problem exceeds the value of the current best primal solution, it is optimal. Next, the value of the dense problem is compared to the best solution to the semi–Lagrangean subproblem. If the value of the dense problem exceeds the value of the best solution to the semi–Lagrangean subproblem, the subproblem has been solved to optimality, and we

Input: An instance of the SSCFLP.
Output: A strengthened formulation of the SSCFLP and a vector, δ^*, of optimal dual variables for the assignment constraints.

Step 1: Set $LB^0 = -\infty$ and set the iteration counter $k = 0$.

Step 2: Set $k = k + 1$, solve the linear relaxation of the SSCFLP, and let $(\underline{x}, \underline{y})$ be an optimal solution to the LP relaxation with objective function value LB^k.

Step 3: If $(\underline{x}, \underline{y})$ is integral, return $(\underline{x}, \underline{y})$ as an optimal solution to the SSCFLP.

Step 4: If $LB^k - LB^{k-1} < \epsilon$, return the strengthened formulation and an optimal dual solution to the assignment constraints, δ^*.

Step 5: For each $i \in \mathcal{I}$ do the following:

- Step 5.1: Add all violated implied upper bounds $x_{ij} - y_i \leq 0$ to the formulation.
- Step 5.2: Separate the solution $(\underline{\mathbf{x}}_i)$ from the knapsack polytope $\text{conv}(\{\mathbf{x}_i \in \{0,1\}^{|\mathcal{J}|} : \sum_{j \in \mathcal{J}} d_j x_{ij} \leq s_i\})$ if possible. If a cutting plane, say $\pi^T \mathbf{x}_i \leq \pi_0$, is generated translate it to $\pi^T \mathbf{x}_i \leq \pi_0 y_i$ and add it to the formulation of the SSCFLP.

Step 6: If Step 4 resulted in violated cutting planes, go to Step 6. Else for each pair $i_1, i_2 \in \mathcal{I}$ with $i_1 < i_2$ do the following:

- Step 6.1 Create the aggregate solution $\underline{z}_j = \underline{x}_{i_1 j} + \underline{x}_{i_2 j}$ for each $j \in \mathcal{J}$ and $\underline{\gamma}_i = 1 - \underline{y}_i$ for each $i = i_1, i_2$.
- Step 6.2 Separate the point $(\underline{z}, \underline{\gamma})$ from $\mathcal{P}^{z\gamma}(\{i_1, i_2\})$, if possible. If a cutting plane, say $\pi^t z + \psi\gamma \leq \pi_0$, is generated, reinsert the original variables and add the cutting plane $\sum_{j \in \mathcal{J}} \pi_j (x_{i_1 j} + x_{i_2 j}) \leq (\pi_0 - \psi_1 - \psi_2) + \psi_1 y_{i_1} + \psi_2 y_{i_2}$ to the formulation of the SSCFLP.

Step 7: If $k < K$ and Step 5 or Step 6 resulted in violated cutting planes, go to Step 2, otherwise return the strengthened formulation and an optimal dual solution to the assignment constraints, δ^*.

Algorithm 3.6: Summary of the cutting plane algorithm.

continue to Step 3. If not, we continue to Step 2.3 where the piercing cut is selected. In Step 2.4 the sparse problem is solved. Note that the choice of piercing cuts ensures that the sparse problems always have a feasible solution as the dummy facility has amble capacity to service all customers simultaneously. The solution value of the sparse problem, Z^{SP}, is compared to the best solution to the current subproblem found so far. If the solution to the sparse problem improves the current best solution, the subproblem solution $(x, y)^k$ is updated and the procedure continues to Step 2.6. Otherwise we continue to Step 2.7. If the procedure continues to Step 2.6, it means that the current best solution to subproblem has just been updated. We therefore check if this solution is primal feasible in which case

Input: Vectors c and b, matrix A, and feasible set $\mathcal{X}$.
Output: An optimal solution x^* to (3.1) and optimal multiplier $\mu^* \in \mathcal{M}$.

Step 1: (Initialization) Run Algorithm 3.6 on the full SSCFLP in order to strengthen the problem and obtain an optimal dual solution to the assignment constraints, say δ_j^*. Next, run a local branching heuristic producing a feasible solution $(\bar{x}, \bar{y})$. If no feasible solution was found, set $UB = \infty$, otherwise set $UB = \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} c_{ij}\bar{x}_{ij} + \sum_{i\in\mathcal{I}} f_i\bar{y}_i$ and $(x^*, y^*) = (\bar{x}, \bar{y})$. Finally, set $k = 0$ and initialize the semi–Lagrangean multiplier to μ^k to suitably small value.

Step 2: (Solving subproblem) Use the cut–and–solve framework to solve the semi–Lagrangean subproblem

Step 2.1: (Initialization of cut–and–solve) Set $sub\_best = \infty$ and initialize the incumbent $(x^k, y^k) = null$.

Step 2.2: (Solution of dense problem) Solve the problem (3.12) with the integrality requirement on the x_{ij}–variables relaxed. Let Z^{DP} be the optimal solution value and $(x, y)^{DP}$ an optimal solution. If $Z^{DP} \geq UB$, return the incumbent as it is optimal. Else if $Z^{DP} \geq sub\_best$, go to Step 3, with $(x, y)^k$ as an optimal solution to the semi–Lagrangean subproblem. Otherwise, continue to Step 2.3.

Step 2.3: (Piercing cut selection) Let $\Omega = \{i \in \mathcal{I} \,:\, y_i^{DP} = 0\}$.

Step 2.4: (Solution of sparse problem) Solve the problem (3.12) with all variables $y_i,\ x_{ij},\ j \in \mathcal{J}, i \in \Omega$ fixed to zero. Let $(x, y)^{SP}$ be an optimal solution to the sparse problem and go to Step 2.5.

Step 2.5: (Incumbent update) If $Z^{SP} \leq sub\_best$, update the subproblem incumbent $(x, y)^k = (x, y)^{SP}$, set $sub\_best = Z^{SP}$, and go to Step 2.6. Else, go to Step 2.7.

Step 2.6: (Feasibility check) If $\sum_{i\in\mathcal{I}} x_{ij}^k = 0$ for some $j \in \mathcal{J}$, go to Step 4. Else, the subproblem solution is primal feasible. If $sub\_best < UB$ update the incumbent $(x^*, y^*) = (x, y)^k$ and set $UB = sub\_best$. Go to Step 2.7.

Step 2.7: (Piercing cut addition) Add the piercing cut $\sum_{i\in\Omega} y_i \geq 1$ to (3.12) and go to Step 2.2.

Step 3: (Termination check) $(x, y)^k$ is optimal to the subproblem. If $\sum_{i\in\mathcal{I}} x_{ij}^k = 1$ for all $j \in \mathcal{J}$ or $sub\_best \geq UB$, stop with $((x, y)^k, \mu^k)$ as an optimal primal–dual pair. Otherwise, go to Step 4.

Step 4: (Multiplier update) Let $\mathcal{J}^0 = \{j \in \mathcal{J} \,:\, \sum_{i\in\mathcal{I}} x_{ij}^k = 0\}$. Update the multipliers according to the rule

$$\mu^{k+1} = \begin{cases} \max\{\min_{i\in\mathcal{I},\ j\in\mathcal{J}^0}\{c_{ij} \,:\, c_{ij} > \mu^k\}, \mu^k + K\}, & \text{if } \max_{i\in\mathcal{I}, j\in\mathcal{J}^0}\{c_{ij}\} > \mu^k \\ \max\{\min_{i\in\mathcal{I},\ j\in\mathcal{J}}\{c_{ij} \,:\, c_{ij} > \mu^k\}, \mu^k + K\}, & \text{if } \max_{i\in\mathcal{I}, j\in\mathcal{J}}\{c_{ij}\} > \mu^k \\ \mu^{\max}, & \text{otherwise,} \end{cases}$$

set $k = k + 1$, and return to *Step 2*.

Algorithm 3.7: A dual ascent algorithm for the SSCFLP that uses the cut–and–solve framework to solve the semi–Lagrangean subproblems.

we test if this primal feasible solution improves the incumbent (x^*, y^*). Furthermore, if the current best solution is primal feasible, we continue the cut–and–solve procedure by going to Step 2.7 where the piercing cut is added to the dense problem as we might need to solve the subproblem to optimality. In Step 3 we check if the current best solution is optimal. If the subproblem was solved to optimality, *sub_best* provides a valid lower bound on the optimal solution, so if $sub\_best \geq UB$, we know that (x^*, y^*) is optimal. Conversely, if the subproblem solution is primal feasible, we know, by the construction of the algorithm, that (x^*, y^*) is an optimal solution to the subproblem. By means of point 2 of Theorem 3.1 we can conclude that $(x, y)^k$ is an optimal solution to the SSCFLP. Otherwise, we go to Step 4 where the semi–Lagrangean multiplier is incremented.

3.4.6 A cut–and–solve algorithm using semi–Lagrangean based dual ascent for sparse problems

Algorithm 3.8 outlines the cut–and–solve algorithm for the SSCFLP that utilizes a dual ascent algorithm to solve the sparse problems. In Step 1, the algorithm is initialized by first running the cutting plane algorithm followed by a local branch algorithm. The dense problem is solved in Step 2, and in Step 3 we check if the dense problem produced a solution larger than the incumbent, in which case the incumbent is optimal. In Step 4 we select the set of variables forming the piercing cut. The sparse problem is solved by a dual ascent algorithm in Step 5; first, the iteration counter is set and the semi–Lagrangean dual bound is set to minus infinity in Step 5.1. In Step 5.2 we solve the semi–Lagrangean subproblem. We start by checking if the multiplier is at its upper bound as the subproblem then has to be solved to optimality. If that is not the case, we start by solving the subproblem by a heuristic and if the resulting solution is primal feasible, we solve the subproblem to optimality in Step 5.2.2. In Step 5.3 we perform the termination check of the dual ascent algorithm. If the dual lower bound $\mathcal{L}$ exceeds the value of the incumbent, the sparse problem cannot improve the current best solution and we therefore go to Step 7 where a piercing cut is added. If $\mu^k = \mu^{max}$, we know that the subproblem has been solved to optimality, and if the corresponding solution is not primal feasible, the sparse problem contains no feasible solutions so we continue to Step 7. But if the solution is actually primal feasible, it is also optimal for the sparse problem and we go to Step 6. Otherwise, we need to increase the multiplier in Step 5.4 and the dual ascent algorithm returns to Step 5.2. We update the incumbent in Step 6 and check if the incumbent value falls below the value of the lower bound obtained from the dense problem as the incumbent is then optimal. Finally, in Step 7, the piercing cut is added to the program and we return to Step 2.

Input: Vectors c and b, matrix A, and feasible set $\mathcal{X}$.
Output: An optimal solution x^* to (3.1) and optimal multiplier $\mu^* \in \mathcal{M}$.

Step 1: (Initialization) Run Algorithm 3.6 on the full SSCFLP in order to strengthen the problem and obtain an optimal dual solution to the assignment constraints, say δ_j^*. Next, run a local branching heuristic producing a feasible solution $(\bar{x}, \bar{y})$. If no feasible solution was found, set $best = \infty$, otherwise set $UB = \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} c_{ij}\bar{x}_{ij} + \sum_{i\in\mathcal{I}} f_i\bar{y}_i$ and $(x^*, y^*) = (\bar{x}, \bar{y})$. Finally, set the set of piercing cuts $H = \emptyset$.

Step 2: (Dense problem) Solve the CFLP relaxation of the SSCFLP with all piercing cuts in H added. Let the solution value be Z^{DP} and an optimal solution be $(x, y)^{DP}$.

Step 3: (Termination check) If $Z^{DP} \geq UB$, return the incumbent (x^*, y^*) as it is optimal.

Step 4: (Piercing cut selection) Set $\Omega = \{i \in \mathcal{I} : y_i^{DP} = 0\}$.

Step 5: (Sparse problem) Use the semi–Lagrangean based dual ascent algorithm to solve the sparse problem.

Step 5.1 (Initialization) Set $k = 0$, $\mathcal{L} = -\infty$, and initialize μ^k to a suitably small value.

Step 5.2 (Solve subproblem) Set $k = k + 1$. If $\mu < \mu^{\max}$, go to Step 5.2.1, else go to Step 5.2.2.

Step 5.2.1 Use a heuristic to obtain a good feasible solution, $(x, y)^k$ to the problem

$$\begin{aligned}
\min\ & \sum_{i\in\mathcal{I}_0} \sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}_0} f_i y_i \\
\text{s.t.:}\ & \sum_{i\in\mathcal{I}_0} = 1, \quad \forall j \in \mathcal{J}, \\
& \sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i y_i, \quad \forall i \in \mathcal{I}_0, \\
& x_{ij} = y_i = 0, \quad \forall i \in \Omega, j \in \mathcal{J}, \\
& x_{ij} = 0, \quad \forall (i,j) \in \mathcal{I} \times \mathcal{J} : c_{ij} \geq \mu^k, \\
& y_0 = 1, \\
& x_{ij}, y_i \in \{0,1\}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}.
\end{aligned} \tag{3.16}$$

If $\sum_{i\in\mathcal{I}} x_{ij} = 0$ for some $j \in \mathcal{J}$, go to Step 5.4, else go to Step 5.2.2.

Step 5.2.2 Solve the problem (3.16) to optimality. Let $(x, y)^k$ be an optimal solution, let $\mathcal{L}$ be the corresponding solution value, and continue to Step 5.3.

Step 5.3 (Termination check) If $\mathcal{L} \geq UB$, the sparse problem cannot contain improving solutions, hence go to Step 7. If $\mu^k = \mu^{\max}$ and $\sum_{i\in\mathcal{I}} x_{ij}^k = 0$ for some $j \in \mathcal{J}$, the sparse problem is infeasible so go to Step 7. If $\sum_{i\in\mathcal{I}} x_{ij}^k = 1$ for all $j \in \mathcal{J}$, the solution $(x, y)^k$ is optimal for the sparse problem, hence go to Step 6. Otherwise, continue to Step 5.4.

Step 5.4 (Multiplier update) Let $\mathcal{J}^0 = \{j \in \mathcal{J} : \sum_{i\in\mathcal{I}} x_{ij}^k = 0\}$. Update the multipliers according to the rule

$$\mu^{k+1} = \begin{cases} \max\{\min_{i\in\mathcal{I},\ j\in\mathcal{J}^0}\{c_{ij} : c_{ij} > \mu^k\}, \mu^k + K\}, & \text{if } \max_{i\in\mathcal{I}, j\in\mathcal{J}^0}\{c_{ij}\} > \mu^k \\ \max\{\min_{i\in\mathcal{I},\ j\in\mathcal{J}}\{c_{ij} : c_{ij} > \mu^k\}, \mu^k + K\}, & \text{if } \max_{i\in\mathcal{I}, j\in\mathcal{J}}\{c_{ij}\} > \mu^k \\ \mu^{\max}, & \text{otherwise,} \end{cases}$$

and return to *Step 5.2*.

Step 6: (Incumbent update) If $\mathcal{L} < UB$, set $UB = \mathcal{L}$ and $(x^*, y^*) = (x, y)^k$. If $Z^{DP} \geq UB$, return (x^*, y^*) as it is optimal.

Step 7: (Adding piercing cut) Add the piercing cut $\sum_{i\in\Omega} y_i \geq \gamma + 1$ to H and go to Step 2.

Algorithm 3.8: A cut–and–solve algorithm for the SSCFLP that integrates a semi–Lagrangean based dual ascent algorithm to solve the sparse problems.

3.5 Computational results

This section describes the results obtained using the algorithms proposed in the previous sections. The primary purpose of the computational study is to test to which extent the integration of the semi–Lagrangean based dual ascent algorithm and the cut–and–solve framework can efficiently solve the single–source capacitated facility location problem (SSCFLP). In order for us to test the algorithms we start by identifying the most promising initialization schemes for the semi–Lagrangean dual multiplier. Next, we run the algorithms on three testbeds that have been proposed in the literature and a fourth testbed that consists of large instances generated for this paper. We compare the performance of the two integrated algorithms to the improved cut–and–solve algorithm from Gadegaard et al. (2016d), which is currently one of the fastest algorithms for the SSCFLP.

3.5.1 Implementation details and test instances

We have tested two different algorithms based on the two frameworks described in Section 3.2 and compared them to a state–of–the–art solver for the SSCFLP. The first algorithm, denoted `DA-CS`, implements Algorithm 3.7 where the cut–and–solve framework is used to solve the semi–Lagrangean subproblems both heuristically and exactly. Algorithm `CS-CPX` is the cut–and–solve algorithm described in Algorithm 3.3. We have used the implementation proposed in Gadegaard et al. (2016d) as this algorithm was proven to be effective for solving the SSCFLP. It serves mainly as a benchmark algorithm as nothing new is proposed here. The algorithm is enhanced with the cutting planes proposed in Section 3.4.4 to make the comparison fair. Finally, `CS-DA` augments the implementation of the cut–and–solve algorithm proposed in Gadegaard et al. (2016d) such that each sparse problem is solved using a semi–Lagrangean based dual ascent algorithm. It uses a standard implementation of Algorithm 3.2 to solve these problems. Each semi–Lagrangean subproblem arising when solving the sparse problems of the cut–and–solve algorithm is solved using CPLEX which has been strengthened by a cut callback employing the cutting plane algorithm Algorithm 3.6. We use CPLEX as a heuristic by setting the relative optimality gap to a small positive value. When a solution of sufficient quality has been found, the best solution is checked for primal feasibility. If the solution is primal feasible, the subproblem is solved to optimality. If, on the other hand, the solution is infeasible, the dual multiplier is increased and a new subproblem is solved.

The cutting plane algorithm, Algorithm 3.6, uses general lifted cover inequalities (GLCI) to separate fractional points from each individual capacity constraint. If no violated GLCI can be generated, we use the separation algorithm for Fenchel inequalities developed in Gadegaard et al. (2016d). Finally, if no cuts could be separated from any of the capacity constraints, we attempt to separate the aggregate solution from the polytope $\mathcal{P}^{z\gamma}$ defined

in (3.14). As in the case of the individual capacity constraints, we first attempt to separate using GLCIs and only if that fails, do we use the exact knapsack separation algorithm.

In all algorithms we initialize the upper bound by running a local branching heuristic on the full problem. The heuristic is similar to the one described in Gadegaard et al. (2016d); first, we run a local branching heuristic on the y–variables in order to get a good initial solution. The second phase then takes the best solution from the first phase and improves it, again by using a local branching heuristic. After each phase we use a fast two–opt local search that swaps customers between open facilities to find a local optimum given the current set of open facilities.

We carried out preliminary experiments using a version of Algorithm 3.1 in which we used the solver developed in Gadegaard et al. (2016d) as a subproblem solver. It turned out that this methodology was very inferior to `DA-CS`, `CS-DA`, and `CS-CPX`, even for the smallest instances. Therefore, we do not report on the results obtained using this methodology.

All algorithms have been coded in C and C++ and compiled using gcc and g++ with optimization option O3 and C++11 features enabled. As solver for the MIPs arising in the subproblems, we have used CPLEX 12.6 with callbacks. The ParallelMode switch is set to deterministic such that different runs can be compared. All codes are publicly available (see Gadegaard, Klose, and Nielsen (2016c)).

The tests have been performed on instances from four different testbeds; the first testbed, TB1, consists of 57 instances used in Díaz and Fernández (2002) ranging in size from instances with 10 facilities and 20 customers to instances with 30 facilities and 90 customers. The second testbed, TB2, contains instances having 10 to 30 facilities and 50 to 200 customers and has been used in Holmberg et al. (1999). Yang et al. (2012) proposed a testbed consisting of relatively large instances ranging in size from 30 facilities and 200 customers to 80 facilities and 400 customers, TB3. These instances range in size from 30 facilities and 200 customers to 80 facilities and 400 customers. The last testbed, TB4, consists of 30 new instances generated according to the procedure proposed by Cornuejols, Sridharan, and Thizy (1991). First, the demand at customer j is generated from $U[5, 35]$, where $U[a, b]$ denotes a uniform distribution on the interval $[a, b]$. Next the positions of the customers and facilities are generated as random points in a unit square. The assignment costs are then set equal to $c_{ij} = d_j\delta_{ij}$, where δ_{ij} denotes the Euclidean distance between customer j and facility i multiplied by 10. Next, the capacity s_i at facility i is generated uniformly from $[10, 160]$. We then scale the capacities such that the ratio $\sum_{i\in\mathcal{I}} s_i / \sum_{j\in\mathcal{J}} d_j = R$ where $R \in \{3, 5, 7\}$. Finally, the fixed operating costs are generated as $f_i = U[0, 90] + U[100, 110]\sqrt{s_i}$. We note that all parameters have been rounded to the nearest integer such that we avoid numerical instability. Two groups of instances were generated: the first group contains 15 instances of size 80×500 and the second group consists of 15 instances of size 100×400. Each group consists of 5 instances

Table 3.1: Column headers of the tables in Section 3.5.

Header	Description
#	Number of instances over which the average is taken.
size	Size of the instance given by $\|\mathcal{I}\| \times \|\mathcal{J}\|$.
R	Ratio between total capacity and total demand, that is $R = \sum_{i\in\mathcal{I}} s_i / \sum_{j\in\mathcal{J}} d_j$.
it	Number of iterations used by the dual ascent algorithm before optimality was proven.
it^{opt}	Number of iterations in which the subproblem was solved to optimality.
% x–left	The percentage of the x_{ij}–left in the final subproblem of the dual ascent algorithm.
CPU	The total time used by the algorithm in question measured in seconds.

having a capacity–to–demand ratio of $R = 3$, $R = 5$, and $R = 7$, respectively. Table 3.1 provides an overview of the table headings.

3.5.2 Testing the initialization of the dual multiplier

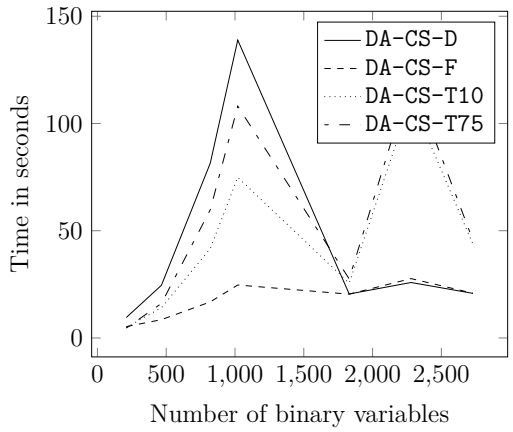
In this section we test three initialization schemes for the semi–Lagrangean multiplier. We run the two algorithms `DA-CS` and `CS-DA` on the smaller instances from TB1 and TB2. Each algorithm is run with the following three initialization schemes:

Dual: First, we apply the cutting plane algorithm separating from the knapsack structures. When no more cutting planes can be found, we set $\mu^0 = \max_{j\in\mathcal{J}}\{\delta_j^*\}$, where δ^* is a vector of optimal dual multipliers for the assignment constraints.

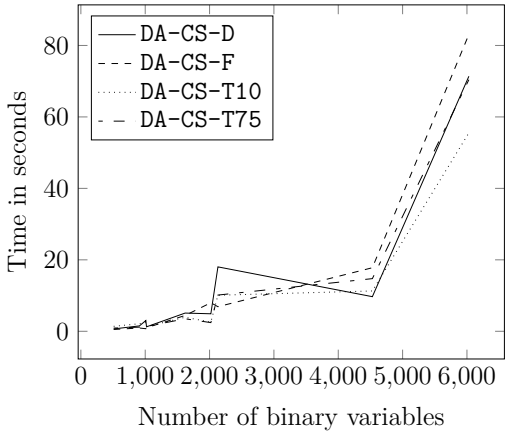
Threshold: We set a minimum number of assignments that should be allowed for each customer, say T. Then $\mu^0 = \min\{n \in \mathbb{N} \,:\, |\{i \in \mathcal{I} \,:\, c_{ij} \leq n\}| \geq T, \forall j \in \mathcal{J}\}$.

Feasible Let $(\bar{x}, \bar{y})$ be a good feasible solution, then $\mu^0 = \min\{c_{ij} \,:\, c_{ij} > c(\bar{x}), i \in I, j \in J\}$, where $c(\bar{x}) = \max\{c_{ij} \,:\, \bar{x}_{ij} > 0\}$.

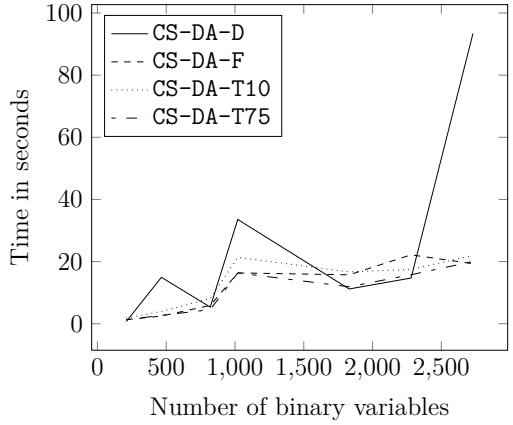
We append a `D`, `T`, or `F` to the name of the algorithm to denote that initialization strategy Dual, Threshold, or Feasible is used. Furthermore, we test both an aggressive and a more conservative value of T for the `T`–option; for the aggressive option we let $T = \lceil |\mathcal{I}|0.75 \rceil$, thereby allowing for relatively large subproblems and consequently fewer iterations of the dual ascent algorithm. For the more conservative value, we let $T = \lceil |\mathcal{I}|0.1 \rceil$, effectively eliminating about 90 percent of the x_{ij}–variables in the first subproblem. This strategy might lead to more, but smaller subproblems solved. We denote the aggressive and the conservative versions of the threshold strategy `T75` and `T10`, respectively. Thus, we get a total of 8 different algorithms, namely `DA-CS-i` and `CS-DA-i`, for `i`$\in$ {`D,F,T10,T75`}.



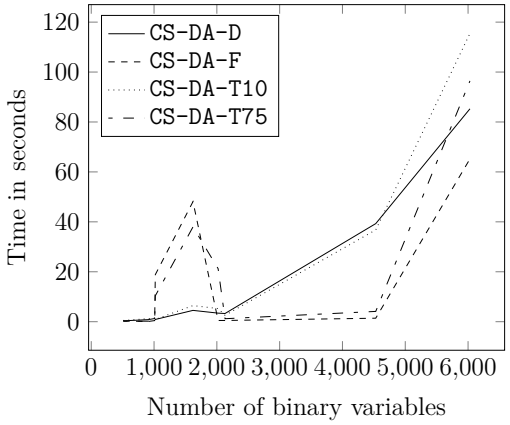
(a) Results of the three initialization strategies obtained from TB1 using `DA-CS`.



(b) Results of the three initialization strategies obtained from TB2 using `DA-CS`.



(c) Results of the three initialization strategies obtained from TB1 using `CS-DA`.



(d) Results of the three initialization strategies obtained from TB2 using `CS-DA`.

Figure 3.2: Overview of the performance of the four different initialization strategies. Figures 3.2a and 3.2b show results for `DA-CS` whereas Figures 3.2c and 3.2d show the results obtained with `CS-DA`.

Note that we do not have dual information available for the algorithm `CS-DA` as the cutting plane algorithm used for this algorithm is run at the root node of the cut–and–solve tree, and the dual ascent algorithm is run at each sparse problem. This means that we use the optimal dual vector found at the root node of the cut–and–solve tree to initialize the semi–Lagrangean multiplier when the initialization strategy `D` is specified.

Figure 3.2 summarizes the performance of the 2 different algorithms and the four initialization schemes. For the `DA-CS` algorithm, the initialization strategy that seems to be the overall best and the most stable is `F`. The results obtained on TB2 are very consistent over all four initialization strategies, but in TB1 all strategies become very unstable except for `F`. As regards the `CS-DA` algorithm, however, we get the best performance using option `T75`,

Table 3.2: Overview of the average number of dual ascent iterations and the average number of x_{ij}–variables left in the last iteration of the dual ascent algorithms.

	DA-CS				CS-DA			
	D	F	T10	T75	D	F	T10	T75
Dual ascent iterations	1.1	4.7	5.42	1.52	5.7	4.8	9.9	2.5
Percentage of variables left	67.1	65.2	63.1	95.8	32.9	50.0	35.3	43.5

Table 3.3: Results obtained for TB3 using the semi–Lagrangean based dual ascent algorithms using cut–and–solve to solve the semi–Lagrangean subproblems, DA-CS.

				DA-CS-F				DA-CS-D			
TB	#	size	R	it	it^{opt}	%x-left	CPU	it	it^{opt}	x_{ij}–left	CPU
3	5	30×200	$1.73 - 1.98$	4.6	1.0	81.2	171.38	1.0	1.0	99.2	100.25
	5	30×300	$2.88 - 3.49$	7.6	1.0	79.0	677.47	1.0	1.0	98.1	256.34
	5	60×300	$3.42 - 5.78$	5.4	1.0	73.0	1627.22	1.0	1.0	94.1	364.66
	5	80×400	$3.50 - 7.30$	5.4	1.0	78.9	5035.14	1.0	1.0	92.4	1712.35
Averages				5.8	1.0	78.0	1877.80	1.0	1.0	95.9	608.40
4	5	80×500	3.00[1]	3.0	1.0	67.9	183.15	1.0	1.0	75.5	2353.94
	5		5.00	1.8	1.0	77.7	1751.20	1.0	1.0	69.4	1760.34
	5		7.00	1.8	1.0	85.4	4861.61	1.6	1.0	71.1	4346.43
	5	100×400	3.00	3.8	1.0	69.8	1455.07	1.0	1.0	84.7	573.26
	5		5.00	2.4	1.0	76.1	635.93	1.2	1.0	75.3	577.80
	5		7.00	2.0	1.0	79.5	1499.78	1.0	1.0	70.1	1547.11
Averages				2.4	1.0	76.6	1796.27	1.1	1.0	74.1	1888.11

[1] We were not able to solve instance $p4$ of size 80×500. The program was killed by the operating system as the size of the branching tree grew too large.

initializing the multiplier using the aggressive threshold value. In Table 3.2, we have summarized the number of dual ascent iterations and the percentage of the variables left in the last iteration of the dual ascent algorithms, that is $|\{(i,j) \in \mathcal{I} \times \mathcal{J} : c_{ij} < \mu^*\}|/(|\mathcal{I}| \cdot |\mathcal{J}|)$. For the algorithm CS-DA, these numbers are calculated as an average over all sparse problems solved in the cut–and–solve algorithm for the particular instance. For DA-CS the D initialization strategy results in few iterations of the algorithm and at the same time, the final subproblem remains small. For the CS-DA, Table 3.2 seems to support the observations from Figure 3.2, namely that the initialization strategy T75 works the best. Therefore, based on Figure 3.2 and Table 3.2, the tests on the larger instances of TB3 and TB4 will be carried out with the initialization strategies D and F for the DA-CS algorithm, whereas CS-DA will only be initialized using the T75 strategy.

3.5.3 Analysis of the DA-CS algorithms

In this section we analyze the results obtained with the algorithms DA-CS-F and DA-CS-D. Table 3.3 shows the results obtained with DA-CS-F and DA-CS-D for the instances of testbeds TB3 and TB4. We have chosen not to reproduce the optimal solution values and have instead aggregated the results by taking the average over five instances having the same size (for detailed information about the instances of TB3 consult Yang et al. (2012) and for information on the new instances of TB4 see Appendix 3.A).

Table 3.3 shows that for the instances of TB3, DA-CS-D clearly outperforms DA-CS-F, as the latter uses more than three times as much CPU time on average compared to CS-DA-D. Even though the final subproblem solved by DA-CS-F is reduced to 78 percent of the original problem on average, this is not enough to counterbalance the additional iterations needed when the initialization strategy F is used compared to D. It is worth noting that the initialization strategy D provides an optimal dual multiplier for all instances of TB3, leading to only one subproblem that needs to be solved. In TB4 the performance of the two initialization strategies seem more equal as the CPU times used by the two algorithms are almost identical and differ by less than 5 percent. In TB4 the initialization strategy D is again very good at predicting an optimal dual multiplier as only 1.1 iterations are needed by the dual ascent algorithm on average. A very interesting point is that the modified cut–and–solve algorithm seems to work very well as a subproblem solver as it needed to solve only one subproblem to optimality in all instances of TB3 and TB4. This suggests that the truncated cut–and–solve algorithm is indeed a very effective heuristic for the SSCFLP.

3.5.4 Analysis of the CS-DA algorithm

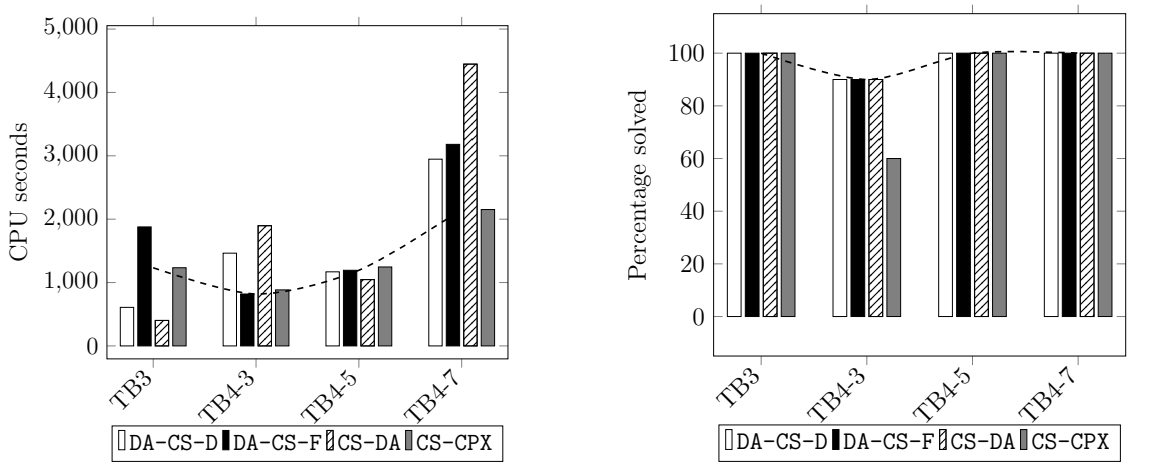
Table 3.4 summarizes the results obtained with the cut–and–solve algorithm using the dual ascent algorithm for solving the sparse problems. It is evident that the cut–and–solve framework in conjunction with the semi–Lagrangean based dual ascent algorithm is very effective in reducing the size of the integer linear programs that need to be solved. In the column headed “SP size” we report the average size of the final sparse problem as a percentage of the original instance size. On average, the sparse problems consist of less than 10 percent of the variables in the original problem. We note that the results confirm the findings from Gadegaard et al. (2016d), that is that the CFLP relaxation of the SSCFLP provides a good approximation as regards the facilities to be opened, and we note that the cut-and-solve algorithm requires less than 3 iterations to find an optimal solution. In addition, the dual ascent algorithm used to solve the subproblems only uses about 2 iterations on average to solve the sparse problems to optimality. Furthermore, we observed that CPLEX very quickly found near optimal solutions to the semi–Lagrangean subproblems and that the main

Table 3.4: Results obtained on TB3 and TB4 using cut–and–solve algorithm incorporating a dual ascent algorithm for solving the sparse problems, `CS-DA`.

				Iterations			
TB	#	size	R	CS[1]	DA[2]	SP size	CPU
3	5	30×200	$1.73 - 1.98$	2.4	2.2	15.5	75.55
	5	30×300	$2.88 - 3.49$	3.8	1.7	5.1	1115.67
	5	60×300	$3.42 - 5.78$	2.0	2.4	8.9	80.32
	5	80×400	$3.50 - 7.30$	3.0	1.7	4.8	337.31
Averages				2.8	2.0	8.5	402.21
4	5	80×500[3]	3	2.0	2.0	9.0	2347.01
	5		5	2.3	1.6	7.0	2406.79
	5		7	2.0	1.8	7.1	7157.59
	5	100×400[3]	3	2.5	1.0	8.4	709.42
	5		5	2.6	1.6	5.9	659.62
	5		7	2.4	2.2	5.2	1735.84
Averages				2.3	1.65	7.3	2398.84

[1] Number of iterations of the cut–and–solve algorithm.
[2] Average number of iterations of the dual ascent algorithm used to solve a sparse problem. [3] We were not able to solve instance $p4$ of size 80×500. The program was killed by the operating system as the size of the branching tree grew too large.

effort in the subproblems was spent on increasing the lower bound. Memory consumption therefore only became prohibitive in one instance. In Section 3.5.5, we discuss how the solver proposed in Gadegaard et al. (2016d) was immensely challenged by the task of finding a feasible solution to the sparse problems when the ratio between total capacity and total demand was small. The `CS-DA` algorithm performs slightly better on the instances of TB3 compared to the `DA-CS` algorithms analyzed in the previous section. However, for the larger instances of TB4, the roles are reversed. One reason for this change in performance is that the `CS-DA` solves a CFLP as a relaxation of the full problem. When the instances get larger, this large CFLP becomes more time consuming to solve. The CFLP solved as a relaxation of the semi–Lagrangean subproblem in `DA-CS` is, however, significantly reduced by the rule $c_{ij} \geq \mu \Rightarrow x_{ij} = 0$.



(a) Time used by the four algorithms on the large instances.

(b) Number of instances solved by the algorithms.

Figure 3.3: Comparison with the cut–and–solve algorithm proposed in Gadegaard et al. (2016d). The i in TB4-i indicates the ratio between total demand and total capacity.

3.5.5 Comparing with a state–of–the–art solver for the SSCFLP

Figure 3.3 depicts the performance of the three algorithms `DA-CS`, `CS-DA`, and `CS-CPX`. From Figure 3.3a we see that none of the algorithms perform consistently better than the others. For the instances of TB3, `CS-DA` performs extremely well, but this performance deteriorates as the instances and the capacity to demand ratios increase. We also see that the augmented cut–and–solve algorithm, `CS-CPX`, performs very well on most of the instances, which confirms that this algorithm is indeed very efficient for solving the SSCFLP. Figure 3.3a shows that the `CS-CPX` algorithm performs just as well or even better than the semi–Lagrangean based algorithms. It should be noted that the time–averages are taken over instances that were actually solved. This makes the `CS-CPX` look better than is actually the case. When we look at Figure 3.3b we see that the dual ascent based algorithms are more robust compared to `CS-CPX` in case the instances have a small ratio between total capacity and total demand. `CS-CPX` failed on four of the 10 instances of TB4 having a ratio $R = 3$. In all four cases CPLEX failed to find a feasible solution to the first sparse problem and consequently no nodes could be fathomed based on bound. The operating system would relatively quickly kill the program due to the excessive memory consumption.

The results we have obtained suggests that a hybrid solver, which based on the characteristics of the instance chooses the algorithmic approach that should be taken. If the instance has a large R–value or if it is small, then the cut–and–solve algorithm `CS-CPX` should be used. On the other hand, if the instance is large and has a small R–value the solver

should use the dual ascent algorithm `DA-CS-F`. The dashed line in Figure 3.3 depicts the performance of such an approach, showing that in most cases this would be a very efficient solution procedure for the SSCFLP.

3.6 Conclusion

In this paper we have proposed two new ways of integrating semi–Lagrangean based dual ascent and cut–and–solve for the single–source capacitated facility location problem (SSCFLP). The first approach used a cut–and–solve algorithm as a subproblem solver in a dual ascent algorithm where the cut–and–solve algorithm only solved the subproblems to optimality if a good primal feasible solution was found. The second approach augmented an existing cut–and–solve algorithm by solving the sparse problems using a dual ascent algorithm based on a semi–Lagrangean relaxation of the SSCFLP. We proposed to update the semi–Lagrangean multiplier based on the solution to the semi–Lagrangean subproblem and tested four distinct initialization strategies and found that for the cut–and–solve algorithm using dual ascent for solving the subproblems, the initialization strategy that worked the best was setting the initial value of the dual multiplier such that a 25 percent reduction of the subproblem was obtained. The results were slightly more ambiguous for the dual ascent algorithm using cut–and–solve as the subproblem solver. Here, initializing using a feasible solution or using the dual multipliers of the assignment constraints worked equally well. In order to speed up the computations and limit the memory consumption we proposed a new way of generating cutting planes for a substructure of the SSCFLP combining the capacity constraints and the assignment constraints.

We showed empirically that the dual ascent based algorithms solved more instances compared to an efficient cut–and–solve algorithm proposed in the literature, when the instances were large and had a tight capacity to demand ratio. We argued that the primary reason is that the feasibility problem corresponding to the Lagrangean subproblem is polynomially solvable in contrast to the $\mathcal{NP}$–hard feasibility problem of the SSCFLP. With these new methodologies it was possible to increase the size of the instances solvable by almost 25 percent.

In the future, it will be interesting to investigate whether improved updating strategies for the dual multiplier and fixation of variables based on Lagrangean penalty tests can lead to improved performance through fewer subproblems. It is also of interest to examine if the methodology can be used to solve more complex location problems such as dynamic or multi-stage models.

3.7 Acknowledgments

The author would like to thank Associate Professors Andreas Klose and Lars Relund Nielsen for insightful comments and suggestions. This work was partially supported by a grant from Købmand Ferdinand Sallings Mindefond.

3.A Detailed information on the new instances

Table 3.5 gathers information on the new instances generated for this paper. The columns headed *id*, *size*, and R contain the name of the instance, the size of the instance ($|\mathcal{I}| \times |\mathcal{J}|$), and the ratio between total supply and total demand, respectively. The next three columns headed LB^{LP}, LB^{CP}, and Gap closed report the lower bound obtained from the LP relaxation before adding cuts, the lower bound produced by the cutting plane algorithm, Algorithm 3.6, and the percentage gap closed by the cutting planes. The last two columns headed UB and $\%gap$ contain the value of the best known upper bound and the percentage gap between this upper bound and the best known lower bound. All instances are publicly available (see Gadegaard, Klose, and Nielsen (2016e)).

Table 3.5: Detailed information on the new instances generated for this paper.

id	size	R	LB^{LP}	LB^{CP}	%Gap closed	UB	%gap
p1	80×500	3	29194.7	31341.5	90.34	31571	0
p2			29291.0	31352.4	91.17	31552	0
p3			30417.0	32714.5	93.93	32863	0
p4			30087.6	32179.3	48.68	34384	7
p5			28405.2	30781.1	92.92	30962	0
p6		5	20183.6	24957.7	96.36	25138	0
p7			20038.5	24927.1	96.41	25109	0
p8			20998.6	25836.9	96.33	26021	0
p9			20537.3	24829.1	95.63	25025	0
p10			19784.6	24830.4	98.39	24913	0
p11		7	16214.0	22981.4	97.58	23149	0
p12			15792.8	22501.6	98.77	22585	0
p13			16847.2	23936.1	97.75	24099	0
p14			16395.2	22928.8	97.87	23071	0
p15			15948.1	23246.1	97.54	23430	0
p16	100×400	3	34992.1	36278	87.60	36460	0
p17			32707.5	33933.4	83.82	34170	0
p18			32533.1	33909.8	88.94	34081	0
p19			30153.2	31584.7	86.56	31807	0
p20			32463.6	33570.8	86.20	33748	0
p21		5	22870.7	25783.2	95.76	25912	0
p22			21720.8	24936	96.49	25053	0
p23			21782.7	25268	95.35	25438	0
p24			20207.6	23699.9	94.84	23890	0
p25			21578.2	24845.9	94.09	25051	0
p26		7	17609.2	22219.7	98.48	22291	0
p27			16823.0	21765.6	96.44	21948	0
p28			17073.3	22234.1	97.09	22389	0
p29			15835.8	21059.9	98.23	21154	0
p30			16771.3	21696.5	95.71	21917	0

Fourth Chapter

A bi–objective approach to discrete cost–bottleneck location problems

This chapter is based on the paper Gadegaard, Klose, and Nielsen (2016a).
The research was conducted during 2015 and 2016 and submitted to
Annals of Operations Research in March 2016
The work was presented at IFORS 2014, Barcelona, Spain.

A bi–objective approach to discrete cost–bottleneck location problems

Sune Lauth Gadegaard[†], Andreas Klose[*] and Lars Relund Nielsen[†]

[†]Department of Economics and Business Economics, Aarhus University, Denmark,
{sgadegaard, larsrn}@econ.au.dk
[*]Department of Mathematics, Aarhus University, Denmark, aklose@imf.au.dk

Abstract

This paper considers a family of bi–objective discrete facility location problems with a cost objective and a bottleneck objective. A special case is, for instance, a bi-objective version of the (vertex) p-centdian problem. We show that bi-objective facility location problems of this type can be solved efficiently by means of an ε-constraint method that solves at most $(n-1) \cdot m$ minisum problems, where n is the number of customer points and m the number of potential facility sites. Additionally, we compare the approach to a lexicographic ε-constrained method that only returns efficient solutions and to a two–phase method relying on the perpendicular search method. We report extensive computational results obtained from several classes of facility location problems. The proposed algorithm compares very favorably to both the lexicographic ε-constrained method and to the two phase method.

Keywords: discrete facility location; bi-objective optimization; ε-constrained method; lexicographic optimization.

4.1 Introduction

Single objective location analysis usually distinguishes between two major types of objective functions. Whilst the objective of a *minisum* location problem consists in minimizing average (weighted) costs, a solution to a *minimax* location problem aims at minimizing the maximal (weighted) distance between customer points and facilities. We will refer to the two objectives as a *cost objective* and a *bottleneck objective*, respectively. On networks, the prototype cost and bottleneck location models are the p-median (Hakimi, 1965) and p-center problem (Hakimi,

1964), respectively. The survey papers by Reese (2006) and Revelle, Eiselt, and Daskin (2008) review the extensive literature on these two important network location problems. Within the subject of discrete facility location, the simplest and also most studied problem with cost–objective is the uncapacitated facility location problem proposed by Balinski (1965). For discrete facility location the bottleneck–objective has not received as much attention as the cost–objective, but Dearing and Newruck (1979) study a capacitated facility location problem with such a bottleneck–objective.

As the cost–objective focuses on minimizing the total/average cost of supplying demand points from supply points, it often provides solutions in which remote and sparsely populated areas are discriminated in terms of accessibility compared to centrally situated and highly populated areas. On the other hand, locating facilities according to the bottleneck objective (focussing on equity rather than efficiency) may cause a large increase in the total cost, thus generating a substantial loss in cost efficiency. This fact was recognized by Halpern (1976) who minimized a convex combination of the two objectives such that both efficiency and equity were expressed in the resulting solution. On a network, this model is known as the centdian problem. Halpern (1978) showed that when placing *one* facility on an undirected graph so as to minimize the centdian objective, only a finite set composed of the set of vertices and the set of so–called local centers need to be examined. For the case where multiple facilities should be placed, this finite set needs to be expanded to the (still finite) set presented in Perez-Brito, Moreno-Perez, and Rodriguez-Martin (1997).

Recognizing the fact that only supported efficient solutions can be found using convex combinations of the objectives, it seems obvious to apply a bi-objective approach to the cost–bottleneck problem. Instead of only considering a fixed convex combination of the two objectives, the entire set of non-dominated outcomes will then be generated. The need to balance efficiency and equity does, however, also arise in the case of other, similar problems. In case of the transportation problem, the literature distinguishes between the classical Hitchcock-type and the bottleneck transportation problem (Hammer, 1969; Garfinkel and Rao, 1971). Whilst the objective of the former is to minimize total transportation cost, the latter aims at minimizing the maximum time required to transport all supplies to the destinations. In this case it seems relevant to find a balance between these two objectives, leading to the bi-objective "bottleneck-cost" transportation problem studied by Pandian and Nataraja (2011). This problem can in fact be solved in polynomial time. Combining a cost and a bottleneck objective is also highly relevant in the context of discrete facility location problems such as, for instance, the uncapacitated and capacitated facility location problems. The bottleneck objective may then, in particular, refer to a customer service objective that aims at keeping maximum delivery times small. Facility location models also play a role in the area of supplier selection (Current and Weber, 1994). In this case, the cost objective

may refer to the total procurement cost, whereas the bottleneck objective is to minimize the (largest) lead time.

In the literature on planar and network multi–objective facility location problems, most papers concentrate on specific problems and on methodological and theoretical results (see e.g. Hamacher and Nickel (1996) for planar and Hamacher, Labbé, and Nickel (1999) for network location problems). The opposite is true for discrete multi–objective facility location, where the attention has primarily been on applications (see e.g. the survey in Nickel, Puerto, and Rodríguez-Chía (2005)). However, a few contributions with a methodological perspective on multi–criteria discrete location problem exist. In Ross and Soland (1980) the set of Pareto optimal solutions is characterized by rewriting the location problem as a generalized assignment problem with an additional constraint. However, the authors argue not to generate all efficient solutions, and propose an *interactive* approach instead. Fernández and Puerto (2003) consider a multi–objective version of the uncapacitated facility location problem (UFLP). A multi–objective dynamic programming approach is proposed based on a decomposition of the UFLP into a facility selection problem and a demand allocation problem. We have, however, not been able to find any papers investigating, in a discrete bi–objective setting, the combination of a cost objective and a bottleneck. For surveys on multi–objective facility location problems we refer the reader to Nickel et al. (2005) and Farahani, SteadieSeifi, and Asgari (2010).

In this paper we, therefore, suggest a bi-objective approach to balance cost minimization and maximum transportation times for a family of discrete facility location problems comprising many well studied location problems as special cases. We establish the computational complexity of the problem and prove the problem to be tractable. Furthermore, we propose a scheme for solving the problem by means of an ε–constrained method. We suggest two ways to accommodate the issue of generating weakly efficient solutions: First, a simple change of the cost matrix that imposes the non–linear ε–constraint on the center objective. Second, we outline a scheme based on lexicographic branch–and–bound which generates no weakly efficient solutions. Last, we compare the ε–constraint algorithm to a two–phase implementation. The main contributions of the paper can thus be summarized as follows:

1. We propose a family of discrete bi–objective facility location problems, encompassing many problems known from the literature.
2. We show that even though the problems are generally $\mathcal{NP}$–hard, they are computationally tractable in a certain sense.
3. We propose a simple methodology, based on the ε–constraint method, for solving these problems.
4. Through extensive experiments we show that the methodology is indeed very efficient compared to other algorithms from the literature.

The remaining of this paper is organized as follows: Section 4.2 introduces the cost–bottleneck location problem and Section 4.3 outlines the preliminaries of bi–objective combinatorial optimization and establishes the computation complexity of the problem. Section 4.4 outlines the ε–constraint algorithm proposed to solve the problem and finally, results from extensive computational experiments are reported in Section 4.5.

4.2 The bi–objective cost–bottleneck location problem

A large number of facility location problems are special cases of the general integer program

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in I} f_i y_i \tag{4.1a}$$

$$\text{s.t.:} \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \tag{4.1b}$$

$$x_{ij} - y_i \leq 0, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}, \tag{4.1c}$$

$$(\mathbf{x}_i, y_i) \in \mathcal{X}_i, \quad \forall i \in \mathcal{I}, \tag{4.1d}$$

$$y \in \mathcal{Y}. \tag{4.1e}$$

Here $\mathcal{J}$ is the set of demand points which need to be served by a set of open facilities picked among the potential facility sites in $\mathcal{I}$. The cost of servicing all of customer j's demand from facility i amounts to $c_{ij} \geq 0$ while opening a facility i results in a fixed charge of $f_i \geq 0$. It is assumed, that both c_{ij} and f_i are *non–negative integers*. Constraints (4.1b) ensure that all demand at customer j is covered by allocating the demand to at least one open facility while constraints (4.1c) ensure that demand can only be allocated to open facilities. Constraints (4.1d) restrict the possible assignments $\mathbf{x}_i = (x_{ij})_{j\in\mathcal{J}}$ of demand points to facilities. Possible assignments are *single–sourced* if demand points are assigned to only one facility, that is $\mathcal{X}_i \subseteq \{0,1\}^{|\mathcal{J}|+1}$. Otherwise we have $\mathcal{X}_i \subseteq [0,1]^{|\mathcal{J}|} \times \{0,1\}$. Finally, $\mathcal{Y} \subseteq \{0,1\}^{|\mathcal{I}|}$ may introduce further restrictions on the locational decisions y_i.

Special cases of the program (4.1) comprise among others the uncapacitated facility location problem, the capacitated facility location problem with and without single-sourcing, and the p–median problem.

When the sum of costs is minimized, the relation to the individual demand point is not taken into account meaning that in a spatial setting some customers might be located far from the open facilities. To overcome this issue we introduce yet another objective, namely to minimize the travel time of the *worst* assignment, effectively introducing the bottleneck objective

$$\min \max_{i\in\mathcal{I}, j\in\mathcal{J}} \{t_{ij} \ :\ x_{ij} > 0\}. \tag{4.2}$$

It is again assumed that t_{ij} is a non–negative integer for all $i \in \mathcal{I}$ and all $j \in \mathcal{J}$. In this setting, one can consider the t_{ij} as the *travel time* from demand point j to facility i while c_{ij} is the *cost* incurred by this assignment. The introduction of this additional objective function leads to the BOCO problem

$$\begin{aligned} \min \ & \left(\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i \ , \ \max_{i\in\mathcal{I}, j\in\mathcal{J}} \{t_{ij} \ : \ x_{ij} > 0\} \right) \\ \text{s.t.:} \ & \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \\ & x_{ij} - y_i \leq 0, \quad \forall i \in \mathcal{I}, \ j \in \mathcal{J}, \\ & (\mathbf{x}_i, y_i) \in \mathcal{X}_i, \quad \forall i \in \mathcal{I}, \\ & y \in \mathcal{Y}, \end{aligned} \tag{4.3}$$

which will be referred to as the bi–objective cost–bottleneck location problem (BO–CBLP).

4.3 Preliminaries

In the remainder of this paper, we will adopt the notation from Ehrgott (2005) to induce orderings on $\mathbb{R}^2$. Let $z^1, z^2 \in \mathbb{R}^2$. Then

$$\begin{aligned} z^1 \leq z^2 \Leftrightarrow & z^1_k \leq z^2_k, && k = 1,2 \text{ and } z^1 \neq z^2 \\ z^1 <_{\text{lex}} z^2 \Leftrightarrow & z^1 \neq z^2 \text{ and } z^1_q < z^2_q, && \text{where } q = \min\{k = 1,2 \ : \ z^1_k \neq z^2_k\} \\ z^1 \leq_{\text{lex}} z^2 \Leftrightarrow & z^1 = z^2 \text{ or } z^1 <_{\text{lex}} z^2 \end{aligned}$$

If $z^1 \leq z^2$ we say that z^1 *dominates* z^2. Similarly, if $z^1 \leq_{\text{lex}} z^2$ we say that z^1 *lexicographically dominates* z^2. Note the implication that if $z^1 \leq z^2$ then $z^1 \leq_{\text{lex}} z^2$.

The focus of this section will be on a generic *bi-objective combinatorial optimization* (BOCO) problem of the form

$$\min\{(f_1(x), f_2(x)) \ : \ x \in \mathcal{X}\} \tag{4.4}$$

where $f_1 : \mathcal{X} \to \mathbb{R}$ and $f_2 : \mathcal{X} \to \mathbb{Z}$ are defined over the mixed integer set $\mathcal{X} \subseteq \{0,1\}^{n_1} \times [0,1]^{n_2}$. The set $\mathcal{X}$ is the set of feasible solutions, also referred to as the feasible set in *decision space*. The image of $\mathcal{X}$, $\mathcal{Z} \coloneqq f(\mathcal{X}) \subseteq \mathbb{Z}^2$, is referred to as the feasible set in *outcome space*. For brevity of notation, we will often write $f(x) = (f_1(x), f_2(x))$. It is not obvious what is meant by (4.4). To clarify this we use the concept of Pareto optimality or efficiency:

Definition 4.1. A feasible solution $\hat{x} \in \mathcal{X}$ is called Pareto optimal or *efficient* if there does not exist any $\bar{x} \in \mathcal{X}$ such that $f(\bar{x}) \leq f(\hat{x})$. The image $f(\hat{x}) = (f_1(\hat{x}), f_2(\hat{x}))$ is then called *non-dominated*.

A feasible solution $\hat{x} \in \mathcal{X}$ is called weakly efficient if there does not exist any $\bar{x} \in \mathcal{X}$ such that $f_1(\bar{x}) < f_1(\hat{x})$ *and* $f_2(\bar{x}) < f_2(\hat{x})$.

From Definition 4.1 we have the following connection between the lexicographic ordering $\leq_{\text{lex}}$ and efficiency:

Lemma 4.1 (Ehrgott (2005))**.** *Let $\hat{x} \in \mathcal{X}$ satisfy $f(\hat{x}) \leq_{\text{lex}} f(x)$ for all $x \in \mathcal{X}$. Then $\hat{x}$ is efficient.*

Let $\mathcal{X}_E$ denote the set of efficient solutions and let $\mathcal{Z}_N = f(\mathcal{X}_E)$ be the image of this set. The set of non-dominated solutions $\mathcal{Z}_N$ will also be referred to as the *non–dominated frontier*. We distinguish between efficient solutions which are supported, extreme supported, and unsupported.

Definition 4.2. 1. A solution $x \in \mathcal{X}$ is a *supported* efficient solution if there exists a $\lambda > 0$ such that x is an optimal solution to

$$\min\{(\lambda c^1 + (1-\lambda)c^2)x \ : \ x \in \mathcal{X}\}$$

The corresponding outcome vector, $z := f(x)$, is called a *supported* non–dominated outcome vector. The set of supported non–dominated outcomes is denoted $\mathcal{Z}_{sN}$.

2. If z is also an extreme point of $\text{conv}(\mathcal{Z}_N)$, then x is called an *extreme* supported efficient solution and the outcome vector z is called an *extreme* supported non–dominated outcome vector.

3. If $x \in \mathcal{X}_E$ and $z := f(x) \notin \mathcal{Z}_{sN}$, then x is said to be an *unsupported* efficient solution and z is called an *unsupported* non–dominated outcome.

If two feasible solutions $x_1, x_2 \in \mathcal{X}$ map into the same outcome vector, $z \in \mathcal{Z}$, that is $f(x_1) = f(x_2) = z$, the solutions x_1 and x_2 are called *equivalent*. The literature is not always precise on the outcome of a proposed algorithm and we therefore employ the following definition from Hansen (1980):

Definition 4.3. A *complete* set $\mathcal{C}$ is a set of efficient solutions such that all $x \in \mathcal{X} \backslash \mathcal{C}$ are either dominated by or equivalent to at least one $\hat{x} \in \mathcal{C}$. Moreover we distinguish between two types of complete sets:

1. A *minimal complete set*, $\mathcal{C}_{\min}$ is a complete set without equivalent solutions. Any complete set contains a minimal complete set.

2. The *maximal complete set*, $\mathcal{C}_{\max}$, is the complete set including all efficient solutions, i.e., $\mathcal{C}_{\max} = \mathcal{X}_E$.

In this paper the focus is on generating a set $\mathcal{C}_{\min}$, i.e., the efficient solutions found for (4.4) constitute a minimal complete set. Finally, the concept of computational tractability is defined:

Definition 4.4 (Ehrgott (2005))**.** The BOCO problem (4.4) is called *intractable* if the cardinality of $\mathcal{Z}_N$ can be exponential in the size of the input and tractable otherwise.

4.3.1 Characteristics of the BO–CBLP problem

Despite the relatively simple nature of the problem, the BO–CBLP (4.3) is a difficult BOCO problem. The computational complexity of the BO–CBLP is stated in the following proposition.

Proposition 4.1. *The BO–CBLP* (4.3) *is $\mathcal{NP}$–hard.*

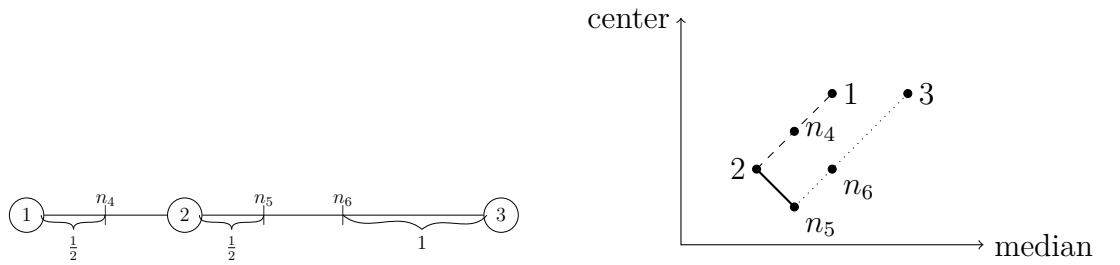
Proof. We show this by reducing the set covering problem to BO–CBLP. Let $c_{ij} = 0$ if demand point $j \in \mathcal{J}$ can be covered by facility $i \in \mathcal{I}$ and $c_{ij} = M$ otherwise. Furthermore, let $t_{ij} = c_{ij}$ for all $i \in \mathcal{I}$ and all $j \in \mathcal{J}$ and $f_i = 1$ for all $i \in \mathcal{I}$. Let K be the minimum number of facilities needed to cover all customers. Then the problem exhibits two non–dominated solutions. The first opens the K facilities needed to cover the demand points, resulting in the outcome vector $(K, 0)$. Any efficient solution mapping to another non–dominated outcome does not cover at least one customer. Hence such an efficient solution opens any single facility resulting in the outcome $(1, M)$. The latter is trivial to compute, but the first requires to solve the set covering problem. □

However, despite the fact that the BO–CBLP is $\mathcal{NP}$–hard, the problem is in fact *tractable* in the sense of Definition 4.4:

Proposition 4.2. *The BO–CBLP is tractable.*

Proof. As the travel time matrix $(t_{ij})_{i\in\mathcal{I},j\in\mathcal{J}}$ comprises at most $|\mathcal{I}| \times |\mathcal{J}|$ different values, the center objective (4.2) can attain no more than $|\mathcal{I}| \times |\mathcal{J}|$ different values, implying $|\mathcal{Z}_N| \leq |\mathcal{I}| \times |\mathcal{J}|$, which is polynomial in the input size. □

Proposition 4.2 tells us, that even though the BO–CBLP is $\mathcal{NP}$–hard, we only need to solve a polynomial number of $\mathcal{NP}$–hard problems in order to generate the non–dominated frontier, $\mathcal{Z}_N$.



(a) Illustration of the graph used in the proof of Theorem 4.1.

(b) Solutions in outcome space for the graph used in the proof of Theorem4.1.

Figure 4.1: Illustrations of the example used in the proof of Theorem 4.1.

4.3.2 A link to the weighted *p*–centdian problem

The p–centdian problem is a combination of the p–median and the p–center problems where the objective function is a convex combination of the median/cost and the center/bottleneck objectives. Given a graph $G = (V, E)$, let $P(G)$ be the set of all points on G and let $d(l, h)$ be the distance of a shortest path between points l and h on G (note that the points l and h might be *interior* points on the edges of G). Furthermore, given a set of points $S \subseteq P(G)$ we define $d(S, h) = \min\{d(l, h) \ : \ l \in S\}$. In addition, let $w_j \geq 0$ and $v_j \geq 0$ be weights of the node $j \in V$ representing, for example, the number of customers or some other measure of attractiveness of the node j. For $0 \leq \lambda \leq 1$ the p–centdian problem may then be stated as

$$\begin{aligned} &\min \ \lambda \sum_{j \in V} w_j d(S, j) + (1 - \lambda) \max_{j \in V} v_j d(S, j) \\ &\text{s.t.: } S \subseteq P(G) \\ &\qquad |S| = p \end{aligned} \tag{4.5}$$

It has been shown in Perez-Brito et al. (1997) that there exists a finite set of points on G containing an optimal solution to the p–centdian. In what follows, the problem

$$\min\{\left(\sum_{j \in V} w_j d(S, j), \max_{j \in V} v_j d(S, j)\right) \ : \ S \subseteq P(G), \ |S| = p\}$$

is denoted the bi–objective p–centdian problem. It turns out that there can be infinitely many Pareto optimal solutions to this problem.

Theorem 4.1. *The set of Pareto optimal solutions to the bi–objective p–centdian problem on a network can be uncountably infinite even for $p = 1$ when $S \subseteq P(G)$.*

Proof. We show the result by giving an example having this property. Consider the graph given in Figure 4.1a, where the edge lengths are given by $d(1, 2) = d(2, 1) = 1$, $d(2, 3) = d(3, 2) = 2$, and $w_j = v_j = 1$ for all $j \in V$. Furthermore, suppose $p = 1$, that is, we want to

place one facility on G. The intersection points (see Kariv and Hakimi (1979) for a definition) on G are denoted by n_4, n_5, and n_6. It is easily verified that locating a facility at node 2 is an optimal solution to the 1–median problem with outcome vector $(3, 2)$ whereas placing a facility at n_5 is an optimal solution for the 1–center problem, resulting in the outcome vector $(3.5, 1.5)$. As the solutions are unique optimal solutions they are *efficient*. In Figure 4.1b the outcome vectors for all points on the graph G have been plotted. All points on the edge $(1, 2)$ map into the dashed line, while all solutions on the edges (n_5, n_6) and $(n_6, 3)$ map into the dotted line. However, all points on the edge $(2, n_5)$ map into the solid line which is non–dominated. As there are an uncountable infinite number of points on this edge, the result follows. □

Note that the bi–objective p–centdian is *not* covered by the general BO–CBLP as the set I of potential facility sites has to be finite in the definition of the BO–CBLP. However, letting the sets I and J equal the set V, the (vertex) p–centdian problem (4.5) can be stated as the following *scalarized* version of the BO–CBLP:

$$\begin{aligned}
\min\ & \lambda \left(\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i \right) + (1-\lambda) \max_{i\in\mathcal{I}, j\in\mathcal{J}} \{t_{ij} \ : \ x_{ij} > 0\} \\
\text{s.t.:}\ & \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \\
& (\mathbf{x}_i, y_i) \in \mathcal{X}_i, \quad \forall i \in \mathcal{I}, \\
& y \in \mathcal{Y},
\end{aligned}$$

where $\mathcal{X}_i = \{(\mathbf{x}_i, y_i) \in \{0,1\}^{|\mathcal{J}|+1} \ : \ x_{ij} \leq y_i\}$, $\mathcal{Y} = \{y \in \mathbb{R}^n \ : \ \sum_{i\in\mathcal{I}} y_i = p\}$, $f_i = 0$, $c_{ij} = w_j d(i,j)$, and $t_{ij} = v_j d(i,j)$ for all $i \in \mathcal{I}$ and $j \in \mathcal{J}$. In many practical applications it will suffice to consider placing facilities only at the nodes of the graph. For $0 < \lambda < 1$ a solution to the vertex–p–centdian problem corresponds to a supported efficient solution of the BO–CBLP (4.3) with the sets $\mathcal{I}$, $\mathcal{J}$, $\mathcal{X}_i$, $\mathcal{Y}$ and the coefficients f_i, c_{ij}, and t_{ij} defined as above. This means that solving the BO–CBLP yields a solution to the vertex–p–centdian problem *for each value of* $\lambda > 0$. Most literature considers special structured graphs such as trees when solving the p–centdian problem. This approach offers a means to solve the vertex–p–centdian problem for all values of $0 < \lambda < 1$ on a general graph with no assumption other than that facilities should be located on nodes only.

4.4 Solution methodologies

One of the most well–known approaches for establishing the complete set of Pareto optimal solutions to a bi–objective optimization problem is probably the *ε-constrained method*. This method turns one of the objectives into a constraint. The scalar ε represents the upper bounds

Input: Functions f_1 and f_2 and a feasible set $\mathcal{X}$.
Output: A solution $(\mathcal{C}_{\min}, \mathcal{Z}_N) \subseteq \mathcal{X} \times \mathcal{Z}$ to (4.4) or a proof that $\mathcal{X}_E = \emptyset$.

Step 0: (Initialization) Set $\mathcal{C}_{\min} = \emptyset$, $\mathcal{Z}_N = \emptyset$, $\varepsilon = \infty$ and $k = 1$.

Step 1: (Subproblem) If $\min\{f_i(x) \ : \ x \in \mathcal{X}, \quad f_j(x) \leq \varepsilon\}$ is feasible let x^k be an optimal solution. Else go to *Step 3*.

Step 2: (Update) Set $\mathcal{C}_{\min} = \mathcal{C}_{\min} \cup \{x^k\}$, $\mathcal{Z}_N = \mathcal{Z}_N \cup \{f(x^k)\}$, $\varepsilon = f_2(x^k) - 1$ and $k = k + 1$. Return to *Step 1*.

Step 3: Remove dominated solutions from $\mathcal{C}_{\min}$ and their outcome vectors from $\mathcal{Z}_N$ and return $(\mathcal{C}_{\min}, \mathcal{Z}_N)$ as an optimal solution.

Algorithm 4.1: Summary of the ε–constraint based algorithm.

on the objective, and by varying this scalar in an appropriate way, the complete efficient frontier can be generated. Recently, Bérubé, Gendreau, and Potvin (2009) successfully applied this method to the traveling salesman problem with profits. The literature also suggests a number of variations of this method. Two recent versions are: Filippi and Stevanato (2013) combine the weighted sum scalarization technique with the ε-constrained method and show that exactly $2|\mathcal{Z}_N| - 1$ single objective optimization problems have to be solved in order to produce the entire set of non-dominated outcomes. Two box algorithms based on a combination of lexicographic optimization and the ε-constrained method (the lexicographic ε-constrained method) are proposed by Hamacher, Pedersen, and Ruzika (2007). The proposed algorithm also solves at most $2|\mathcal{Z}_N| - 1$ lexicographic optimization problems.

In this section we describe an ε–constrained method for finding a minimal complete set for the BO–CBLP. The two ε–constrained problems arising in terms of the general BOCO (4.4) are

$$\min\{f_i(x) \ : \ x \in \mathcal{X}, \ f_j(x) \leq \varepsilon_j\}, \tag{P_j^i}$$

where $i, j = 1, 2$ and $i \neq j$. It is well known that all non–dominated solutions can be found by varying the ε–parameter in an appropriate manner (see e.g. Ehrgott (2005)). BOCO problems yield a straightforward variation scheme for the ε parameter. One simply constructs a sequence where ε is initially set equal to a substantially large value and progressively lowered. An outline of a generic ε–constrained method is given in Algorithm 4.1.

Note that it is crucial for the procedure that $f_2(x) \in \mathbb{Z}$ for all $x \in \mathcal{X}$ such that a strict improvement in the objective moved into the constraints can be modeled as a less or equal constraint.

A major drawback of the ε–constrained method is that the set of solutions produced by Steps 1 and 2 in Algorithm 4.1 usually contains (weakly) dominated solutions, such that the method presumably solves far more ε–constrained programs than actually required. One

way of preventing this is to obtain a *lexicographically* optimal solution to the ε–constrained subproblem in *Step 1* in Algorithm 4.1. This way, by Lemma 4.1, the solution obtained in this step is guaranteed to be efficient, and *Step 3* can be skipped. We describe how such a lexicographically optimal solution can be obtained in Section 4.4.2. Another weak point of the ε–constrained algorithm is that the ε–constraint might ruin the structure of the underlying problem. This issue is addressed in Section 4.4.1.

4.4.1 The ε–constrained method for BO–CBLP

In the ε–constrained method for BO–CBLP we choose to move the non–linear center–objective into the constraints and obtain the problem

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i \tag{4.6a}$$

$$\text{s.t.:} \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J} \tag{4.6b}$$

$$x_{ij} - y_i \leq 0, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}, \tag{4.6c}$$

$$\max_{i\in\mathcal{I},j\in\mathcal{J}} \{t_{ij} \ :\ x_{ij} > 0\} \leq \varepsilon, \tag{4.6d}$$

$$(\mathbf{x}_i, y_i) \in \mathcal{X}_i, \quad \forall i \in \mathcal{I} \tag{4.6e}$$

$$y \in \mathcal{Y}. \tag{4.6f}$$

The program (4.6a)–(4.6f) differs from the general facility location problem (4.1) only in the non–linear constraint (4.6d). The presence of this constraint might ruin the structure of the problem and, furthermore, the program cannot be handed directly to a MILP–solver.

Fortunately, for the BO–CBLP the ε–constraint (4.6d) can be taken into account by simple variable elimination:

Lemma 4.2. *The constraints*

$$x_{ij} = 0, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J} : t_{ij} > \varepsilon \tag{4.7}$$

are equivalent to the constraint $\max_{i\in\mathcal{I},j\in\mathcal{J}}\{t_{ij} \ :\ x_{ij} > 0\} \leq \varepsilon$.

This variable elimination, constraints (4.7), can easily be implemented while maintaining the structure of the general facility location problem, simply by changing the cost matrix as follows

$$c_{ij} = \begin{cases} c_{ij}, & \text{if } t_{ij} \leq \epsilon \\ M, & \text{otherwise,} \end{cases}$$

where M is a sufficiently large number. With this transformation of the cost matrix, the ε–constrained facility location problem, (4.6a)–(4.6f), can be solved as an ordinary facility

Input: Cost matrix c_{ij} and travel time matrix t_{ij}.
Output: A solution $(\mathcal{C}_{\min}, \mathcal{Z}_N)$ to the BO-CBLP problem (4.3).

Step 0: (Initialization) Set $\mathcal{C}_{\min} = \emptyset$, $\mathcal{Z}_N = \emptyset$, $\varepsilon = \infty$, $k = 1$, and $M = \infty$.

Step 1: (Subproblem) Solve the facility location problem (4.1) with assignment cost matrix c. Let (x^k, y^k) be an optimal solution and set $z_1^k = \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij}^k + \sum_{i\in\mathcal{I}} f_i y_i^k$ and $z_2^k = \max_{i\in\mathcal{I},j\in\mathcal{J}}\{t_{ij} \,:\, x_{ij}^k > 0\}$. If $z_2^k = M$ go to *Step 3* else go to *Step 2*.

Step 2 (Update) Set $\mathcal{C}_{\min} = \mathcal{C}_{\min} \cup \{(x^k, y^k)\}$, $\mathcal{Z}_N = \mathcal{Z}_N \cup \{(z_1^k, z_2^k)\}$ and $\varepsilon = z_2^k - 1$. Update the cost matrix

$$c_{ij} = \begin{cases} c_{ij}, & \text{if } t_{ij} \le \epsilon \\ M, & \text{otherwise,} \end{cases}$$

and set $k = k + 1$. Return to *Step 1*.

Step 3 Remove dominated solutions from $\mathcal{C}_{\min}$ and their outcome vectors from $\mathcal{Z}_N$ and return $(\mathcal{C}_{\min}, \mathcal{Z}_N)$ as an optimal solution.

Algorithm 4.2: Summary of the ε–constrained method applied to the BO–CBLP.

location problem as links (i, j) will not be used unless the travel time on the arc is less than or equal to ε. This leads to the ε–constrained algorithm for the BO-CBLP problem given in Algorithm 4.2. The adaptation of the generic ε–constrained algorithm to the BO–CBLP happens in *Step 1* and *Step 2* of Algorithm 4.2. Instead of solving an ε–constrained problem in *Step 1*, we simply solve the single objective *cost modified* facility location problem which preserves its structure. In *Step 2* the ε–constraint is "added" to the model by changing the cost matrix.

Rather intriguingly, this implies that a specialized algorithm for the single objective minisum facility location problem (4.1) can be used to solve the subproblems arising in *Step 1* of Algorithm 4.2, often resulting in quite large problem instances solved relatively fast.

Furthermore, as the center objective $\max_{i\in\mathcal{I},j\in\mathcal{J}}\{t_{ij} \,:\, x_{ij} > 0\}$ attains at most $|\mathcal{I}| \times |\mathcal{J}|$ different values, we have Proposition 4.3:

Proposition 4.3. *The ε–constraint algorithm in Algorithm 4.2 solves at most $(|\mathcal{I}| - 1) \times |\mathcal{J}|$ general facility location problems.*

Proof. The maximum number of problems solved is reached if the algorithm fixes only one x_{ij}–variable in each iteration. However, each customer point $j \in \mathcal{J}$ must have at least one possible assignment for the problem to be feasible. This implies that the algorithm performs at most $(|\mathcal{I}| - 1) \times |\mathcal{J}|$ iterations. □

Corollary 4.1. *The BO–CBLP is $\mathcal{NP}$–hard if and only if the single objective facility location problem* (4.1a)–(4.1e) *is $\mathcal{NP}$–hard.*

Proof. The $\Leftarrow$ direction of the bi–implication follows immediately. The converse follows from the fact that if the single objective facility location problem (4.1a)–(4.1e) can be solved in polynomial time, then so can the BO–CBLP by Proposition 4.3. □

4.4.2 Solving a lexicographic BOCO problem

In this section we describe how a lexicographic optimization problem can be dealt with (the problem is also discussed by Ralphs, Saltzman, and Wiecek (2006) who apply a weighted Chebyshev norm approach). To make the exposition as general as possible, we use the terminology of the general BOCO problem (4.4).

One simple possibility is to employ a scalarization of the BOCO that only puts little weight on the second objective, obtaining the scalarized problem

$$\min\{\lambda f_1(x) + (1-\lambda) f_2(x) \ : \ x \in \mathcal{X}, f_2(x) \leq \varepsilon\}$$

where λ is very close to one. This approach does, however, suffer from "bad scaling" of the objective function coefficients which often leads to problems that are very hard to solve and numerically unstable.

Another approach is to implicitly enumerate all optimal solutions to the ε–constrained problem

$$\min\{f_1(x) \ : \ x \in \mathcal{X}, f_2(x) \leq \varepsilon\}. \tag{4.8}$$

This can be done by modifying the branch–and–cut algorithm used to solve the ε–constraint problem. To that end, let x^* be the current incumbent of the ε–constrained problem and let $z_1^* = f_1(x^*)$ be the corresponding solution value. A node in the branching tree is then pruned only if it is infeasible or if it shows a lower bound *strictly* greater than z_1^*. Conversely, if the lower bound of the node equals z_1^* then the first objective cannot strictly improve in subsequent subproblems. However, there might exist solutions improving the second objective. Solutions in the subsequent subproblems need to strictly improve the second objective in order to improve the solution and therefore the constraint $f_2(x) \leq f_2(x^*) - 1$ can be used as a local cutting plane (or branching constraint).

The incumbent is updated during the modified branch–and–cut algorithm whenever a feasible solution, $\bar{x}$, is found such that $f_1(\bar{x}) < z_1^*$ or (non–exclusively) such that $f_1(\bar{x}) \leq z_1^*$ *and* $f_2(\bar{x}) < f_2(x^*)$.

This way, all optimal solutions to (4.8) are implicitly enumerated and the lexicographically best solution is found. This guarantees that the solution returned is efficient.

Lexicographic branch–and–bound applied to the BO–CBLP problem

The lexicographic combinatorial optimization problem that need to be solved in each iteration of the ε–constrained algorithm in Algorithm 4.2 is

$$\begin{aligned}
\text{lex}\min\ & \left(\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i\ ,\ \max\{t_{ij}\ :\ x_{ij}>0\} \right) \\
\text{s.t.:}\ & \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j\in\mathcal{J}, \\
& x_{ij} - y_i \le 0, \quad \forall i\in\mathcal{I},\ j\in\mathcal{J}, \\
& \max\{t_{ij}\ :\ x_{ij}>0\} \le \varepsilon, \\
& (\mathbf{x}_i, y_i)\in\mathcal{X}_i, \quad \forall i\in\mathcal{I}, \\
& y\in\mathcal{Y}.
\end{aligned}$$

The branch–and–cut algorithm used to solve this problem is implemented as a best first search. A node in the branching tree is pruned if the node is LP–infeasible or if a lower bound, say LB, is strictly greater than the incumbent, z_1^* (tolerances are used to ensure numerical stability). If, on the other hand, $LB \le z_1^*$, it is necessary to distinguish between two different cases. In the first case, when the node is integer feasible, a single child node is created by adding the branching constraints $x_{ij} = 0$ for all $i \in \mathcal{I}$ and $j \in \mathcal{J}$ where $t_{ij} \ge \max_{i\in\mathcal{I},j\in\mathcal{J}}\{t_{ij}\ :\ x_{ij}^*>0\}$. In the second case, when the node shows a fractional LP solution, the node is separated using a variable dichotomy by selecting an integer infeasible variable and creating two child nodes: one forcing the variable to zero, the other to one. The constraints described above that force x_{ij} to zero in order to improve the bottleneck objective are added to both child nodes as well. To determine the branching variable, we adopt the *Pseudocost branching* rule described in the excellent paper by Achterberg, Koch, and Martin (2005).

4.5 Computational results

The purpose of the computational tests is fourfold: first, we examine whether it is worthwhile to compute a lexicographically optimal solution to the subproblems arising in the ε–constraint algorithm. Secondly, we examine the extent to which the proposed ε–constrained algorithm is efficient for solving different BO–CBLP problems. Next, we test if the methodology is appropriate for solving these problems. This is done by comparing the ε–constrained method to an implementation of the two–phase method. And finally, a customized solver for the single–source capacitated facility location problem is used to test to which extent such a solver can be used to speed up the computation of the non–dominated set. In total, 1398 different problems have been solved and four algorithmic approaches have been tested.

Table 4.1: Description of the column headings

Heading	Description
#	Number of instances over which the averages are taken in the corresponding row.
Size	Displays the size of the instances of the corresponding row as $\lvert\mathcal{I}\rvert \times \lvert\mathcal{J}\rvert$.
Time	Reports the average time consumption in CPU seconds for calculating the entire frontier.
% times	Average of the percentage $\frac{\text{CPU time}[\varepsilon-\text{alg}]}{\text{CPU time [two-phase method]}}100$.
Δz_2	Reports the average difference in the second coordinate of the lexicographic minima. Note that $\Delta z_2 + 1$ gives an upper bound on the number of non–dominated outcomes.
$\lvert\mathcal{Z}_N\rvert$	Reports the average number of non–dominated solutions for the instances of the corresponding row.
N	Contains the average number of dominated solutions generated by the algorithm. That is, the number of *unnecessary* iterations of the ε–constrained algorithm.
$\lvert\mathcal{Z}_N\rvert^{P2}/\lvert\mathcal{Z}_N\rvert^{P1}$	Reports the average ratio between the points found in phase two of the two–phase method and those found in phase one.

All algorithms have been coded in C++11 and compiled using the GNU GCC compiler with optimization option O3. All the experiments were carried out on a Fujitsu Esprimo Q920 desktop with 16GB RAM and a 2.20 GHz Intel Core i7-4785T processor running a 64 bit version of Linux Ubuntu. The lexicographic branch–and–cut algorithm described in Sections 4.4.2 was coded using the branch– and incumbent callbacks of the C++ API of CPLEX concert technology. The integer feasible solutions are kept in an external data-structure and CPLEX is then told to reject all solutions such that it does not terminate when a zero gap is obtained. The ParallelMode switch in CPLEX is set to *deterministic*. The absolute and the relative optimality gaps are set to 0.0. All other parameters are at their default values. The code, instances, and detailed results for each instance are all publicly available (Gadegaard, Klose, and Nielsen, 2016b).

To ease the reading, we summarize our test statistics in Table 4.1. For all tests except the ones carried out with the lexicographic branch–and–bound algorithm, we used a time limit of one hour for the generation of the non–dominated frontier.

4.5.1 Test classes

We report the results of tests conducted on three different classes of facility location problems known from the literature. The problem classes are the uncapacitated facility location

problem, and the capacitated facility location problem with and without single-source constraints. In the following, we give a short description of the three problem classes.

The capacitated facility location problem

The capacitated facility location problem (CFLP) is a widely studied combinatorial optimization problem. It consists of opening a set of facilities and assigning demand points to these facilities in such a way that all capacities are respected and the cost of assigning demand points and opening facilities is minimized. The CFLP assumes that

1. Each demand point has a fixed and known demand, $d_j > 0$.
2. Each facility has a fixed and known capacity, $s_i > 0$, which must be respected.
3. Each customer does not need to be assigned to a single facility; its demand can be split between several open facilities.
4. The cost of assigning a customer j to facility i depends linearly on the fraction of the demand d_j transported on the link (i, j).

This leads to the set of possible assignments, $\mathcal{X}_i$, being defined as

$$\mathcal{X}_i = \{(\mathbf{x}_i, y_i) \in [0,1]^{|\mathcal{J}|} \times \{0,1\} \ : \ \sum_{j \in \mathcal{J}} d_j x_{ij} \leq s_i y_i, \ x_{ij} - y_i \leq 0, \forall i \in \mathcal{I}, j \in \mathcal{J}\}$$

The set $\mathcal{Y}$ is equal to $\{y \in \{0,1\}^{|\mathcal{J}|} \ : \ \sum_{i \in \mathcal{I}} s_i y_i \geq \sum_{j \in \mathcal{J}} d_j\}$. The constraints $x_{ij} - y_i \leq 0$ and $\sum_{i \in \mathcal{I}} s_i y_i \geq \sum_{j \in \mathcal{J}} d_j$ are implied by the other constraints, but they often strengthen the LP relaxation. We denote the BO–CBLP arising from the CFLP the *capacitated* bi–objective cost–bottleneck location problem (capacitated BO–CBLP). The instances for the capacitated BO–CBLP are a subset of the test bed created for the paper Klose and Görtz (2007) consisting of 45 instances ranging from 100 customers and 100 facility sites to 200 customers and 200 facility sites.

The uncapacitated facility location problem

The uncapacitated facility location problem (UFLP) is very similar to the CFLP. As the name suggests it is a version of the CFLP where all facilities have sufficient capacity to potentially serve all demand points. Therefore, no additional restrictions on the assignments are needed and the UFLP can be defined by the sets $\mathcal{X}_i$ given by

$$\mathcal{X}_i = \mathbb{R}^{|\mathcal{J}|+1}.$$

As there is no restriction on the number of facilities to be open in a feasible solution either, we have $\mathcal{Y} = \{0,1\}^{|\mathcal{I}|}$. We denote the BO–CBLP arising from the UFLP the *uncapacitated*

bi–objective cost–bottleneck location problem (uncapacitated BO–CBLP). For the uncapacitated BO–CBLP we use a slightly larger subset of the instances created by Klose and Görtz (2007) consisting of 60 instances ranging in size from 100 customers and 100 facilities to 100 customers and 500 facility sites.

The single–source capacitated facility location problem

The single–source capacitated facility location problem (SSCFLP) is also a variant of the CFLP where each demand point has to be assigned to exactly one open facility. The SSCFLP can be described in terms of the general facility location problem (4.1) by setting

$$\mathcal{X}_i = \{(\mathbf{x}_i, y_i) \in \{0,1\}^{|\mathcal{J}|+1} \ : \ \sum_{j \in \mathcal{J}} d_j x_{ij} \leq s_i y_i, \ x_{ij} - y_i \leq 0, \forall i \in \mathcal{I}, j \in \mathcal{J}\},$$

and the extra requirement on the y-variables is given by

$$\mathcal{Y} = \{y \in \{0,1\}^{|\mathcal{I}|} \ : \ \sum_{i \in \mathcal{I}} s_i y_i \geq \sum_{j \in \mathcal{J}} d_j\}.$$

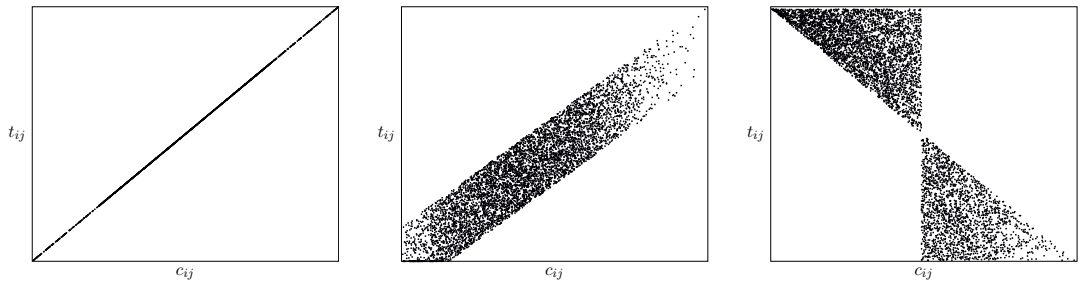
Note that the constraint defining the set $\mathcal{Y}$ and the constraints $x_{ij} - y_i \leq 0$ are redundant as was the case for the capacitated BO–CBLP. We denote the resulting bi–objective location problem the *single–source capacitated* bi–objective center location problem (single–source capacitated BO–CBLP).

In order to test the algorithms, we include the test bed proposed in Holmberg, Rönnqvist, and Yuan (1999) that consists of 71 instances as well as 57 instances from the testbed used in Díaz and Fernández (2002).

4.5.2 Cost structures

We use the cost matrix and cost vector provided by the instances for the travel costs and the fixed opening costs, respectively. Regarding the travel times we generate three new instances for each instance with travel costs defined as follows:

1. $t_{ij} = c_{ij}$. This suggests that travel time is equivalent to travel distance/cost. A plot of this cost structure, here referred to as C1, is provided in Figure 4.2a.
2. $t_{ij} = \max\{0, c_{ij} + U(-d, d)\}$. Here $U(k_1, k_2)$ denotes a discrete uniform distribution on the interval $[k_1, k_2]$. This implies that travel times are positively correlated with the travel cost, but that there is some *noise* which increases (decreases) the travel time for some distances. This is, for example, the case in geographically challenging countries like Denmark. A plot of cost structure C2 is given in Figure 4.2b.
3. Finally, we generate the t_{ij}'s such that $\text{corr}(t_{ij}, c_{ij}) < 0$, suggesting that large values of c_{ij} lead to small values of t_{ij} and vice versa. Such instances occur in e.g. supplier



(a) Cost structure C1. (b) Cost structure C2. (c) Cost structure C3.

Figure 4.2: A plot of the three different cost structures for the assignment of customers to facilities.

selection problems (see Current and Weber (1994) for more on location problems used in supplier selection problems). One can think of c_{ij} as the cost of procuring the required amount per period of product j from supplier i, where f_i is a fixed cost of placing an order, whilst t_{ij} is the delivery time. This model then minimizes total cost and the time until the last order arrives. We have generated the travel times in the following way: let $C_{\max}$ and $C_{\min}$ be the largest and the smallest assignment costs, respectively. Then

$$c_{ij} < \frac{C_{\max} - C_{\min}}{2} \Rightarrow t_{ij} = U(C_{\max} + C_{\min} - c_{ij}, C_{\max})$$
$$c_{ij} \geq \frac{C_{\max} - C_{\min}}{2} \Rightarrow t_{ij} = U(C_{\min}, C_{\max} + C_{\min} - c_{ij})$$

The cost structure C3 is illustrated in Figure 4.2c.

4.5.3 Performance of the lexicographic branch–and–bound approach

We carried out the experiments for the lexicographic branch–and–bound procedure only for the capacitated BO–CBLP as this problem exhibits characteristics of all three problem classes, and only the locational decisions, the y–variables, need to be integer. We found that only some of the smaller problems could be solved directly using the lexicographic branch–and–bound algorithm. As can be seen in Table 4.2 we succeeded in solving some problems with 100 facility sites and up to 200 customers. It is quite obvious that when the ratio between the total capacity and total demand becomes larger, the lexicographic branch–and–bound algorithm becomes more time consuming and less effective. This seems to be due to the larger number of feasible solutions to the problems, as the algorithm has a very hard time fathoming branching nodes after branching on integer feasible nodes. When we increase the size of the instances, the time consumption increases drastically. We were not

Table 4.2: Results obtained using the lexicographic branch–and–bound algorithm.

		Time		
#	Size	C1[2]	C2	C3
R[1]= 3				
5	100x100	1101.34	1865.48	—
5	100x200	1945.25	—	—
R[1]= 5				
5	100x100	5242.64	—	5817.79
R[1]= 10				
5	100x100	7274.67	—	6137.56

[1] $R = \sum_{i \in I} s_i / \sum_{j \in J} d_j$.
[2] Ci = cost structure $i = 1, 2, 3$.
— Indicates that none of the instances of that particular (row,column)–combination could be solved within one hour of computation time.

able to solve larger instances as the branching tree became too large to fit in memory. The implemented lexicographic branch–and–bound algorithm thus seems unsuitable as a solution procedure for these problem types. Even though no weakly efficient solutions are generated, enumerating all optimal solutions of the subproblems of the ε–constrained algorithm becomes prohibitive even for smaller problems. Therefore, we did not perform further tests with the lexicographic branch–and–bound algorithm, and the following results are carried out using CPLEX as a single objective solver for the subproblems.

4.5.4 Performance of the ε–constrained algorithm

In this section we report the results obtained with the ε–constrained algorithm using CPLEX as a single objective solver for the subproblems arising in the ε–algorithm described in Algorithm 4.2.

Results for the capacitated BO–CBLP

We report the results obtained when applying the ε-algorithm to the 45 instances in the capacitated BO-CBLP class that could be solved within one hour of computation time. Table 4.3 shows that problems of up to 200 facilities and 200 customers could be solved. Furthermore, the number of non-dominated solutions does not vary much within each cost structure, but substantial variations are seen between cost structures: when considering the cost structure C2 many more non–dominated points are generated compared to C1 and C3. Intuitively, one would expect cost structure C3 to produce more non–dominated

Table 4.3: Summary of the results obtained for the capacitated BO–CBLP.

		Time			Δz_2			$\lvert\mathcal{Z}_N\rvert$			N		
#	Size	C1[‡]	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
$R^\dagger = 3$													
5	100×100	17.91	81.24	45.35	34.80	117.20	32.60	8.20	43.00	33.60	0.00	0.00	0.00
5	100×200	68.92	243.62	130.02	29.40	74.40	31.00	13.40	44.80	32.00	0.00	0.00	0.00
5	200×200	291.73	775.31	285.59	37.00	91.60	34.80	18.80	53.40	35.80	0.00	0.00	0.00
$R^\dagger = 5$													
5	100×100	52.24	150.11	85.05	53.00	90.40	36.40	20.80	50.60	35.40	0.00	0.00	0.00
5	100×200	122.42	569.50	353.63	39.80	117.00	33.80	19.40	64.80	34.80	0.00	0.00	0.00
5	200×200	437.54	1163.85	425.62	39.60	82.60	32.40	21.20	60.80	33.40	0.00	0.00	0.00
$R^\dagger = 10$													
5	100×100	67.89	166.90	213.00	53.00	82.60	37.20	23.80	38.20	38.00	0.00	0.00	0.00
5	100×200	231.57	893.59^4	2521.07^3	67.60	82.60	32.20	34.00	51.80	33.20	0.00	0.00	0.00
5	200×200	384.43	1070.58	584.13	55.80	111.20	37.60	22.00	64.20	36.00	0.00	0.80	0.00

[†] $R = \sum_{i \in \mathcal{I}} s_i / \sum_{j \in \mathcal{J}} d_j$. [‡] C$i$ = cost structure $i = 1, 2, 3$.
Superscripts: Indicates the number of instances of that particular (row,column)–combination that could be solved within one hour of computation time. If no superscript, all instances were solved.

points as the coefficients are negatively correlated. However, when we divide the range of the coefficients t_{ij} into three equal segments, it turns out that for the cost structure C3, about two thirds of the entries in the travel time matrix lie in the largest third. This means that many variables are fixed in the first couple of iterations. Thus, producing solutions improving the center/bottleneck objective quickly becomes infeasible, implying smaller values of $|\mathcal{Z}_N|$ for this cost structure. By comparison, for cost structures C1 and C2, about 60 percent of the travel time coefficients lie in the middle third. This explains why the cost structure C3 produces less efficient solutions than expected.

Another interesting point in relation to cost structure C3 is that in many cases the upper bound on the number of non–dominated outcomes given by $\Delta z_2 + 1$ is strict. That is, there is almost always a non–dominated solution for each value between $z_2^{\min} = \min\{z_2 : z \in \mathcal{Z}_N\}$ and $z_2^{\max} = \max\{z_2 : z \in \mathcal{Z}_N\}$ of the bottleneck objective. This seems to be due to the antagonistic relationship between the objectives when the coefficients c_{ij} and t_{ij} are negatively correlated: The cost objective tries to pick the assignments with small values of c_{ij} resulting in long travel times for that assignment.

Note that strikingly few weakly efficient solutions are generated for all of the three cost structures. The low numbers of weakly efficient solutions confirm the suitability of the ε–constraint approach for these cost–bottleneck location problems.

Results for the uncapacitated BO–CBLP

We were able to solve slightly larger problems of the uncapacitated BO–CBLP class compared to the capacitated version. This means that instances ranging from 100 facilities and 100 customers to instances with 100 facilities and 500 customers were solved to optimality. As

Table 4.4: Summary of the results obtained for the uncapacitated BO–CBLP.

		Time			Δz_2			$\lvert\mathcal{Z}_N\rvert$			N		
#	Size	C1[1]	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
15	100×100	19.73	55.96	30.18	84.53	137.87	34.73	27.00	57.33	35.20	0.07	0.40	0.00
15	100×200	74.71	227.93	145.45	77.33	122.20	32.33	27.67	62.87	33.33	0.00	0.33	0.00
15	200×200	459.46	1190.83	791.66	67.80	117.93	34.07	39.27	82.00	35.07	0.00	0.53	0.00
15	500×500	402.88	1464.44	1043.93	25.47	76.73	29.33	23.13	69.73	30.33	0.00	0.40	0.00

[1] Ci = cost structure $i = 1, 2, 3$

there is no capacity limit on the facilities in this problem type, the instances are grouped by size only.

In Table 4.4, we see the same pattern as for the capacitated BO–CBLP, namely that the cost structure C2 yields larger values of $|\mathcal{Z}_N|$. And again, the number of non–dominated outcomes in cost structure C3 reaches the upper bound $\Delta z_2 + 1$ in most of the instances. Furthermore, a slightly higher number of non–dominated solutions is generated compared to the capacitated BO–CBLP. This phenomenon was expected as there is no capacity constraints to conflict with the fixation of assignment variables.

The reader should again note the remarkably small number of weakly efficient solutions which underpins the appropriateness of the ε–constraint approach.

Results for the single–source capacitated BO–CBLP

The 71 problems from Holmberg et al. (1999) are divided into five subsets and the 57 instances of Díaz and Fernández (2002) are divided into 7 subsets based on the dimensions of the problems. Table 4.5 summarizes the results obtained for the single–source capacitated BO–CBLP.

As opposed to the capacitated and the uncapacitated cost–bottleneck location problems, the size of $\mathcal{Z}_N$ is significantly larger for the instances with negatively correlated travel costs and travel times compared to the two other cost structures. The number of weakly efficient solutions described in column N is again remarkably small, which suggests that the problem instances for the single–source case also only have a few alternative optimal solutions and that, therefore, the ε–constraint method is a well–suited approach.

It is interesting to note, that for the Holmberg et al. instances of size 30×150, only one efficient solution is found for all of the 15 instances with cost structure C1. This happens, because the c_{ij}'s are relatively large compared to the fixed opening costs, f_i, leading to more facilities being open in an optimal solution to the minisum problem. As the travel time coefficients $t_{ij} = c_{ij}$ in cost structure C1, the cost objective and the bottleneck objectives are not in conflict at all, and hence, the optimal solution to the cost objective, turns out to be optimal for the bottleneck objective as well.

Table 4.5: Summary of the results obtained for the single–source capacitated BO–CBLP.

		Time			Δz_2			$\lvert\mathcal{Z}_N\rvert$			N		
#	Size	C1[2]	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
Intances from (Holmberg et al., 1999)													
12	10×50	0.13	0.42	8.75	60.42	161.83	574.25	3.75	9.92	137.33	0.00	0.08	0.08
12	20×50	0.45	1.08	44.21	93.17	142.00	571.25	12.08	16.00	172.33	0.00	0.00	0.33
16	10×90[1]	0.93	3.50	46.22	53.20	81.27	143.93	10.27	15.40	93.53	0.00	0.07	0.47
15	30×150	0.80	10.42	16.87	0.00	162.06	42.38	1.00	25.19	43.38	0.00	0.44	0.00
16	30×200	13.10	41.22	957.86^{15}	88.44	176.00	335.69	23.25	53.00	313.80	0.19	0.13	0.27
Instances from Díaz and Fernández (2002)													
6	10×20	0.48	0.72	4.01	31.83	43.00	83.33	4.33	7.67	33.67	0.00	0.00	0.17
11	15×30	2.19	3.02	16.67	32.27	39.36	88.82	10.45	13.45	45.73	0.00	0.09	0.18
8	20×40	4.93	7.12	51.78	25.38	34.25	93.29	12.00	15.00	59.25	0.00	0.00	0.00
8	20×50	372.47	353.99	45.52^{6}	29.50	40.00	92.33	12.50	18.25	64.33	0.13	0.13	0.67
8	30×60	27.41	49.62	153.80	33.25	39.88	94.63	17.67	23.78	74.11	0.22	0.44	0.33
8	30×75	84.55^{7}	225.38^{7}	1407.9^{7}	28.00	44.50	94.60	16.00	28.14	78.43	0.14	0.00	0.57
8	30×90	492.08	706.92^{6}	220.43^{6}	25.38	30.50	95.33	15.00	17.00	81.17	0.13	0.17	0.17

[1] These instances ranges from 10×90 to 30×70. [2] Ci = cost structure $i = 1, 2, 3$.
Superscripts: Indicates the number of instances of that particular (row,column)–combination that could be solved within one hour of computation time. If no superscript, all instances were solved.

4.5.5 Comparison with the two–phase method

In order to validate the effectiveness of the ε–constrained method proposed in this paper, we have implemented a two–phase method for solving the BO–CBLP as well. We have chosen to implement this solution methodology as it is probably the most widely used solution method for bi–objective combinatorial optimization next to the ε–constrained method (for a thorough treatment of two–phase methods the reader is referred to Przybylski, Gandibleuz, and Ehrgott (2011) and references therein).

In the first phase of the two–phase method, the set $\mathcal{Z}_{sN}$ is generated by solving weighted sum scalarized versions of the BO–CBLP for different weight vectors. Phase two consists of a method capable of generating the remaining non–dominated outcomes, $\mathcal{Z}_{nN}$. We have implemented the so–called "Perpendicular Search Method" (PSM) suggested by Chalmet, Lemonidis, and Elzinga (1986) for generating the non–extreme supported non–dominated outcomes. Contrary to the ε–constrained method, the PSM algorithm computes no weakly non–dominated solutions. However, similar to the ε–constrained method the implementation is straightforward.

To meet the potential critique that the two–phase method is badly implemented, we here mention that between 99.6% and 100.0% of the running time was spent by CPLEX solving the subproblems. We also note that a first implementation where the second phase consisted of ranking solutions using no–good inequalities performed very poorly. Thus, we chose to implement the PSM method instead.

Using the two–phase method and the PSM method requires the non–linear min–max

objective to be linearized. For the SSCFLP and the UFLP this is easily done by replacing the objective with a new continuous variable ρ and including the set of constraints:

$$\rho \geq \sum_{i \in \mathcal{I}} t_{ij} x_{ij}, \quad \forall j \in \mathcal{J}.$$

For the CFLP, where the x_{ij}–variables are continuous, we need to introduce a set of new binary variables ξ_{ij} equaling one if and only if $x_{ij} > 0$. The following constraints are then added to the formulation of the CFLP:

$$\begin{aligned} \rho &\geq t_{ij}\xi_{ij}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}, \\ \xi_{ij} &\geq x_{ij}, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}, \\ \xi_{ij} &\in \{0,1\}, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}. \end{aligned}$$

The results obtained with the two–phase method has been aggregated in Table 4.6. If the algorithm failed to solve all instances corresponding to a row in Table 4.6 within one hour of computation time, the number of actually solved instances is indicated in the "%–time" columns using superscript. It is obvious that the overall performance of the two-phase method is very poor compared to the ε–constrained method. In fact, the ε–constrained method is 2 to 160 times faster than the two–phase method on average. For the uncapacitated as well as for the capacitated BO–CBLP, the relative performance across the cost structures seems stable. However, for the single–source capacitated BO–CBLP, the ε–constraint algorithm becomes better relative to the two–phase method when the coefficients c_{ij} and t_{ij} become negatively correlated. The explanation for this behavior is easily found in the columns entitled "$|\mathcal{Z}_N|^{P2}/|\mathcal{Z}_N|^{P1}$". These columns display the ratio between the number of solutions found in phase one and phase two, respectively. This ratio increases significantly for most of the instances in cost structure C3 indicating that only few extreme supported non–dominated outcomes exists for these problems. As this ratio grows, the two–phase method loses its power as the first phase cannot divide the search space into sufficiently small regions for the second phase to be effective.

Furthermore, it was only possible to solve the small instances of the capacitated BO–CBLP and the uncapacitated BO–CBLP. In particular, we cannot solve many of the capacitated BO–CBLP within one hour of computation time. This is mainly due to the linearization of the objective function which requires the introduction of $|\mathcal{I}| \times |\mathcal{J}|$ new binary variables as well as $|\mathcal{I}| \times |\mathcal{J}|$ new constraints. In the linearization of the min–max objective in the uncapacitated and the single–source capacitated BO–CBLP, only one additional continuous variable and $|\mathcal{J}|$ new constraints are needed. Therefore these problems scale slightly better. It should, however, be very clear, that the two–phase method is also less suited for these problems than the ε–constraint method.

Table 4.6: Comparison of the results obtained with the two phase method and the ε–constrained method.

#	Size	% time			$\lvert\mathcal{Z}_N\rvert^{P2}/\lvert\mathcal{Z}_N\rvert^{P1}$		
		C1[3]	C2	C3	C1	C2	C3
Capacitated BO–CBLP							
$R^2 = 3$							
5	100×100	0.62[3]	0.47[1]	—	1.16	1.83	—
$R^2 = 5$							
5	100×100	—	—	—	—	—	—
$R^2 = 10$							
5	100×100	1.84[1]	3.83[1]	—	1.22	1.20	—
Uncapacitated BO–CBLP							
15	100×100	3.55	3.56	3.60	2.15	1.96	0.95
15	100×200	3.33	4.18	2.76	1.61	2.03	0.43
Single–source capacitated BO–CBLP							
Intances from (Holmberg et al., 1999)							
12	10×50	16.58	18.59	4.75	0.32	1.11	49.72
12	20×50	11.33	17.07	2.38	0.83	1.67	70.13
16	10×90[1]	17.58	12.80	2.07	0.96	1.60	40.15
15	30×150	38.07	2.91	1.03	0.00	2.46	8.16
16	30×200	1.19	1.12	—	2.27	4.25	—
Intances from (Díaz and Fernández, 2002)							
6	10×20	8.51	8.47	10.46	0.25	0.62	5.63
11	15×30	14.63	15.85	4.72	0.96	1.53	7.61
8	20×40	0.72	0.84	3.99[5]	0.97	1.03	6.79
8	20×50	12.01[6]	15.82[5]	8.97[3]	1.20	1.71	6.98
8	30×60	55.96[5]	22.55[4]	13.12[3]	1.05	1.18	4.90
8	30×75	—	—	—	—	—	—
8	30×90	—	—	—	—	—	—

[1] These instances ranges from 10×90 to 30×70. [2] $R = \sum_{i \in \mathcal{I}} s_i / \sum_{j \in \mathcal{J}} d_j$. [3] C$i$ = cost structure $i = 1, 2, 3$. Superscripts: Indicates the number of instances of that particular (row,column)–combination that could be solved within one hour of computation time. If no superscript, all instances were solved. — Indicates that none of the instances of that particular (row,column)–combination could be solved using the two phase method within one hour of computation time.

4.5.6 Utilizing a customized solver for the single–source capacitated BO–CBLP

As a quite efficient solver for the single–source capacitated facility location problem was available to the authors, we wanted to test the capabilities of the method. The specialized solver is based on a cut–and–solve framework with upper bounds generated using a local branching heuristic. The lower bound of the problem is strengthened by exact knapsack

Table 4.7: Summary of the results obtained for the single–source capacitated BO–CBLP using a specialized solver.

		Time			$\lvert\mathcal{Z}_N\rvert$			N		
#	Size	C1[2]	C2	C3	C1	C2	C3	C1	C2	C3
12	10×50	0.26	0.74	11.20	3.75	9.50	104.00	0.00	0.00	0.00
12	20×50	1.44	3.79	92.81	1.44	3.79	92.81	0.00	0.08	0.17
16	10×90[1]	0.73	9.95	12.30	1.00	21.44	22.06	0.00	0.38	0.00
15	30×150	6.45	12.11	36.58	8.40	12.07	53.60	0.00	0.07	0.00
16	30×200	33.02	93.63	1086.10	17.44	39.19	161.88	0.00	0.13	0.06
6	10×20	0.81	1.10	4.12	4.00	6.83	22.50	0.00	0.00	0.00
11	15×30	3.36	4.69	17.75	7.82	9.55	28.45	0.09	0.18	0.18
8	20×40	5.78	6.89	34.33	8.25	11.00	34.75	0.00	0.00	0.13
8	20×50	9.77	14.20	73.01	8.50	11.88	37.25	0.00	0.00	0.13
8	30×60	19.78	202.18	48.49	11.25	13.75	40.88	0.13	0.00	0.25
8	30×75	17.51	23.71	159.28	10.00	15.75	42.50	0.00	0.00	0.00
8	30×90	26.80	33.70	408.78	9.38	12.50	44.25	0.00	0.00	0.13

[1] These instances ranges from 10×90 to 30×70. [2] Ci = cost structure $i = 1, 2, 3$

separation (the code is available on request). In Table 4.7, we report the results obtained using this specialized solver as a black box engine for solving the subproblems arising in each iteration of the ε–constraint algorithm.

With this solver as black box engine, we were easily able to solve the largest instances in both of the testbeds and we saved a significant amount of time on the largest instances. The smaller instances were often solved in less time using CPLEX (see Table 4.5), however. This basically boils down to our implementation: The implementation using CPLEX fixes the x_{ij}–variables and *resolves* the model. This allows CPLEX to utilize basis, incumbent, and branching information from previous problems. When we use the specialized solver, the problem is solved *from scratch* every time, implying that no information from previously solved problems is used.

4.6 Conclusion

In this paper we investigated the very general bi-objective center location problem. We proved that even though the problem is $\mathcal{NP}$–hard, it is in fact tractable, in the sense that the size of the efficient frontier of the bi–objective problem is always limited by a polynomial in the input size.

We proposed a scheme for solving the BO-CBLP that relies on an ε–constrained method. We suggested two ways to accommodate the issues of generating weakly efficient solutions in the ε–constraint: First, a simple change of the cost matrix which is both a necessary and

sufficient condition in order to impose the non–linear ε–constraint on the center objective. This implies, the structure of the underlying location problem is kept, and specialized solvers can be used as a subroutine reducing the computation time. Secondly, we outlined a scheme based on lexicographic branch–and–bound leading to no weakly efficient solutions being generated. Furthermore, the number of iterations performed by the algorithm is bounded by a polynomial in the input size.

We also linked the BO–CBLP to the p–centdian location problem studied in the literature. We found that the bi–objective p–centdian problem *on a graph* might have an uncountably infinite number of Pareto optimal solutions and noted that given a computationally efficient way of solving the BO–CBLP, the vertex–p–centdian problem can be solved efficiently for all values of the scaling parameter λ. As many very efficient algorithms for the p-median problem exist, and the p–centdian on a graph can be reduced to a discrete problem, our algorithm may constitute an efficient way of solving the p–centdian problem.

Through extensive computational tests we have shown that the proposed method is capable of efficiently and exactly solving even large BO–CBLPs. In addition, the tests showed that remarkably few weakly non–dominated solutions exist for these very large combinatorial optimization problems. This leads to the lexicographic branch–and–bound based algorithm being very inferior compared to the ε–constrained algorithm. Furthermore, the proposed algorithm outperforms a two–phase method by several orders of magnitude as well.

Directions for further research include the testing of the approach for the p–centdian problem. That is, to investigate if it is in fact possible to solve large problem instances of the p–centdian problem using an algorithm tailored for the p–median problem. It would also be interesting to apply the methodology on other types of more complex facility location models, such as multi–stage and dynamic models.

4.7 Acknowledgments

The authors would like to thank Professor Kim Allan Andersen for insightful comments and suggestions. This work was supported by a grant from Købmand Ferdinand Sallings Mindefond.

Bound set based branch–and–cut algorithms for bi–objective combinatorial optimization

This chapter is based on the paper Gadegaard, Ehrgott, and Nielsen (2016). The research was conducted from the autumn of 2014 to the spring of 2016 and submitted to optimization-online.org in April 2016

The work was presented at the MCDM 2015, Hamburgh, Germany, and at the INFORMS annual meeting 2015, Philadelphia, Pennsylvania, USA

Bound set based branch–and–cut algorithms for bi–objective combinatorial optimization

Sune Lauth Gadegaard[†], Matthias Ehrgott[*], and Lars Relund Nielsen[†]

[†]Department of Economics and Business Economics, Aarhus University, Denmark,
{sgadegaard, larsrn}@econ.au.dk
[*]Department of Mathematics, Aarhus University, Denmark, m.ehrgott@lancaster.ac.uk

Abstract

Most real–world optimization problems are of a multi–objective nature, involving objectives which are conflicting and incomparable. Solving a multi–objective optimization problem requires a method which can generate the set of rational compromises between the objectives. In this paper, we propose two distinct bound set based branch–and–cut algorithms for bi–objective combinatorial optimization problems, based on implicitly and explicitly stated lower bound sets, respectively. The algorithm based on explicitly given lower bound sets computes for each branching node a lower bound set and compares it to an upper bound set. The implicit bound set based algorithm, on the other hand, fathoms branching nodes by generating a single point on the lower bound set for each local Nadir point. We outline several approaches for fathoming branching nodes and we propose an updating scheme for the lower bound sets that prevents us from solving the bi–objective LP–relaxation of each branching node. To strengthen the lower bound sets, we propose a bi–objective cutting plane algorithm that dynamically adjusts the weights of the objective functions such that different parts of the feasible set are strengthened by cutting planes. In addition, we suggest an extension of the branching strategy "Pareto branching". Extensive computational results obtained for the bi–objective single–source capacitated facility location problem prove the effectiveness of the algorithms.

Keywords: bi–objective branch–and–cut; bi–objective optimization; combinatorial optimization; branch–and–cut.

5.1 Introduction

A general definition of a multi–objective decision problem with k objectives can be given as follows: Let $\mathcal{X}$ be a set of decisions available to a decision maker, and let $x \in \mathcal{X}$ be a specific decision. Now assume that the decision x can be quantified by real valued functions $f_i : \mathcal{X} \to \mathbb{R}$, $i = 1, \ldots, k$. The multi–objective optimization problem then consists in finding all decisions constituting *rational compromises* (Pareto–optimal solutions), with respect to the objective functions f_i, $i = 1, \ldots, k$. When the decisions $\mathcal{X}$ are implicitly given by a set of constraints, the problem belongs to the set of multi–objective programming problems, and if the objective functions f_i and the constraints are all linear, the problem is denoted a multi–objective linear programming problem. In this paper, we will consider the problem where all variables are restricted to be either zero or one and for which only two linear objective functions are present. Such a problem is usually named a *bi–objective combinatorial optimization* (BOCO) problem.

In the following paragraphs we provide an overview of some of the most important contributions to the development of branch–and–bound algorithms for general BOCO problems. A schematic overview is also given in Table 5.1. In the column headed "x" we have illustrated the domain of the variables where we use the convention that $\mathbb{B} = \{0, 1\}$. Furthermore, when the problem is a *mixed integer* BOCO, we write MI$\mathcal{S}$ where ($\mathcal{S}$) is the discrete domain for the integer restricted variables. The "f" column shows the domain of the objective functions, that is, "$(\mathbb{R}, \mathbb{Z})$" means that the first objective maps to the real numbers and the second is restricted to map into the integers. Furthermore, if the corresponding paper treats problems with more than two objectives, we simply write "$(\mathbb{R}, \ldots, \mathbb{R})$". Surprisingly little research has been devoted to branch–and–bound algorithms for general BOCO problems although many problems can be fitted into this framework, for example the bi-objective knapsack problem (Ulungu and Teghem, 1997), the bi–objective assignment problem (Przybylski, Gandibleux, and Ehrgott, 2008; Pedersen, Nielsen, and Andersen, 2008), bi–objective facility location problems (Fernández and Puerto, 2003), and the bi–objective TSP (Bérubé, Gendreau, and Potvin, 2009).

However, more effort has been put into the development of branch–and–bound algorithms for BOCO problems lately. Klein and Hannan (1982) propose what is probably the first branch–and–bound algorithm for BOCO problems. During the eighties and nineties, only very few researchers followed up on this idea, as examples we mention Kiziltan and Yucaoğlu (1983), Ulungu and Teghem (1997), Ramos et al. (1998) and Visée et al. (1998) (although the latter three are problem specific). Since the turn of the millennium more attention has been brought to this solution approach and even generalizations where both integer and continuous variables are allowed (mixed integer BOCO problems) were considered. Mavrotas and Diakoulaki (1998) are among those who develop a branch-and-bound algorithm for

Table 5.1: Overview of the bi–objective branch–and–bound algorithms proposed in the literature. All approaches has been converted to the minimization case for better comparison.

Reference	Node selection[1]	Lower bound[2]	Upper bound	Cuts	Branching[3]	x	f_i	Notes
Klein and Hannan (1982)	—	UP	Incumbent set	—	Variable fixing	$\mathbb{N}_0$	$(\mathbb{Z},\dots,\mathbb{Z})$	Propose to use post–optimality techniques to solve a series of integer programs.
Kiziltan and Yucaoğlu (1983)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{R},\dots,\mathbb{R})$	Probing on variables is applied in each node of the branching tree.
Ulungu and Teghem (1997)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{Z},\mathbb{Z})$	Adapt the methods presented in Martello and Toth (1990) to a multi–objective framework.
Ramos et al. (1998)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{R},\mathbb{R})$	Pure branch–and–bound. Feasible solutions are found as leaf nodes, no heuristics.
Visée et al. (1998)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{Z},\mathbb{Z})$	Use the two phase method and employ the method of Ulungu and Teghem (1997) in the resulting triangles.
Mavrotas and Diakoulaki (1998)	DF	UP	Incumbent set	—	Variable fixing	MI($\mathbb{B}$)	$(\mathbb{R},\dots,\mathbb{R})$	Allow continuous variables. Problematic, as some dominated solutions might be considered non–dominated.
Mavrotas and Diakoulaki (2005)	DF	UP	Incumbent set	—	Variable fixing	MI($\mathbb{B}$)	$(\mathbb{R},\dots,\mathbb{R})$	Computationally improve the solution of the LP–relaxation. Add a final dominance test. Still problematic.
Sourd and Spanjaard (2008)	DF	HS	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{R},\mathbb{R})$	Propose to use hypersurfaces as lower bound set.
Florios et al. (2010)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{R},\mathbb{R})$	Application of the algorithm proposed in Mavrotas and Diakoulaki (1998, 2005).
Vincent et al. (2013)	DF	UP	Incumbent set	—	Variable fixing	MI($\mathbb{B}$)	$(\mathbb{R},\mathbb{R})$	Correct and improve the algorithm of Mavrotas and Diakoulaki (1998). Characterize the non–dominated frontier of a mixed integer BOCO problem.
Stidsen et al. (2014)	—	S–LP	Incumbent set	—	Variable fixing, PB, SL	MI($\mathbb{B}$)	$(\mathbb{Z},\mathbb{R})$	Introduce Pareto branching and slicing of the outcome space. One objective must be integer valued.
This paper	BB	S–LP BO–LP	Incumbent set	Yes	Variable fixing, extended PB	$\mathbb{B}$	$\mathbb{Z}$	Introduces cutting plane algorithm. Proposes updating of lower bound sets. Extends PB.

[1] Node selection (DF: Depth first, BB: Best bound) [2] Lower bound (HS: Hyper surface, UP: Utopian or ideal point, S–LP: LP–relaxation of scalarized problem, BO–LP: Bi–Objective LP–relaxation) [3] Branching: (PB: Pareto branching, SL: Slicing)

BOCO problems. They develop a depth first branch–and–bound algorithm capable of finding all non–dominated outcome vectors of a mixed integer BOCO problem. Whenever a leaf node is reached (that is, when all integer variables have been fixed), the resulting bi–objective linear program is solved. The resulting outcome vectors are compared to solutions in the incumbent set; all outcomes in the set of yet non–dominated outcomes dominated by the outcome vector of a new solution are removed and only non–dominated outcomes are added. A node is fathomed in the branching tree if it is infeasible or if the *ideal point* of the node is dominated by a point in the set of yet non–dominated points. Later, Mavrotas and Diakoulaki (2005) published a number of improvements. The algorithm was adapted to binary combinatorial optimization problems (all variables are either zero or one) and was applied to multi–objective, multi–dimensional knapsack problems in Florios et al. (2010).

Unfortunately the algorithm proposed in the two papers by Mavrotas and Diakoulaki (1998, 2005) might return dominated solutions. The issue originates in the fact that not only the extreme points of the non–dominated frontier found at the leaf nodes might be non–dominated. Also the line segments joining these need to be considered. This issue is addressed in Vincent (2009) and corrected for the bi–objective case by Vincent et al. (2013). In the latter paper a number of lower bound sets are introduced and promising results are reported with up to 60 constraints and 60 variables, of which 30 are binary.

Sourd, Spanjaard, and Perny (2006) and Sourd and Spanjaard (2008) develop a branch–and–bound framework where the branching part is identical to a single objective branch–and–bound algorithm. However, the bounding part is performed via a set of points rather than the single ideal point. The current node can be discarded if a hypersurface separates the set of feasible solutions in the subproblem from the incumbent set (the set of non–dominated points). Sourd and Spanjaard use a rather sophisticated problem–specific hypersurface and obtain promising experimental results for the bi–objective spanning tree problem.

Very recently Stidsen et al. (2014) introduced the concept of *Pareto branching* where branching is performed in outcome space. Also, they proposed *slicing* the outcome space and thereby obtaining better upper bounds for fathoming nodes in the branching tree. Promising test results are provided for a range of bi–objective combinatorial optimization problems.

Although several novel and efficient approaches have been proposed in the past, none of these takes advantage of the lower bound set available from the bi–objective LP–relaxation. Furthermore, none of the previously mentioned algorithms incorporates cutting planes, even though effective separation routines have resulted in a significant speedup for single objective problems. Therefore, we propose two novel bound set based branch–and–cut algorithms for bi–objective linear combinatorial optimization problems. The algorithms rely on either explicitly or implicitly given lower bound sets obtained from the bi–objective LP–relaxation. To summarize, the main contributions of this paper are as follows:

1. We propose a bi–objective cutting plane algorithm which dynamically changes weights of the objectives in order to approximate the best possible lower bound set obtainable from the LP–relaxation.
2. We develop a simple updating scheme for explicit lower bound sets that reduces the number of bi–objective LPs that need to be solved.
3. We propose a simple method for implicitly describing the lower bound set obtained from the bi–objective LP–relaxation.
4. The Pareto branching strategy is strengthened to what we call extended Pareto branching.
5. In total 8 different branch–and–cut implementations are evaluated on the bi–objective single source–capacitated facility location problem, which, to the best of our knowledge, is solved for the first time in the literature.

The remainder of this paper is organized as follows: Section 5.2 gives the basic definitions of bi–objective optimization. Section 5.3 starts with a short theoretical description of a generic bi–objective branch–and–cut algorithm and afterwards we describe the main components of the branch–and–cut algorithm in detail. Finally, different implementations of the algorithm developed in the paper are tested in Section 5.4.

5.2 Preliminaries

The focus of this section will be on a generic linear *bi-objective combinatorial optimization* (BOCO) problem of the form

$$\min\{Cx \,:\, x \in \mathcal{X}\} \tag{5.1}$$

where $C = (c^1, c^2)$ is a $2 \times n$ dimensional matrix with all entries being integral and the *feasible set* is defined by $\mathcal{X} = \{x \in \{0,1\}^n \,:\, Ax \leqq b\}$. The set $\mathcal{X}$ of feasible solutions is also referred to as the feasible set in *decision space* and the image of $\mathcal{X}$ under the linear mapping C is called the feasible set in *objective space* and is here denoted $\mathcal{Z}$.

To compare vectors in $\mathbb{R}^2$ we adopt the notation from Ehrgott (2005). Let $z^1, z^2 \in \mathbb{R}^2$, then

$$\begin{aligned} z^1 \leqq z^2 &\Leftrightarrow z^1_k \leqq z^2_k, \text{ for } k = 1,2 \\ z^1 \leq z^2 &\Leftrightarrow z^1 \leqq z^2 \text{ and } z^1 \neq z^2 \\ z^1 < z^2 &\Leftrightarrow z^1_k < z^2_k, \text{ for } k = 1,2. \end{aligned}$$

We define the set $\mathbb{R}^2_{\leqq} = \{z \in \mathbb{R}^2 \,:\, z \geqq 0\}$ and analogously $\mathbb{R}^2_{\leq}$ and $\mathbb{R}^2_{<}$. Furthermore, given a set $S \subseteq \mathbb{R}^2$ let $S_N = \{s \in S \,:\, (\{s\} - \mathbb{R}^2_{\leqq}) \cap S = \{s\}\}$, where $(\{s\} - \mathbb{R}^2_{\leqq}) = \{z \in \mathbb{R}^2 \,:\, z = s - r,\ r \in \mathbb{R}^2_{\leqq}\}$. The set S_N is called the *non–dominated set of* S.

The problem (5.1) does not immediately reveal what an "optimal solution" should be. To clarify this we use the concept of Pareto optimality or efficiency:

Definition 5.1. A feasible solution $\hat{x} \in \mathcal{X}$ is called Pareto optimal or *efficient* if there does not exist any $x \in \mathcal{X}$ such that $Cx \leq C\hat{x}$. The image $C\hat{x}$ is then called *non-dominated*.

A feasible solution $\hat{x} \in \mathcal{X}$ is called weakly efficient if there does not exist any $x \in \mathcal{X}$ such that $Cx < C\hat{x}$.

Let $\mathcal{X}_E$ denote the set of all efficient solutions. Then the image of $\mathcal{X}_E$ under the linear mapping C is exactly $\mathcal{Z}_N$, that is, $\mathcal{Z}_N = C\mathcal{X}_E$. The set $\mathcal{Z}_N$ is referred to as the set of non–dominated outcomes. A subset $\mathcal{X}^* \subseteq \mathcal{X}_E$ where $C\mathcal{X}^* = \mathcal{Z}_N$ and $Cx \neq Cx'$ for all $x, x' \in \mathcal{X}^*$ will be considered an optimal solution to (5.1). Note that an optimal solution $\mathcal{X}^*$ is a *set* of efficient solutions.

The sets $\mathcal{X}_E$ and $\mathcal{Z}_N$ need to be further divided into two subsets. An efficient solution $\hat{x} \in \mathcal{X}_E$ is said to be a *supported* efficient solution if there exists a weight $\lambda \in (0, 1)$ such that $\lambda c^1\hat{x} + (1-\lambda)c^2\hat{x} \leqq \lambda c^1 x + (1-\lambda)c^2 x$ for all $x \in \mathcal{X}$. The set of supported efficient solutions is denoted $\mathcal{X}_{SE}$. The elements in $\mathcal{X}_{NE} = \mathcal{X}_E \setminus \mathcal{X}_{SE}$ are called *non–supported* efficient solutions. Analogously, the set $\mathcal{Z}_N$ is partitioned into two subsets, namely $\mathcal{Z}_{SN} = C\mathcal{X}_{SE}$ and $\mathcal{Z}_{nN} = C\mathcal{X}_{NE}$.

5.3 Bi–objective bound set based branch–and–cut

A branch–and–cut framework provides a very successful standard method for solving single objective combinatorial optimization problems (see e.g. Nemhauser and Wolsey (1988) or Martin (1999) for a detailed description). Here, the set of feasible solutions to the optimization problem is partitioned into disjoint subproblems which can be displayed in a tree–structure where each *node* represents a *subproblem*. We say that a branching node is *fathomed* if it has been proven that the subproblem corresponding to that branching node cannot contain solutions improving the current best solution or if the corresponding subproblem is infeasible. The algorithm keeps a set H of *active* nodes, that have not been fathomed. A specific active branching node is denoted η. Let $\mathcal{X}(\eta)$ denote the set of feasible solutions of the subproblem corresponding to branching node η. That is, the solutions in $\mathcal{X}$ satisfying all branching constraints added on the unique path from the root node to the branching node η. Furthermore, we let $\underline{\mathcal{X}}$ ($\underline{\mathcal{X}}(\eta)$) denote the set $\mathcal{X}$ ($\mathcal{X}(\eta)$) with all integrality constraints removed.

For single objective optimization problems only a single optimal solution value exists, say $z^* \in \mathbb{R}$. Thus, upper and lower bounds on z^* are given by numbers $u, l \in \mathbb{R}$ satisfying $l \leqq z^* \leqq u$. To adapt a branch–and–cut framework to BOCO problems we need to consider

bounds on the set $\mathcal{Z}_N$ of non–dominated solution values, hence we naturally need to extend the concept of bounds to *bound sets*. We use the definition of bound sets given in Ehrgott and Gandibleux (2007), stated for the bi–objective case below.

Definition 5.2 (Bound sets)**.** Lower and upper bound sets are defined as follows:

1. A *lower bound set* on $\mathcal{Z}_N$ is a subset $L \subseteq \mathbb{R}^2$ such that L is an $\mathbb{R}^2_{\leqq}$–closed and $\mathbb{R}^2_{\leqq}$–bounded set with $L = L_N$ such that

$$\mathcal{Z}_N \subseteq (L + \mathbb{R}^2_{\leqq}).$$

   Given two lower bound sets L_1 and L_2 we say that L_1 *dominates* L_2 if $L_1 \subseteq L_2 + \mathbb{R}^2_{\leqq}$. If furthermore $L_1 + \mathbb{R}^2_{\leqq} \neq L_2 + \mathbb{R}^2_{\leqq}$ we say that L_1 *strictly* dominates L_2.

2. An *upper bound set* on $\mathcal{Z}_N$ is a subset $U \subseteq \mathbb{R}^2$ such that U is an $\mathbb{R}^2_{\leqq}$–closed and $\mathbb{R}^2_{\leqq}$–bounded set with $U = U_N$ such that

$$\mathcal{Z}_N \subseteq \mathrm{cl}\left(\mathbb{R}^2 \setminus (U + \mathbb{R}^2_{\leqq})\right),$$

   where $\mathrm{cl}(S)$ denotes the closure of a set $S \subseteq \mathbb{R}^2$.

The lower bound set L is called $\mathbb{R}^2_{\leqq}$–*convex* if the set $(L + \mathbb{R}^2_{\leqq})$ is convex. In this paper we will focus on the $\mathbb{R}^2_{\leqq}$–convex lower bound set available from the non–dominated frontier of the LP-relaxation of the BOCO, that is

$$(C\underline{\mathcal{X}})_N = \{z \in \mathbb{R}^2 \,:\, z = Cx,\ Ax \leqq b,\ x \in [0,1]^n\}_N.$$

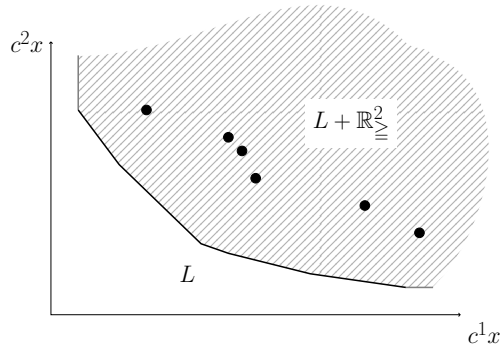
From Definition 5.2 it is also readily seen that any set of feasible solutions filtered by dominance gives rise to an upper bound set. A $\mathbb{R}^2_{\leqq}$–convex lower bound set and an upper bound set is illustrated in Figure 5.1a.

For single objective optimization problems an active node in the branching tree can be fathomed if the subproblem corresponding to the branching node is infeasible or if the lower bound of the subproblem is greater than or equal to the global upper bound. To extend this result to a multi–objective branch–and–cut algorithm, we need the following definition.

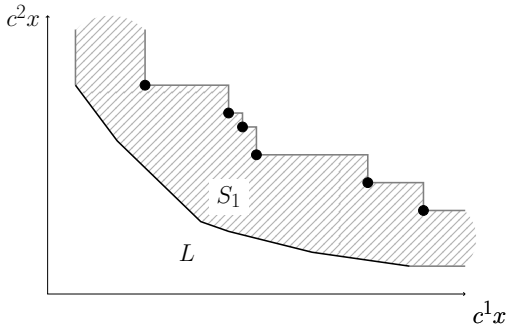
Definition 5.3 (Local Nadir point)**.** Let $U = \{z^1, \ldots, z^{|U|}\} \subseteq \mathcal{Z}$ be an upper bound set of feasible points ordered such that $z^u_1 < z^{u+1}_1$, for all $z^u, z^{u+1} \in U$. Then the set of *local Nadir points* is given by

$$\mathcal{N}(U) = \bigcup_{u=1}^{|U|-1} \{(z^{u+1}_1, z^u_2)\}.$$

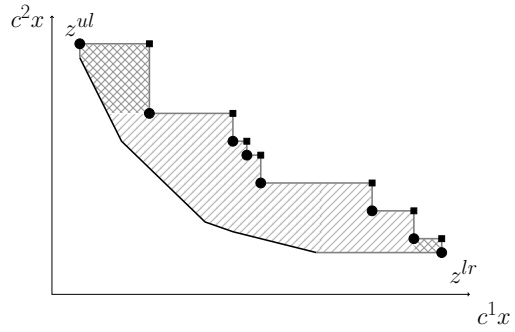
In a multi–objective branch–and–cut algorithm an active node η in the branching tree, can be fathomed if the subproblem corresponding to the branching node is infeasible or if



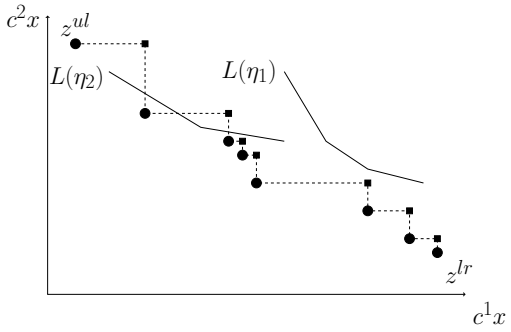
(a) Upper and lower bound sets. The hatched area is $L + \mathbb{R}^2_{\leqq}$.



(b) Search area S_1 defined in (5.2) (hatched area).



(c) Search area S_2 defined in (5.3) (hatched and cross-hatched area). Note that if the lexicographic minima are not part of the upper bound set, then the cross-hatched areas are missing.



(d) The node η_1 *can* be fathomed, since no local Nadir point is positioned "above" the lower bound set. The node η_2 can not be fathomed.

Figure 5.1: Illustrations of the search area. Lower bound sets are illustrated with solid black lines, upper bound sets with circles, and local Nadir points with squares.

every solution in the subproblem corresponding to η is dominated by at least one solution in the upper bound set U. That is, the *search area* between the lower and upper bound sets must be empty.

The search area may be defined in different ways. As noted by Przybylski, Gandibleux, and Ehrgott (2010), given a branching node η, its feasible points in objective space $\mathcal{Z}(\eta) \coloneqq C\mathcal{X}(\eta)$, a lower bound set $L(\eta)$ of $\mathcal{Z}(\eta)$, and an upper bound set U we have

$$(\mathcal{Z}(\eta) \cap \mathcal{Z}_N) \subseteq \left(L(\eta) + \mathbb{R}^2_{\leqq}\right) \setminus \left(U + \mathbb{R}^2_{>}\right) = S_1. \tag{5.2}$$

That is, the search area is defined as S_1 (see Figure 5.1b for an illustration). This representation is, however, not that useful in an algorithmic sense. By assuming that the lexicographic minima are part of the upper bound set U, we get the inclusion

$$(\mathcal{Z}(\eta) \cap \mathcal{Z}_N) \subseteq \left(L(\eta) + \mathbb{R}^2_{\leqq}\right) \cap \left(\mathcal{N}(U) - \mathbb{R}^2_{\leqq}\right) = S_2. \tag{5.3}$$

Step 1: Initialize the upper bound set U and set H equal to the root node.

Step 2: If $H = \emptyset$, then return $U = \mathcal{Z}_N$ and stop; otherwise select an active branching node $\eta \in H$.

Step 3: Add cutting planes.

Step 4: Obtain a lower bound set $L(\eta)$ of node η.

Step 5: Update the upper bound set U.

Step 6: If the node η can be fathomed, go to Step 1.

Step 7: Perform branching on η. Go to Step 1.

Algorithm 5.1: Generic multi–objective branch–and–cut algorithm based on bound sets.

The search area S_2 is illustrated in Figure 5.1c (hatched and cross-hatched area). Note that if the lexicographic minima are not part of the upper bound set, then (5.3) does not hold (the cross-hatched areas are missing). Given an upper bound set U including the lexicographic minima and a lower bound set $L(\eta)$ on $\mathcal{Z}(\eta)$, the search for non–dominated points can be restricted to the search area S_2 and a sufficient condition for fathoming a branching node η is

$$\left(L(\eta) + \mathbb{R}^2_{\leqq}\right) \cap \left(\mathcal{N}(U) - \mathbb{R}^2_{\leqq}\right) = \emptyset \tag{5.4}$$

Moreover, the result can be strengthened further.

Proposition 5.1. *Given an active branching node η, a lower bound set $L(\eta)$, and an upper bound set U including the lexicographic minima, the branching node η can be fathomed if*

$$\left(L(\eta) + \mathbb{R}^2_{\leqq}\right) \cap \mathcal{N}(U) = \emptyset. \tag{5.5}$$

Proof. We show that (5.5) implies (5.4). Assume that (5.5) holds true and that $\bar{z} \in \left(L(\eta) + \mathbb{R}^2_{\leqq}\right) \cap \left(\mathcal{N}(U) - \mathbb{R}^2_{\leqq}\right) \neq \emptyset$. Then $\bar{z} \in (\mathcal{N}(U) - \mathbb{R}^2_{\leqq}) \setminus \mathcal{N}(U)$ implying there exists a $\tilde{z} \in \mathcal{N}(U)$ such that $\tilde{z} \in \{\bar{z}\} + \mathbb{R}^2_{\leqq}$. Since $\bar{z} \in L(\eta) + \mathbb{R}^2_{\leqq}$ then so is $\tilde{z}$. This contradicts the starting assumption. □

An illustration of the proposition is given in Figure 5.1d. Here the branching node η_1 can be fathomed whereas node η_2 cannot.

Based on the above results a general bi–objective branch–and–cut framework can now be described as given in Algorithm 5.1. The algorithm is initialized by setting the upper bound set equal to the lexicographic minima of each criterion in Step 0 and an active node is chosen in Step 1. At each node, cuts can be added (Step 2) in order to strengthen the lower bound set obtained in Step 3. If a feasible solution can be obtained from the node it

might be a non–dominated solution, and the upper bound set is therefore updated in Step 4. After obtaining a lower bound set and updating the upper bound set, Step 5 is used to test whether the branching node can be fathomed. If not, the branching node is split into several new disjoint *child nodes* (Step 6). Note that Algorithm 5.1 implicitly enumerates all solutions to the BOCO problem 5.1 implying that when $H = \emptyset$, the set of non–dominated outcomes has been found. Furthermore, as the number of solutions to the BOCO problem is finite, Algorithm 5.1 terminates in finite time if the individual steps can be performed in finite time. In the following subsections we elaborate on Steps 2 through 6.

5.3.1 Step 2 - Adding cutting planes

A major challenge when designing algorithms for BOCO problems is to find a way to efficiently utilize the numerous methods and strategies available for the single objective versions of the BOCO problems. In this section, we propose a way to utilize cutting planes to reduce the "gap" between the lower bound set provided by the bi–objective LP–relaxation and the set of non–dominated outcomes to the BOCO problem (4.4).

In a single objective branch–and–cut algorithm, cutting planes are used before a branch–and–bound algorithm is started, as a way to improve the lower bound. Or seen from another perspective cuts are added to generate a tighter representation of the convex hull of integer solutions *in the direction of the objective function*. In case of a BOCO problem there is not a single direction of the objective function, and it is not obvious in which part of the polyhedron corresponding to the LP-relaxation, cutting planes would be most beneficial.

Therefore, we solve the LP–relaxation of a weighted sum scalarization of the BOCO problem and add cutting planes (in decision space) for different weights. The goal is to generate a relaxation of the BOCO problem which provides a lower bound set as close to $\text{conv}(\mathcal{Z}_N)_N$ as possible. Note, that the strongest $\mathbb{R}^2_{\leqq}$–convex lower bound set is

$$\text{conv}(\mathcal{Z}_N)_N = \{z \in \mathbb{R}^2 \,:\, z = Cx,\ x \in \text{conv}(\mathcal{X})\}_N.$$

From this description, we see that by approximating $\text{conv}(\mathcal{X})$ using cutting planes, we also approximate the strongest possible $\mathbb{R}^2_{\leqq}$–convex lower bound set without having to solve a series of integer programming problems.

An overview of the cutting plane algorithm is given in Algorithm 5.2. The algorithm starts by choosing a search direction (or weight) in Step 2.1. When the weight has been chosen, an ordinary cutting plane algorithm is used to separate cutting planes in the part of the decision space identified by the search direction λ (Step 2.2 and Step 2.3). In Step 2.4 a stopping criterion for the cutting plane algorithm is checked, allowing for multiple *rounds* of cuts. Finally, in Step 2.5, we check if a new search direction should be chosen or not. The algorithm described in Algorithm 5.2 distinguishes itself from a single objective cutting plane

Step 2.1: Choose weight $\lambda \in (0,1)$.

Step 2.2: Solve the weighted sum scalarization of the BOCO problem

$$\min\{(\lambda c^1 + (1-\lambda)c^2)x \ : \ x \in \underline{\mathcal{X}}\},$$

and let x^* be an optimal solution.

Step 2.3: If possible, separate x^* using cutting planes and append the cutting planes to the description of $\underline{\mathcal{X}}$.

Step 2.4: If we should add further cuts (e.g. if new cuts were added in Step 2.2), go to Step 2.1.

Step 2.5: If we should apply a new search direction λ, go to Step 2.1; otherwise stop.

Algorithm 5.2: Step 2 of Algorithm 5.1.

algorithm only by the outer loop where different search directions are used. This means that problem specific cutting planes can be used in Step 2.3, if effective separation algorithms are known.

The algorithmic framework in Algorithm 5.2 leaves two obvious ways of choosing the search direction in Step 2.1: An *a priori* and a *dynamic* approach. An example of an a priori approach would be to pick the values for λ from the set $\{\frac{1}{k}, \frac{2}{k}, \ldots, \frac{k-1}{k}\}$, for some number $k > 0$, whereas a dynamic strategy would be to chose λ based on the previous iteration of the algorithm. In this paper we have implemented a dynamic updating scheme based on a modification of the so–called *Non–Inferior Set Estimation* framework proposed by Cohon (1978), Aneja and Nair (1979), and Dial (1979).

5.3.2 Step 3 - Obtaining a lower bound set

In this section, we describe how we derive lower bound sets of the current branching node η in Step 3 of Algorithm 5.1. Consider an active branching node η and let

$$L^C(\eta) = (C\underline{\mathcal{X}}(\eta))_N$$

denote the lower bound set equal to the set of non–dominated outcome vectors of the bi–objective LP–relaxation of the current node. One approach to obtaining lower bound sets would then be to solve the bi–objective LP–relaxation in each branching node and use Proposition 5.1 to test if the node can be fathomed. However, it may be computationally expensive to solve a bi–objective LP and checking condition (5.5) in Proposition 5.1 at every branching node. Therefore, we only want to solve the bi–objective LP at branching nodes

where there is a possibility that condition (5.5) holds (the search area S_2 defined in (5.2) is empty). Given $\lambda \in (0,1)$, let

$$\Lambda^{\lambda}(\eta) = \min\{(\lambda c^1 + (1-\lambda)c^2)x \ : \ x \in \underline{\mathcal{X}}(\eta)\} \tag{5.6}$$

denote the optimal solution value of the *λ–scalarized LP–relaxation* of the node η and

$$x^{\lambda}(\eta) \in \arg\min\{(\lambda c^1 + (1-\lambda)c^2)x \ : \ x \in \underline{\mathcal{X}}(\eta)\},$$

an optimal solution to (5.6). The following proposition gives sufficient conditions to ensure that S_2 is non-empty.

Proposition 5.2. *Consider upper bound set U and lower bound set $L(\eta) = L^C(\eta)$ and assume that the solution $x^{\lambda}(\eta)$ satisfies*

$$Cx^{\lambda}(\eta) \leq z^n$$

for some $z^n \in \mathcal{N}(U)$. Then

$$\left(L(\eta) + \mathbb{R}^2_{\leqq}\right) \cap \mathcal{N}(U) \neq \emptyset.$$

Proof. Suppose there exists a local Nadir point $z^n \in \mathcal{N}(U)$ such that $Cx^{\lambda}(\eta) \leq z^n$. Since $Cx^{\lambda}(\eta) \in L(\eta)$, we have that $z^n \in L(\eta) + \mathbb{R}^2_{\leqq}$ implying $\left(L(\eta) + \mathbb{R}^2_{\leqq}\right) \cap \mathcal{N}(U) \neq \emptyset$. □

Proposition 5.2 suggests to always solve a λ–scalarized LP before solving the bi–objective LP relaxation. If Proposition 5.2 holds, then there is no reason to solve the bi–objective LP, since the branching node η cannot be fathomed. Furthermore, solving the λ–scalarized LP provides us with a lower bound set

$$L^{\lambda}(\eta) = \{z \in \mathbb{R}^2 \ : \ \lambda z_1 + (1-\lambda)z_2 = \Lambda^{\lambda}(\eta)\},$$

which may be used to find a lower bound set $L(\eta)$ at branching node η as described in Proposition 5.3.

Proposition 5.3. *Given an active branching node η and parent node η_0 with lower bound set $L(\eta_0)$, the set*

$$L(\eta) = \left(\left(L(\eta_0) + \mathbb{R}^2_{\leqq}\right) \cap \left(L^{\lambda}(\eta) + \mathbb{R}^2_{\leqq}\right)\right)_N,$$

is a lower bound set of branching node η dominating both $L(\eta_0)$ and $L^{\lambda}(\eta)$.

Proof. Obviously, $L^{\lambda}(\eta)$ is a lower bound set for η, and by relaxation so is $L(\eta_0)$. The rest follows from Proposition 2 in Ehrgott and Gandibleux (2007). □

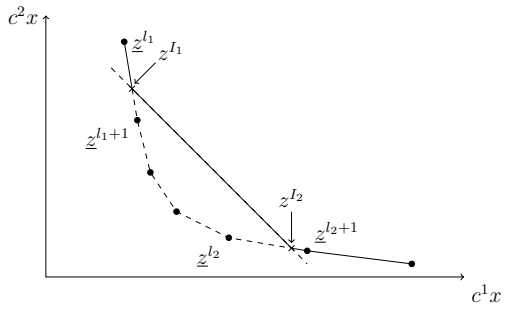


Figure 5.2: Updating the lower bound set using Proposition 5.3.

Step 3.1: If the branching at the parent node η_0 has been performed in objective space, solve the LP–relaxation and return $L^C(\eta)$.

Step 3.2: Solve the λ–scalarized LP and find $L(\eta)$ defined in Proposition 5.3.

Step 3.2: If $L(\eta)$ strictly dominates $L(\eta_0)$, then solve the LP–relaxation and return $L^C(\eta)$; otherwise return $L(\eta)$.

Algorithm 5.3: Step 3 of Algorithm 5.1.

Determining the set $L(\eta)$ defined in Proposition 5.3 is straightforward. Assume that for branching node η_0 the lower bound set $L(\eta_0)$ is represented by an ordered list of extreme points, $\{\underline{z}^1, \ldots, \underline{z}^L\}$, with $\underline{z}^l_1 < \underline{z}^{l+1}_1$. The updating is then simply done by first finding two pairs of points $(\underline{z}^{l_1}, \underline{z}^{l_1+1})$ and $(\underline{z}^{l_2}, \underline{z}^{l_2+1})$ satisfying

$$\lambda \underline{z}^{l_1}_1 + (1-\lambda)\underline{z}^{l_1}_2 \geqq \Lambda^\lambda(\eta) > \lambda \underline{z}^{l_1+1} + (1-\lambda)\underline{z}^{l_1+1}_2$$

and

$$\lambda \underline{z}^{l_2}_1 + (1-\lambda)\underline{z}^{l_2}_2 < \Lambda^\lambda(\eta) \leqq \lambda \underline{z}^{l_2+1} + (1-\lambda)\underline{z}^{l_2+1}_2,$$

as illustrated in Figure 5.2.

Having determined these two pairs of points, we simply calculate the intersection point between the straight line defined by the two points $\underline{z}^{l_1}$ and $\underline{z}^{l_1+1}$ (respectively $\underline{z}^{l_2}$ and $\underline{z}^{l_2+1}$) and the line defined by the lower bound set $L^\lambda(\eta)$. The two intersection points, say z^{I_1} and z^{I_2}, are inserted in the list and we obtain the new lower bound set $L(\eta)$ defined by

$$L(\eta) = \text{conv}(\{\underline{z}^1, \ldots, \underline{z}^{l_1}, z^{I_1}, z^{I_2}, \underline{z}^{l_2+1}, \ldots, \underline{z}^L\})_N.$$

Step 3 can now be specified in Algorithm 5.3. We are interested in solving the bi–objective LP–relaxation as few times as possible as long as the updated lower bound set is of sufficient quality. We use the heuristic rule that if the branching performed at the parent node was performed in objective space (see Section 5.3.5), the bi–objective LP–relaxation is calculated since we want new lower bound sets for different regions in the objective space (Step 3.0). Otherwise we solve the λ–scalarized LP (Step 3.1) and if the lower bound set of the parent node is strictly dominated by $L(\eta)$ found in Proposition 5.3 (all the extreme points of $L(\eta_0)$ lie below $L^\lambda(\eta)$), then we resolve the bi–objective LP–relaxation.

Step 5.1: If the subproblem corresponding to node η is infeasible, fathom η and go to Step 1.

Step 5.2: If Proposition 5.2 holds, go to Step 6.

Step 5.3: If Proposition 5.1 holds, go to Step 1.

Algorithm 5.4: Step 5 of Algorithm 5.1.

5.3.3 Step 4 - Update the upper bound set

Throughout the branch–and–bound process an upper bound set, U, of feasible points in objective space filtered by dominance is maintained and ordered such that $z_1^u < z_1^{u+1}$ for all $z^u, z^{u+1} \in U$. In Step 0 of Algorithm 5.1 U is initialized with the two lexicographic minima. Whenever a feasible solution is found, the outcome vector of the solution is inserted into the upper bound set U and the augmented set is filtered by dominance. In this way, the set U is constantly improving and will at any time form an upper bound set. Note that solutions can be obtained using heuristics from all active branching nodes even though the node is not a leaf node (all integer variables are fixed).

5.3.4 Step 5 - Bound fathoming

Algorithm 5.4 checks if an active node η in the branching tree, can be fathomed. If the subproblem corresponding to the branching node is infeasible, we fathom η and pick a new active node (Step 5.0); otherwise Proposition 5.2 is used as a preliminary check for not fathoming the node (Step 5.1). Finally, Proposition 5.1 is checked in Step 5.2. Proposition 5.1 can be checked using both an explicit and an implicitly given lower bound set $L(\eta)$. Below we describe three ways of testing Proposition 5.1.

Bound fathoming using an explicit lower bound set and LP

Assume that a lower bound set $L(\eta)$ is stored as an explicit set using the extreme points $\{\underline{z}^1, \ldots, \underline{z}^L\}$ of $L(\eta)$. Note that when $L(\eta)$ is $\mathbb{R}^2_{\leqq}$–convex, the set $L(\eta) + \mathbb{R}^2_{\leqq}$ forms an unbounded convex polygonal domain in $\mathbb{R}^2$, and the verification of the condition in Proposition 5.1 amounts to verifying if any of the points in $\mathcal{N}(U)$ lie in this convex polyhedron. A simple, and straightforward way of performing this bound fathoming check is to solve the

linear program

$$\begin{aligned} Z(z^n) = \min\ & s_1 + s_2 \\ \text{s.t.:}\ & \sum_{l=1}^{L} \underline{z}_1^l \lambda_k - s_1 \leqq z_1^n \\ & \sum_{l=1}^{L} \underline{z}_2^l \lambda_k - s_2 \leqq z_2^n \\ & \sum_{l=1}^{L} \lambda_l = 1 \\ & \lambda_l, s_1, s_2 \geqq 0,\ \forall l = 1, \ldots, L \end{aligned} \tag{5.7}$$

for all $z^n \in \mathcal{N}(U)$. If $Z(z^n) > 0$ for all $z^n \in \mathcal{N}(U)$, the node η can be fathomed based on bounding. Note that the linear programs (5.7) only have three constraints, leading to linear programs which can be solved very quickly. Furthermore, for two different local Nadir points z^n and $z^{n'}$, the linear programs (5.7) differs in the right hand sides only. This means that very few dual simplex iterations are usually needed in order to resolve these linear programs.

Bound fathoming using an explicit lower bound set and a point–in–polytope algorithm

Let $\{\underline{z}^1, \ldots, \underline{z}^L\}$ denote the extreme points of $L(\eta)$ and note that

$$\mathcal{Z}_N \subseteq \left(\{(z_1^{lr}, z_2^{ul})\} - \mathbb{R}^2_{\leqq} \right),$$

where $z^{ul} = \operatorname{lex\,min}\{(c^1 x, c^2 x) : x \in \mathcal{X}\}$ and $z^{lr} = \operatorname{lex\,min}\{(c^2 x, c^1 x) : x \in \mathcal{X}\}$ are the two lexicographic minima. Hence Proposition 5.1 does not hold if a local Nadir point is in the polygon given by

$$\operatorname{conv}(\{(\underline{z}_1^1, z_2^{ul}), \underline{z}^1, \ldots, \underline{z}^L, (z_1^{lr}, \underline{z}_2^L), (z_1^{lr}, z_2^{ul})\}). \tag{5.8}$$

This means that the condition in Proposition 5.1 can be tested by calling, for each local Nadir point, a *point–in–polytope* algorithm which tests for inclusion in the polygon (5.8). As soon as the algorithm declares that a local Nadir point is within the polygon, we know the branching node cannot be fathomed. Fortunately, much research has gone into point–in–polytope (PIP) algorithms and we refer the interested reader to the computational study by Schirra (2008) of the reliability and speed of a number of different algorithmic approaches for the PIP problem.

Bound fathoming using an implicit lower bound set and LP

The bound fathoming can also be done without explicitly maintaining a lower bound set. Note that the condition in Proposition 5.1 asks if any point on the lower bound set dominates a local Nadir point and that the lower bound set of the branching node η we are using is

$(C\mathcal{X}(\eta))_N$. Deciding if there exists a point $\underline{z} \in (C\underline{\mathcal{X}}(\eta))_N$ which (strictly) dominates a local Nadir point $z^n \in \mathcal{N}(U)$ is equivalent to having $\tilde{Z}(z^n) = 0$ where

$$\begin{aligned} \tilde{Z}(z^n) = \min\ & s_1 + s_2 \\ \text{s.t.:}\ & c^1 x - s_1 \leqq z_1^n \\ & c^2 x - s_2 \leqq z_2^n \\ & x \in \underline{\mathcal{X}}(\eta) \\ & s_1, s_2 \geqq 0. \end{aligned} \tag{5.9}$$

Note that an optimal solution (x^*, s^*) to the linear program (5.9) either has $\tilde{Z}(z^n) = 0$ implying $Cx^* \leq z^n$ (that is, there is a solution to the LP-relaxation which dominates z^n) or we have $\tilde{Z}(z^n) = s_1^* + s_2^* > 0$. This leads to Proposition 5.4.

Proposition 5.4. *The branching node η can be fathomed if the optimal solution value $\tilde{Z}(z^n)$ of the program* (5.9) *is strictly positive for all $z^n \in \mathcal{N}(U)$.*

This implicit approach eliminates the need for generating the entire efficient frontier of the LP-relaxation and only one linear program needs to be solved for each local Nadir point. The approach might then have its merits for problems with few non–dominated outcomes compared to the number of extreme points of $(C\underline{\mathcal{X}}(\eta))_N$, but this is not a trivial matter to decide a priori. A possible drawback compared to generating the complete set $(C\underline{\mathcal{X}}(\eta))_N$ is that the updating scheme described in Proposition 5.3 no longer applies, implying that additional linear programming problems need to be solved at each branching node. Furthermore, the program (5.9) might be large and solving it may consequently be rather time consuming. As the program (5.9) must potentially be solved for several local Nadir points, this might lead to prohibitive computation times.

5.3.5 Step 6 - Performing branching

As the set of feasible solutions does not differ compared to a single objective combinatorial optimization problem, the branching rules devised for these problems can be applied. However, much information can be gained by utilizing the definition of an efficient solution and its non–dominated outcome vector. As mentioned in Section 5.3.2, the bi–objective LP–relaxation is not solved at each branching node. However, the weighted sum scalarization

$$\min\{(\lambda c^1 + (1-\lambda)c^2)x \ :\ x \in \underline{\mathcal{X}}(\eta)\} \tag{5.10}$$

is solved. Let $\underline{x}(\eta)$ be an optimal solution to problem (5.10), and let $\underline{z}(\eta) = C\underline{x}(\eta)$ be the corresponding outcome vector. First, we outline the very effective branching strategy proposed in Stidsen et al. (2014) called *Pareto branching* (PB). PB is based on the observation

that if $z \leqq \underline{z}(\eta)$ for all z in the *ordered* sublist $\{\bar{z}^{u_1}, \ldots, \bar{z}^{u_K}\} \subseteq U$ of the current upper bound set, then the branching node η can be split by the disjunction

$$c^1 x \leqq \bar{z}_1^{u_1} - 1 \quad \vee \quad c^2 x \leqq z_2^{u_K} - 1.$$

We note, that in our implementation we update the upper bound set U *before* branching. This means, that if a node results in an integer feasible solution, the outcome vector of this solution is part of the upper bound set when branching is performed. Therefore, if a branching node results in an integer feasible solution, Pareto branching can always be performed and there is no need to add the weaker so–called no–good inequalities

$$\sum_{i:\underline{x}(\eta)_i=0} x_i + \sum_{i:\underline{x}(\eta)_i=1} (1 - x_i) \geqq 1,$$

used in ranking based two–phase methods.

The idea of PB can be expanded to *extended Pareto branching* (EPB). From the definition of the search area given in (5.3), it is evident that non–dominated outcomes can only exist in the set

$$\bigcup_{z^n \in \left(L(\eta)+\mathbb{R}^2\right) \cap \mathcal{N}(U)} \left(z^n - \mathbb{R}^2_{\leqq}\right).$$

Before making a branching decision, we already check whether local Nadir points exist in the set $L(\eta) + \mathbb{R}^2_{\leqq}$ (see Proposition 5.1 and Proposition 5.4),and therefore we can simply create a child node for each local Nadir point found. Note that this split of the branching node η might not separate the current LP–solution as we may have that

$$Cx(\eta) \in \bigcup_{z \in \left(L(\eta)+\mathbb{R}^2\right) \cap \mathcal{N}(U)} \left(z - \mathbb{R}^2_{\leqq}\right).$$

Therefore, we only perform extended Pareto branching when in fact there exits a $\bar{z} \in U$ such that $\bar{z} \leq \underline{z}(\eta)$. This guarantees separation of the branching node.

Note that the EPB requires that all local Nadir points are checked. If the EPB rule is not applied, finding a single local Nadir point dominated by the lower bound set leads to the conclusion that branching node η cannot be fathomed. This implies a tradeoff between stronger branching rules and faster treatment of branching nodes that cannot be fathomed.

If we cannot perform either extended Pareto branching or Pareto branching (note that this means, that $x(\eta) \notin \{0,1\}^n$), the branching node is simply separated by a variable dichotomy. The branching variable is chosen based on pseudo–cost information provided by the solver (see for example Achterberg, Koch, and Martin (2005) for a discussion of variable selection strategies in single objective combinatorial optimization).

Table 5.2: Different implementations of the bi–objective branch–and–cut algorithm.

Abbreviation[1]	`E-PB-PIP`	`E-PB-LP`	`E-EPB-PIP`	`I-PB-LP`	`I-EPB-LP`
Node selection	Best first	Best first	Best first	Best first	Best first
Cuts	At root node	At root node	At root node	At root note	At root node
Lower bound set	Explicit	Explicit	Explicit	Implicit	Implicit
Fathoming nodes	PIP	LP	PIP	LP	LP
Pareto Branching	Yes	Yes	Yes	Yes	Yes
Extended PB	No	No	Yes	No	Yes

[1] A–B–C. A: Lower bound set (E: explicit, I: implicit). B: Branching strategy (PB: Pareto branching, EPB: extended Preto branching). C: Method for testing the condition in Proposition 5.1 (LP: linear programming, PIP: point–in–polytope).

5.4 Computational results

In this section we report on the computational experiments conducted with the bi–objective branch–and–cut algorithms for the bi–objective single–source capacitated facility location problem (BO–SSCFLP) (see Appendix 5.A). The purpose of the computational study is to answer the following questions:

(i) Which implementations based on explicit or implicit lower bound sets perform the best?

(ii) Given an explicit lower bound set, does node fathoming based on linear programming or point–in–polytope algorithms perform the best?

(iii) Is it worth performing extended Pareto branching?

(iv) Is adding cutting planes effective in improving the running time?

(v) Does the lower bound updating scheme given in Proposition 5.3 improve the performance?

(vi) Is the bound set based branch–and–cut algorithm competitive with state–of–the art algorithms?

To answer the first three questions above, we have implemented different versions of Algorithm 5.1. An overview is given in Table 5.2. The algorithms prefixed with an `E` are all based on explicitly generated lower bound sets while algorithms prefixed with an `I` rely on implicit lower bound sets (see Section 5.3.4). The branching strategy is indicated using the abbreviations `PB` for Pareto branching and `EPB` for extended Pareto branching (see Section 5.3.5). For the explicit lower bounds, we proposed two ways of fathoming nodes; one based on linear programming (`LP`) (see Section 5.3.4) and one based on the point–in–polytope (`PIP`) algorithm (see Section 5.3.4). For the algorithms with implicitly given lower

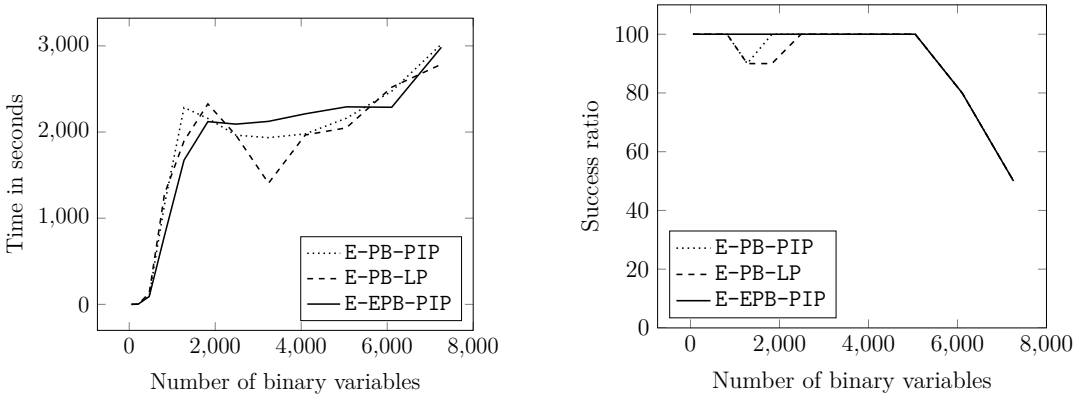
bound sets, nodes can only be fathomed using linear programming (see Section 5.3.4). Note that all implementations use a best first search, where the node having the smallest value of $\Lambda^{\lambda}(\eta)$ is chosen as the next node to be processed. Since computational experience for single objective optimization problems shows that cutting planes have a larger effect at the root note compared to nodes deeper in the tree, we call Algorithm 5.2 only at the root node. We use general lifted cover inequalities and Fenchel inequalities as cutting planes for the knapsack structures arising from the capacity constraints of the BO–SSCFLP as they have been shown to be effective for the SSCFLP (see e.g. Gadegaard, Klose, and Nielsen (2016d)). Furthermore, Stidsen et al. (2014) established that Pareto branching in bi–objective branch–and–bound gives a significant speed up compared to only branching on variables. We therefore include Pareto branching in all algorithms.

After testing the five implementations specified in Table 5.2, we address the remaining three questions as follows:

- We answer Question (iv) by comparing the best explicit and implicit implementations with and without cutting planes added.
- We answer Question (v) by comparing the explicit lower bound set based algorithm with and without the updating strategy.
- We finally answer Question (vi) by comparing the overall best implementation of Algorithm 5.1 with two different implementations of the two–phase method.

5.4.1 Implementation details and test instances

All implementations have been coded in C and C++ and compiled using gcc and g++ with optimization option O3 and C++11 enabled. They are all publicly available (see Gadegaard, Nielsen, and Ehrgott (2016a)). All implementations use CPLEX 12.6 with callbacks as solver. The ParallelMode switch is set to deterministic such that different runs can be compared, the Reduce switch is set such that neither primal nor dual reduction is performed, and all internal cuts of CPLEX are turned off. For all instances a fixed time limit of 3600 CPU seconds (one hour of computation time) is set after which the search is aborted. As CPLEX 12.6 with callbacks is limited to creating at most two child nodes when branching, we only perform extended Pareto branching when there are one or two local Nadir points in the set $(L(\eta) + \mathbb{R}^2_{\leqq})$. If there are more local Nadir points in $(L(\eta) + \mathbb{R}^2_{\leqq})$, we resort to Pareto branching as explained above. For `E-PB-PIP` and `E-EPB-PIP` the point–in–polytope problem is solved using the `PNPOLY` algorithm developed by Franklin (2006) while all implementations using linear programming for node fathoming are solved by CPLEX using the dual simplex algorithm. Since CPLEX does not allow for changes in the objective function in the callbacks, we use a fixed value of $\lambda = 0.5$ during the branch–and–cut process.



(a) Average CPU time in seconds.

(b) Instances solved within the time limit (one hour).

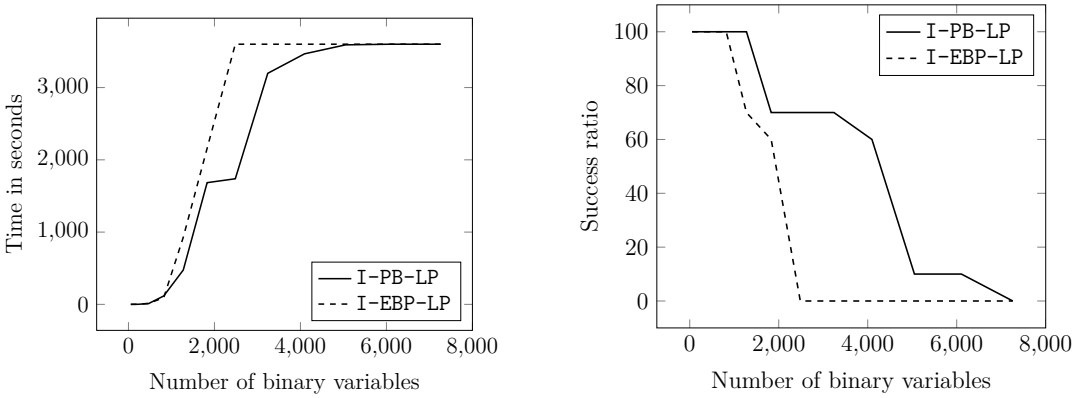
Figure 5.3: Performance of implementations based on explicit lower bound sets.

For the computational study we have generated a number of instances of the BO-SSCFLP. These instances were generated in the same way as was done by Stidsen et al. (2014) for the uncapacitated version of the BO–SSCFLP. The demands were generated from the set $\{5, \ldots, 10\}$ and the capacities from the set $\{10, \ldots, 20\}$, both according to a uniform distribution. The ratio between the total capacity and total demand is then scaled to equal $r \in \mathbb{R}$, where r is uniformly generated from the interval $[1.5, 4]$. For each instance size, defined by $|\mathcal{I}| \times |\mathcal{J}|$, we have generated 10 instances. The instance generator as well as the instances are all publicly available (see Gadegaard, Nielsen, and Ehrgott (2016c)). The number of facilities is $|\mathcal{I}| \in \{5, 10, 15, \ldots, 60\}$ and the number of customers is set to $|\mathcal{J}| = 2|\mathcal{I}|$. This leads to 120 instances of the SSCFLP ranging in sizes from 5×10 to 60×120, implying the number of binary variables ranges from 55 to 7,260.

5.4.2 Questions (i)–(iii) – Comparison of implementations

Figure 5.3 shows a comparison of the implementations based on an explicitly given lower bound set. From Figure 5.3a we see that all implementations based on explicit lower bounds perform equally well. The time in CPU seconds is an average over all instances which could be solved within an hour. All algorithms are able to solve most of the instances with up to 6,000 variables within an hour. However, when the instances grow beyond this size, the success ratio begins to decrease, but as shown in Figure 5.3b, 50% of the instances having more than 7,000 variables were still solvable within an hour. Note that `E-PB-PIP` and `E-PB-LP` both fail to solve a single instance having 1,275 variables and `E-PB-LP` also fails to solve one instance having 1,830 binary variables. The algorithm `E-EPB-PIP` does, however, solve all these instances within an hour.

Considering the results obtained for implementations using explicitly given bound sets



(a) Average CPU time in seconds.

(b) Instances solved within the time limit (one hour).

Figure 5.4: Performance of implementations based on implicit lower bound sets.

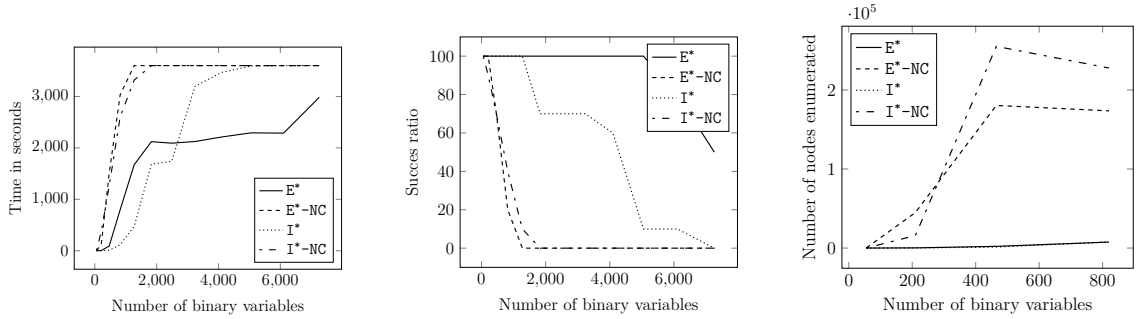
there is no clear winner, but as `E-EPB-PIP` solves slightly more instances within an hour compared to `E-PB-PIP` and `E-PB-LP`, it therefore seems to be more robust.

Now consider the two implementations based on implicit lower bounds. As is shown in Figure 5.4a the two implementations are comparable for relatively small problem instances, but as the number of binary variables increases, `I-EPB-LP` becomes more time consuming than `I-PB-LP`. From Figure 5.4b we see that `I-EPB-LP` was able to solve instances of up to 1,830 binary variables only, while I–PB–LP performed better by solving instances of up to 5,000 binary variables. The reason is that `I-EPB-LP` has to solve one LP for each local Nadir point whereas `I-PB-LP` only has to find one local Nadir point for which the linear program (5.9) has a strictly positive solution value.

When we compare the results against the implementations based on explicit lower bound sets, we see that only significantly smaller instances could be solved using implicit lower bound sets. The computational study of the `E`–implementations showed that in approximately half of the branching nodes the lower bound set is updated using Proposition 5.3 (see Section 5.4.4 for a computational study of the effect of Proposition 5.3), whereas the `I`–implementations need to solve the rather large LPs in (5.9) several times for each branching node. Especially the `I-EPB-LP` suffers from this problem as more LPs need to be solved in order to perform the extended Pareto branching.

The above tests clearly show, that the best explicit and implicit implementations are `E-EPB-PIP` and `I-PB-LP`.

The overall best implementation seems to be `E-EPB-PIP` which is based on explicit lower bound sets, extended Pareto branching, and a PIP algorithm to solve the fathoming test. It outperforms the best implementation using implicit lower bound sets, the `I-PB-LP`. The use of extended Pareto branching seems to make the algorithm more robust compared to



(a) Average CPU time in seconds.

(b) Instances solved within the time limit (one hour).

(c) Number of nodes enumerated.

Figure 5.5: The effect of cutting planes.

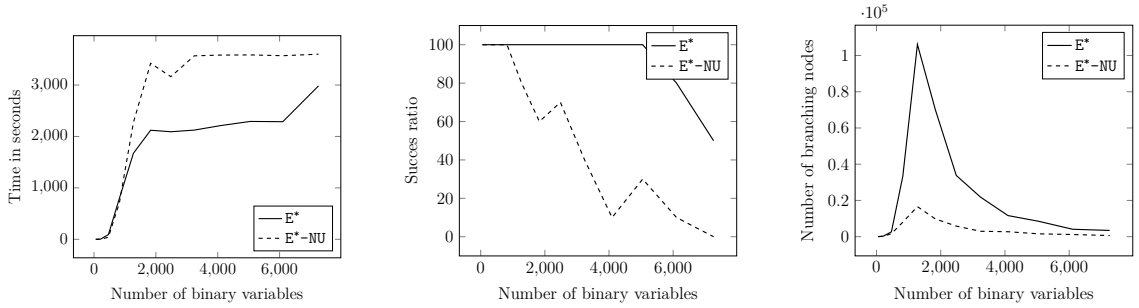
the other implementations with explicit lower bound sets. Hence in the following, we will only perform further tests with the best implementations under explicit and implicit lower bound sets (`E-EPB-PIP` and `I-PB-LP`). In the remainder of the paper we denote these two implementations `E*` and `I*`, respectively.

5.4.3 Question (iv) – Is adding cuts worth the effort?

To test whether adding cuts at the root node as explained in Section 5.3.1 contributes positively to the solution time, we compare the performance of `E*` and `I*` with implementations where the cutting plane algorithm is turned off. We denote these two new variants `E*-NC` and `I*-NC`, where `NC` is an abbreviation for "No Cuts". Instead we let CPLEX generate cutting planes at the root node.

Figure 5.5 clearly shows that we get a positive effect by adding cuts to the bi-objective LP. The addition of cuts at the root node has a high positive impact on the solution time and solvability of the instances within the one hour limit. It seems that strong cutting planes are necessary in order to use the bound set branch–and–cut algorithm.

In Figure 5.5c we see that the versions without the initial cutting plane algorithm experience a substantially faster growth in the number of nodes that the algorithms need to enumerate. A peculiar phenomenon is that the `I*-NC` enumerates *more* branching nodes than the `E*-NC`, even though the `E*-NC` updates the lower bound set. This basically means that the lower bound sets used in `E*-NC` are not as strong as those used in `I*-NC`. The explanation seems to be that the solver chooses different search paths for the two algorithms and that the search paths used for `E*-NC` lead to better feasible solutions faster, increasing the fathoming potential. Overall, however, the addition of cuts at the root node has a high impact on the solution times and the instance sizes solvable, and we dare conclude, that cutting planes are necessary in order to solve these instances.



(a) Average CPU time in seconds.

(b) Instances solved within the time limit (one hour).

(c) Number of nodes enumerated.

Figure 5.6: Effect of Proposition 5.3.

5.4.4 Question (v) – Effect of using Proposition 5.3

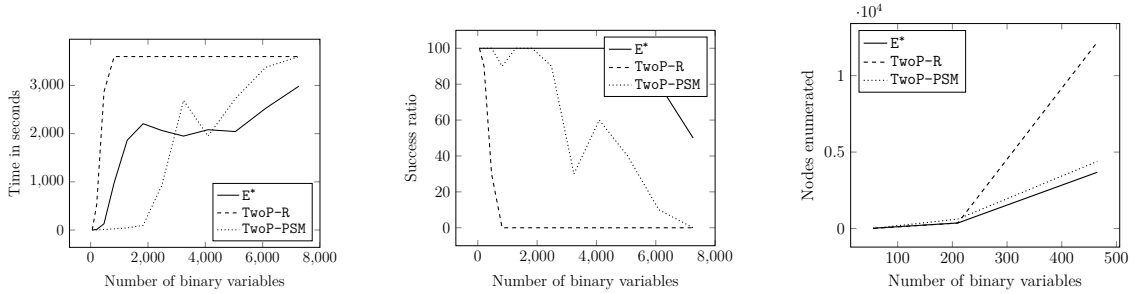
To check if the lower bound updating strategy of Proposition 5.3 contributes positively to the running times, we modified `E*` such that the bi–objective LP–relaxation is solved in all branching nodes. That is, we obtain the best possible $\mathbb{R}^2_{\leqq}$–convex lower bound set and may fathom nodes faster. This may, however, come at the expense of a higher CPU time needed to solve the bi-objective LP. We name this implementation `E*-NU` where `NU` is an abbreviation for "No Updating".

In Figure 5.6c it is seen that `E*` produces an order of magnitude more branching nodes compared to the `E*-NU`. This was expected as the lower bound sets generated by the updating strategy are weaker than those produced by $L^C(\eta)$. Figure 5.6a shows that the updating strategy has a positive effect as the instances grow in size. Furthermore, using the updating strategy makes the procedure more robust as can be seen in Figure 5.6b. When we solve a bi–objective LP at each node, some instances become very time consuming, even for smaller sizes. The reason is that solving a degenerate bi–objective LP requires the solution of several degenerate single objective LPs. The updating strategy circumvents this issue by updating the lower bound set via the solution of a *single* objective degenerate LP.

In summary, the updating scheme significantly improves the running time and robustness of the E* algorithm, and we conclude that the updating scheme is necessary for the branch–and–cut algorithm when solving larger instances.

5.4.5 Question (vi) – Comparing against the two–phase method

To test the effectiveness of the overall best branch–and–cut approach, we compare `E*` with two implementations of the two–phase method. Both implementations are publicly available (see Gadegaard, Nielsen, and Ehrgott (2016b)).



(a) Average CPU time in seconds.

(b) Instances solved within the time limit (one hour).

(c) Number of nodes enumerated by the algorithms (only instances solved by two–phase method).

Figure 5.7: Comparison with the two–phase method.

The first two–phase method is implemented as indicated in Stidsen et al. (2014), namely with a second phase based on ranking. We use the abbreviation `TwoP-R` to denote this two-phase ranking method. The results obtained with the two–phase algorithm seem to be consistent with the results obtained in Stidsen et al. (2014) where facility location problems having up to 20 binary variables could be solved within 300 CPU seconds. Here, we can solve instances of up to 500 binary variables within ten times the computation time. To meet the potential critique that `TwoP-R` is badly implemented, we here mention that between 99.3% and 100.0% of the running time was spent by CPLEX solving the subproblems. Furthermore, we have implemented `TwoP-R` such that CPLEX *reoptimizes* the MIPs arising in the subproblems after adding branching constraints. This gives a significant speedup compared to solving each MIP from scratch.

Figure 5.7 shows the performance of `TwoP-R` (dashed line) compared to that of `E*` (solid line). It is clear from Figure 5.7a and Figure 5.7b that the ranking based two–phase method is very inferior to the branch–and–bound algorithm `E*`. Figure 5.7c gives an explanation of why `TwoP-R` performs badly compared to `E*`. When we look at the instances actually solved by `TwoP-R`, we see a much faster increase in the number of branching nodes to be enumerated in order for the algorithm to solve the instances compared to the `E*`. As `TwoP-R` ranks the solutions between the supported non–dominated solutions found in the first phase, it generates many equivalent solutions leading to a large number of MIPs solved redundantly. Furthermore, even though `TwoP-R` is implemented such that the MIPs are reoptimized after adding no–good constraints, a new root node has to be solved in each MIP leading to excessive computation times.

As the results obtained with this standard ranking based two–phase method were rather disappointing in terms of computation times, we also implemented a two–phase method where the second phase is based on the *perpendicular search method* (PSM) proposed by

Chalmet, Lemonidis, and Elzinga (1986). We denote the two–phase method based on PSM, `TwoP-PSM`. The PSM is basically a branch–and–bound algorithm where an MIP is solved in each node, and where all branching is performed in objective space. The branching strategy in PSM is equivalent to single solution Pareto branching. This implies, that *no* equivalent solutions are generated, and we thereby circumvent this obvious issue with `TwoP-R`.

Figure 5.7a shows that `TwoP-PSM` (dotted line) performs much better than `TwoP-R` and that it even outperforms `E*` for smaller instances. Although `TwoP-PSM` is faster for the smaller instances, it becomes unstable in the sense that some instances become unsolvable within an hour whereas others can be solved within a few minutes. This can be seen in Figure 5.7b where the success ratio for the `TwoP-PSM` drops much earlier than for the `E*` algorithm. One of the main reasons why the `TwoP-PSM` is faster for small instances is that the preprocessing done by CPLEX reduces the instances considerably. Also, CPLEX finds strong upper bounds very quickly which means that the probing techniques implemented in the solver is able to fix many variables at an early stage of the search. But as the instances grow larger, the preprocessing and the internal heuristics seem to perform worse and more branching nodes need to be enumerated. Both two-phase methods are also slowed down by the scaling of the objective functions performed when searching the triangles created by the first phase; even though the weights used are integers, the difference in the coefficients becomes very large and the instances suffer from *bad scaling* which increases the computation times considerably.

In sum, both two-phase methods in our implementation are outperformed by `E*` on large instances. In particular, the ranking based `TwoP-R` performs very poorly on all instances, whereas the `TwoP-PSM` algorithm has its merits in case of smaller instances.

5.5 Conclusions

In this paper we have developed a novel bound set based branch–and–cut algorithm for solving bi–objective combinatorial optimization problems. The algorithm were tested using both explicit and implicit representations of lower bound sets, and we have shown that the best algorithm based on explicit lower bound sets outperforms the best algorithm based on implicit lower bound sets. We proposed an updating scheme that prevented the algorithm from solving a bi–objective LP at each node. Computational results have shown that the cost of weaker lower bound sets was by far outweighed by the improvement in speed when a branching node is processed. The paper also suggests a simple bi–objective cutting plane algorithm that significantly improves the performance of both the explicit and the implicit lower bound based algorithms. Furthermore, we proposed an extension of the Pareto branching strategy suggested in the literature and showed that it makes the explicit lower bound set based algorithm more stable. Finally, we proved the effectiveness of the

branch–and–cut algorithm by comparing it to two different implementations of the two–phase method. Especially for larger instances, did the new branch–and–cut algorithm outperform the two–phase methods. An interesting area for future research would be to investigate extensions of the algorithms to multi–objective problems with more than two objectives. Using a generalization of the local Nadir points to higher dimensions and the LP based fathoming approaches, both the explicit and implicit lower bound based algorithms could be extended to three or more criteria. Another fruitful area could be to extend the reduction and preprocessing techniques developed for single objective combinatorial optimization to the multi-objective versions. Last, a computational study with problems having a totally unimodular constraint matrix is interesting as feasible solutions can be "harvested" when solving the bi–objective LPs arising in the branching nodes.

5.6 Acknowledgements

The authors would like to thank Professor Kim Allan Andersen for insightful comments and literature suggestions. This work was partially supported by a grant from Købmand Ferdinand Sallings Mindefond.

5.A The single–source capacitated facility location problem

We have chosen to test our algorithm on the bi–objective single–source capacitated facility location problem (BO–SSCFLP) which can be described as follows: given are a set $\mathcal{I}$ of potential facility sites and a set $\mathcal{J}$ of demand points. Each facility i has a fixed opening cost of $f_i > 0$ and a capacity $s_i > 0$ while each demand point j has a fixed and known demand of $d_j > 0$. Each demand point j must be serviced by exactly one open facility i, resulting in a cost of c_{ij}. It is assumed that all parameters are non–negative integers. The BO–SSCFLP is then the problem of minimizing the total opening cost and the total servicing cost. Given binary variables y_i equaling one only if facility i is open and binary variable x_{ij} equaling one if and only if customer j is serviced by facility i, the bi–objective SSCFLP can be stated as the BOCO problem

$$\begin{aligned}
\min\ & \left(\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} c_{ij}x_{ij}\ ,\ \sum_{i\in\mathcal{I}} f_i y_i \right) \\
\text{s.t.:}\ & \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \\
& \sum_{j\in\mathcal{J}} d_j x_{ij} \leqq s_i y_i, \quad \forall i \in \mathcal{I}, \\
& x_{ij},\ y_i \in \{0,1\}, \quad \forall j \in \mathcal{J},\ i \in \mathcal{I}.
\end{aligned}$$

The SSCFLP matches the assumption of both objectives having integral values for all feasible solutions. In addition, the bi–objective SSCFLP is a very natural BOCO problem as the objectives are antagonistic by nature: opening an extra facility results in an increase in fixed opening costs and a decrease in servicing costs.

Bibliography

Achterberg, T., T. Koch, and A. Martin (2005). Branching rules revisited. *Operations Research Letters 33*(1), 42 – 54.

Ahuja, R., J. Orlin, S. Pallottino, M. Scaparra, and M. Scutella (2004). A multi-exchange heuristic for the single-source capacitated facility location problem. *Management Science 50*(6), 749–760.

Ahuja, R., J. Orlin, and D. Sharma (2000). Very large-scale neighborhood search. *International Transactions in Operational Research 7*(4-5), 301–317.

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

Avella, P., M. Boccia, and S. Salerno (2011). A computational study of dicut reformulation for the Single Source Capacitated Facility Location problem. *Studia Informatica Universalis 9*(3), 21–42.

Avella, P., M. Boccia, A. Sforza, and I. Vasilév (2008). An effective heuristic for large-scale capacitated facility location problems. *Journal of Heuristics 15*(6), 597–615.

Avella, P., A. Sassano, and I. Vasilév (2007). Computational study of large-scale p-median problems. *Mathematical Programming 109*(1), 89–114.

Balas, E. (1975). Facets of the Knapsack polytope. *Mathematical Programming 8*(1), 146–164.

Balas, E. and E. Zemel (1978). Facets of the Knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics 34*(1), 119–148.

Balinski, M. (1965). Integer programming: Methods, uses, computations. *Management Science 12*(3), 253–313.

Balinski, M. and P. Wolfe (1963). On Benders decomposition and a plant location problem. *ARO–27, Mathematica, Princeton, NJ*, 45.

Bar-Ilan, J., G. Kortsarz, and D. Peleg (1993). How to allocate network centers. *Journal of Algorithms 15*(3), 385 – 415.

Barceló, J. and J. Casanovas (1984). A heuristic Lagrangean algorithm for the capacitated plant location problem. *European Journal of Operational Research 15*(2), 212–226.

Barceló, J., E. Fernández, and K. Jörnsten (1991). Computational results from a new lagrangean relaxation algorithm for the capacitated plant location problem. *European Journal of Operational Research 53*(1), 38–45.

Barceló, J., Å. Hallefjord, E. Fernández, and K. Jörnsten (1990). Lagrangean relaxation and constraint generation procedures for capacitated plant location problems with single sourcing. *Operations-Research-Spektrum 12*(2), 79–88.

Beasley, J. (1993). Lagrangean heuristics for location problems. *European Journal of Operational Research 65*(3), 383 – 399.

Beltran, C., C. Tadonki, and J. Vial (2006). Solving the p-Median Problem with a Semi–Lagrangian Relaxation. *Computational Optimization and Applications 35*(2), 239–260.

Beltran-Royo, C., J. Vial, and A. Alonso-Ayuso (2012). Semi–Lagrangian relaxation applied to the uncapacitated facility location problem. *Computational Optimization and Applications 51*, 287–409.

Bérubé, J., M. Gendreau, and J. Potvin (2009). 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.

Bhaskaran, S. and M. Turnquist (1990). Multiobjective transportation considerations in multiple facility location. *Transportation Research Part A: General 24*(2), 139 – 148.

Bitran, G., V. Chandru, D. Sempolinski, and J. Shapiro (1981). Inverse optimization: An application to the capacitated plant location problem. *Management Science 27*(10), 1120 – 1141.

Boccia, M., A. Sforza, C. Sterle, and I. Vasilyev (2008). A cut and branch approach for the capacitated p-median problem based on Fenchel cutting planes. *Journal of Mathematical Modelling and Algorithms 7*(1), 43–58.

Boyd, E. A. (1993). Generating Fenchel cutting planes for Knapsack polyhedra. *Siam Journal on Optimization 3*(4), 734–750.

Brimberg, J. and C. ReVelle (1998). A bi-objective plant location problem: cost vs. demand served. *Location Science 6*(1–4), 121 – 135.

Ceselli, A. and G. Righini (2005). A branch-and-price algorithm for the capacitated p–median problem. *Networks 45*(3), 125–142.

Chalmet, L., L. Lemonidis, and D. Elzinga (1986). An algorithm for the bi-criterion integer programming problem. *European Journal of Operational Research 25*(2), 292 – 300.

Chanta, S., M. Mayorga, and L. McLay (2011). Improving emergency service in rural areas: a bi-objective covering location model for EMS systems. *Annals of Operations Research 221*(1), 133–159.

Chen, C. and C. Ting (2008). Combining Lagrangian heuristic and ant colony system to solve the single source capacitated facility location problem. *Transportation Research Part E: Logistics and Transportation Review 44*(6), 1099 – 1122.

Church, R. and C. ReVelle (1974). The maximal covering location problem. *Papers in Regional Science 32*(1), 101–118.

Climer, S. and W. Zhang (2006). Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence 170*(8), 714–738.

Cohon, J. (1978). *Multiobjective Programming and Planning*. Academic Presse, INC. (London) LTD.

Contreras, I. and J. Díaz (2008). Scatter search for the single source capacitated facility location problem. *Annals of Operations Research 157*(1), 73–89.

Cooper, L. (1964). Heuristic methods for location-allocation problems. *SIAM Review 6*(1), 37–53.

Cornuejols, G., R. Sridharan, and J. Thizy (1991). A comparison of heuristics and relaxations for the capacitated plant location problem. *European Journal of Operational Research 50*(3), 280–297.

Correia, I., L. Gouveia, and L. Saldanha-da Gama (2010). Discretized formulations for capacitated location problems with modular distribution costs. *European Journal of Operational Research 204*(2), 237 – 244.

Cortinhal, M. and M. Captivo (2003). Upper and lower bounds for the single source capacitated location problem. *European Journal of Operational Research 151*(2), 333 – 351.

Crowder, H., E. Johnson, and M. Padberg (1983). Solving large-scale zero-one linear programming problems. *Operations Research 31*(5), 803–834.

Current, J., C. ReVelle, and J. Cohon (1985). The maximum covering/shortest path problem: A multiobjective network design and routing formulation. *European Journal of Operational Research 21*(2), 189 – 199.

Current, J. and C. Weber (1994). Application of facility location modeling constructs to vendor selection problems. *European Journal of Operational Research 76*(3), 387 – 392.

Davis, P. S. and T. L. Ray (1969). A branch-bound algorithm for the capacitated facilities location problem. *Naval Research Logistics Quarterly 16*(3), 331–344.

de Fermat, P. (1679). Methodus and disquirendam maximam et minimam. In *Varia opera mathematica*.

Dearing, P. and F. Newruck (1979). A capacitated bottleneck facility location problem. *Management Science 25*(11), 1093–1104.

Dial, R. (1979). A model and algorithm for multicriteria route-mode choice. *Transportation Research Part B: Methodological 13*(4), 311–316.

Díaz, J. and E. Fernández (2002). A branch-and-price algorithm for the single source capacitated plant location problem. *Journal of the Operational Research Society 53*, 728–740.

Drezner, T., Z. Drezner, and S. Salhi (2006). A multi-objective heuristic approach for the casualty collection points location problem. *The Journal of the Operational Research Society 57*(6), 727–734.

Drezner, Z. (1984). The planar two-center and two-median problems. *Transportation Science 18*(4), 351–361.

Du, F. and G. Evans (2008). A bi-objective reverse logistics network analysis for post-sale service. *Computers & Operations Research 35*(8), 2617 – 2634.

Ehrgott, M. (2005). *Multicriteria Optimization* (2nd ed.). Springer berlin, Heidelberg.

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

Elloumi, S. (2010). A tighter formulation of the p-median problem. *Journal of Combinatorial Optimization 19*(1), 69–83.

Elloumi, S., M. Labbé, and Y. Pochet (2004). A new formulation and resolution method for the p-center problem. *INFORMS Journal on Computing 16*(1), 84–94.

Farahani, R., M. SteadieSeifi, and N. Asgari (2010). Multiple criteria facility location problems: A survey. *Applied Mathematical Modelling 34*(7), 1689–1709.

Fernández, E. and J. Puerto (2003). Multiobjective solution of the uncapacitated plant location problem. *European Journal of Operational Research 145*(3), 509–529.

Filippi, C. and E. Stevanato (2013). Approximation schemes for bi-objective combinatorial optimization and their application to the tsp with profits. *Computers & Operations Research 40*(10), 2418 – 2428.

Fischetti, M. and A. Lodi (2003). Local branching. *Mathematical Programming 98*(1-3), 23–47.

Fischetti, M., C. Polo, and M. Scantamburlo (2004). A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks 44*(2), 61–72.

Florios, K., G. Mavrotas, and D. Diakoulaki (2010). Solving multiobjective, multiconstraint knapsack problems using mathematical programming and evolutionary algorithms. *European Journal of Operational Research 203*(1), 14 – 21.

Franklin, W. (2006). PNPOLY - point inclusion in polygon test. `http://www.ecse.rpi.edu/~wrf/Research/Short_Notes/pnpoly.html`. Source code.

Gadegaard, S. (2016). Integrating cut–and–solve and semi–lagrangean dual ascent for the single–source capacitated facility location problem. `http://www.optimization-online.org/DB_HTML/2016/04/5415.html`.

Gadegaard, S., M. Ehrgott, and L. Nielsen (2016). Bi–objective branch–and–cut algorithms: Applications to the single source capacitated facility location problem. `http://www.optimization-online.org/DB_HTML/2016/04/5402.html`.

Gadegaard, S., A. Klose, and L. Nielsen (2016a). A bi–objective approach to discrete cost–bottleneck location problem. Submitted.

Gadegaard, S., A. Klose, and L. Nielsen (2016b). A bi–objective approach to discrete cost–bottleneck location problems. `https://github.com/SuneGadegaard/BOCBLP`. Source code (v1.0.0).

Gadegaard, S., A. Klose, and L. Nielsen (2016c). Implementations of a cut–and–solve algorithm for the single source capacitated facility location problem. `https://github.com/SuneGadegaard/SSCLPsolver`. Source code (v1.2.0).

Gadegaard, S., A. Klose, and L. Nielsen (2016d). An improved cut–and–solve algorithm for the single source capacitated facility location problem. Submitted.

Gadegaard, S., A. Klose, and L. Nielsen (2016e). Instances for facility location problems. `https://github.com/SuneGadegaard/Instances`. Source code.

Gadegaard, S., L. Nielsen, and M. Ehrgott (2016a). A branch and cut algorithm for bi–objective combinatorial optimization problems. `https://github.com/SuneGadegaard/BiObjectiveBranchAndCut`. source code (v0.0.1).

Gadegaard, S., L. Nielsen, and M. Ehrgott (2016b). A general two–phase method for bi–objective combinatorial optimization. `https://github.com/SuneGadegaard/TwoPhaseMethod`. Source code (v1.0.0).

Gadegaard, S., L. Nielsen, and M. Ehrgott (2016c). An instance generator for the capacitated facility location problem. `https://github.com/SuneGadegaard/SSCFLPgenerator`. Source code (v1.0.0).

Garfinkel, R. S. and M. Rao (1971). The bottleneck transportation problems. *Naval Research Logistics Quarterly 18*(4), 465–472.

Geoffrion, A. (1974). Lagrangean relaxation for integer programming. In *Approaches to Integer Programming*, Volume 2 of *Mathematical Programming Studies*, pp. 82–114. Springer Berlin Heidelberg.

Gu, Z., G. Nemhauser, and M. Savelsbergh (1998). Lifted Cover Inequalities for 0-1 Integer Programs: Computation. *INFORMS Journal on Computing 10*(4), 427–437.

Gu, Z., G. Nemhauser, and M. Savelsbergh (1999). Lifted cover inequalities for 0-1 integer programs: Complexity. *INFORMS Journal on Computing 11*(1), 117–123.

Guastaroba, G. and M. G. Speranza (2012). Kernel search for the capacitated facility location problem. *Journal of Heuristics 18*(6), 877–917.

Hakimi, S. (1964). Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research 12*(3), 450–459.

Hakimi, S. (1965). Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research 13*(3), 462–475.

Halpern, J. (1976). The location of a center-median convex combination on an undirected tree. *Journal of Regional Science 16*(2), 237–245.

Halpern, J. (1978). Finding minimal center-median convex combination (cent-dian) of a graph. *Management Science 24*(5), 535–544.

Hamacher, H., M. Labbé, and S. Nickel (1999). Multicriteria network location problems with sum objectives. *Networks 33*(2), 79–92.

Hamacher, H. and S. Nickel (1994). Combinatorial algorithms for some 1-facility median problems in the plane. *European Journal of Operational Research 79*(2), 340 – 351.

Hamacher, H. and S. Nickel (1996). Multicriteria planar location problems. *European Journal of Operational Research 94*(1), 66–86.

Hamacher, H., C. Pedersen, and S. Ruzika (2007). Finding representative systems for discrete bicriterion optimization problems. *Operations Research Letters 35*(3), 336 – 344.

Hammer, P. (1969). Time minimizing transportation problems. *Naval Research Logistics Quarterly 16*(3), 345–357.

Hansen, P. (1980). Bicriterion path problems. In G. Fandel and T. Gal (Eds.), *Multiple Criteria Decision Making Theory and Application*, Volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pp. 109–127. Springer Berlin Heidelberg.

Hansen, P., M. Labbé, and J. Thisse (1991). From the median to the generalized center. *RAIRO. Recherche opérationnelle 25*(1), 73–86.

Harkness, J. and C. ReVelle (2003). Facility location with increasing production costs. *European Journal of Operational Research 145*(1), 1 – 13.

Holmberg, K. (1994). Solving the staircase cost facility location problem with decomposition and piecewise linearization. *European Journal of Operational Research 75*(1), 41 – 61.

Holmberg, K., M. Rönnqvist, and D. Yuan (1999). An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research 113*(3), 544–559.

Jörnsten, K. and A. Klose (2015). An improved lagrangian relaxation and dual ascent approach to facility location problems. *Computational Management Science*, 1–32.

Kaparis, K. and A. Letchford (2010). Separation algorithms for 0-1 knapsack polytopes. *Mathematical Programming 124*(1-2), 69–91.

Kariv, O. and S. Hakimi (1979). An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics 37*(3), 513–538.

Karp, R. (1972). Reducibility among combinatorial problems. In R. Miller, J. Thatcher, and J. Bohlinger (Eds.), *Complexity of Computer Computations*, The IBM Research Symposia Series, pp. 85–103. Springer US.

Kiziltan, G. and E. Yucaoğlu (1983, December). An algorithm for multiobjective zero-one linear programming. *Management Science 29*(12), 1444–1453.

Klamroth, K. (2001). Planar weber location problems with line barriers. *Optimization 49*(5-6), 517–527.

Klein, D. and E. Hannan (1982). An algorithm for the multiple objective integer linear programming problem. *European Journal of Operational Research 9*(4), 378 – 385.

Klincewicz, J. and H. Luss (1986). A Lagrangian relaxation heuristic for capacitated facility location with single-source constraints. *Journal of the Operational Research Society 37*, 495–500.

Klose, A. and A. Drexl (2005). Facility location models for distribution system design. *European Journal of Operational Research 162*(1), 4–29.

Klose, A. and S. Görtz (2007). A branch-and-price algorithm for the capacitated facility location problem. *European Journal of Operational Research 179*(3), 1109–1125.

Launhardt, W. (1900). *The theory of the Trace: The principle of location.* Lawrence Asylum Press, Madras. Translated by Bewley, A. (1900).

Letchford, A. and S. Miller (2014). An aggressive reduction scheme for the simple plant location problem. *European Journal of Operational Research 234*(3), 674–682.

Manne, A. S. (1964). Plant location under economies-of-scale-decentralization and computation. *Management Science 11*(2), 213–235.

Martello, S., D. Pisinger, and P. Toth (1999). Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science 45*(3), 414–424.

Martello, S. and P. Toth (1990). *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons, Inc.

Martin, R. (1999). *Large scale linear and integer optimization: a unified approach.* Kluwer Academic Publishers.

Mavrotas, G. and D. Diakoulaki (1998). A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research 107*(3), 530–541.

Mavrotas, G. and D. Diakoulaki (2005). Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming. *Applied mathematics and computation 171*(1), 53–71.

Megiddo, N. and K. Supowit (1984). On the complexity of some common geometric location problems. *SIAM Journal on Computing 13*(1), 182–196.

Mehrez, A. and A. Stulman (1982). The maximal covering location problem with facility placement on the entire plane. *Journal of Regional Science 22*(3), 361–365.

Melachrinoudis, E. (1988). An efficient computational procedure for the rectilinear maximin location problem. *Transportation Science 22*(3), 217–223.

Minieka, E. (1970). The m-center problem. *Siam Review 12*(1), 138–139.

Monabbati, E. (2014). An application of a Lagrangian-type relaxation for the uncapacitated facility location problem. *Japan Journal of Industrial and Applied Mathematics 31*(3), 483–499.

Mulvey, J. and M. Beck (1984). Solving capacitated clustering problems. *European Journal of Operational Research 18*(3), 339 – 348.

Myung, Y., H. Kim, and D. Tcha (1997). A bi-objective uncapacitated facility location problem. *European journal of operational research 100*(3), 608–616.

Neebe, A. and M. Rao (1983). An algorithm for the fixed-charge assigning users to sources problem. *Journal of the Operational Research Society 34*, 1107–1113.

Nemhauser, G. and L. Wolsey (1988). *Integer and Combinatorial Optimization*. Wiley New York.

Nickel, S. and J. Puerto (1999). A unified approach to network location problems. *Networks 34*(4), 283–290.

Nickel, S., J. Puerto, and A. Rodríguez-Chía (2005). MCDM location problems. In J. Figueira, S. Greco, and M. Ehrgott (Eds.), *Multiple Criteria Decision Analysis: State of the Art Surveys*, International Series in Operations Research and Management Science, Chapter 19, pp. 761–795. Berlin, Heidelberg: Springer.

Nickel, S., J. Puerto, and A. Rodríguez-Chía (2015). Location problems with multiple criteria. In G. Laporte, S. Nickel, and F. Gama (Eds.), *Location Science*, pp. 205–248. Springer.

Ogryczak, W. (1997a). On cent-dians of general networks. *Location Science 5*(1), 15–28.

Ogryczak, W. (1997b). On the lexicographic minimax approach to location problems. *European Journal of Operational Research 100*(3), 566 – 585.

Pandian, P. and G. Nataraja (2011). A new method for solving bottleneck-cost transportation problems. *International Mathematical Forum, Vol. 6, 2011, no. 10, 451 - 460 6*(10), 451–460.

Pedersen, C., L. Nielsen, and K. Andersen (2008). The bicriterion multimodal assignment problem: Introduction, analysis, and experimental results. *INFORMS Journal on Computing 20*(3), 400–411.

Perez-Brito, D., J. Moreno-Perez, and I. Rodriguez-Martin (1997). Finite dominating set for the p-facility cent-dian network location problem. *Studies in Locational Analysis 11*, 27–40.

Pirkul, H. (1987). Efficient algorithms for the capacitated concentrator location problem. *Computers & Operations Research 14*(3), 197 – 208.

Pisinger, D. (1997, September). A Minimal Algorithm for the 0-1 Knapsack Problem. *Operations Research 45*(5), 758–767.

Przybylski, A., X. Gandibleux, and M. Ehrgott (2008). Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research 185*(2), 509 – 533.

Przybylski, A., X. Gandibleux, and M. Ehrgott (2010). A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization 7*(3), 149 – 165.

Przybylski, A., X. Gandibleuz, and M. Ehrgott (2011). The two phase method for multiobjective combinatorial optimization problems. In R. Mahjoub (Ed.), *Progress in Combinatorial Optimization*, pp. 559–595. ISTE Ltd and John Wiley Sons Inc.

Ralphs, T., M. Saltzman, and M. Wiecek (2006). An improved algorithm for solving biobjective integer programs. *Annals of Operations Research 147*(1), 43–70.

Ramos, R., S. Alonso, J. Sicilia, and C. González (1998). The problem of the optimal biobjective spanning tree. *European Journal of Operational Research 111*(3), 617 – 628.

Reese, J. (2006). Solution methods for the p-median problem: An annotated bibliography. *Networks 48*(3), 125–142.

Revelle, C., H. Eiselt, and M. Daskin (2008). A bibliography for some fundamental problem categories in discrete location science. *European Journal of Operational Research 184*(3), 817–848.

ReVelle, C. and R. Swain (1970). Central facilities location. *Geographical Analysis 2*(1), 30–42.

Rönnqvist, M., S. Tragantalerngsak, and J. Holt (1999). A repeated matching heuristic for the single-source capacitated facility location problem. *European Journal of Operational Research 116*(1), 51–68.

Ross, G. and R. Soland (1980). A multicriteria approach to the location of public facilities. *European Journal of Operational Research 4*(5), 307 – 321.

Schirra, S. (2008). How reliable are practical point-in-polygon strategies? In D. Halperin and K. Mehlhorn (Eds.), *Algorithms - ESA 2008*, Volume 5193 of *Lecture Notes in Computer Science*, pp. 744–755. Springer Berlin Heidelberg.

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

Sourd, F., O. Spanjaard, and P. Perny (2006). Multi-objective branch and bound. Application to the bi-objective spanning tree problem. In *7th International Conference in Multi-Objective Programming and Goal Programming*.

Sridharan, R. (1993). A Lagrangian heuristic for the capacitated plant location problem with single source constraints. *European Journal of Operational Research 66*(3), 305 – 312.

Stidsen, T., K. Andersen, and B. Dammann (2014). A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science 60*(4), 1009–1032.

Toregas, C., R. Swain, C. ReVelle, and L. Bergman (1971). The location of emergency service facilities. *Operations Research 19*(6), 1363–1373.

Ulungu, E. and J. Teghem (1997). Solving multi-objective knapsack problem by a branch-and-bound procedure. In C. J. (Ed.), *Multicriteria Analysis*, pp. 269–278. Springer Berlin Heidelberg.

Villegas, J., F. Palacios, and A. Medaglia (2006). Solution methods for the bi-objective (cost-coverage) unconstrained facility location problem with an illustrative example. *Annals of Operations Research 147*(1), 109–141.

Vincent, T. (2009). Multi-objective branch and bound for mixed 0-1 linear programming: Corrections and improvements. Master's thesis, Universität Kaiserslautern.

Vincent, T., F. Seipp, S. Ruzika, A. Przybylski, and X. Gandibleux (2013). Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case. *Computers & Operations Research 40*(1), 498–509.

Visée, M., J. Teghem, M. Pirlot, and E. Ulungu (1998). Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization 12*(2), 139–155.

Weber, A. (1922). *Über den Standort der Industrien.* Tübingen, J.C.B. Mohr.

Weismantel, R. (1997). On the 0/1 knapsack polytope. *Mathematical Programming 77*(3), 49–68.

Weiszfeld, E. (1937). Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tohoku Mathematical Journal 43*, 355–386.

Weiszfeld, E. and F. Plastria (2009). On the point for which the sum of the distances to n given points is minimum. *Annals of Operations Research 167*(1), 7–41.

Yang, Z., F. Chu, and H. Chen (2012). A cut-and-solve based algorithm for the single-source capacitated facility location problem. *European Journal of Operational Research 221*(3), 521–532.

DEPARTMENT OF ECONOMICS AND BUSINESS ECONOMICS

AARHUS UNIVERSITY
SCHOOL OF BUSINESS AND SOCIAL SCIENCES
www.econ.au.dk

PhD dissertations since 1 July 2011

2011-4 Anders Bredahl Kock: Forecasting and Oracle Efficient Econometrics

2011-5 Christian Bach: The Game of Risk

2011-6 Stefan Holst Bache: Quantile Regression: Three Econometric Studies

2011:12 Bisheng Du: Essays on Advance Demand Information, Prioritization and Real Options in Inventory Management

2011:13 Christian Gormsen Schmidt: Exploring the Barriers to Globalization

2011:16 Dewi Fitriasari: Analyses of Social and Environmental Reporting as a Practice of Accountability to Stakeholders

2011:22 Sanne Hiller: Essays on International Trade and Migration: Firm Behavior, Networks and Barriers to Trade

2012-1 Johannes Tang Kristensen: From Determinants of Low Birthweight to Factor-Based Macroeconomic Forecasting

2012-2 Karina Hjortshøj Kjeldsen: Routing and Scheduling in Liner Shipping

2012-3 Soheil Abginehchi: Essays on Inventory Control in Presence of Multiple Sourcing

2012-4 Zhenjiang Qin: Essays on Heterogeneous Beliefs, Public Information, and Asset Pricing

2012-5 Lasse Frisgaard Gunnersen: Income Redistribution Policies

2012-6 Miriam Wüst: Essays on early investments in child health

2012-7 Yukai Yang: Modelling Nonlinear Vector Economic Time Series

2012-8 Lene Kjærsgaard: Empirical Essays of Active Labor Market Policy on Employment

2012-9 Henrik Nørholm: Structured Retail Products and Return Predictability

2012-10 Signe Frederiksen: Empirical Essays on Placements in Outside Home Care

2012-11 Mateusz P. Dziubinski: Essays on Financial Econometrics and Derivatives Pricing

2012-12 Jens Riis Andersen: Option Games under Incomplete Information

2012-13 Margit Malmmose: The Role of Management Accounting in New Public Management Reforms: Implications in a Socio-Political Health Care Context

2012-14 Laurent Callot: Large Panels and High-dimensional VAR

2012-15 Christian Rix-Nielsen: Strategic Investment

2013-1 Kenneth Lykke Sørensen: Essays on Wage Determination

2013-2 Tue Rauff Lind Christensen: Network Design Problems with Piecewise Linear Cost Functions

2013-3 Dominyka Sakalauskaite: A Challenge for Experts: Auditors, Forensic Specialists and the Detection of Fraud

2013-4 Rune Bysted: Essays on Innovative Work Behavior

2013-5 Mikkel Nørlem Hermansen: Longer Human Lifespan and the Retirement Decision

2013-6 Jannie H.G. Kristoffersen: Empirical Essays on Economics of Education

2013-7 Mark Strøm Kristoffersen: Essays on Economic Policies over the Business Cycle

2013-8 Philipp Meinen: Essays on Firms in International Trade

2013-9 Cédric Gorinas: Essays on Marginalization and Integration of Immigrants and Young Criminals – A Labour Economics Perspective

2013-10 Ina Charlotte Jäkel: Product Quality, Trade Policy, and Voter Preferences: Essays on International Trade

2013-11 Anna Gerstrøm: World Disruption - How Bankers Reconstruct the Financial Crisis: Essays on Interpretation

2013-12 Paola Andrea Barrientos Quiroga: Essays on Development Economics

2013-13 Peter Bodnar: Essays on Warehouse Operations

2013-14 Rune Vammen Lesner: Essays on Determinants of Inequality

2013-15 Peter Arendorf Bache: Firms and International Trade

2013-16 Anders Laugesen: On Complementarities, Heterogeneous Firms, and International Trade

2013-17	Anders Bruun Jonassen: Regression Discontinuity Analyses of the Disincentive Effects of Increasing Social Assistance
2014-1	David Sloth Pedersen: A Journey into the Dark Arts of Quantitative Finance
2014-2	Martin Schultz-Nielsen: Optimal Corporate Investments and Capital Structure
2014-3	Lukas Bach: Routing and Scheduling Problems - Optimization using Exact and Heuristic Methods
2014-4	Tanja Groth: Regulatory impacts in relation to a renewable fuel CHP technology: A financial and socioeconomic analysis
2014-5	Niels Strange Hansen: Forecasting Based on Unobserved Variables
2014-6	Ritwik Banerjee: Economics of Misbehavior
2014-7	Christina Annette Gravert: Giving and Taking – Essays in Experimental Economics
2014-8	Astrid Hanghøj: Papers in purchasing and supply management: A capability-based perspective
2014-9	Nima Nonejad: Essays in Applied Bayesian Particle and Markov Chain Monte Carlo Techniques in Time Series Econometrics
2014-10	Tine L. Mundbjerg Eriksen: Essays on Bullying: an Economist's Perspective
2014-11	Sashka Dimova: Essays on Job Search Assistance
2014-12	Rasmus Tangsgaard Varneskov: Econometric Analysis of Volatility in Financial Additive Noise Models
2015-1	Anne Floor Brix: Estimation of Continuous Time Models Driven by Lévy Processes
2015-2	Kasper Vinther Olesen: Realizing Conditional Distributions and Coherence Across Financial Asset Classes
2015-3	Manuel Sebastian Lukas: Estimation and Model Specification for Econometric Forecasting
2015-4	Sofie Theilade Nyland Brodersen: Essays on Job Search Assistance and Labor Market Outcomes
2015-5	Jesper Nydam Wulff: Empirical Research in Foreign Market Entry Mode

2015-6 Sanni Nørgaard Breining: The Sibling Relationship Dynamics and Spillovers

2015-7 Marie Herly: Empirical Studies of Earnings Quality

2015-8 Stine Ludvig Bech: The Relationship between Caseworkers and Unemployed Workers

2015-9 Kaleb Girma Abreha: Empirical Essays on Heterogeneous Firms and International Trade

2015-10 Jeanne Andersen: Modelling and Optimisation of Renewable Energy Systems

2015-11 Rasmus Landersø: Essays in the Economics of Crime

2015-12 Juan Carlos Parra-Alvarez: Solution Methods and Inference in Continuous-Time Dynamic Equilibrium Economies (with Applications in Asset Pricing and Income Fluctuation Models)

2015-13 Sakshi Girdhar: The Internationalization of Big Accounting Firms and the Implications on their Practices and Structures: An Institutional Analysis

2015-14 Wenjing Wang: Corporate Innovation, R&D Personnel and External Knowledge Utilization

2015-15 Lene Gilje Justesen: Empirical Banking

2015-16 Jonas Maibom: Structural and Empirical Analysis of the Labour Market

2015-17 Sylvanus Kwaku Afesorgbor: Essays on International Economics and Development

2015-18 Orimar Sauri: Lévy Semistationary Models with Applications in Energy Markets

2015-19 Kristine Vasiljeva: Essays on Immigration in a Generous Welfare State

2015-20 Jonas Nygaard Eriksen: Business Cycles and Expected Returns

2015-21 Simon Juul Hviid: Dynamic Models of the Housing Market

2016-1 Silvia Migali: Essays on International Migration: Institutions, Skill Recognition, and the Welfare State

2016-2 Lorenzo Boldrini: Essays on Forecasting with Linear State-Space Systems

2016-3 Palle Sørensen: Financial Frictions, Price Rigidities, and the Business Cycle

2016-4 Camilla Pisani: Volatility and Correlation in Financial Markets: Theoretical Developments and Numerical Analysis

2016-5 Anders Kronborg: Methods and Applications to DSGE Models

2016-6 Morten Visby Krægpøth: Empirical Studies in Economics of Education

2016-7 Anne Odile Peschel: Essays on Implicit Information Processing at the Point of Sale: Evidence from Experiments and Scanner Data Analysis

2016-8 Girum Dagnachew Abate: Essays in Spatial Econometrics

2016-9 Kai Rehwald: Essays in Public Policy Evaluation

2016-10 Reza Pourmoayed: Optimization Methods in a Stochastic Production Environment

2016-11 Sune Lauth Gadegaard: Discrete Location Problems – Theory, Algorithms, and Extensions to Multiple Objectives

ISBN: 9788793195424