FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Two-Phase Local Search for the Bi-objective Connected Facility Location Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Thomas Petelin

Matrikelnummer 0525199

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Dipl.-Ing. Dr.techn. Markus Leitner

Wien, 04.12.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Thomas Petelin
Stettnerweg 20, 2100 Korneuburg

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Danksagung

An dieser Stelle möchte ich mich recht herzlich bei ein paar Personen bedanken, die mich während meines Studiums und vor allem während meiner Masterarbeit unterstützt haben.

Bei Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl bedanke ich mich für die Genehmigung des Themas und die Betreuung während der Masterarbeit.

Bei Dipl.-Ing. Dr.techn. Markus Leitner bedanke ich mich für die kompetente fachliche Unterstützung in den verschiedenen Arbeitsphasen und dafür, dass er mir immer mit Rat und Tat zur Seite gestanden hat.

Bei meinen Eltern bedanke ich mich für sämtliche Unterstützungen während meines Studiums.

Abstract

In this thesis a two-phase local search based metaheuristic algorithm for the Bi-objective Connected Facility Location Problem (BoConFL) is presented.

The Connected Facility Location Problem (ConFL) is an NP-hard combinatorial optimization problem that has been recently proposed to model the extension of fiber-optic-networks using so-called fiber-to-the-curb (FTTc) strategies. In FTTc scenarios telecommunication providers aim to extend their fiber-optic networks to mediation points (facilities) that bridge between fiber-optic and copper technology. Customers are finally connected to facilities using the previously existing copper network. Thus, the bandwidth of customers can be significantly increased with less investment costs compared to connecting each customer using fiber-optics, i.e., compared to fiber-to-the-home scenarios.

The main drawbacks of previously considered variants of ConFL are that they either aim to mandatorily connect all potential customers or that they simply optimize the difference between the revenue obtained by connecting a subset of customers and the resulting network construction costs. In many realistic settings, however, customer „revenues“ may be given e.g. by means of demands rather than in monetary units. Thus, simply maximizing the previously mentioned difference is not meaningful. Hence, the Bi-objective Connected Facility Location Problem (BoConFL) which is addressed in this thesis aims to simultaneously maximize the collected revenue and minimize the resulting costs for constructing the network. In many relevant scenarios, the addition of a second objective function provides a better representation of the real world since these two objectives are conflicting, rather than finding a single optimal solution in the BoConFL we are interested in identifying the set of non-dominated, i.e., Pareto-optimal, solutions.

Based on previous work for single-objective variants of the problem and successful approaches for different bi-objective combinatorial optimization problem a two-phase algorithm is developed in order to get a good approximation of the Pareto front with the following two main steps:

a) Construction of a set of promising solutions by aggregation of the two objectives to a single one. Variable neighborhood descent is used to further improve the obtained set of initial solution candidates.

b) Application of a Pareto local search algorithm that takes both objectives explicitly into account to further improve the quality of the solution set.

The influence of the algorithms components and parameters on the runtime and the quality of the obtained approximation is analyzed using a computational study.

Kurzfassung

In dieser Arbeit wird ein auf lokaler Suche basierender zwei phasiger metaheuristischer Algorithms für das Bi-objective Connected Facility Location Problem (BoConFL) präsentiert.

Das Connected Facility Location (ConFL) Problem ist ein NP hartes kombinatorisches Optimierungsproblem, welches erst kürzlich als Modell für die Erweiterung von Glasfaserkabelnetzwerken zu sogenannten Fiber-to-the-Curb (FTTc) Strategien vorgeschlagen wurde. Bei solchen FTTc Szenarien versuchen Telekommunikationsanbieter ihr Glasfasernetzwerk zu sogenannten Vermittlungspunkten (Facilities) zu erweitern, welche den Übergang von Glasfaser zu Kupfer handhaben. Dadurch wird dem Kunden deutlich mehr Bandbreite geboten und zusätlich werden die Ausbaukosten geringer gehalten, als würde jeder Kunde einen direkten Glasfaseranschluss bekommen.

Der größte Nachteil in den bisher präsentierten Varianten des ConFL Problems liegt darin, dass entweder versucht wird jeden Kunden mit einer Facility zu verbinden oder es wird einfach versucht die Differenz zwischen Ausbaukosten und Gewinn zu minimieren indem nur ein bestimmter Teil aller möglichen Kunden angebunden wird. In vielen realistischen Szenarien kann der „Ertrag“ eines Kunden eher durch dessen Anforderung definiert werden als über erhaltenen Gewinn. Dadurch ist die einfache Maximierung von Gewinn minus Ausbaukosten nicht aussagekräftig. Auf Grund dessen haben wir versucht das BoConFL Problem zu lösen, mit dem Ziel den Kundennutzen zu maximieren während wir die Ausbaukosten minimieren. Da die Erweiterung des ConFL Problems um eine weitere Zielfunktion eine bessere Abbildung der Realität darstellt, weil sich die beiden Zielfunktionen widersprechen, haben wir versucht eine Menge von sich nicht gegenseitig dominierenden (Pareto-optimale) Lösungen zu suchen anstatt einer einzelnen optimierten Lösung.

Basierend auf bisherigen Arbeiten zu dem normalen, single-objective ConFL Problem und diversen erfolgreichen Ansätzen für bi-objective kombinatorische Optimierungsprobleme werden wir einen zwei-Phasen Algorithmus entwickeln, welcher versucht die Paretofront so gut als möglich anzunähern. Dieser Algorithmus besteht aus folgenden zwei Schritten:

a) Konstruktion einer Menge von vielversprechenden Lösungen durch Aggregation der beiden Zielfunktionen zu einer einzigen, unter Verwendung von Gewichten. Weiters verwenden wir eine variable Nachbarschaftssuche um die initialen Lösungen nochmals zu verbessern.

b) Anwendung einer Pareto-lokalen Suche, welche beide Zielfunktionen berücksichtigt, um die Lösungen weiter zu verbessern.

Um den Einfluss der Komponenten des Algorithmus und seiner Parameter auf die Laufzeit und die Qualität der berechneten Lösungen zu analysieren, werden wir den Algorithmus auf ein Set verschiedener Testinstanzen anwenden.

Contents

CHAPTER 1

Introduction

With the increasing availability of video-on-demand and other internet based products customers need access to higher bandwidth internet connections. Thus the telecommunication industry needs to extend their existing networks. Among other strategies so-called fiber-to-the-home (FTTh), fiber-to-the-building (FTTb), or fiber-to-the-curb (FTTc) networks are considered as particularly relevant by telecommunication providers to increase the bandwidth available to customers.

Starting with copper only networks telecommunication providers soon extended their networks by replacing the connections between their main distribution centers by fiber-optic cables that are capable of handling a higher traffic volume. Due to the increased availability of different services on the internet, customers needed a better connection to their service providers. At first only the main hubs of telecommunication providers were connected with fiber optic cables which improved the performance significantly. Still, however, the end customers were not satisfied. To solve this problem the fiber optic networks should be brought closer to the consumer. In FTTc scenarios telecommunication providers aim to extend their fiber-optic networks to mediation points (facilities) that bridge between fiber-optic and copper technology. Customers are finally connected to facilities using the previously existing copper network. If the length of the copper path between a customer and the facility is not too long, the bandwidth of customers can be significantly increased. The associated investment costs are, however, typically significantly smaller than connecting each customer using fiber-optics, i.e., compared to fiber-to-the-home scenarios where the customer would be directly connected to the fiber-optic network of the provider.

To model such extensions of fiber-optic-networks the Connected Facility Location Problem (ConFL) has been proposed recently. The main drawbacks of previously proposed variants of ConFL are that they either aim to mandatorily connect all potential customers or that they simply optimize the difference between the revenue obtained by connecting a subset of customer and the resulting network construction costs. In many realistic settings, however, customer „revenues“

may be given, e.g., by means of demands rather than in monetary units. Thus, simply maximizing the previously mentioned difference is not meaningful. In all previous attempts these two conflicting objectives were merged and solved as a single objective problem. However the addition of a second objective function provides a better representation of the real world. As these two objectives are conflicting, rather than finding a single optimal solution in the BoConFL we are interested in identifying the set of non-dominated, i.e., Pareto-optimal, solutions. Hence, the Bi-objective Connected Facility Location Problem (BoConFL) which is addressed in this thesis aims to simultaneously maximize the collected revenue and minimize the resulting costs for constructing the network.

1.1 Aim of the work

In this thesis different existing methods to solve the single objective ConFL and variants thereof are presented. Furthermore an overview of approaches to solve bi- and multi-objective problems, either with exact methods or heuristics is given. Our aim was to develop a metaheuristic algorithm based on local search for the bi-objective connected facility location problem that is capable of finding a good approximation of the set of non-dominated solutions. Focusing on a local search based approach was somehow natural, since this technique has been successfully applied to single objective problems with similar characteristics. Furthermore the bi-/multi-objective ConFL was not tackled before by a local search based algorithm.
Finally, by a computational study, the influence of the components and parameters on the runtime and the quality of the obtained approximations of the developed approach is investigated.

1.2 Outline of the Thesis

In Chapter 2 the ConFL and the bi-objective ConFL which is tackled in this thesis are formally defined. Chapter 3 gives an overview of common approaches to solve the single objective combinatorial optimization problems, including exact algorithms, heuristics as well as genetic algorithms. Chapter 4 which gives an overview of the most commonly used approaches in the literature to conquer the single objective ConFL and briefly summarizes relevant algorithms for bi- and multi-objective problems. Chapter 5 describes the algorithm developed in this thesis and discusses the implementation. In the next Chapter the results obtained from a computational study are discussed. Finally, in Chapter 6 some conclusions are drawn and ideas for future work based on this thesis are presented.

CHAPTER 2

Problem Definition

The Connected Facility Location Problem is a generalization of the Facility Location Problem and the Steiner Tree Problem and defined as follows:

Definition 2.1.1 (Connected Facility Location Problem)**.** *We are given a graph $G = (V, E)$ where the node set is the disjoint union of customer nodes $R \subseteq V$, facility nodes $F \subseteq V$ and potential Steiner nodes $T = V \setminus (F \cup R)$. The set of edges E is the disjoint union of core edges $E_C = \{\{i, j\} \in E : i, j \in F \cap T\}$ and assignment edges $E_A = \{\{i, j\} \in E : i \in F, j \in R\}$. For all core edges $e \in E_C$ we are given an edge cost $c_e \geq 0$, for all facilities $i \in F$ we are given facility opening costs $f_i \geq 0$, and each edge $e \in E_A$ we are given costs $a_{ij} \geq 0$ for assigning customer $j \in R$ to facility $i \in F$.*

A solution to the ConFL consists of a set of open facilities $F' \subseteq F$, a set of chosen assignment edges $E'_A \subseteq E_A$ such that each customer $j \in R$ is connected to exactly one open facility $i(j) \in F'$ and a Steiner tree (V', E'_C) which is a subgraph of $(F \cup T, E_C)$.

The objective value of a solution is given by the sum of the opening costs, the costs of assigning customers to open facilities and the edge costs of the Steiner tree connection the open facilities, i.e.

$$\min \sum_{i \in F'} f_i + \sum_{j \in R} a_{i(j)j} + \sum_{e \in E'_C} c_e, \tag{2.1}$$

and the overall objective is to identify a solution with minimal costs. Note that in this formulation closed facility nodes can also be used as pure Steiner nodes.

The bi-objective ConFL can be modeled equivalently with the addition that a potential revenue is assigned to each customer and not every customer has to be assigned to a facility in a feasible solution.

Definition 2.1.2 (Bi-objective Connected Facility Location Problem)**.** *We are given a graph $G = (V, E)$ where the node set is the disjoint union of customer nodes $R \subseteq V$, facility nodes*

$F \subseteq V$ and potential Steiner nodes $T = V \setminus (F \cup R)$. Further we are given a root node $b \in (F \cup T)$ which can either be a facility (open or closed) or a Steiner node. The set of edges E is a disjoint union of core edges $E_C = \{\{i,j\} \in E : i,j \in F \cap T\}$ and assignment edges $E_A = \{\{i,j\} \in E : i \in F, j \in R\}$. For all core edges $e \in E_C$ we are given an edge cost $c_e \geq 0$, for all facilities $i \in F$ we are given facility opening costs $f_i \geq 0$. Additionally a potential revenue $r_k > 0$ is assigned for each customer $k \in R$. For each edge $e \in E_A$ we are given the costs of assigning a customer $j \in R$ to a facility $i \in F$ as $a_{ij} \geq 0$.

A solution $S = ((V', E'_C), F', R', E'_A)$ to the BoConFL consists of a set of open facilities $F' \subset F$, a set of chosen assignment edges $E'_A \subseteq E_A$ such that each selected customer $j \in R'$, $R' \subseteq R$, is connected to exactly one open facility $i(j) \in F'$ and a Steiner tree (V', E'_C) which is a subgraph of $(F \cup T, E_C)$ where $F' \subseteq V'$.

In the BoConFL we have not one but two objective values for each solution, the cost $z_1(S)$ which is given by the sum of the opening costs, the costs of assigning customers to open facilities and the edge costs of the Steiner tree connection the open facilities and the revenue $z_2(S)$ which is the sum of the revenue of each selected customer.

In the BoConFL we want to simultaneously minimize the total costs $z_1(S)$ and maximize the collected revenue $z_2(S)$ of a solution S, i.e.,

$$z_1(S) = \sum_{i \in F'} f_i + \sum_{j \in R'} a_{i(j)j} + \sum_{e \in E'_C} c_e, \tag{2.2}$$

$$z_2(S) = \sum_{k \in R'} r_k. \tag{2.3}$$

The objective is to identify the set of Pareto optimal solutions, i.e., the set of feasible solutions for which one cannot find another feasible solution that is better w.r.t. to one of the objectives without deteriorating the other objective.

In order to create a minimization problem, instead of calculating the revenue which would be maximized we calculate the lost revenue $z'_2(S)$ which needs to be minimized. This results in the following two objective functions:

$$z_1(S) = \sum_{i \in F'} f_i + \sum_{j \in R'} a_{i(j)j} + \sum_{e \in E'_C} c_e, \tag{2.4}$$

$$z'_2(S) = \sum_{k \in R \setminus R'} r_k. \tag{2.5}$$

CHAPTER 3

Methodologies

The following chapter will provide an introduction to exact and heuristic approaches for solving combinatorial optimization problems (COPs) Furthermore an overview of methods used to conquer bi- and multi-objective problems is given. Finally Section 3.3 introduces basic definitions and terminology of multi-objective optimization.

Exact methods usually take a form of branching and other forms of exhaustive search in order to find optimal solutions for given problems if given enough time. Due to their behavior these methods tend to find good if not exact solutions but the drawback is the long runtime because of the large solution space which needs to be searched. The most common and widely used approach for COPs is mixed integer linear programming.

3.1 Integer Linear programming

This topic was first studied by Kantorovich [23] and gained military interest during the second world war. A few years later, independent of Kantorovich, Dantzig [9] formalized linear programming and published the well known simplex algorithm to solve linear programs (LPs). Von Neumann [50] proposed the theory of duality of LPs stating that every minimization problem has an equivalent maximization problem and vice versa.

Formulation (3.1)-(3.3) is the standard for a linear program (LP),

$$\text{min. } \mathbf{c}^T\mathbf{x} \tag{3.1}$$

$$\text{s.t. } \mathbf{A}\mathbf{x} \geq \mathbf{b} \tag{3.2}$$

$$\mathbf{x} \geq \mathbf{0} \tag{3.3}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the vector of variables which needs to be determined, $\mathbf{c} \in \mathbb{R}^n$ is the cost vector, $\mathbf{b} \in \mathbb{R}^m$ is a coefficient vector and matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a matrix of coefficients. $\mathbf{c}^T$

denotes the transposed vector of $\mathbf{c}$. The objective function (3.1) is to be optimized regarding to the given criterion. The inequality (3.2) is the constraint which specify a convex polytope over which the objective function will be optimized. The domain of the variables (3.3) defines the value range of $\mathbf{x}$.

The most well known algorithms to solve linear programs are the Simplex algorithm by Dantzig [8] and the Interior point method [25, 47] as well as adaptations of them.

Some problems are formulated as maximization problems so these need to be transformed to minimization problems by application of the theorem of von Neumann (3.1) - (3.3):

$$\text{max. } \mathbf{b}^T\mathbf{y} \tag{3.4}$$

$$\text{s.t. } \mathbf{A}^T\mathbf{y} \geq \mathbf{c} \tag{3.5}$$

$$\mathbf{y} \in \mathbb{R}^n \tag{3.6}$$

This transformation can also be used in the other direction, from minimization to maximization problems.

Based on the domain is the variable (3.3) we differentiate between the following four forms of LPs: The domain of $\mathbf{x}$ need not to be restricted to $\mathbb{R}$. Due to a more restricted domain of $\mathbf{x}$ we can differentiate some special forms:

- **Integer Linear Program** (ILP): $\mathbf{x} \in \mathbb{Z}^n$
- **Binary Integer Program** (BIP): $\mathbf{x} \in \{0, 1\}^n$
- **Mixed Integer Program** (MIP): $\mathbf{x} \geq 0, \mathbf{x}_i \in \mathbb{Z}, i \in S \subset \{1, ..., n\}$

In order to solve ILPs, BIPs or MIPs some advanced algorithms that implement the simplex algorithm are used, e.g., branch and bound method [29], branch and cut algorithm [41]. This is done by relaxing the integer condition of variables from $x \in \mathbb{N}$ to a continuous value range $x \in \mathbb{R}$. Due to this relaxation of a constraint the solution obtained might not be integral anymore. This fact can be used to get a lower bound of the relaxed minimization problem as well as an upper bound by solving the dual maximization problem or vice versa. The Branch and Bound algorithm is used to solve such relaxation problems. It uses a divide and conquer approach to break the problem into smaller and easier subproblems which can be solved independently. If a solution to a relaxed problem is found with one or more variables being a fractal by using binary branching, e.g.: a solution of a ILP with $x = 3.5$ will be branched to two new subproblems each having an additional constraint $x \leq 3$ and the other $x \geq 4$.

After a new solution is obtained, it is checked if it worse than the best upper / lower bound already found the node is pruned and all resulting subproblems as well. The branch and bound procedure is finished when the lower and and the upper bound respectively the the primal and dual solution are equal and integral.

3.2 Metaheuristic Approaches

Metaheuristics often try to optimize a solution for a problem by iteratively trying to improve a solution with regard to a specific measure of quality with no assumption about the optimality of the solution. They can search a very large set of candidate solutions but they do not guarantee that an optimal solution will be found. In the following subsections, which follows the lecture slides of Heuristic Optimization Techniques, an overview of the most common metaheuristic algorithms is given.

3.2.1 Greedy Heuristic

A greedy heuristic (also called a construction heuristic) is typically used to create an initial solution. Due to their nature they usually fail to find optimal solutions and do not derive any information on the distance to the optimum. Usually they make a decision with no concern if this was good on the long run, thus they are called short sighted. They are typically fast and often find good approximations of the optimal solution. Algorithm 1 shows a pseudo code implementation of a greedy heuristic. Starting from an empty solution and continuously adding the cheapest item to the solution until it has discovered a complete solution.

Algorithm 1 Greedy construction heuristic

```
1: $x \leftarrow$ empty solution
2: **while** $x$ is no complete solution **do**
3:     $e \leftarrow$ current cheapest extension for $x$
4:     $x \leftarrow x \oplus e$
5: **end while**
```

3.2.2 Greedy Randomized Adaptive Search Procedure

A Greedy Randomized Adaptive Search Procedure (GRASP), introduced by Feo and Resende [12], is also used as a construction heuristic like the greedy heuristic. As opposed to the latter, a GRASP is usually used to generate a set of independent initial solutions which are needed, e.g., population based metaheuristics.

As a greedy heuristic, GRASP, starts with an empty solution. Rather than just adding the next best element to the solution it evaluates all existing solution candidates that are extensions to the current solution. These candidates are added to a Candidate List (CL) and a subset of these, which are most likely the better ones, are added to the Restricted Candidate List (RCL). How many of the solutions are added to the RCL determines the randomness of the GRASP. If the size of the RCL is one, then the GRASP would resemble a deterministic greedy algorithm as described in Section 3.2.1. Next the algorithm selects a random element from the RCL and extends the actual solution by the selected candidate. This process is repeated until a complete solution has been generated.

Algorithm 2 details a single iteration of GRASP, e.g. the construction of a single solutions.

Algorithm 2 Greedy Randomized Adaptive Search Procedure

```
1: $x \leftarrow$ empty solution
2: while $x$ is no complete solution do
3:     CL $\leftarrow$ all possible extension of $x$
4:     RCL $\leftarrow$ promising subset of CL
5:     $e \leftarrow$ random element of RCL
6:     $x \leftarrow x \oplus e$
7: end while
```

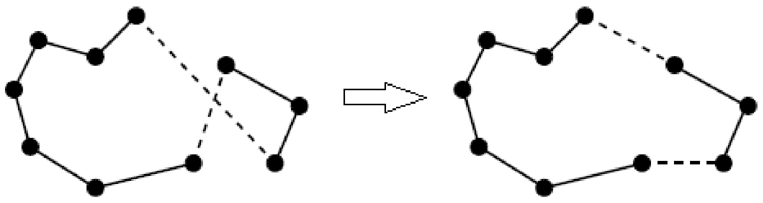

Figure 3.1: Example of a move within the 2-exchange neighborhood on the TSP. Two edges are selected and their endpoints are exchanged in order to check if the solution derived is better than the initial one.

3.2.3 Local Search

The Local Search (LS) algorithm (see Algorithm 3) tries to find an optimal solution by iteratively improving candidate solutions within a given neighborhood. The main parts of a LS are the definition of a solution representation, the generation of an initial solution, a selection of the neighborhood structure to use and of a step function and finally a stopping criterion.

Algorithm 3 Local Search

```
1: $x \leftarrow$ initial solution
2: repeat
3:     select an $x' \in N(x)$
4:     if $f(x') \leq f(x)$ then
5:         $x \leftarrow x'$
6:     end if
7: until termination criterion met
```

A neighborhood structure is a function $N : S \rightarrow 2^S$ that assigns to each solution $x \in S$ a set of neighbors $N(x) \subseteq S$. $N(x)$ is often called the neighborhood of x. Usually a neighborhood is defined by a set of possible moves. For the symmetric Traveling Salesman Problem (TSP), e.g., the k-exchange neighborhood defines a neighborhood structure in which each tour differs from the initial tour by a maximum of k edges (see Figure 3.1 for an exemplary 2-exchange move).

As one can imagine from the example the number of neighbors can be large. There exist three common strategies (step functions) to select neighboring solutions (cf. Step 3 of Algorithm

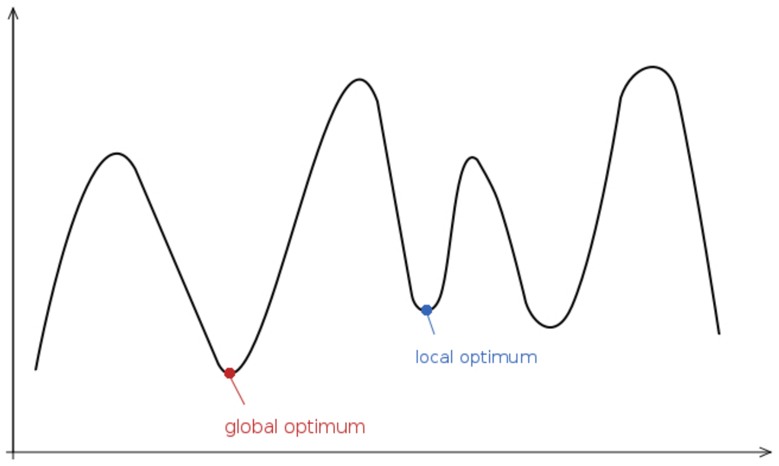

Figure 3.2: Difference between local and global optimum

3):

- **Random Improvement:** use a random neighboring solution from $N(x)$.
- **Next Improvement:** search $N(x)$ in a fixed order and take the first neighboring solution that is better than x.
- **Best Improvement:** search all $N(x)$ and take the best neighboring solution x.

The main drawback of the basic LS algorithm is that it only finds a local optimum x in the neighborhood structure N, i.e., a solution x such that $f(x) \leq f(x'), \forall x' \in N(x)$. As also shown in Figure 3.2 such a local optimum may not be a global optimum, i.e., there may exist a solution x'' with $f(x'') \leq f(x)$.

To overcome that problem the following extensions of the basic LS algorithm can be used.

3.2.4 Variable Neighborhood Descent

The Variable Neighborhood Descent method [19] can be used to overcome the problem of the basic local search (LS). The disadvantage of the LS which only uses a single neighborhood is that one might find a local optima in that specific neighborhood but which probably not a global optima for the whole solution space. They used the fact that a local optima of one neighborhood structure is not necessarily an optima of another as well as that a global optima is a local optima for each possible neighborhood structures and that for most problems the local optima lie relatively close together.

A VND iterates deterministically over a set of neighborhood structures $\mathcal{N}_1, ..., \mathcal{N}_{l_{\max}}$ with the step function usually being next or best improvement (see Algorithm 4). The solution that is obtained by a VND will be optimal with respect to all neighborhood structures but still need not be a global optimum. In VND, the neighborhood structures are usually ordered either by ascending size or complexity.

Algorithm 4 Variable Neighborhood Descent (x)

```
1:  $x \leftarrow$initial Solution
2:  $l \leftarrow 1$
3:  repeat
4:      find an $x'$ satisfying $f(x') \leq f(x''), \forall x'' \in \mathcal{N}_l(x)$
5:      if $f(x') < f(x)$ then
6:          $x \leftarrow x'$;
7:          $l \leftarrow 1$
8:      else
9:          $l \leftarrow l + 1$
10:     end if
11: until $l > l_{\max}$
12: return x
```

3.2.5 Variable Neighborhood Search

Hansen and Mladenovic [42] proposed the Variable Neighborhood Search (VNS) which utilizes a method which is called *shaking* and improve this solution by applying LS. Shaking is a process which takes a solution and modifies it to resemble another solution which might not be achieved by only applying local search because it does not follow any improvement strategy. It applies k random moves in the given neighborhood. The greater the value of k the more the new solution diverges from the initial one. It is basically a means to escape local optima by applying a number of random moves in one or different neighborhoods and to avoid cycling which might occur when applying deterministic rules.

The basic pseudo code for a VNS can be seen in Algorithm 5, where N_l describes the l^{th} neighborhood with $l \in 1, ..., l_{\max}$. Before each local search application shaking is performed in order to alter the existing solution more or less, depending on k. The local search performed in line 5 is the same as described in Section 3.2.3 with the extension that the N_l defines the neighborhood structure in which the local search will be searching. In case a better solution is found the neighborhood is reset back to N_1 else shaking and local search is applied in the next neighborhood until the stopping criterion is reached.

There exist some different VNS variants. The Basic Variable Neighborhood Search (BVNS) [5] is the the basic variant described above (see Algorithm 5). A simplified variant is the Reduced Variable Neighborhood Search (RVNS) [20], where the LS (Step 5 from Algorithm 5) is removed. For the General Variable Neighborhood Search (GVNS) [5] the shaking and the LS steps of the BVNS are removed and instead a VND is performed. Another variant is the Variable Neighborhood Decomposition Search (VNDS) [20] which is also an extension of the BVNS where instead of a normal LS a specialized version of LS applied where all elements which are in solution x and x' are fixed and the optimization only considers elements which are different.

Algorithm 5 Basic Variable Neighborhood Search (x, k)

```
1: repeat
2:    $l \leftarrow 1$
3:    repeat
4:       $x' \leftarrow$ generate random neighbor in $N(x)$
5:       $x' \leftarrow \mathrm{localSearch}(x', N_l)$
6:       if $f(x') \leq f(x)$ then
7:          $x \leftarrow x'$;
8:          $l \leftarrow 1$
9:       else
10:         $l \leftarrow l + 1$
11:      end if
12:   until $l > l_{\max}$
13: until termination criterion met
14: return x
```

3.2.6 Tabu Search

Tabu Search was first presented by Hansen [18] and later described by Glover [14]. Its idea is based on a memory (called Tabu list (TL)) which keeps track of the course of optimization and uses this knowledge to escape local optima. To avoid cycles already or recently visited solutions are temporarily forbidden and can not be visited again for a certain period. Tabu search is mostly guided deterministically and in most cases a best neighbor step function is used. Algorithm 6 details its basic principles. As shown on line 4 in each iteration all possible neighbors of x which are not prohibited due to the tabu list TL are considered. Then, based on X', the best possible solution is obtained. This solution is then added to TL and the oldest solution is removed from the tabu list.

Algorithm 6 Tabu Search

```
1: $TL \leftarrow \varnothing$
2: $x \leftarrow$ initial solution
3: repeat
4:    $X' \leftarrow$ subset of $N(x)$ considering $TL$
5:    $x' \leftarrow$ best solution of $X'$
6:    $TL \leftarrow TL \oplus x'$
7:    delete elements from $TL$ which are older than $t_L$
8:    if $f(x') \leq f(x)$ then
9:       $x \leftarrow x'$;
10:   end if
11: until termination criterion met
```

Typically one either stores whole solutions or the relevant attributes of visited solutions. In a tabu search approach for the TSP based on a two-exchange neighborhood one could, e.g., simply store a current move and forbids to reverse it for the next t_L iterations with t_L being the length of the TL. In practice different tabu lists for different attributes have shown to be useful. Also parameter t_L is very important because it decides how long a move or a solution is restricted which influences the direction in which the solution is optimized in the next iterations. Optimal t_L values have to be evaluated experimentally.

Battiti and Tecchiolli [4] proposed Reactive Tabu Search which is a variant of the Tabu Search mentioned above which was originally developed for the 0/1-knapsack problem and used an adaptive tabu list length and a diversification strategy to get better solutions.

3.2.7 Population based Approaches

Contrary to the methods described before, population based methods maintain a whole set of solutions in a population. These solutions evolve, might merge and exchange information to generate new solutions which possibly replace other solutions from a previous generation. In the following subsections three common population based metaheuristics are described.

3.2.7.1 Evolutionary Algorithms

In contrast to previously mentioned methods Evolutionary Algorithms (EA) work on a set of candidate solutions, called the *population*, instead of just on one single solution. They follow a principle that is easily applicable for various problems from combinatorial optimization to continuous parameter optimization as well as optimization of non-linear structures.

Algorithm 7 shows a basic evolutionary algorithm. Various adaptions of this EA have been proposed in the literature and some of them presented in the following subsections.

Algorithm 7 Evolutionary algorithm

```
1: $P \leftarrow$ set of initial solutions
2: $\mathrm{evaluate}(P)$
3: **repeat**
4:     $Q \leftarrow \mathrm{generateNewSolutionsByVariation}(P)$
5:     $\mathrm{evaluate}(Q)$
6:     $P \leftarrow \mathrm{selectBetterSolutionsFrom}(P, Q)$
7: **until** termination criterion met
```

3.2.7.2 Genetic Algorithms

The idea for genetic algorithms (GA) came from J. H. Holland and was later adapted by D. E. Goldberg [15]. A GA is a special search heuristic which is based upon the process of natural evolution. The algorithm works not only on a single solution but on a set of candidate solutions and every individual can be mutated and altered in the evolutionary process. In each iteration some individuals will be selected for reproduction, based on a fitness value, of which two or more parents will create offspring. Next the new individuals can mutate to vary a bit from their parents and in the end of an iteration the offspring will replace the original population, however some individuals of the original population may survive as well, if they have proven worthwhile to keep, which depends on the evolution strategy used. The structure of a GA is shown by Algorithm 8.

Algorithm 8 Genetic algorithm

```
1: $t \leftarrow 0$
2: Initialization($P(t)$)
3: Evaluation($P(t)$)
4: while not termination-condition do
5:     $t \leftarrow t + 1$
6:     $Q_s(t) \leftarrow$ Selection($P(t-1)$)
7:     $Q_r(t) \leftarrow$ Recombination($Q_s(t)$)
8:     $Q_m(t) \leftarrow$ Mutation($Q_r(t)$)
9:     $P(t) \leftarrow$ Replacement($P(t-1)$, $Q_m(t)$)
10:    Evaluation($P(t)$)
11: end while
```

In a GA each solution has to be represented in a genetic form (called the *chromosome* or *genotype*) that contains all the properties of a candidate solution. A fitness function $f(i)$ is needed to evaluate a solution. Usually a high fitness value equals a good solution and a low fitness value a bad solution. Each iteration of the algorithm produces a new *generation* of the population P.

Initialization: Initial solutions can either be generated randomly or for example using a construction heuristic described in Section 3.2.1 or 3.2.2. The size of the initial population P highly depends on the problem considered and can range from a few to several thousand individuals.

Evaluation: Each solution $s \in P$ gets evaluated and a fitness value is assigned which is then used to rank the solutions.

Selection: In each generation a set $Q_S \subseteq P$ of *individuals* (chromosomes) is selected to breed the next generation. There exists various types of selection strategies such as, e.g., fitness proportional selection, linear ranking, rank selection, tournament selection, weights tournament selection, each with its own advantages and disadvantages, have been proposed in the literature.

Recombination: This process, which is also called *crossover*, is like the biological reproduction process where the offspring is produced from their parents. The crossover process can take two or more parents from the selected individuals (Q_s) and the new individual should be build upon attributes that are inherited from its parents. There exist various approaches which should reproduce offspring with whom and different crossover techniques such as, e.g., one-point crossover, two-point crossover, cut and splice, uniform crossover.

The recombination process is repeated until a new population (Q_r) of appropriate size is generated.

Mutation: In this step, small random changes are made to some individuals in order to introduce new characteristics into the population.

Replacement: Here another selection process is done, deciding which individual will survive from the current generation and the new offspring to to next generation. The size of the population usually stays constant, but which individual will be be chosen can vary. The extremes are that the offspring completely replaces the parents or only one parent is replaced by a new individual, but commonly some form in between is chosen.

3.2.7.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) tries to optimize a problem iteratively by improving a candidate solution with regard to a fitness function. This method was first proposed by Kennedy and Eberhart [26], see also [27]. The idea of this optimization scheme is inspired by the movement of bird flocks and fish schools which both try to find the optimal position in the swarm.

As a GA, a PSO algorithm also works on a set of candidate solutions (*swarm*). Each individual solution is called *particle*. Each particle i, $1 \leq i \leq P$, has a *velocity* and they store their best known position pBest_i. Also the position of the globally fittest particle, *gBest*, is tracked by the metaheuristic. In each iteration each particle is accelerated toward pBest and gBest with its velocity which is weighted by a random value. Algorithm 9 shows the basic implementation of PSO.

The velocity $v_{\max}$ is an important parameter as it determines the resolution with which regions are searched. A too high value might cause particles to move past good solutions and a too low value can make it unable to move over local optima and trap it there. The acceleration constants α and β are usually set to a fixed value depending on the application.

Another option to escape from local optima, besides changing $v_{\max}$ is to not use *gBest* but rather use *lBest*, which is the best particle of a predefined neighborhood. The neighborhood size is a value set to, e.g., two and thus defining the neighbors of $\mathrm{particle}_i$ as $\mathrm{particle}_{i-1}$ and $\mathrm{particle}_{i+1}$ and the neighbors do not change during a run. In order to use this local variant of PSO one has only to change to calculation of the velocity v the following way:

$$v_i \leftarrow v_i + \alpha * \mathrm{rnd}() * (pBest_i - x_i) + \beta * \mathrm{rnd}() * (lBest_i - x_i)$$

3.3 Basic Definitions for Multi-objective Optimizations

Let us consider the following, general multi-objective optimization problems with $l \geq 2$ objectives

$$\text{„min“ } z(x)$$

$$\text{s.t. } x \in X$$

Algorithm 9 Particle Swarm Optimization

```
1: initialize swarm
2: while termination criterion not met do
3:     for $i \leftarrow 1$ to $P$ do
4:         if $\text{fitness}(x_i) < \text{fitness}(pBest_i)$ then
5:             $pBest_i \leftarrow x_i$
6:         end if
7:         if $\text{fitness}(x_i) < \text{fitness}(gBest_i)$ then
8:             $gBest \leftarrow x_i$
9:         end if
10:    end for
11:    for $i \leftarrow 1\mathrm{to}P$ do
12:        $v_i \leftarrow v_i + \alpha * \text{rnd}() * (pBest_i - x_i) + \beta * \text{rnd}() * (gBest_i - x_i)$
13:        if $v_i > v_{\max}$ then
14:            $v_i \leftarrow v_{\max}$
15:        end if
16:        $x_i \leftarrow x_i + v_i$
17:    end for
18: end while
```

where $x \in \mathbb{R}^n$ is the variable vector, $z = (z_1, ..., z_l)$ with $\mathbb{R}^n \rightarrow \mathbb{R}^l$ is the objective function, „min“ $(z(x))$ refers the component-wise minimization and $X \in \mathbb{R}^n$ is the feasible set. If some of z_i's, $1 \leq i \leq l$ are conflicting usually no single solution to this problem exists, but the objective usually is to identify all so-called non dominated solutions. More precisely one typically aims to identify one solution for each point of the so-called Pareto front.

Definition 3.3.1 (Pareto dominance)**.** *A vector $u^* = (u_1, ..., u_n)$ dominates a vector $v^* = (v_1, ..., v_n)$, denoted by $u \prec v$ if and only if $\forall i \in \{1, ..., n\} : u_i \leq v_i$ and $\exists j \in \{1, ..., n\} : u_j < v_j$.*

Definition 3.3.2 (Non-dominated solution)**.** *A feasible solution x^* is non-dominated if no other solution y^* with $z(y^*) \prec z(x^*)$ exists.*

Definition 3.3.3 (Non-dominated point)**.** *The point of a non-dominated solution x^* in the objective space, $z(x^*)$, is called non-dominated point.*

Definition 3.3.4 (Extreme non-dominated point)**.** *The point of a non-dominated solution x^* in the objective space is called an extreme non-dominated point if it exists no other solution y^* where any single objective $\hat{c}(y^*) < \hat{c}(x^*)$.*

Definition 3.3.5 (Equivalent solutions)**.** *Two solutions x^* and y^* are called equivalent if $z(x^*) = z(y^*)$.*

Definition 3.3.6 (Non-dominated set)**.** *A non-dominated set is a set of non-dominated solutions.*

Definition 3.3.7 (Pareto front)**.** *The Pareto front is the image of the non-dominated set in the objective space.*

Definition 3.3.8 (Minimal complete set)**.** *The minimal complete set contains all existing non-dominated solutions.*

When considering metaheuristic approaches for multi-objective problems we do not know whether a currently non dominated solution, with respect to the set of known solutions, is really non dominated. Therefore, the following concepts will be used:

Definition 3.3.9 (Non-dominated solution with respect to S)**.** *Let S be a set of feasible solutions. A feasible solution x^* is non dominated w.r.t. S if solution $y^* \in S$ with $z(y^*) \prec z(x^*)$ exists.*

Definition 3.3.10 (Non-dominated point with respect to S)**.** *The point of a non-dominated solution x^* w.r.t. S in the objective space is called non-dominated point w.r.t. S.*

Definition 3.3.11 (Extreme Non-dominated point with respect to S)**.** *Let S be a set of feasible solutions. The point of a non-dominated solution x^* in the objective space is called an extreme non-dominated point w.r.t. S if no other solution $y^* \in S$ where any single objective $z_i(y^*) < z_i(x^*), \forall i \in \{1, ..., l\}$.*

Definition 3.3.12 (Non-dominated set with respect to S)**.** *A non-dominated set w.r.t. S is a set of non-dominated solutions w.r.t. S.*

Whenever it is clear from the context we will simply use non-dominated solution (extreme non-dominated set, non-dominated set) instead of non-dominated solution (extreme non-dominated set, non-dominated set) with respect to S.

CHAPTER 4

Previous & Related Work

In this chapter an overview of the relevant previous work is given starting with single objective variants of the ConFL. Since the multi-objective ConFL is not explored yet a brief review on applied multi-objective approaches for different problems is given later on.

4.1 State of the art

ConFL has been introduced by Karger and Minkoff [24] who also presented an approximation algorithm for the problem. Subsequently various algorithmic approaches for several problem variants have been proposed.

In the single objective case three different approaches have been mainly used to solve the connected facility location problems. Depending on the time available and the desired quality of the delivered solution one has the options to choose between metaheuristic approaches, approximation algorithms and exact methods which are usually based on methods from mixed integer programming. Leitner and Raidl [34] considered a prize-collecting variant of ConFL, i.e., not all customers need to be connected, which also considers capacity constraints on facilities and proposed exact methods based on mixed integer programming which are solved using branch-and-cut and branch-and-cut-and-price algorithms. Gollowitzer et al. [16] developed an integer programming model based on single commodity flow for the capacitated connected facility location problem. There also exists a set of mixed integer programming (MIP) formulations for the ConFL modeled by Gollowitzer and Ljubić [17].

Leitner and Raidl propose a combination of Lagrangian decomposition with very large scale neighborhood search [33] for ConFL, a variable neighborhood search (VNS) for the prize collecting capacity constrained ConFL [32] and a VNS for the capacitated ConFL [35]. Tomazic and Ljubić [48] developed a greedy randomized adaptive search procedure (GRASP) algorithm and Ljubić [36] proposed a hybrid VNS, combining a VNS with a reactive tabu search for the

ConFL.

Swamy and Kumar [46] propose a primal-dual approximation algorithm for the ConFL by solving an exponential size linear program relaxation. Eisenbrand et al. [11] present a simple randomized algorithm framework to approximate the ConFL problem via random facility sampling and core detouring.

In the single-objective case one can always say that a solution is better, equal or worse than another one. In the multi-objective scenario, however, the different candidate solutions are not necessarily comparable with each other. Thus optimality depends on the preferences of the decision maker, who may value one objective more than another. If the weight of the objectives is known we can reduce the problem to the single objective case. If nothing is known about the decision maker's preferences, it is common to tackle problems in terms of Pareto optimality, to obtain a set of Pareto optimal solutions approximating the Pareto front as good as possible.

As the bi-/multi-objective ConFL has not been considered previously the following part gives an overview of the currently used methods to solve bi-/multi-objective optimization problems.

Evolutionary algorithms (EAs) are widely used in multi-objective optimization because of their ability to possibly find multiple Pareto optimal solutions in a single run. Fonseca and Fleming [13] present a review of the early work done in the area and discussing the similarities and differences of the various approaches. The best-known EA in the field was proposed by Deb et al. [10]. This Non-Dominated Sorting Genetic Algorithm for multi-objective optimization (NSGA-II) features low computational requirements, an elitist approach, a parameter-less sharing approach, and has been successfully applied to a number of problems. Coello and Lechuga [7] propose another population based algorithm based on particle swarm optimization (PSO) [27]. They use the concept of Pareto dominance to determine the flight direction of the particles and use a global storage for non-dominated solutions to guide the algorithm toward optimality. Other PSO approaches for multi-objective problems are presented by Parsopoulos and Vrahatis [45] and Zhang [51] respectively.

A different way to tackle multi-objective problems is to use local search (LS) variants instead of EAs. Most of these LS based methods are applied to the bi-objective TSP and use a two-phase approach. Paquete and Stützle [44] propose a two-phase algorithm for the bi-objective TSP that generates initial solutions by considering only one of the objective functions. In the second phase a LS algorithm, using an aggregated weight function (Marler and Arora [38]) which varies the weights until all aggregations are explored is applied. In contrast to Parquet and Stützle, Lust and Teghem [37] present another two-phase algorithm. In the first phase they aggregate the two objective functions with different weights to generate a set of initial non-dominated solutions and subsequently apply a Pareto local search (PLS) [1, 2, 43] to the initial set to get a better approximation of the Pareto front. Lagnua et al. [28] propose a combination of a genetic algorithm (GA) and a LS for a beacon layout problem. An overview of different metaheuristic methods used to tackle multi-objective combinatorial problems is given by Basseur et al. [3]. Several exact approaches for bi-objective problems including, e.g., the ϵ-constraint [6,30] or the two-phase method have also been proposed.

CHAPTER 5

The Two-Phase Local Search Algorithm

As discussed in Chapter 4 there exist a lot of different algorithms for multi-objective problems with population based approaches being the majority of it. Due to the structure of the ConFL it is, however, very hard to find a good solution representation that can be used for a population based method like the NSGA-II [10]. Thus we had to look for other alternatives.

Based on the previous work done on the single-objective variants of the problem and successful approaches for different bi-objective combinatorial optimization problems a two-phase algorithm [37, 44] approach seemed to be promising to conquer the bi-objective ConFL.

Hence we decided to implement a two-phase algorithm which in phase 1 (see Section 5.2) aggregates the two objective functions with different weights and runs a single objective variant of a VND algorithm on each weight set to generate different solutions. Each solution is inserted into a paretofilter (see Section 5.1 which keeps track of the set of non dominated solutions, i.e., which removes dominated solutions. This set of non dominated solutions is then used as input for the second step (see Section 5.3) of the two-step procedure in which a Pareto Local Search (see Section 5.3.1) is used to improve the already found solutions to find an even better approximation of the Pareto front.

In order to implement such a two-step procedure as described above one more crucial method needs to be introduced first - the Paretofilter for which we first need some definitions already presented in Section 3.3 for how to compare solutions for a bi-objective ConFL instance.

5.1 Paretofilter

A Paretofilter is a method that takes as input a set S of solutions and a newly generated solution s_n. It checks whether s_n is non-dominated w.r.t. S in which case s_n will be added to S. Further the method will remove all solutions from S that are dominated by the newly found solution thus securing that only non-dominated solutions are kept.

This procedure, which is given in Algorithm 10, removes all solutions that are dominated by s_n by iterating over all members from S and s_n is added to S in case s_n is not dominated by any other solution currently in S.

Algorithm 10 Paretofilter

```
1: add ← true
2: for all $s \in S$ do
3:     if $z(s_n) \prec z(s)$ then
4:         $S \leftarrow S \setminus \{s\}$
5:     else
6:         if $z(s) \prec z(s_n)$ then
7:             add ← false
8:         end if
9:     end if
10: end for
11: if add then
12:     $S \leftarrow S \cup \{s_n\}$
13: end if
```

5.2 Phase 1

In this phase weights $\omega_1 \geq 0$ and $\omega_2 \geq 0$, $\omega_1 + \omega_2 = 1$, are set for each objective function, which are used to aggregate the two functions to one and then a construction heuristic is run to get an initial solution s. This initial solution is subsequently optimized using a VND to obtain an even better solution s'. The solution of this procedure will then be inserted into the paretofilter for evaluation against the non dominated solution set S. After each iteration the weights are adjusted by Δ_ω with $0 < \Delta_\omega \geq 1$. This is needed in order to weight the cost and revenue differently for the next run to create a new initial solution which should diverge from the last one. Then the whole procedure is rerun which can be seen in Algorithm 11.

In order to generate initial solutions and apply a single-objective VND in the first phase the two objective functions

$$z_1(s) = \sum_{i \in F_s} f_i + \sum_{e \in E_s} c_e \tag{5.1}$$

$$z_2(s) = \sum_{k \in R \setminus R_s} r_k \tag{5.2}$$

Algorithm 11 Phase 1

```
1: $S \leftarrow \varnothing$
2: $\omega_1 \leftarrow 0$
3: $\omega_2 \leftarrow 1$
4: while $\omega_1 < 1$ do
5:     $s \leftarrow \mathrm{constructionHeuristic}(\omega_1, \omega_2)$  see Algorithm 12
6:     $s' \leftarrow \mathrm{VND}(s, \omega_1, \omega_2)$  see Section 5.2.4
7:     $S \leftarrow \mathrm{Paretofilter}(S, s')$  see Algorithm 10
8:     $\omega_1 \leftarrow \omega_1 + \Delta_\omega$
9:     $\omega_2 = 1 - \omega_1$
10: end while
```

are aggregated into a new single objective function Z based on which single objective optimization can be performed.

5.2.1 Aggregation of objective Functions

Several methods that can be used to aggregate objective functions have been proposed (see, e.g., [38]) - the most commonly used ones are described below for general multi-objective optimization problems with l objectives.

Weighted exponential sum: This is one of the most general functions for weight aggregation. There are two commonly used variants.

$$Z = \sum_{i=1}^{k} \omega_i [z_i(s)]^p \tag{5.3}$$

$$Z = \sum_{i=1}^{k} [\omega_i z_i(s)]^p \tag{5.4}$$

The parameter p is set to a fixed value that has to be determined be experimentation and the weights are typically $\omega_i > 0$ with $\sum_{i=1}^{k} \omega_i = 1$.

Weighted sum: This is the method of choice for this master thesis as it has no parameters that could vary (besides the weights ω_i). The weighted sum method is a special case of the weighted exponential sum where $p = 1$ and by far the most common approach used to aggregate multiple objectives.

$$Z = \sum_{i=1}^{k} \omega_i z_i(s) \tag{5.5}$$

Exponential weighted criterion: The aim of this method is to capture points on non-convex portions of the Pareto optimal surface.

$$Z = \sum_{i=1}^{k} (e^{p * \omega_i} - 1) e^{p * z_i(s)} \tag{5.6}$$

To implement that aggregation function one needs to take care as it can easily lead to a numerical overflow depending on the selection of p.

Weighted product: The following method allows functions with different orders of magnitude to have similar significance.

$$Z = \prod_{i=1}^{k} [z_i(s)]_i^{\omega} \tag{5.7}$$

Usually when aggregation is done a single run will just yield one solution but the Pareto front consists of a set of non-dominated solutions thus making it necessary that several runs with different weight sets are done in order to find a good initial approximation of the Pareto front.

5.2.2 Adaptation of Weights

As there are different methods for the weight aggregation there also exist several approaches for the weight adaption process in each iteration [21]. Some of them are listed below.

Conventional Weighted Aggregation (CWA): When CWA is used a priory knowledge of the search space is required as the weights are fixed and in a run only one Pareto optimal solution can be found. Thus, using this method one needs to rerun the algorithm with different weight values in order to find other Pareto optimal points. A common approach here is to use a fixed step size $\Delta_\omega = 1/(n_r - 1)$ where n_r is the number of different weight sets that should be used. This was the aggregation method of choice for this thesis, cf. Algorithm 11.

Bang-Bang Weighted Aggregation (BWA): To overcome the limitation of CWA that in each iteration only one Pareto optimal point can be found Bang-Bang weighted aggregation modifies the weights during the optimization process and determines the weights based on the iteration's index t and the change frequency F.

$$\omega_1(t) = \text{sign}(\sin(\frac{2\pi t}{F})), \omega_2(t) = 1 - \omega_1(t) \tag{5.8}$$

BWA changes the weights rather abruptly due to the sign function.

Dynamic Weighted Aggregation (DWA): This method also changes the weights during an iteration but as is does not use the sign function it is not as harsh as BWA. Its slower weight change leads the optimization algorithm to head toward the Pareto front.

$$\omega_1(t) = |\sin(\frac{2\pi t}{F})|, \omega_2(t) = 1 - \omega_1(t) \tag{5.9}$$

There also exist some adaptions of the DWA method proposed by Jin: Evolutionary Dynamic Weighted Aggregation (EDWA) [21] and Randomly Weighted Aggregation (RWA) [22] which are normally used in multi-objective evolutionary algorithms.

As there are some good applications of local search heuristics on the ConFL and variants thereof in the single objective case [32, 33, 35, 36] we decided to apply a variant of such a method to our given aggregated problem using the simple CWA approach.

5.2.3 Construction Heuristic

To create initial solutions there exist different approaches and how to obtain starting solutions which serve as input for the various VND applications with changed weights.

The following three approaches to create initial solutions for further improvement by the VND seem natural:

Random Solutions: The easiest way to create a starting solution is to generate a random solution and use this as a starting point with the big disadvantage that this could lead to a long runtime of the VND as that solution could be very far from optimal w.r.t. the chosen weighted sum objective.

Independent Heuristic Solutions: In order to get a better initial solution a common way is to use a construction heuristic every time an initial solution is needed. The advantage of this approach is that the starting solutions is generated w.r.t. the chosen weighted sum objective and thus could reduce the runtime of the VND significantly. The only disadvantage is that depending on the complexity of the problem the algorithm could be very time consuming and it is not adviceable to rerun it every time a new starting solution is needed.

Iterative Heuristic Solutions: In order to conquer the long runtime of the creation of independent heuristic solutions a common practice is to run a construction heuristic once at the beginning and then after each application of the optimization heuristic the last found solution is modified by applying random moves in one or more neighborhood structures with the advantage that it might take less time to alter an existing solution compared to running the complete construction heuristic again. The big advantage of this procedure is that it can potentially save a lot of time but on the downside there are two major disadvantages depending on the number of random moves in the neighborhood structures. In case too few moves are applied the newly generated initial solution could lie relatively close to the original solution and thus the next optimization run with a different weight set might lead to a new optimized solution which is close to another existing non-dominated solution in the solution space thus leading to a bad approximation of the Pareto front. On the other hand, if too many moves are applied the altered solution is more or less equivalent to a random generated solution which, as mentioned above, might lead to a long run of the optimization heuristic and thus wasting the saved time of not running the construction heuristic again.

In order to use such an approach to generate initial solutions some tests are necessary to determine the right amount of moves in the neighborhood structures to not go too far away from the last found solution and stay close enough to not resemble a random solution.

We decided to use a construction heuristic each time (i.e., the „independent“ approach) since tests showed that the construction heuristic is rather fast.

The construction heuristic starts with a solution that initially consists of the root node only.

Let $S' = ((V', E'_C), F', R', E'_A)$ be the partial solution of the current iteration, where $F' \subset F$ is the set of open facilities, $E'_A \subseteq E'$ the set of chosen assignment edges such that each selected customer $j \in R'$ with $R' \subset V'$ is connected to exactly one open facility $i(j) \in F'$ and a Steiner tree (V', E'_C) which is a subgraph of $(F \cup T, E_C)$ where $F' \subseteq V'$.

For $s \in F \setminus F'$, let $\tilde{c}(s) \geq 0$ be the minimum costs of extending (V', E') by a path to contain s and $(V'(s), E'(s)))$ denote the corresponding path (which is calculated for all $s \in F \setminus F'$) by computing a shortest path with source b where costs f edges $e \in E'$ are set to zero.

Furthermore for each $s \in F \setminus F'$ let $c(s, \omega_1, \omega_2) = \{j \in R : \exists \{s, j\} \in E_A \wedge a_{ij} < \omega_2 r_j\}$ be the set of customer nodes that can be assigned to s in a profitable way w.r.t. ω_1 and ω_2. In each iteration facility $s' = \operatorname{argmin}_{s \in F \setminus F'}\{\omega_1[\tilde{c}(s) + f_s + \sum_{j \in c(s,\omega_1,\omega_2)} a_{ij}] - \omega_2 \sum_{j \in c(s,\omega_1,\omega_2)} r_j\}$ is selected and added to the solution (together with the necessary extensions of the Steiner tree and customer assignments). Thus the new solution is given by $((V' \cup V'(s'), E' \cup E'(s')), F' \cup \{s'\}, R' \cup R'(s'))$. This process is repeated until there is no more facility found which improves the objective value.

Algorithm 12 Construction Heuristic

```
1: $S' \leftarrow \text{emptySolution}()$
2: add start node to $S'$
3: **repeat**
4:     get $s' = \operatorname{argmin}_{s \in F \setminus F'}\{\omega_1[\tilde{c}(s) + f_s + \sum_{j \in c(s,\omega_1,\omega_2)} a_{ij}] - \omega_2 \sum_{j \in c(s,\omega_1,\omega_2)} r_j\}$
5:     add $s'$ to $S'$
6:     add edges and Steiner nodes $((V'(s), E'(s)))$ between $s'$ and $S'$ to $S'$
7:     add customers and assignment edges to $S'$
8: **until** termination-condition met
```

Due to the fact that during the construction heuristic a customer node $j \in R$ could be assigned to more than one facility if the condition

$$a_{s'j} * \omega_1 - r_j * \omega_2 < 0 \tag{5.10}$$

holds, the solution S could be not valid, because the problem definition states that each customer can only be assigned to a maximum of one facility. In order to secure this condition and also to make some small improvements to the solution generally we apply a set of improvement/post-processing strategies after the construction of the initial solution is finished.

5.2.3.1 Solution Post-processing:

Figure 5.1 visualizes the improvement strategies which were applied in order to further optimize the solution.

Method 1: Remove multiple customer connections As mentioned above, it can happen during the construction phase that a customer is assigned to multiple open facilities. In order to solve the problem of the multiple connections all edge from each customer in the solution S will be removed except the cheapest one. Let $E'_A(j) = \{\{i, j\} \in E'_A\}$ be the set of assignment edges for customer $j \in R'$ to each facility $i \in F'$ it is assigned to. This improvement

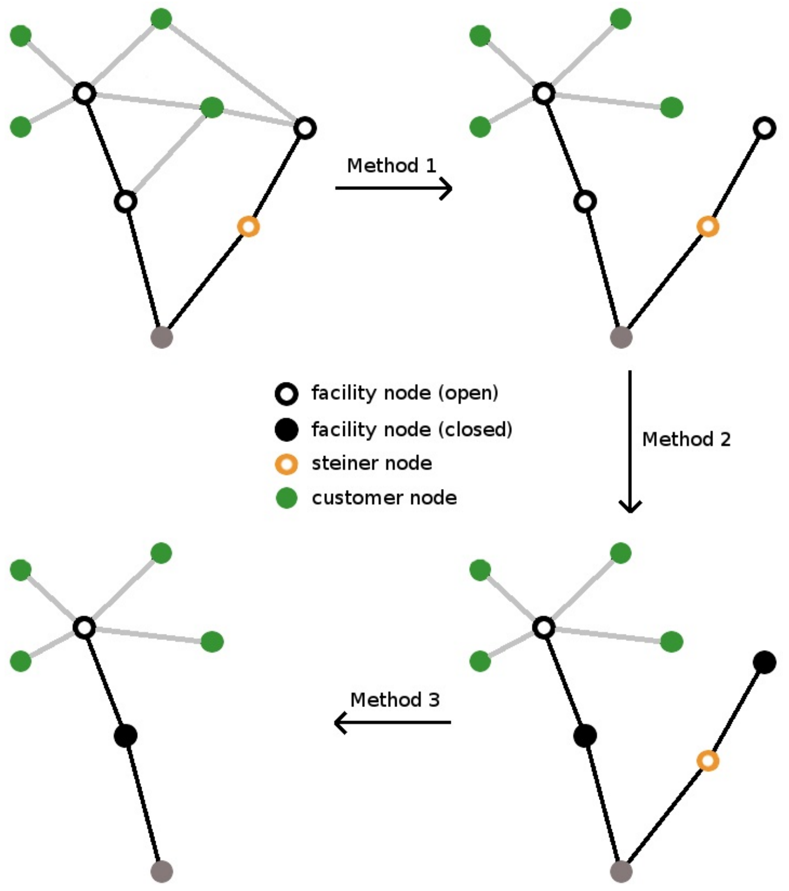

Figure 5.1: Improvement Strategies visualization - Shown in this figure are the 3 methods used to improve the solution and make it valid. Method 1 removes multiple customer assignments because during the construction heuristic it can happen that a customer is assigned to more than one open facility. In Method 2 previously opened facilities which have lost all the customer assignments because of Method 1 are now closed because open facilities without customers worsen the objective value. Finally Method 3 prunes all leaf nodes which are neither open facilities nor customers.

strategy aims to remove all assignment edges for each selected customer except the „cheapest“ $E''_A = \bigcup_{j \in R'} \{\{i, j\} \in E'_A : a_{ij} = \text{argmin}_{s \in F'}(\{i, j\} \in E'_A)\}$ w.r.t. to the objective value.

Method 2: Close facilities After removing multiple customer connections it could happen that a facility ends up with no customer attached to it but initially it was an open facility because it had some customers connected to it. These facilities will be closed because an open facility creates costs and earns no revenue. The remaining set of open facilities can be described as $F' \leftarrow F' \setminus \{s \in F' : \{j : \{s, j\} \in E'_A\} = \emptyset\}$.

Method 3: Prune leaf facilities and Steiner nodes After the first two improvement methods it could happen that a closed facility node or a Steiner node ends up as a leaf node in the core graph of the solution. These leaf nodes create unnecessary costs because they serve no customer thus moving the solution away from the Pareto front and making it mandatory to remove them. In order to achieve this all degree-one Steiner nodes $t \in V' \setminus (F' \cup R')$ where $\deg(t) = 1$ and their corresponding edge $e_t \in E'_C$ are removed recursively.

5.2.4 Variable Neighborhood Descent

The second step in the first phase is to apply a VND heuristic for each constructed solution.

We used the following neighborhood structures which have already been used for ConFL with a single objective. In the VND the neighborhood structures are considered in the same order as presented here.

5.2.4.1 Key-Path Improvement

As the Steiner tree problem (STP) is a subproblem of ConFL we decided to use a Key Path Improvement neighborhood structure as is has been successfully applied to the STP as well as in some of its generalizations [31, 39, 49].

In order to define a key path it is necessary to define key nodes first. The set of key nodes is defined as $K = \{b\} \cup F_s \cup \{v \in R_s | \deg_s(v) \geq 3\}$ where b is the root node, i.e., it consists of the root node, all open facilities and all nodes of degree ≥ 3 of the core graph of the current solution S. A key path is a path between any two nodes $u, v \in K$ of s which contains no other key node along its path.

Leitner and Raidl [33] proposed to use Key-Path improvement on a neighborhood search for the capacitated ConFL which we adopted. Their algorithm, described in Algorithm 13, considers every key path $P = (\mathcal{V}, \mathcal{E}) \in \bar{P}(s)$, where $\bar{P}(s)$ is the set of all key paths, of a solution S. In each iteration one key-path is replaced by a shortest path between its key nodes in a best improvement manner. Thereby, when identifying a shortest replacement of key path $(\mathcal{V}, \mathcal{E}) \in \bar{P}(S)$ the costs of all other core edges in the solution are set to zero.

5.2.4.2 Facility Swap

This neighborhood structure opens a currently closed facility or closes an open one. Thereby to update a solution in an appropriate way, the following cases need to be distinguished. For $s \in F \setminus F'$, i.e., a new facility is opened, two possibilities exist:

a) $s \in V'$: The facility s is used as Steiner node in the current solution S', see Figure 5.2. In this case s is simply marked as open, i.e., $F' = F' \cup \{s\}$ and all unassigned customers of s $R'_U(s) = \{j \in R : \exists\{s, j\} \in E_A \wedge \omega_1 c_{sj} < \omega_2 r_j\}$ and their corresponding edges are added to S'.

Algorithm 13 Key Path Improvement

```
1: **repeat**
2:   $c'_e = \begin{cases} 0 \\ c_e \end{cases} , \forall e \in E$
3:   $\delta = 0$
4:   **for all** key paths $P = (\mathcal{V}, \mathcal{E}) \in \bar{P}(s)$ **do**
5:     // key path end nodes are $u$ and $v$
6:     $c'_e = c_e, \forall e \in \mathcal{E}$
7:     find shortest path $P' = (\mathcal{V}', \mathcal{E}')$ in $(V', E')$ between $u$ and $v$ w.r.t. $c'$
8:     $\delta' = \sum_{e \in \mathcal{E}'} c'_e - \sum_{e \in \mathcal{E}} c_e$
9:     **if** $\delta' < \delta$ **then**
10:      $\delta = \delta'$
11:      store replacement of path $P$ by $P'$ as best move
12:    **end if**
13:    $c'_e = 0, \forall e \in \mathcal{E}$
14:  **end for**
15:  **if** $\delta < 0$ **then**
16:    apply best move
17:  **end if**
18: **until** $\delta \geq 0$
```

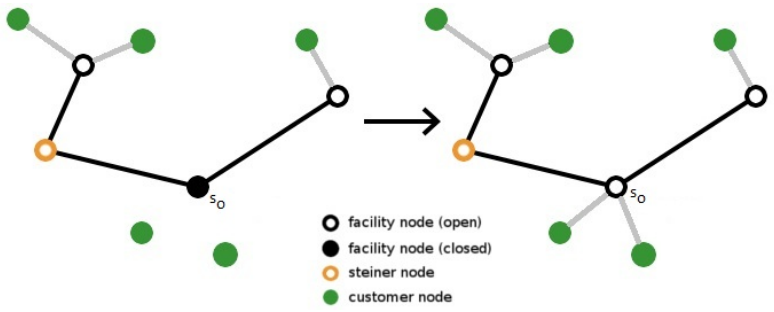

Figure 5.2: Example of facility swap neighborhood - the facility to open (s) is already in the current solution ($s \in V'$), thus only neighboring customers where equation $\omega_1 c_{sj} < \omega_2 r_j$ holds are be added.

b) $s \notin V'$: Next to opening and assigning customers to it, facility s must be connected to the Steiner tree (V', E'). As for the construction heuristic, this is achieved by computing a cheapest path $P = (\mathcal{V}, \mathcal{E})$ between s and the solutions core graph (V', E'). Thus, the new solutions S'' is given as $S'' = (V' \cup \mathcal{V}, E' \cup \mathcal{E} \cup \{\{s, l\} : l \in R'_U(s)\}, F' \cup \{s\}, R' \cup R'_U(s))$, see also Figure 5.3.

If facility s should be closed, i.e., $s \in F'$, we need to distinguish the following cases (shown in Figure 5.4,5.5 and 5.6) depending on the degree of node s ($\deg'_G(s)$) in $G' = (V', E')$.

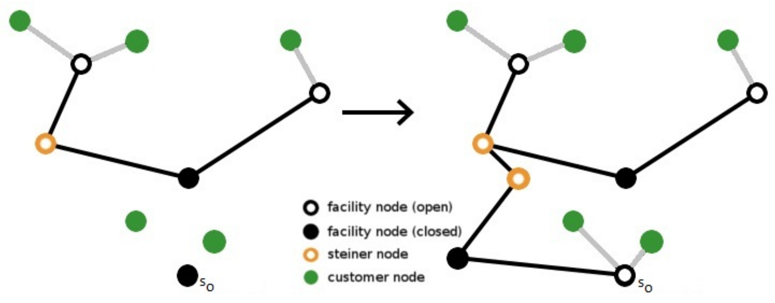

Figure 5.3: Example of facility swap neighborhood - the facility to open (s) is not in the current solution ($s \notin V'$), thus it needs to be opened and connected to (V', E'_C) and then all customers which suffice $\omega_1 c_{sj} < \omega_2 r_j$ are connected.

a) $\deg_{G'}(s) \geq 3$: In that case node s must be used as Steiner node to ensure connectivity after closing facility s. Hence the new solution is defined by closing s, i.e., $F' = F' \setminus \{s\}$, and removing all customers assignments $R'(s)$ for s, i.e., $R' = R' \setminus R'(s)$ and their corresponding assignment edges $E'_A(s)$. See Figure 5.4 for an example.

b) $\deg_{G'}(s) = 2$: In this case all assigned customers $R'(s)$ are removed, i.e., $R' = R' \setminus R'(s)$, as well as the assignment edges $E'_A(s)$. Further s is removed , i.e., $F' = F' \setminus \{s\}$ as well as both edges connected to s, $E'_C(s)$. This procedure separates the core graph $S_C = (F' \cup T', E'_C)$ into two subtrees $S_{C1} = (F'_1 \cup T'_1, E'_{C1})$ and $S_{C2} = (F'_2 \cup T'_2, E'_{C2})$. Due to the removal of s and its corresponding edges it can happen, like during the construction heuristic, that Steiner nodes in S_{C1} with $\deg_{S_{C1}}(t') = 1$, where $t' \in T'_1$, exist. These will be removed iteratively from S_{C1} along with their connecting edge. This removal if also performed for S_{C2}. In order to reconnect the two subtrees S_{C1} and S_{C2} the shortest path $P' = (\mathcal{V}', \mathcal{E}')$ from S_{C1} to S_{C2} is calculated by using modified edge costs

$$c'_e = \begin{cases} 0, \forall e \in (E'_{C1} \cup E'_{C2}) \\ c_e, \forall e \in E_C \setminus (E'_{C1} \cup E'_{C2}) \end{cases} .$$

At last P' is added two $S_C = (F' \cup T' \cup \mathcal{V}', E'_C \cup \mathcal{E}')$ (see Figure 5.5).

c) $\deg_{G'}(s) = 1$: When s is a leaf node after disconnecting all customers, i.e., $\deg_S(s) = 1$, it will be removed from S along with its connecting edge. See Figure 5.6 for an example. This removal process is repeated for all subsequent Steiner nodes where $\deg_S(t') = 1$, with $t' \in T'$.

5.2.4.3 Facility Exchange

The facility exchange neighborhood is just an extension of the facility swap neighborhood by doing both, opening and closing a facility. We decided to close a facility before opening a new one because if first the new facility is opened and afterwards the other one is removed the

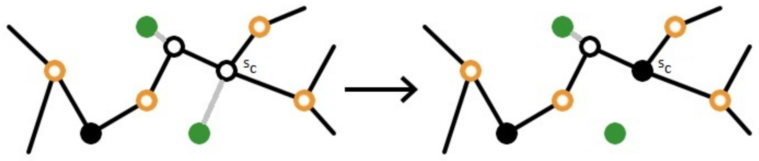

Figure 5.4: Example of facility swap neighborhood - the facility to close (s) has more than 2 adjacent open facilities or Steiner nodes($\deg_{G'}(s) \geq 3$) thus it is a node which is important for the solution so this facility will just be closed and all adjacent customers $R'(s)$ their assignment edges $E'_A(s)$ are removed from S, i.e., $R' = R' \setminus R'(s)$ and $E'_A = E'_A \setminus E'_A(s)$.

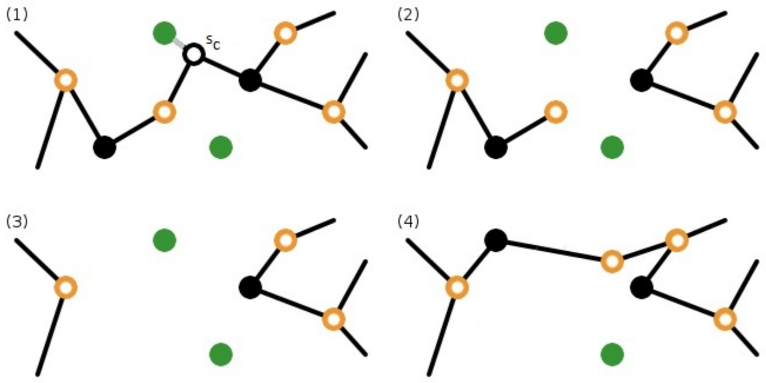

Figure 5.5: Example of facility swap neighborhood - the facility to close (s) has exactly two neighboring open facilities or Steiner nodes ($\deg_{G'}(s) = 2$) then s will be closed and all adjacent customers and their corresponding assignment edges will be removed from S. As it can be seen in subfigure 2, when s and its adjacent edges are removed G' is split in two parts $S_{C1} = (F'_1 \cup T'_1, E'_{C1})$ and $S_{C2} = (F'_2 \cup T'_2, E'_{C2})$. The left subgraph has a leaf node which is a Steiner node, followed by a closed facility and then again a Steiner node. To remove these leaf nodes an adaption of Method 3, described in Section 5.2.3.1, is applied. The result can be seen in subfigure 3. Now the two disconnected graphs, $S_{C1} = (F'_1 \cup T'_1, E'_{C1})$ and $S_{C2} = (F'_2 \cup T'_2, E'_{C2})$, are then reconnected by the shortest path $P' = (\mathcal{V}', \mathcal{E}')$ w.r.t. modified edge weights c'_e from S_{C1} to S_{C2} such that the reconnected graph is again a valid solution.

following could happen: The facility to open $s_o \notin V'$ and the facility to close s_c is a leaf node. In order to add s_o to S the shortest path $P = (\mathcal{V}, \mathcal{E})$ between s_o and S is calculated. Assume s_o is added before s_c is removed it is possible that $s_c \in P$ and therefore s_c is no leaf node anymore and thus would just be closed and not removed. The problem is, in case a new facility is opened before the other one is removed that it could exist a shortest path $P' = (\mathcal{V}, \mathcal{E})$ between s_o and $(V' \setminus \{s_c\}, E'_C \setminus E'_C(s_c))$ for which $\sum_{e \in P} c_e < \sum_{e \in P'} c_e$. Considering that at the time of the shortest path calculation $s_c \in S$ it could be that $\sum_{e \in P} c_e + E'_C(s_c) > \sum_{e \in P'} c_e$ which makes P' the better path but P would be selected. Figure 5.7 visualized the case that s_o is added before removing s_c and 5.8 shows the other way round.

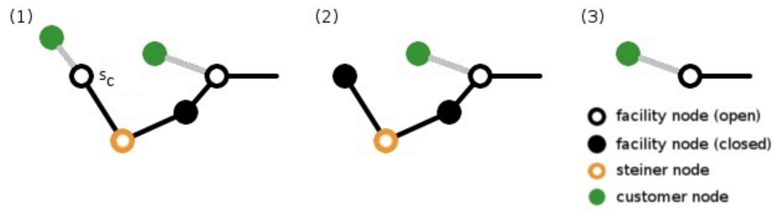

Figure 5.6: Example of facility swap neighborhood - the facility to close (s) has only one neighboring open facility or Steiner node ($\deg_{G'}(s) = 1$). By removing all connected customers and their assignment edges s becomes a leaf node, and then by application of an adapted version of Method 3, described in Section 5.2.3.1, s and all subsequent leaf nodes created by the removal will be removed iteratively.

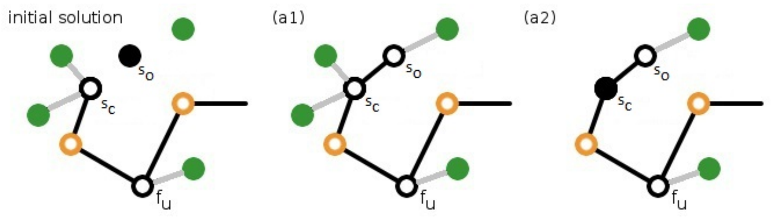

Figure 5.7: Example of facility exchange neighborhood - Opening s_o before removing s_c can lead to the problem that the shortest path $P = (\mathcal{V}, \mathcal{E})$ between s_o and S includes s_c and thus it wont be removed although when taking the decision to swap it was a leaf facility. This could lead to the problem that there exists another shortest path $P' = (\mathcal{V}, \mathcal{E})$ connecting s_o to $S \setminus \{s_c\}$ such that the overall objective value after the swap is done is better. This can be seen in Figure 5.8

5.3 Phase 2

In the second phase of the 2-phase algorithm a Pareto local search (PLS) algorithm is applied to all non dominated solutions acquired in phase to further improve the approximation of the Pareto front. Figure 5.9 shows the basic concept of a PLS.

5.3.1 Pareto Local Search

The Pareto Local Search method was introduced by different authors [1, 2, 43]. It is based on a local search algorithm for multi-objective problems and works without the need for objective aggregation. The basic concept of PLS is to explore the neighborhood of each solution of the non-dominated set of solutions. In case a not dominated neighbor is found, it is added to the set of non-dominated solutions. This process iterates until no further non-dominated solution can be found. Algorithm 14 shows the pseudo code of the PLS method. The input for a PLS algorithm can be a single solution or a set of solutions. For our given problem we used the same neighborhood structures that are used in Phase 1.

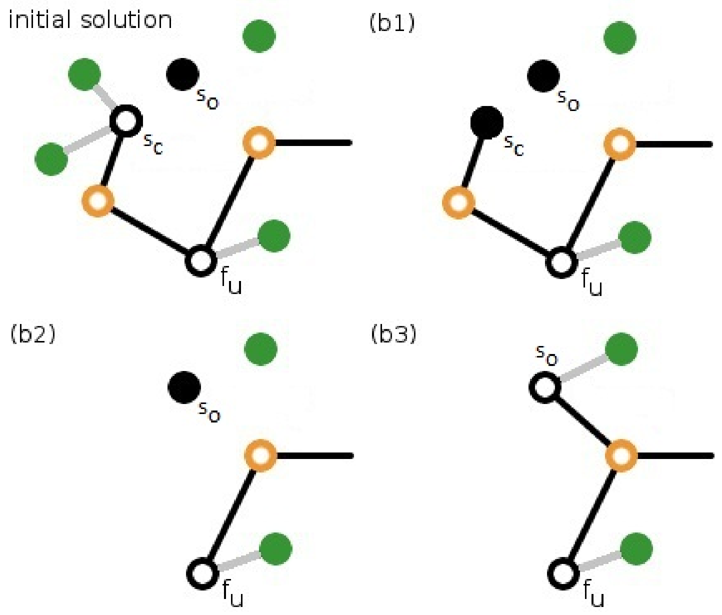

Figure 5.8: Example of facility exchange neighborhood - Exemplary facility exchange move as done by our implementation. At first s_c is closed and all assigned customers are removed from S. Next s_c is removed from S along with its adjacent edge because it was a leaf node. This process is repeated for the next Steiner node until a node $v' \in V'$ where $\deg_{G'}(v') > 1$ is reached. Now the removal steps are done and s_o is added to via the shortest path $P' = (\mathcal{V}, \mathcal{E})$ to the solution.

After applying the PLS procedure to the set of non-dominated solutions acquired in phase 1, **S** contains the set of non-dominated solutions which approximate the Pareto front.

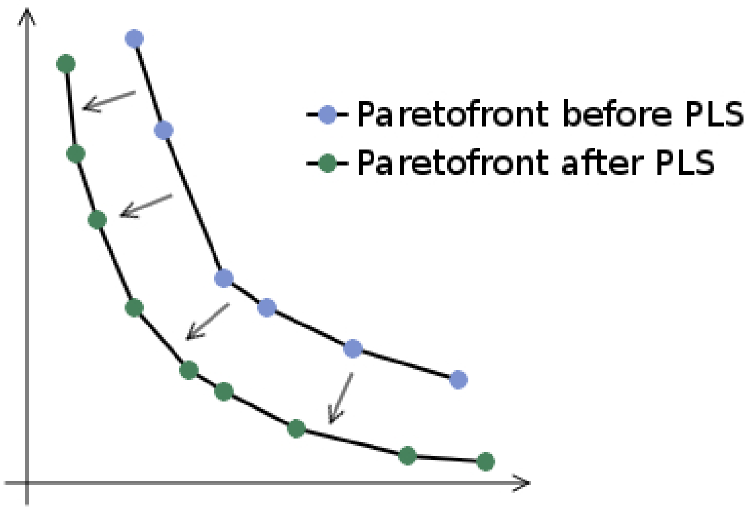

Figure 5.9: Example: A PLS algorithm is applied to improve the approximation of the Pareto front.

Algorithm 14 Pareto Local Search

```
1: $\mathbf{S} \leftarrow$ solutions generated in phase 1
2: explored $\leftarrow \emptyset$
3: for all $s \in \mathbf{S}$ do
4:   explored$(s) \leftarrow$ false
5: end for
6: while $\exists x \in \mathbf{S}$ : explored$(x) =$ false do
7:   $y \leftarrow$ random Solution from $\mathbf{S}$ : explored$(y) =$ false
8:   for all $y' \in N(y)$ do
9:     if $y' \succ y$ then
10:      $\mathbf{S} \leftarrow$ Paretofilter$(\mathbf{S}, y')$
11:      explored$(y') \leftarrow$ false
12:    end if
13:  end for
14:  explored$(y) \leftarrow$ true
15: end while
```

CHAPTER 6

Computational Tests and Results

6.1 Test instances

Our test instances (I01 - I64) were originally test instances for the ConFL and used by Ivana Ljubic in a paper about a hybrid VNS for the ConFL [36]. The problem was that in all instances all customer nodes were connected to each facility node and almost all cases the optimum was to connected all valuable customers to the root node or the root node and a second facility to cover all valuable customers. To overcome that issue we needed test instances where not every customer is connected to every facility. To achieve that we randomly removed 30% of the assignment edges of each customer node. Further as the test instances were from a single objective ConFL they did not have a revenue value for each customer which we also assigned randomly for each customer in order to create usable test instances for the bi-objective case.

See table 6.1 for an overview of instance properties (facility, Steiner and customer node and edge count).

Table 6.1: Node and edge count of instances

	Nodes				Edges		
Instance	$\|V\|$	$\|F\|$	$\|T\|$	$\|R\|$	$\|E\|$	$\|E_C\|$	$\|E_A\|$
ID01	800	300	200	300	27425	625	26800
ID02	1200	200	800	200	13862	2000	11862
ID03	800	300	200	300	27429	625	26804
ID04	750	250	250	250	19213	625	18588
ID05	700	200	300	200	14369	2500	11869
ID06	1200	200	800	200	16869	5000	11869
ID07	1500	500	500	500	99663	25000	74663
ID08	1300	300	700	300	28807	2000	26807

Continued on next page

Table 6.1: Node and edge count of instances

	Nodes				Edges		
Instance	$\|V\|$	$\|F\|$	$\|T\|$	$\|R\|$	$\|E\|$	$\|E_C\|$	$\|E_A\|$
ID09	1250	250	750	250	19840	1250	18590
ID10	1500	500	500	500	75941	1250	74691
ID11	800	300	200	300	27814	1000	26814
ID12	1200	200	800	200	13126	1250	11876
ID13	1250	250	750	250	20596	2000	18596
ID14	1200	200	800	200	36876	25000	11876
ID15	1500	500	500	500	76676	2000	74676
ID16	750	250	250	250	21093	2500	18593
ID17	1500	500	500	500	79672	5000	74672
ID18	750	250	250	250	31084	12500	18584
ID19	1250	250	750	250	23584	5000	18584
ID20	1000	500	0	500	77177	2500	74677
ID21	1200	200	800	200	13116	1250	11866
ID22	1000	500	0	500	87175	12500	74675
ID23	800	300	200	300	27806	1000	26806
ID24	1250	250	750	250	43589	25000	18589
ID25	1000	500	0	500	75680	1000	74680
ID26	1000	500	0	500	75302	625	74677
ID27	800	300	200	300	39308	12500	26808
ID28	1300	300	700	300	28805	2000	26805
ID29	1250	250	750	250	23579	5000	18579
ID30	700	200	300	200	12500	625	11875
ID31	1300	300	700	300	28055	1250	26805
ID32	1500	500	500	500	76679	2000	74679
ID33	1500	500	500	500	99672	25000	74672
ID34	1300	300	700	300	51798	25000	26798
ID35	1000	500	0	500	75665	1000	74665
ID36	1200	200	800	200	36878	25000	11878
ID37	1000	500	0	500	87169	12500	74669
ID38	1250	250	750	250	20593	2000	18593
ID39	1250	250	750	250	43593	25000	18593
ID40	1250	250	750	250	19837	1250	18587
ID41	700	200	300	200	24373	12500	11873
ID42	800	300	200	300	29305	2500	26805
ID43	1200	200	800	200	16865	5000	11865
ID44	800	300	200	300	29299	2500	26799
ID45	1000	500	0	500	77165	2500	74665
ID46	1000	500	0	500	75304	625	74679
ID47	700	200	300	200	12874	1000	11874

Continued on next page

Table 6.1: Node and edge count of instances

	Nodes				Edges		
Instance	$\lvert V\rvert$	$\lvert F\rvert$	$\lvert T\rvert$	$\lvert R\rvert$	$\lvert E\rvert$	$\lvert E_C\rvert$	$\lvert E_A\rvert$
ID48	700	200	300	200	24374	12500	11874
ID49	700	200	300	200	12499	625	11874
ID50	1300	300	700	300	31806	5000	26806
ID51	1500	500	500	500	75927	1250	74677
ID52	1300	300	700	300	51804	25000	26804
ID53	750	250	250	250	19591	1000	18591
ID54	700	200	300	200	14375	2500	11875
ID55	1200	200	800	200	13871	2000	11871
ID56	700	200	300	200	12871	1000	11871
ID57	750	250	250	250	19589	1000	18589
ID58	1500	500	500	500	79668	5000	74668
ID59	1300	300	700	300	31802	5000	26802
ID60	750	250	250	250	19219	625	18594
ID61	750	250	250	250	31094	12500	18594
ID62	1300	300	700	300	28051	1250	26801
ID63	800	300	200	300	39301	12500	26801
ID64	750	250	250	250	21079	2500	18579

6.2 Test Environment

All the algorithms described in chapter 5 have been implemented in the C++ language. The C++/C compiler used was gcc version 4.6.1. The software was tested on a Linux-cluster of 14 Intel Xeon E5540 machines (each processor with quad-core, 2.53 GHz kernels, 8MB L3 Cache). Each evaluation run was performed on a single core.

6.3 Evaluation

Since no other approaches for solving the bi-objective ConFL have been previously proposed, we could not evaluate our results against other solutions. Also the well known genetic algorithm NSGA-II which is widely used for several multi-objective combinatorial optimization problem could not be applied here because finding a good evolutionary encoding of solutions is not obvious.

Hence, we will mainly compare the results after the first phase to the final results of our algorithm. Thus, we analyze the quite high computational effort of the second phase and its pay off.

In the following subsection we are going to introduce some common quality metrics that are used to evaluate multi-objective solutions/Pareto fronts.

All statistical evaluations were performed using R and the package emoa [40].

6.3.1 Quality Metrics

To evaluate the performance of bi- and multi-objective algorithms and their found solutions there is the need for some quality metrics to assign numerical values to solution sets in order to compare them to each other. Thus several methods have been proposed in the literature with the intention to measure different preferences, see, e.g., Zitzler et al. [53] for an overview.

Basically one can say quality metrics are a way to map solution sets to the set of real numbers in order to enable us to quantify quality differences between solution sets by applying common mathematical metrics.

Often not only just a single quality indicator is used to assess an solution set but rather a combination of different quality indicators is used to measure the quality of a Pareto front.

6.3.1.1 Unary Indicators

Unary indicators are widely used due to the fact that they assign to each solution set a real value independent of the other solution sets available but therefore often the optimal Pareto front has to be known or a reference point has to be chosen in the solution space to evaluate the indicator values.

Hypervolume indicator The hypervolume indicator [56] is a widely used measure to evaluate bi- and multi-objective solutions because whenever an solution completely dominates another solution, the hypervolume indicator of the latter will be lower than hypervolume indicator of the former. It was first introduced by Zitzler and Thiele [54, 55] who called it the „size of the space covered“. Today it is one of the most popular measures for the performance assessment of multi-objective optimization algorithms.

For a definition of the hypervolume indicator see Zitzler et al. [52]. Figure 6.1 shows an example of the hypervolume indicator $I_H(A)$ of approximation set $A = (x_1, x_2, x_3, x_4)$. The hypervolume is the area covered between the Pareto front A and the reference point r. The drawback is that the value of I_H depends on the selection of the reference point so for each presented solution also the reference point should be noted in order to make the results reproduceable.

There exist several implements to efficiently calculate the hypervolume indicator. In this thesis an R implementation was used to calculate the corresponding values.

6.3.1.2 Binary Indicators

Binary indicators overcome some limitations of unary indicators as they do not need the optimal Pareto front or a reference point but evaluate one solution set against another one. The drawback

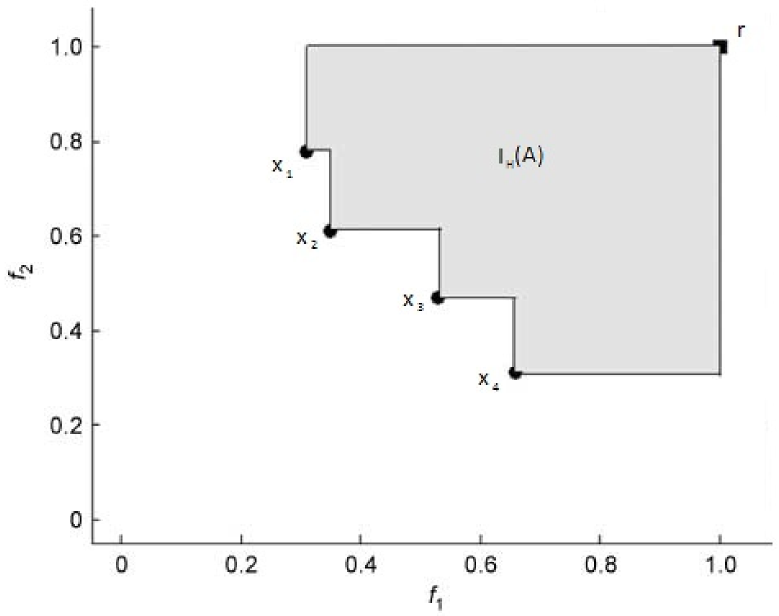

Figure 6.1: Example: Hypervolume indicator $I_H(A)$ of solution set $A = (x_1, x_2, x_3, x_4)$.

is that for n approximation sets one would get $n * (n - 1)$ indicator values which makes the representation and evaluation more difficult.

Epsilon indicator Zitzler et al. [56] also proposed this indicator (see Figure 6.2 for an example) as a measure to compare two Pareto front approximations, A and B. The binary multiplicative epsilon indicator $I_\epsilon(A, B)$ calculates the minimum factor ϵ by which objective vectors from B have to be multiplied in order to move the Pareto front approximation in the objective space that B is weakly dominated by A.

$$I_\epsilon(A, B) = \inf_{\epsilon \in \mathbb{R}} \{\forall x_2 \in B \exists x_1 \in A : x_1 \preceq_\epsilon x_2\} \tag{6.1}$$

The ϵ-dominance relation is defined as:

$$x^1 \preceq_\epsilon x^2 \iff \forall i \in \{1, \ldots, n\} : f_i(x^1) \leq \epsilon * f_i(x^2). \tag{6.2}$$

It is also possible to use the ϵ indicator as an unary indicator in which case a reference set R (optimal Pareto front) is needed:

$$I_\epsilon(A) = I_\epsilon(A, R). \tag{6.3}$$

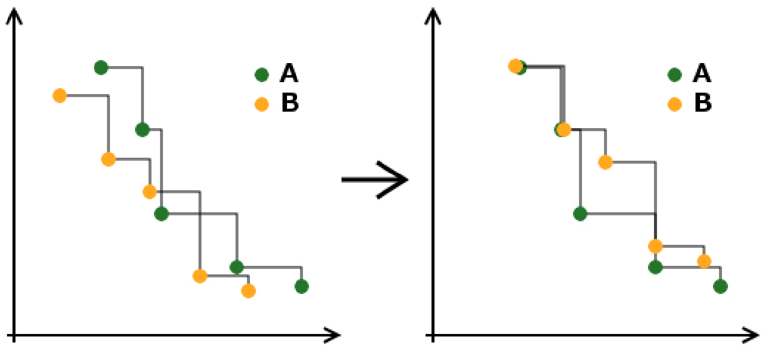

Figure 6.2: Example: Epsilon indicator $I_\epsilon(A, B)$ - shows how B is moved by multiplying its vectors with ϵ that it is weakly dominated by A.

6.4 Computational Results

Due to the design of our algorithm the only parameter which is needed to be selected in advance is the aggregation step size Δ_ω. This value defines with how many different weight sets phase 1 (see Section 5.2) is run before starting phase 2. Paquete and Stützle [44] showed in their work that a $\Delta_\omega < 1/100$ does not provide any significant improvements in the resulting solutions. We also came to that conclusion after experimenting with different Δ_ω values that for lower step sizes there are no more diverging results than compared to greater values.

Also the lower the Δ_ω the higher the run time, which is also a factor to consider. We finally did evaluation runs for the following $\Delta_\omega = \{1/100, 1/20, 1/10, 1/5, 1/2\}$.

After an initial look on the results of the different Δ_ω values the first interesting point were the runtimes. Generally one can observe that the lower the Δ_ω the longer the runtime.

Figure 6.3 compares the runtimes of the no-PLS variant (i.e. only phase 1) with the full two phase approach. Such a runtime behavior is caused by the fact that a small Δ_ω leads to more initial solutions found during phase 1 which helps the Pareto local search in phase 2. This is clearly visible if you look at the runtimes for $\Delta_\omega = 0.50$ where phase 1 only finds up to 3 solutions that can be used as initial solution set for phase 2 which forces the Pareto local search to iterate much more possible solutions. For the full details of all runtimes see Table A.1 in Appendix A. Also very interesting is Figure 6.4 which shows boxplots of the runtimes of phase 1 and the runtimes of phase 2 in minutes.

Next to the runtimes, we also compared for each step size the hypervolume values between the no-PLS and PLS variant.

Figure 6.5 shows exactly what we assumed. The smaller the Δ_ω the better the solutions found in phase 1. This is easily explained because a lower value of Δ_ω automatically increases the number of runs of the construction heuristic which leads to a higher number of initial solution for Phase 2 (e.g., if $\Delta_\omega = 0.05$ this means there are $1/0.05 = 20$ steps from 0 to 1 and the construction heuristic will be run $1/0.05 + 1 = 21$ times) and the higher the number of initial solutions the better is the approximation of the Pareto front in Phase 1. Furthermore, it can be

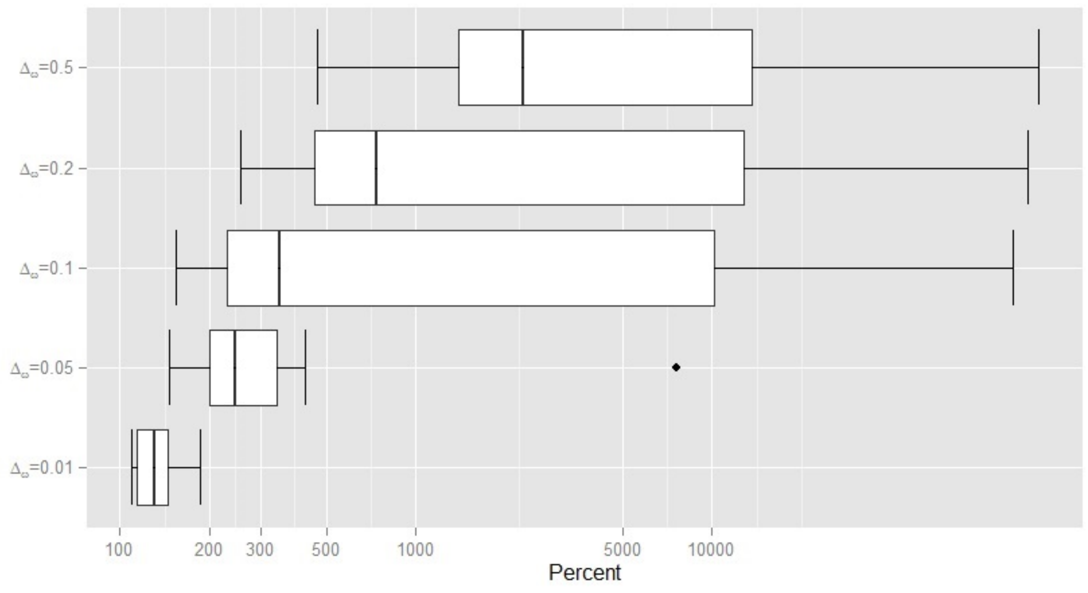

Figure 6.3: Comparison of runtime differences between the no-PLS variant and the PLS variant depending on Δ_ω. The x-axis shows runtime in percent on the basis of the no-PLS variant which represents 100% and the PLS variant has in all cases $> 100\%$, e.g., for $\Delta_\omega = 0.05$ the PLS variant needed on average 2.5 times the time to complete than the no-PLS variant.

seen ,e.g., for $\Delta_\omega = 0.5$ that the hypervolume of the full algorithm is on average double the size the variant without the Pareto local search. This difference gets lower and lower the smaller the Δ_ω.

Table 6.2: Hypervolume values of instances in percent in relation to best solution found for each Δ_ω value with and without PLS. The underlined values represent the instances where the time limit of 2 hours was reached. The reference point used was individually selected for each instance and composed by the maximum of the cost and the maximum of the lost revenue. This explains why some of the values are 0 because in such a case there were only 2 points on the Pareto front which either had the same value for the costs or the lost revenue.

	no PLS, $\Delta_\omega =$					with PLS, $\Delta_\omega =$				
Instance	0.01	0.05	0.10	0.20	0.50	0.01	0.05	0.10	0.20	0.50
ID01	99.5	93.8	90.3	85.1	54.8	100	98.3	96.3	95.2	90.9
ID02	99	94	87.2	73.7	53.1	100	98	95.2	91.2	90.6
ID03	99.4	95.1	89.1	79.8	49.3	100	98.8	96.7	93.4	90.2

Continued on next page

Table 6.2: Hypervolume values of instances in percent in relation to best solution found for each Δ_ω value with and without PLS. The underlined values represent the instances where the time limit of 2 hours was reached. The reference point used was individually selected for each instance and composed by the maximum of the cost and the maximum of the lost revenue. This explains why some of the values are 0 because in such a case there were only 2 points on the Pareto front which either had the same value for the costs or the lost revenue.

	no PLS, $\Delta_\omega =$					with PLS, $\Delta_\omega =$				
Instance	0.01	0.05	0.10	0.20	0.50	0.01	0.05	0.10	0.20	0.50
ID04	96.2	63.2	0	0	0	$\underline{100}$	$\underline{88.4}$	76.2	76.2	76.2
ID05	99.3	95.2	87.6	72.2	44.1	100	98.7	96.2	93.3	89.4
ID06	99.2	93.4	88.1	73.4	43.4	100	98.3	96.1	93.7	90.3
ID07	98	65.2	0	0	0	$\underline{100}$	$\underline{86.1}$	$\underline{74.9}$	$\underline{74.9}$	$\underline{74.9}$
ID08	99.6	95.4	92.6	85	61.3	100	98.8	98	96.1	92.7
ID09	95.5	62.6	0	0	0	$\underline{100}$	$\underline{90.8}$	76.9	76.9	76.9
ID10	97.7	65.3	0	0	0	$\underline{100}$	$\underline{87.6}$	$\underline{76.5}$	$\underline{76.4}$	$\underline{76.4}$
ID11	99.7	95.6	90.6	83.5	51	100	98.7	96.9	94.4	89.8
ID12	99.1	93.6	89.8	80.6	49.6	100	97.7	96.4	93.6	90
ID13	96.1	64	0	0	0	$\underline{100}$	$\underline{88.4}$	73.9	73.9	73.9
ID14	98.5	95	88.8	80.1	45.9	100	98.8	97.9	96.1	91.9
ID15	$\underline{100}$	78.4	0	0	0	$\underline{100}$	$\underline{88.8}$	$\underline{92.1}$	$\underline{92.1}$	$\underline{92.1}$
ID16	94.3	62.6	0	0	0	$\underline{100}$	$\underline{88.4}$	73.9	73.9	73.9
ID17	97.8	65.1	0	0	0	$\underline{100}$	$\underline{87}$	$\underline{77.2}$	$\underline{77.1}$	$\underline{77.1}$
ID18	94.4	62.7	0	0	0	$\underline{100}$	$\underline{89.2}$	72.7	72.7	72.7
ID19	93.5	62.2	0	0	0	$\underline{100}$	$\underline{90.9}$	75.9	75.9	75.9
ID20	97.4	64.8	0	0	0	$\underline{100}$	$\underline{90.2}$	$\underline{76.1}$	$\underline{76.1}$	$\underline{76.1}$
ID21	98.9	92.6	88.1	76.5	41.5	100	97.3	95.4	92.7	86.2
ID22	97	64	0	0	0	$\underline{100}$	$\underline{88.4}$	73.9	73.9	$\underline{73.9}$
ID23	99.4	93.9	87.4	80.2	46.8	100	98.8	96.6	94.8	92.1
ID24	95.2	61.7	0	0	0	$\underline{100}$	$\underline{88.7}$	75.5	75.5	75.5
ID25	98.3	65.5	0	0	0	$\underline{100}$	$\underline{88.7}$	75.9	75.9	75.9
ID26	98	65.7	0	0	0	$\underline{100}$	$\underline{89.4}$	$\underline{75.6}$	$\underline{75.6}$	$\underline{75.6}$
ID27	98.7	91.4	88.6	79.8	42.9	100	99.2	97.6	95.3	91.9
ID28	98.7	95	90.4	81.1	55.8	100	98.6	96.7	95.1	91.6
ID29	94.7	62.3	0	0	0	$\underline{100}$	$\underline{89.1}$	75.2	75.2	75.2
ID30	99.4	94.2	88.1	82.3	57.5	100	97.4	94.5	94	90.2
ID31	99.3	95	91	81.1	56.1	100	98.2	96.9	94.3	91.6
ID32	99.4	65.8	0	0	0	$\underline{100}$	$\underline{88.2}$	$\underline{78.3}$	$\underline{78.3}$	$\underline{78.3}$
ID33	98.5	65.4	0	0	0	$\underline{100}$	$\underline{89}$	$\underline{75.7}$	$\underline{75.7}$	$\underline{75.7}$
ID34	99	92	85.6	80.8	41.6	100	97.9	96.5	94.9	87.9

Continued on next page

Table 6.2: Hypervolume values of instances in percent in relation to best solution found for each Δ_ω value with and without PLS. The underlined values represent the instances where the time limit of 2 hours was reached. The reference point used was individually selected for each instance and composed by the maximum of the cost and the maximum of the lost revenue. This explains why some of the values are 0 because in such a case there were only 2 points on the Pareto front which either had the same value for the costs or the lost revenue.

	no PLS, $\Delta_\omega =$					with PLS, $\Delta_\omega =$				
Instance	0.01	0.05	0.10	0.20	0.50	0.01	0.05	0.10	0.20	0.50
ID35	97.9	65.1	0	0	0	100	90.4	75.3	75.3	75.3
ID36	98.5	91.2	85.3	74.3	34.3	100	98.5	97	94.3	90.3
ID37	97.9	65.9	0	0	0	100	90.5	74.6	74.6	74.6
ID38	96.3	64.3	0	0	0	100	91.8	79.9	79.9	79.9
ID39	94.4	63.2	0	0	0	100	90.3	73.8	73.8	73.8
ID40	96.3	64	0	0	0	100	91.6	77.8	77.8	77.8
ID41	99.2	94.8	88.4	78.2	47.5	100	98.4	96.6	94	88.8
ID42	99.1	92.8	88.4	80.7	41.9	100	98.5	96.9	95.5	89.7
ID43	98.7	95.6	91.2	79.9	51.4	100	99	96.8	94.5	90.9
ID44	98.9	95.3	90.6	85.6	53.1	100	99	98	96.8	92.4
ID45	98.2	65.5	0	0	0	100	90.1	74.8	74.8	74.8
ID46	98.1	65.5	0	0	0	100	84.3	76.1	76.1	76.1
ID47	99.5	94	87.5	75.9	47	100	99.1	96.6	95	91.7
ID48	99	93	86.2	75.8	47	100	98.7	97.1	94.1	90.7
ID49	98.4	91.7	86.7	71.8	47.4	100	98.1	95.9	92.5	89.4
ID50	99.2	91.6	87.8	82	49.3	100	97.6	96.2	94.4	89.7
ID51	97.7	64.6	0	0	0	100	88.5	76.3	76.3	76.3
ID52	99	92.2	86.9	76.4	38.9	100	98.6	96.8	95.2	89.8
ID53	93.9	62.3	0	0	0	100	88.5	73.6	73.6	73.6
ID54	99	92.8	87.1	78	46.2	100	98	96.5	93.8	90.5
ID55	98.9	94.9	88.9	79.2	59.7	100	98.8	96.7	94.3	92.3
ID56	99.4	95	90.5	78.4	45.7	100	99.2	97.7	93.8	90.7
ID57	94.4	62.8	0	0	0	100	90.2	73.9	73.9	73.9
ID58	98.8	66.2	0	0	0	100	87.1	76.8	76.8	76.8
ID59	99.2	91.8	87.9	79.8	42	100	98.2	95.8	93.4	88.2
ID60	95.5	63.7	0	0	0	100	88.8	76.5	76.5	76.5
ID61	94.8	62.8	0	0	0	100	87.4	72.7	72.7	72.7
ID62	99.4	92.3	89.7	81.3	38.1	100	98.3	96.9	94.7	88.8
ID63	98.9	94	89.4	82.2	45.5	100	98.9	96.7	94.8	89.3
ID64	94.1	63.2	0	0	0	100	89.4	72.1	72.1	72.1

Table 6.2 shows the calculated Hypervolume values for each instance and Δ_ω value. The un-

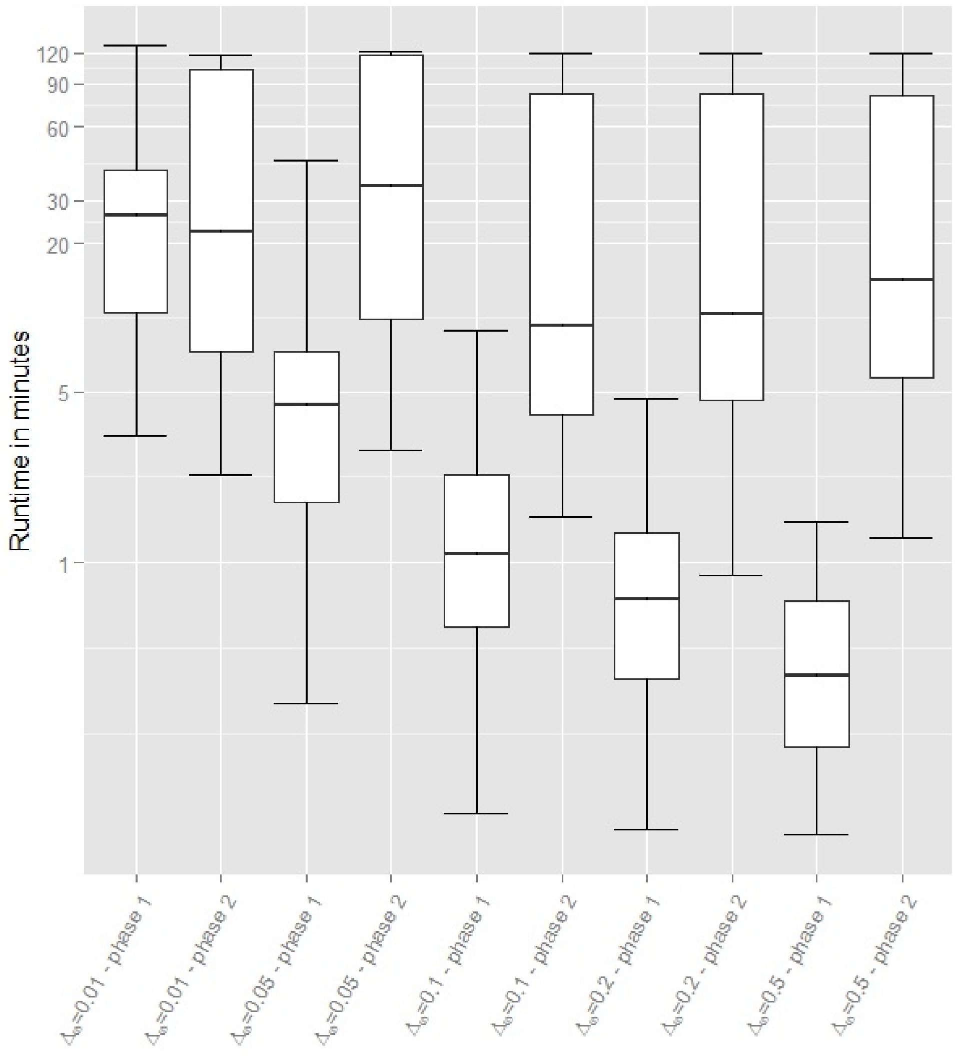

Figure 6.4: Comparison of runtimes of phase 1 and phase 2 of the algorithm depending on Δ_ω.

derlined values represent the instances where the algorithm didn't finish in the given timeframe of 2 hours. It is clearly visible that the best hypervolume value for each instance in either the no PLS or the PLS variant is the found with the lowest Δ_ω value which completed in the given timeframe. Figure 6.6 displays the difference between best solutions found $(\mathrm{PLS} - \mathrm{noPLS})$. One might wonder why for some instances the no PLS solution is equal compared to the PLS variant (e.g., Instance ID15). The reason why this happened is because we had a maximum runtime limit and in each case where the no PLS variant equals the PLS variant is that the Pareto local search did not finish in the given timeframe. In case the Pareto local search would have finished, the $\Delta_\omega = 0.01$ solution would be at least as good as the no PLS variant, most likely

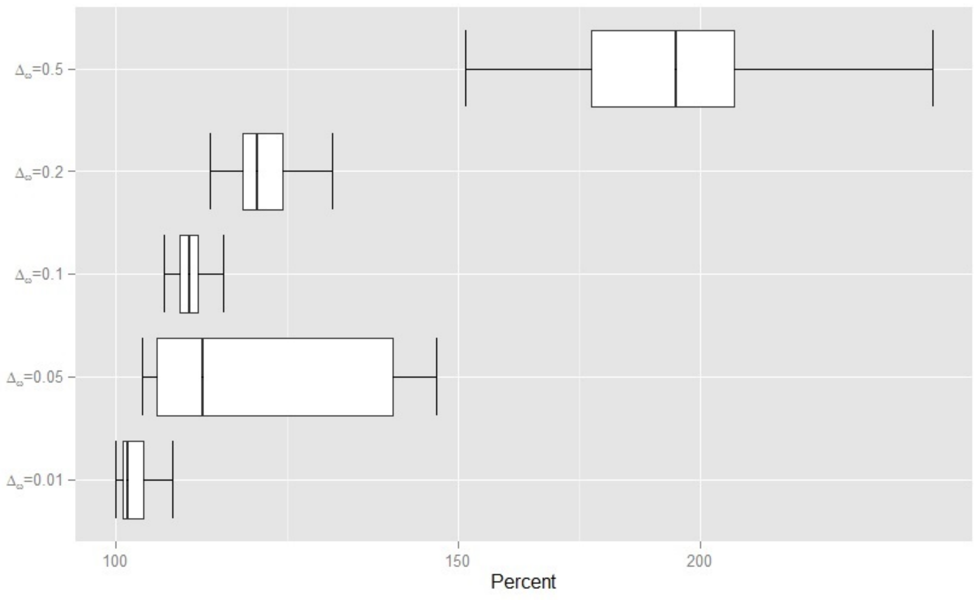

Figure 6.5: Comparison of hypervolume values between the no-PLS variant (100%) and the PLS variant separate for all Δ_ω we used for evaluation.

better, judged by the derived results of our tests.

Figure 6.7 shows a scatter plot of the differences in percent of the hypervolume values versus the runtimes for the best no PLS and the best PLS solution. This graphic gives a good summary of the facts mentioned before. The average difference in the hypervolume values is relatively low thus making the solutions found by additional application of Pareto local search not significantly better compared to the no PLS solutions. There are some outliers on the right hand side of the figure which indicate that for some instances the hypervolume difference of the PLS variant is 100% compared to the no PLS variant. This can be explained by Figure 6.8, where the average of the Pareto front size for the no PLS variant of $\Delta_\omega = 0.5$ is 2.5 solutions per instance. This means there are instances with only 2 solutions on the Pareto front thus resulting in a hypervolume value of 0, which leads to a 100% difference (see figure 6.7). The differences of runtimes lead us to assume that in general the Pareto local search takes significantly more time than only phase 1 with the addition that the hypervolume increase is just marginal. However, when looking at the calculated Pareto fronts (see Appendix B), the distribution of points on the fronts are much better distributed.

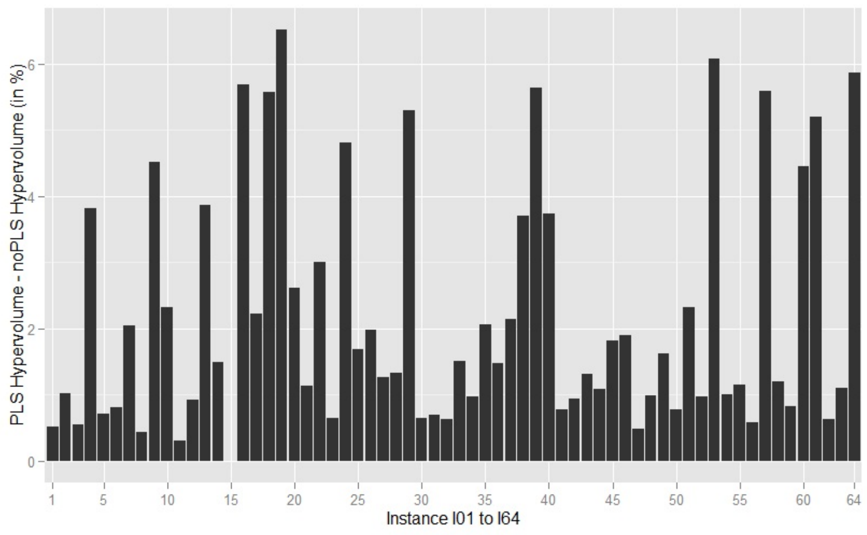

Figure 6.6: Difference of hypervolume values between the best no-PLS and the best PLS variant for each instance.

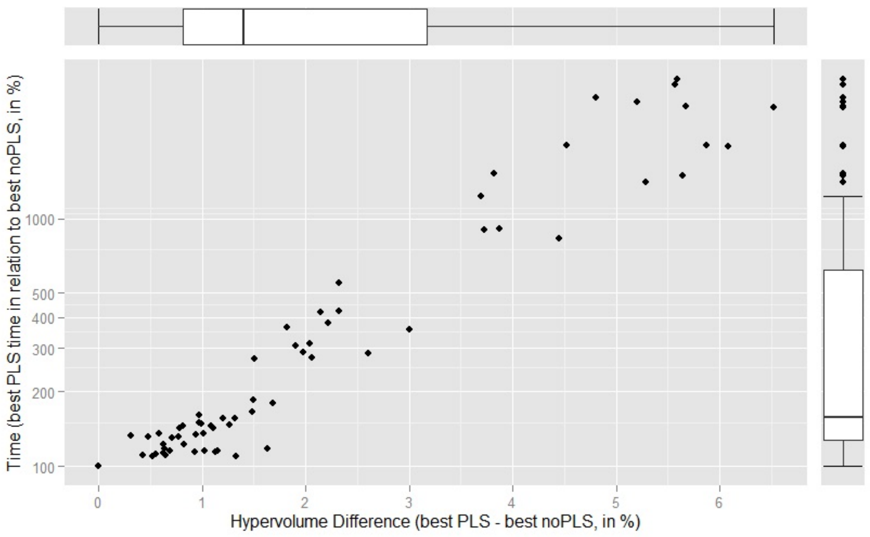

Figure 6.7: Scatter plot of the difference of hypervolume values versus the difference of run-times in percent for the best no PLS and PLS solutions for each instance.

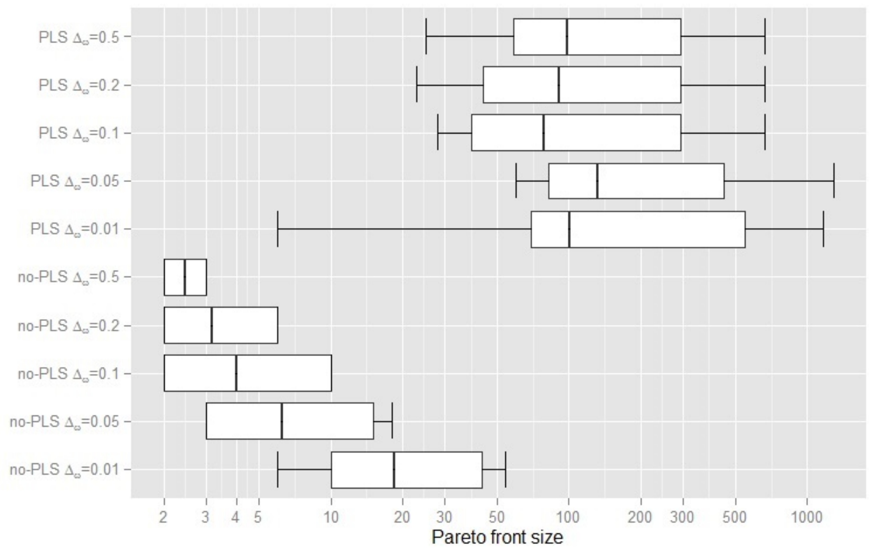

Figure 6.8: Boxplots of Pareto front sizes for each Δ_ω used.

CHAPTER 7

Conclusion

In this thesis we tackled the bi-objective connected facility location (BoConFL) problem with a two-phase local search algorithm. We chose the a two-phase local search algorithm because of the successful applications of local search approaches for the single objective ConFL as well as their efficiency for combinatorial optimization problems and the Pareto Local Search because it showed in various other bi-objective problems (e.g. bi-objective TSP) to further improve the solutions found by the construction heuristic to approximate the Pareto front even better.

After we implemented a suitable construction heuristic based on a greedy algorithm we tested each selected neighborhood structures with a simple local search. We came to the conclusion that for our problem instances the next improvement strategy was the most reliable one. Next to evaluating the neighborhood structures we extended our basic local search to a variable neighborhood decent (VND) in order to take advantage of a lager neighborhood coverage. This led to good starting solutions for phase 2 - the Pareto local search.

Results from our computational study show that phase 1 of our algorithm required significantly less time compared to phase 2. The additional runtime needed by the PLS required to complete a run on an instance compared to the improvement in the quality of the Pareto front achieved is rather insignificant. This fact is clearly visible when comparing the runtimes of only phase 1 and the corresponding hypervolume values to the runtime and hypervolume values of the whole two-phase algorithm. For many instances the PLS did not finish in the given timeframe which also leads to the conclusion that the Pareto local search improves the quality of the Pareto front but also has high costs in terms of runtime, which can be seen for some instances which had a solution in phase 1 but in the given time frame the Pareto local search did not manage to complete its run.

7.1 Future work

As there does not exist an exact method to solve the BoConFL such an implementation would be interesting in order to have a fixed comparison for other approaches like our implementation.

Furthermore the VND could be refined to a VNS which can then be improved by applying a Pareto local search. Our results show that the Pareto local search would need some improvements or modifications in order to cover to objective space more efficiently.

To use a VNS in phase 1 it would also be adviceable to not just increase the weights by Δ_ω stepwise but rather start with the two extreme settings of $\omega_1 = 0, \omega_2 = 1$ and $\omega_1 = 1, \omega_2 = 0$. The next weight set of phase 1 to evaluate should then be $\omega_1 = 0.5, \omega_2 = 0.5$ and then as long as the time limit of the VNS is not reached each of these intervals between these points should be further halved in order to get an evenly distributed set of solutions at the end of phase 1.

Following this improvement of phase 1 also the PLS could be refined in a similar way. Therefore the PLS should not just take a random solution and tries to further optimize it but rather do some kind of Pareto front analysis to choose a solution to improve. Therefore it could be interesting to use some kind of fast clustering algorithm on the points of the current Pareto front and cluster the points in order so find areas where there are only small clusters which should indicate an area which is not well researched and take a solution from there to start its improvement. This procedure should also lead to a better distributed Pareto front approximation then just using random points.

APPENDIX A

Running Time Table

Table A.1: Runtimes (format: hh:mm::ss) of instances for each Δ_ω value with and without PLS - only available for instances which finished in the given timeframe. The underlined values represent the running time of the best no PLS and PLS solution.

	no PLS, $\Delta_\omega =$					with PLS, $\Delta_\omega =$				
Instance	0.01	0.05	0.10	0.20	0.50	0.01	0.05	0.10	0.20	0.50
ID01	1:11:08.3	12:42.7	5:09.6	2:38.1	1:05.1	1:18:01.1	25:38.2	8:30.4	7:36.5	14:09.5
ID02	23:08.5	5:29.0	2:32.2	59.6	28.8	26:36.4	11:21.9	5:43.6	9:08.9	4:38.6
ID03	1:08:09.4	12:44.3	6:18.1	2:14.9	19.6	1:15:55.9	23:22.9	14:14.8	12:17.5	15:19.4
ID04	7:54.6	18.7	5.7	5.1	4.8	2:15.8	2:8	57:40.8	59:41.1	1:02:17.8
ID05	13:29.4	2:39.3	1:13.6	29.9	10.3	17:28.6	5:38.3	3:33.5	3:02.5	2:56.7
ID06	12:43.8	2:49.3	1:9	32.6	6.6	18:30.5	9:31.9	4:38.8	4:05.7	5:11.5
ID07	38:48.4	8:47.5	1:30.9	1:16.5	1:09.9	2:01:41.9	2:34.8	2:14.9	2:8.7	2:37.4
ID08	1:43:40.4	20:13.4	8:47.9	3:30.0	55.0	1:55:15.4	31:38.9	13:31.9	8:54.5	10:16.6
ID09	6:04.9	16.0	6.3	5.9	5.5	2:24.5	2:8.7	44:55.5	46:10.0	46:50.8
ID10	28:29.0	7:02.8	36.5	33.8	34.0	2:39.1	2:01:35.5	2:8.8	2:41.2	2:27.2
ID11	34:3	7:28.7	4:36.5	2:09.7	34.8	45:17.8	18:03.8	10:37.8	8:28.1	11:25.7
ID12	28:41.7	5:13.7	2:43.6	1:19.1	14.7	32:48.5	11:17.7	5:35.3	6:38.0	4:14.4
ID13	13:08.8	2:56.2	6.5	5.5	5.1	2:8.9	2:18.9	37:55.6	37:46.8	38:09.2
ID14	15:1	3:31.5	58.6	32.7	14.2	27:54.8	13:26.0	5:4	6:16.5	6:03.1
ID15	2:08:39.1	44:02.1	39.8	31.9	29.8	2:08:39.1	2:23.1	2:2.6	2:5.3	2:16.6
ID16	4:13.2	43.3	8.8	7.3	6.7	2:10.3	2:0.7	19:26.3	18:58.7	20:33.6
ID17	31:56.5	2:20.7	51.5	51.5	49.4	2:34.6	2:01:26.4	2:15.8	2:8.1	2:40.1
ID18	3:27.8	1:02.4	14.0	13.0	11.8	2:10.1	2:0.4	8:51.3	9:59.2	9:37.4
ID19	4:15.1	1:15.9	8.1	6.8	6.1	2:5.6	2:3	49:45.2	47:24.2	50:07.1
ID20	41:57.5	6:21.8	51.9	48.3	43.8	2:9.2	2:49.9	2:22.2	2:23.5	2:11.0
ID21	23:38.5	4:32.9	2:11.4	1:08.7	19.2	26:49.7	7:59.9	4:15.8	3:22.4	4:19.0
ID22	33:40.6	5:43.5	1:44.2	1:35.4	1:28.6	2:4.5	2:52.8	1:59:00.9	1:53:16.0	2:30.1
ID23	37:49.7	6:58.3	3:09.4	45.0	41.4	44:26.1	16:28.0	7:17.3	5:45.8	9:29.2

Continued on next page

Table A.1: Runtimes (format: hh:mm::ss) of instances for each Δ_ω value with and without PLS - only available for instances which finished in the given timeframe. The underlined values represent the running time of the best no PLS and PLS solution.

	no PLS, $\Delta_\omega =$					with PLS, $\Delta_\omega =$				
Instance	0.01	0.05	0.10	0.20	0.50	0.01	0.05	0.10	0.20	0.50
ID24	3:53.9	1:10.6	16.0	12.3	9.1	2:4.6	2:5.6	24:00.2	27:05.3	25:37.4
ID25	1:07:52.8	20:56.0	45.1	41.0	40.5	2:01:30.5	2:29.9	1:29:04.8	1:26:41.1	1:30:41.0
ID26	41:40.2	18:51.9	38.9	32.7	32.5	2:19.5	2:43.1	2:1.3	2:5.6	2:31.4
ID27	29:01.4	6:19.1	2:54.5	1:20.7	43.4	42:30.6	19:08.8	9:32.8	5:06.1	6:54.5
ID28	1:30:15.6	20:01.1	8:50.6	4:39.6	1:06.3	1:38:57.6	29:22.5	14:03.2	19:30.8	14:19.3
ID29	8:31.6	1:28.5	8.4	7.7	7.2	2:17.1	2:4	41:05.7	41:52.3	41:29.0
ID30	21:54.5	3:52.7	1:30.2	51.2	21.7	24:12.5	7:43.7	3:49.0	2:41.3	5:33.9
ID31	1:07:30.2	12:03.8	6:09.5	2:47.5	1:01.4	1:17:26.3	22:12.5	11:37.4	11:42.4	10:30.8
ID32	1:38:32.4	40:27.2	41.9	33.9	31.0	2:8.9	2:01:32.0	2:1	2:14.4	2:21.7
ID33	44:18.2	8:29.6	1:23.7	1:11.6	1:03.6	2:22.4	2:01:16.9	2:6.6	2:1	2:13.8
ID34	39:08.4	6:07.9	3:17.0	2:29.8	29.4	1:02:50.0	23:46.2	13:13.1	11:53.2	20:25.2
ID35	43:43.4	7:07.3	40.9	39.2	38.5	2:18.0	2:47.2	2:9.7	2:13.7	2:7.7
ID36	15:55.5	3:04.9	1:32.8	46.3	20.4	26:18.1	13:03.7	6:08.5	3:25.4	7:21.6
ID37	28:34.4	5:55.6	1:38.3	1:24.6	1:21.4	2:2.1	2:01:28.1	1:20:18.4	1:20:56.9	1:19:26.1
ID38	9:47.8	1:48.8	6.9	6	5	2:14.6	2:14.0	1:02:42.5	1:04:29.5	1:01:54.0
ID39	8:02.2	1:41.2	23.3	16.7	16.7	2:10.9	2:6.2	23:00.2	24:29.9	27:35.8
ID40	13:22.2	19.8	6.2	5.7	5.4	2:9.8	2:19.5	1:48:06.6	1:49:47.9	1:53:17.9
ID41	10:43.5	2:28.3	1:05.3	27.7	16.3	15:11.5	8:58.4	3:46.9	2:31.5	2:38.3
ID42	30:23.0	5:56.6	2:08.1	1:19.6	17.3	40:45.5	17:26.0	6:04.2	6:09.1	5:24.5
ID43	11:05.4	2:40.8	1:01.9	47.9	11.5	17:14.2	6:46.5	3:13.7	3:51.7	4:33.9
ID44	25:59.0	4:39.7	2:13.0	1:30.8	13.3	37:51.2	14:32.7	6:55.5	5:33.3	3:58.6
ID45	33:00.9	8:44.4	1:02.7	55.4	52.7	2:15.8	2:7.7	2:20.0	2:6.1	2:5.1
ID46	39:28.5	1:35.3	55.4	47.1	47.5	2:01:14.1	2:02:01.8	2:12.2	2:10.3	2:11.8

Continued on next page

Table A.1: Runtimes (format: hh:mm::ss) of instances for each Δ_ω value with and without PLS - only available for instances which finished in the given timeframe. The underlined values represent the running time of the best no PLS and PLS solution.

	no PLS, $\Delta_\omega =$					with PLS, $\Delta_\omega =$				
Instance	0.01	0.05	0.10	0.20	0.50	0.01	0.05	0.10	0.20	0.50
ID47	$\underline{8:35.0}$	2:11.0	1:05.5	32.5	11.0	$\underline{11:13.3}$	5:20.0	3:08.1	3:59.7	2:48.7
ID48	$\underline{9:15.4}$	1:28.2	52.5	21.8	11.0	$\underline{13:45.6}$	5:57.4	2:25.5	1:15.1	2:28.4
ID49	$\underline{16:51.0}$	3:26.2	1:44.8	34.4	10.7	$\underline{19:51.0}$	6:20.1	3:53.3	3:33.9	3:56.3
ID50	$\underline{43:42.9}$	6:42.1	3:18.4	1:22.5	43.5	$\underline{57:21.6}$	23:41.8	17:32.7	12:02.4	12:43.3
ID51	$\underline{22:11.7}$	1:44.3	39.0	35.6	31.4	$\underline{2:01:32.7}$	2:6.6	2:27.4	2:13.2	2:3.8
ID52	$\underline{44:36.0}$	8:00.6	3:35.4	1:40.3	32.6	$\underline{1:06:51.4}$	29:14.3	12:13.3	8:07.1	16:09.8
ID53	$\underline{6:07.6}$	1:02.6	7.1	6.6	6	$\underline{2:4.9}$	2:2.5	31:18.4	31:59.7	30:27.4
ID54	$\underline{12:55.9}$	2:14.2	1:05.4	21.5	21.1	$\underline{17:27.9}$	5:19.8	2:44.5	2:03.8	1:37.3
ID55	$\underline{25:53.9}$	4:15.2	1:57.5	43.7	18.8	$\underline{29:46.6}$	8:47.6	3:57.2	3:34.9	3:01.4
ID56	$\underline{9:42.4}$	1:46.9	1:07.3	28.6	9.8	$\underline{13:11.2}$	4:39.5	2:43.2	3:1	3:13.2
ID57	$\underline{3:17.4}$	34.4	6.3	5.8	5.7	$\underline{2:1.3}$	2:5.5	22:53.6	22:50.7	25:34.2
ID58	$\underline{1:18:29.7}$	22:27.8	55.0	50.7	47.3	$\underline{2:02:08.4}$	2:17.6	2:27.3	2:18.5	2:45.2
ID59	$\underline{34:17.2}$	6:34.9	2:55.7	1:15.8	28.9	$\underline{42:02.6}$	17:43.0	10:59.1	10:15.1	6:52.5
ID60	$\underline{14:24.2}$	2:26.3	5.8	4.9	4.7	$\underline{2:11.2}$	2:4.8	32:08.6	31:34.5	32:54.2
ID61	$\underline{4:02.6}$	44.3	13.3	13.2	13.0	$\underline{2:9}$	56:04.4	6:54.0	8:10.5	9:06.8
ID62	$\underline{1:01:40.1}$	11:39.3	5:10.1	2:03.7	29.8	$\underline{1:09:36.9}$	22:44.3	10:19.3	8:31.2	13:44.0
ID63	$\underline{30:40.2}$	6:11.9	3:15.0	1:36.4	59.9	$\underline{43:32.4}$	18:21.6	7:18.7	5:32.1	10:14.3
ID64	$\underline{6:04.3}$	1:14.2	7.1	6.5	5.7	$\underline{2:4.2}$	2:0.8	13:21.8	13:50.8	13:09.1

APPENDIX B

Pareto Fronts

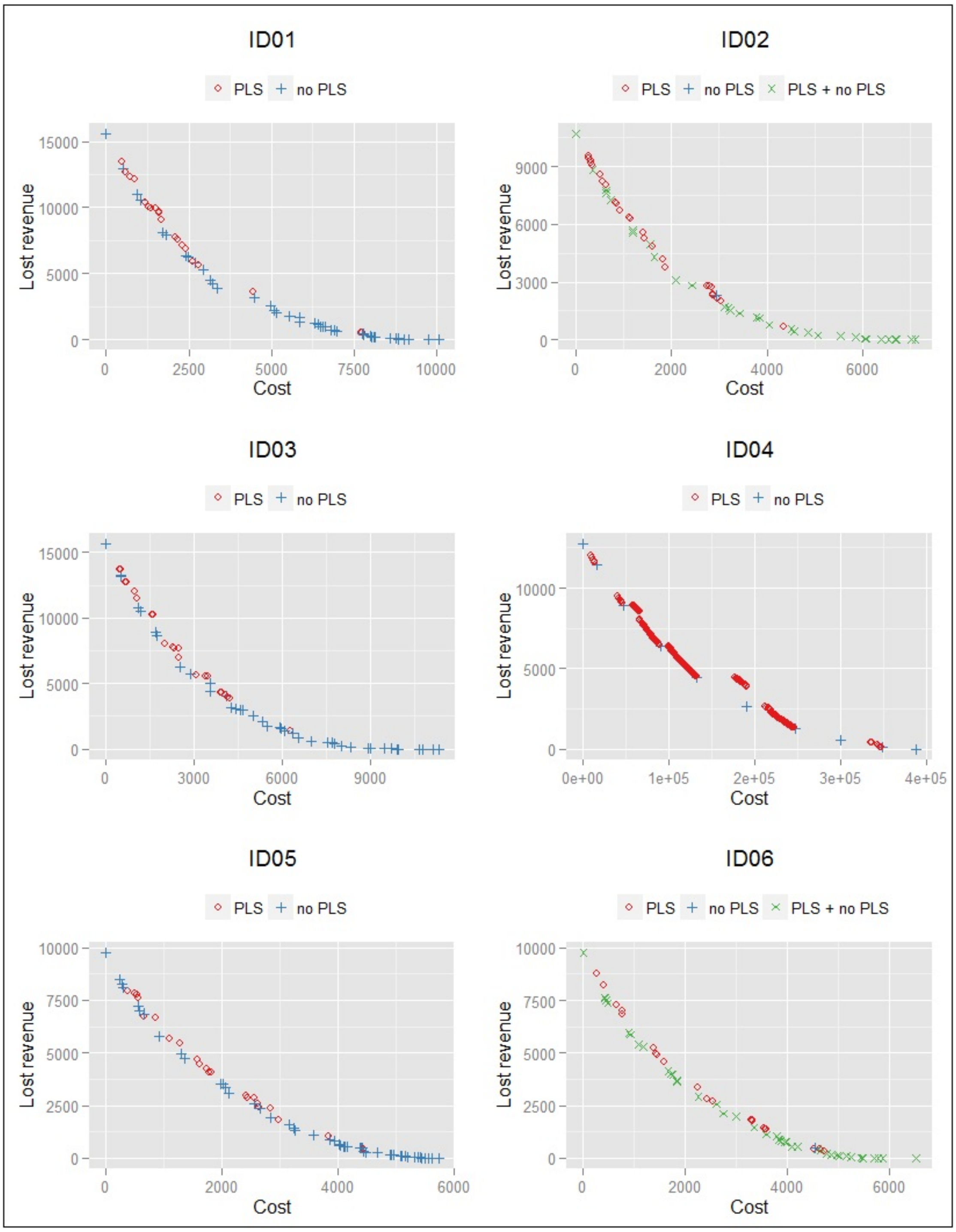

Figure B.1: Pareto Fronts of best no PLS and best PLS solution.

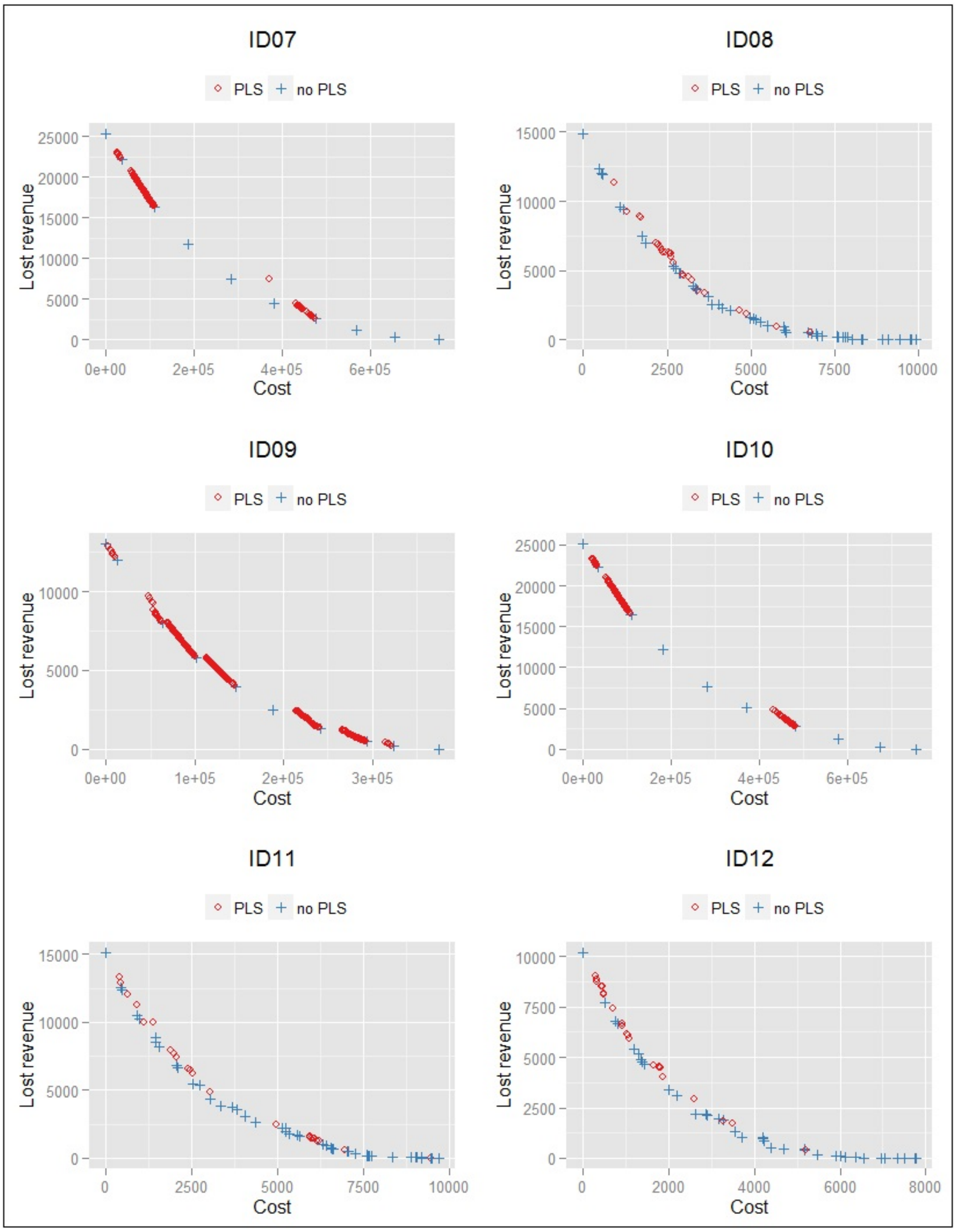

Figure B.2: Pareto Fronts of best no PLS and best PLS solution.

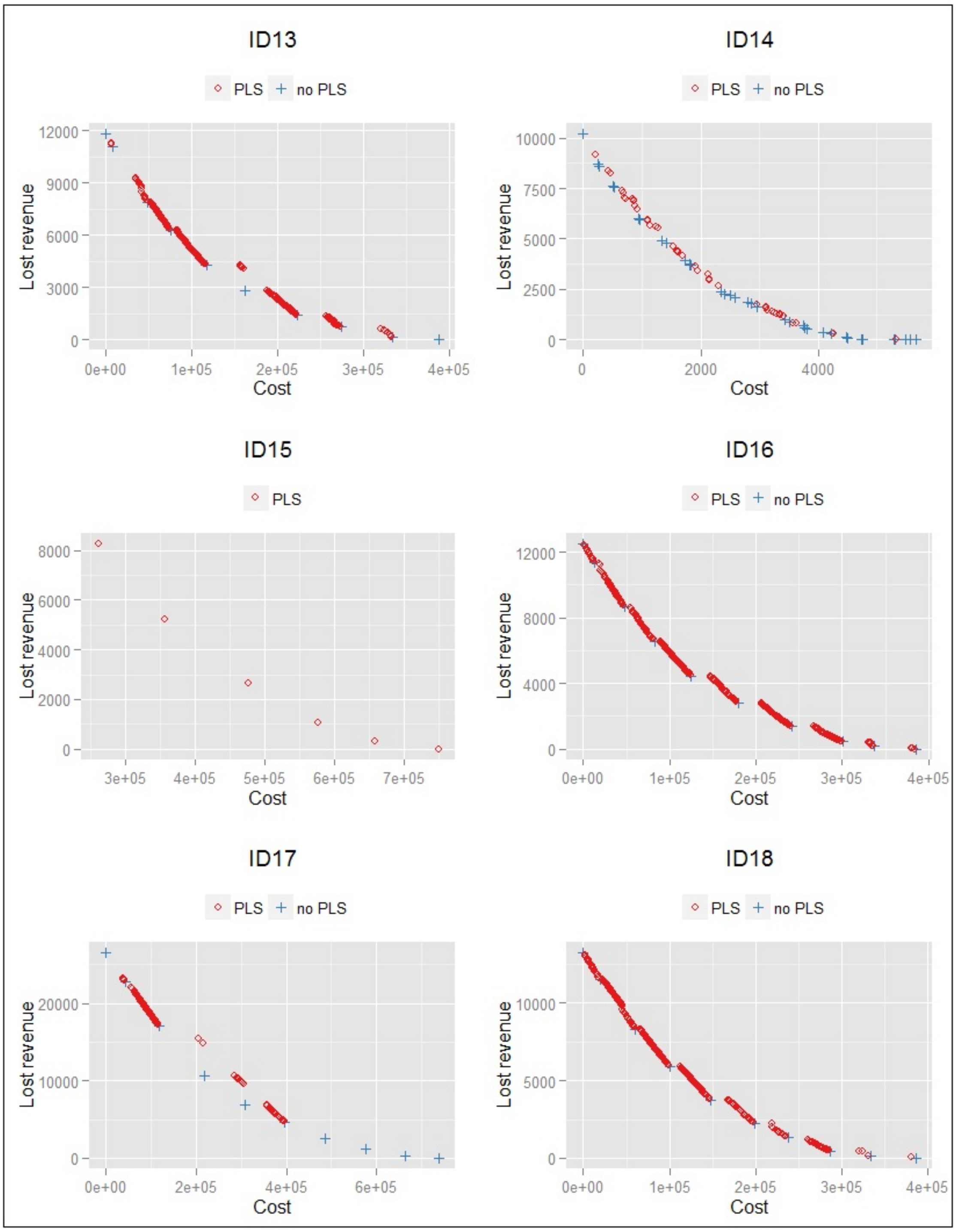

Figure B.3: Pareto Fronts of best no PLS and best PLS solution.

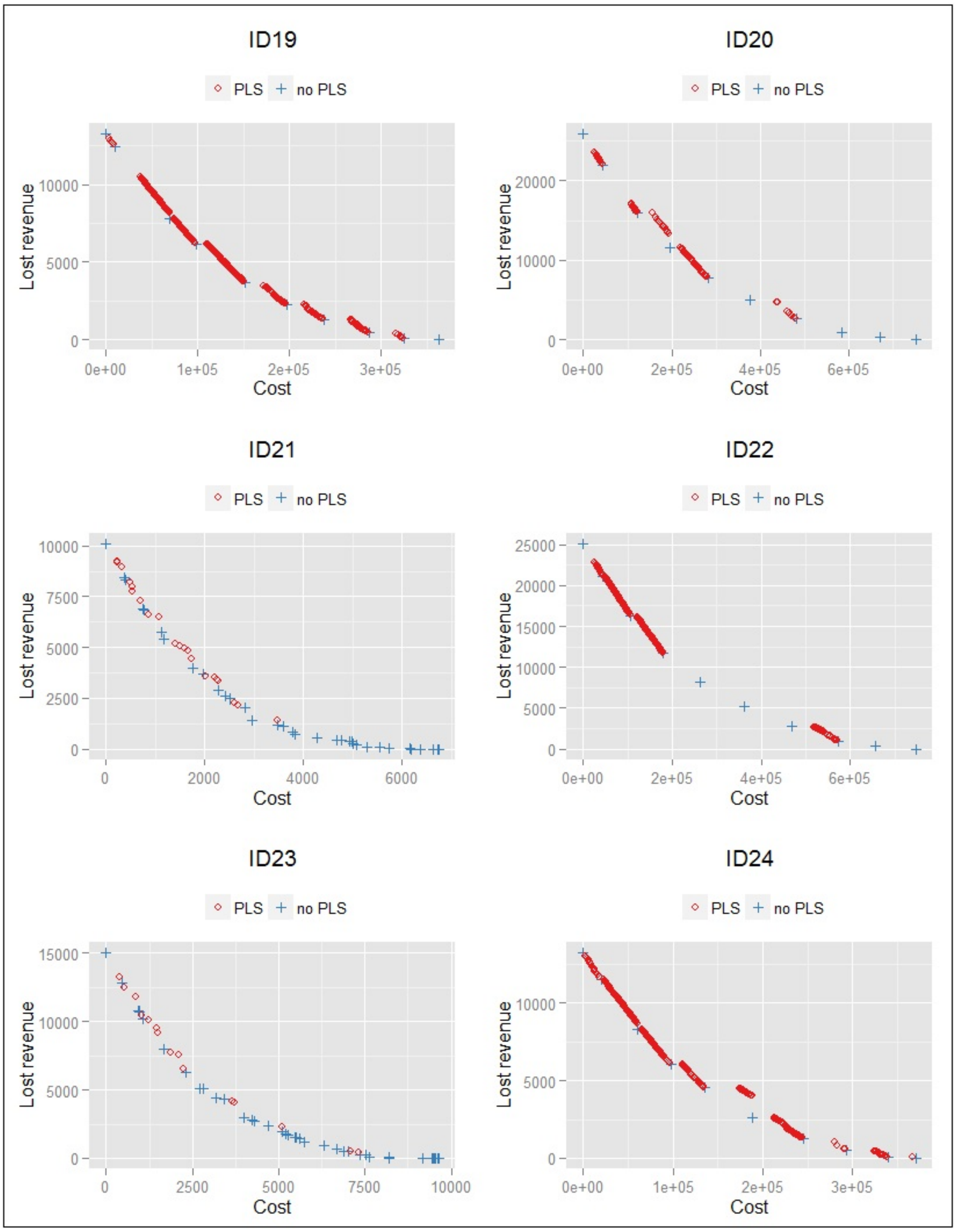

Figure B.4: Pareto Fronts of best no PLS and best PLS solution.

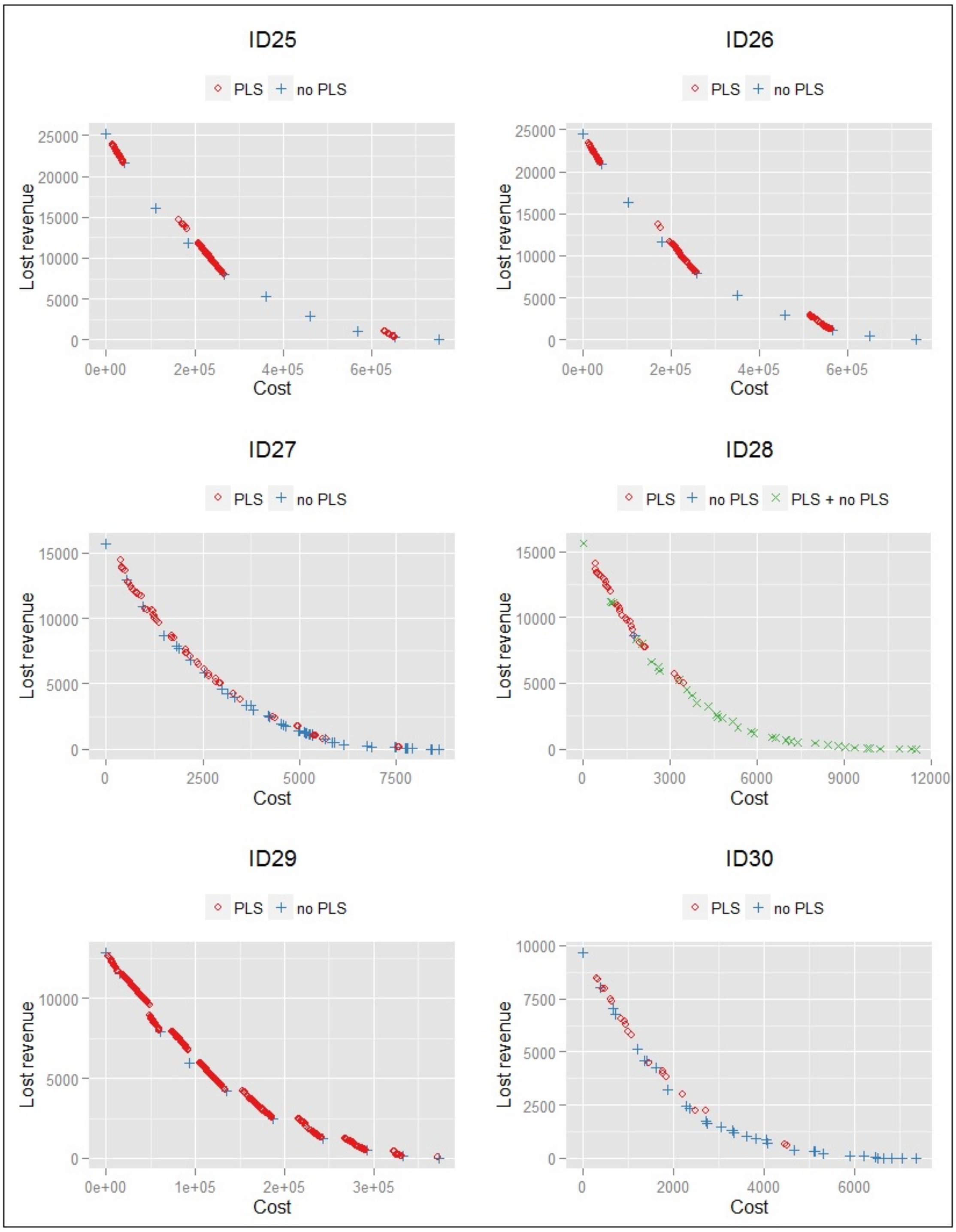

Figure B.5: Pareto Fronts of best no PLS and best PLS solution.

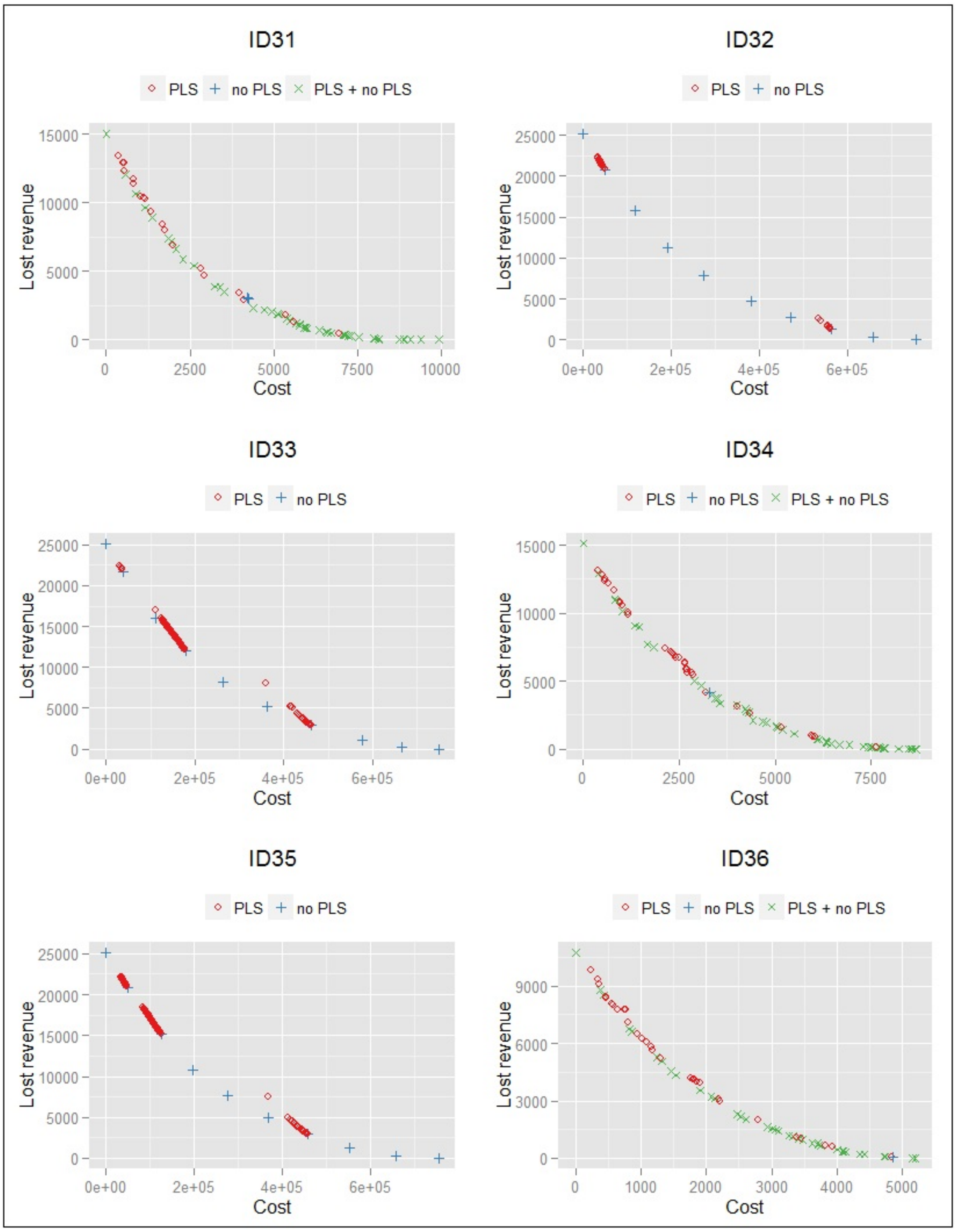

Figure B.6: Pareto Fronts of best no PLS and best PLS solution.

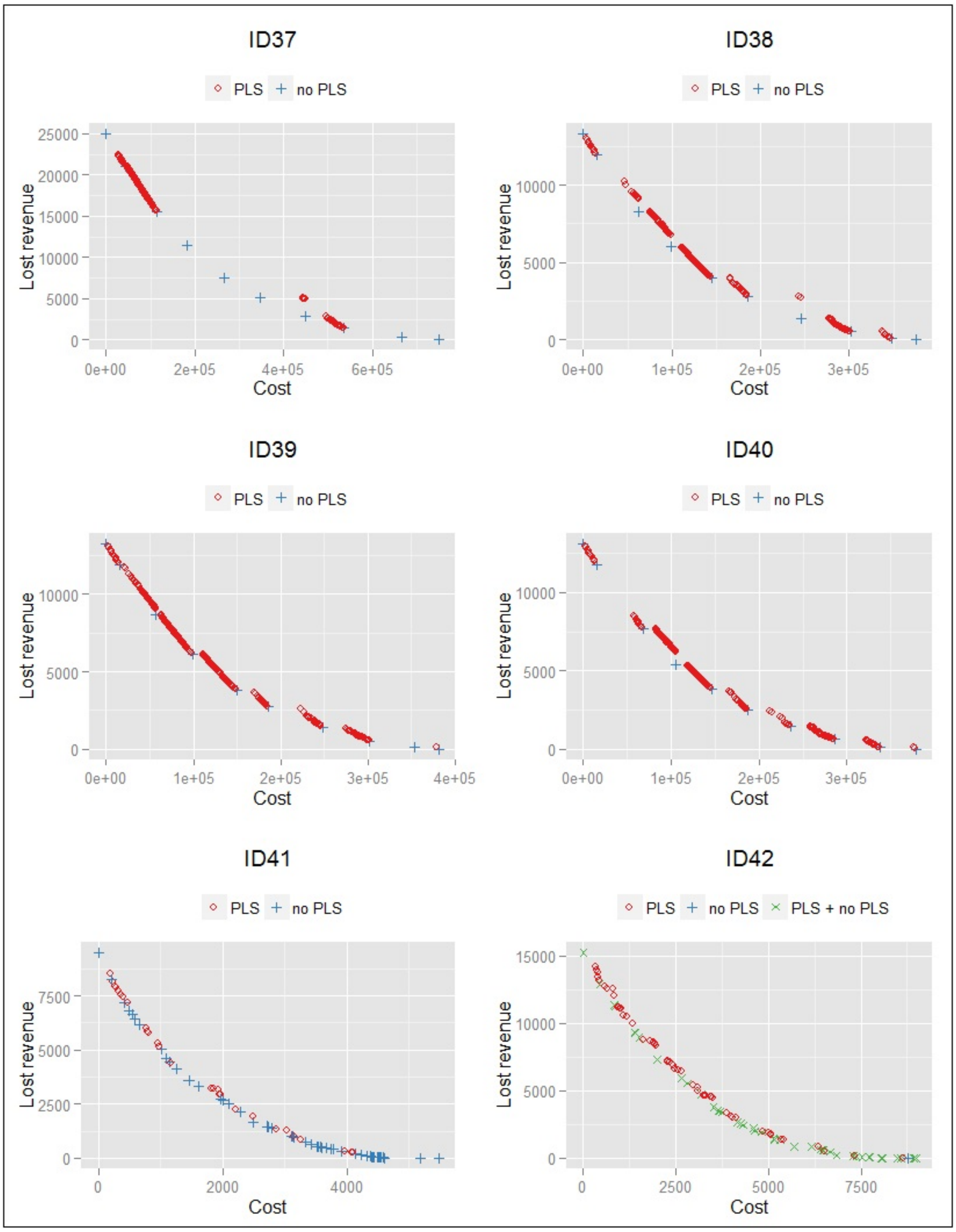

Figure B.7: Pareto Fronts of best no PLS and best PLS solution.

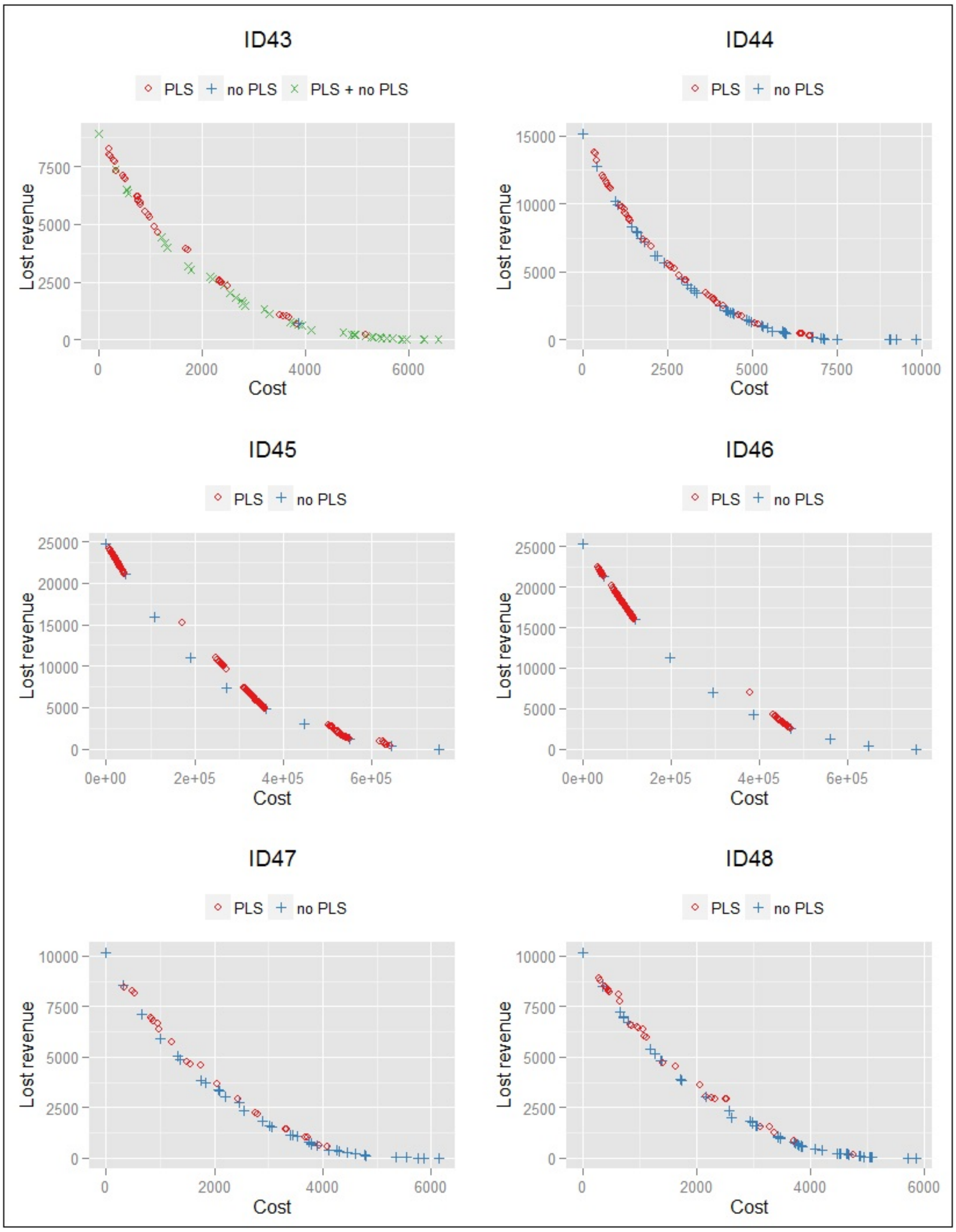

Figure B.8: Pareto Fronts of best no PLS and best PLS solution.

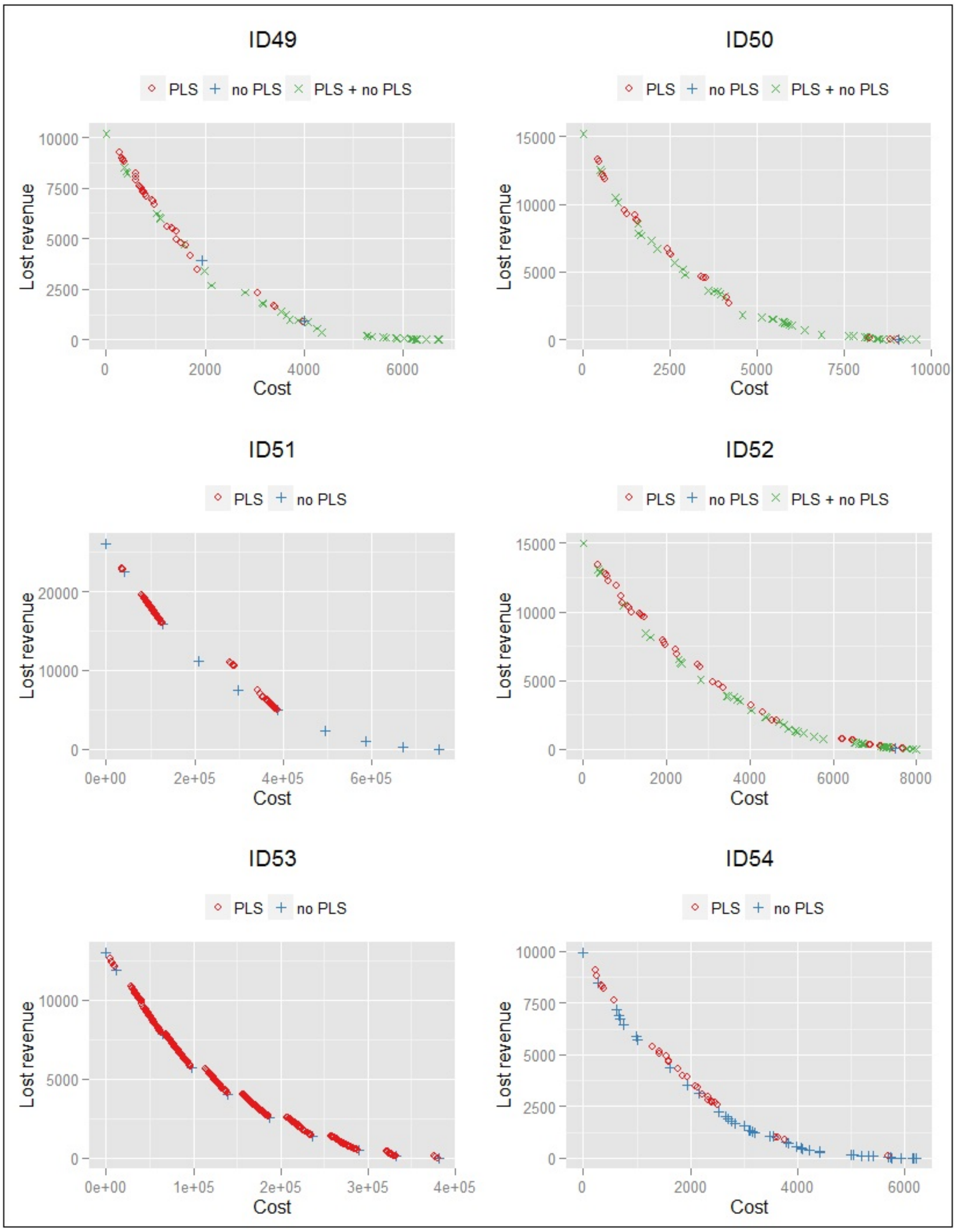

Figure B.9: Pareto Fronts of best no PLS and best PLS solution.

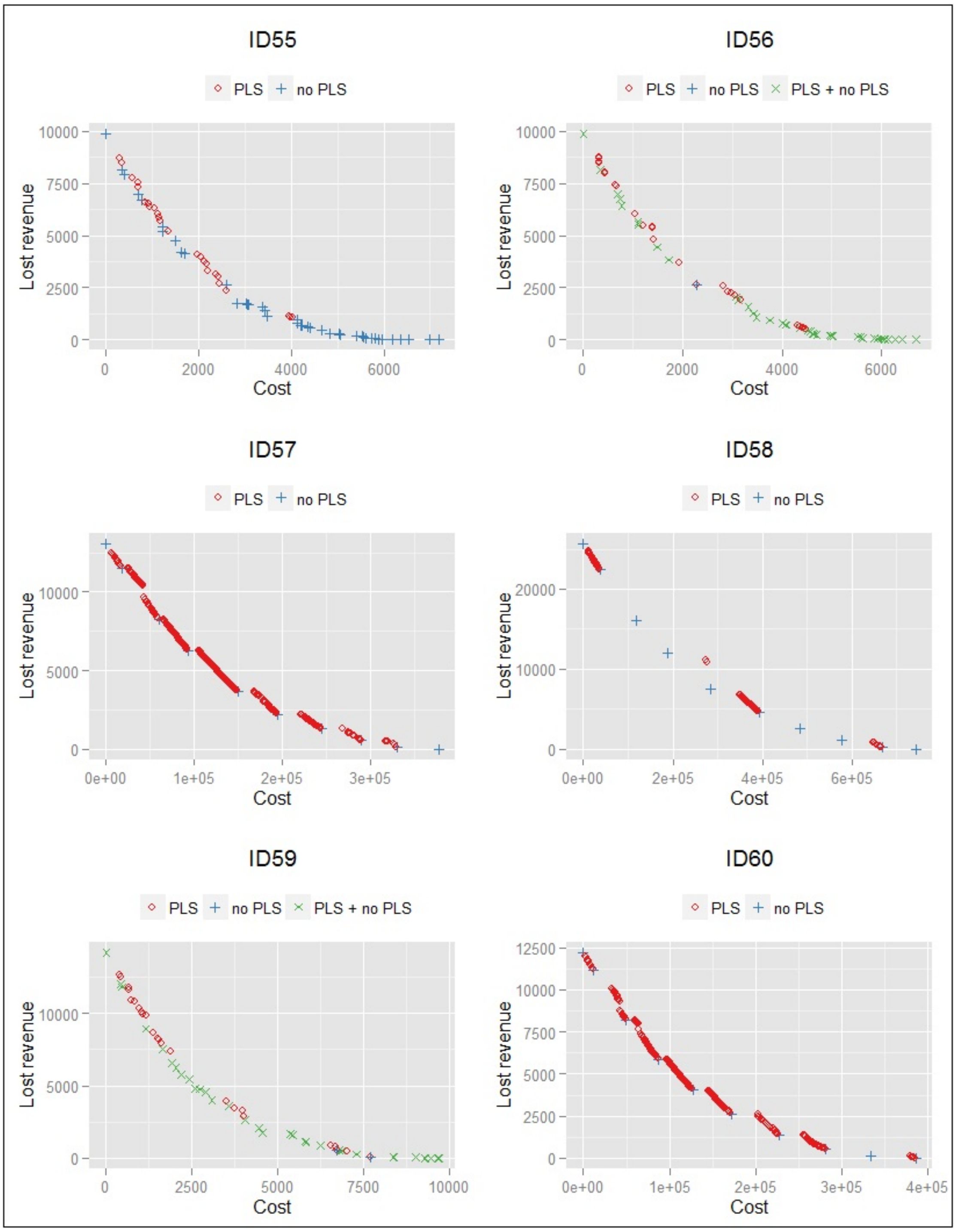

Figure B.10: Pareto Fronts of best no PLS and best PLS solution.

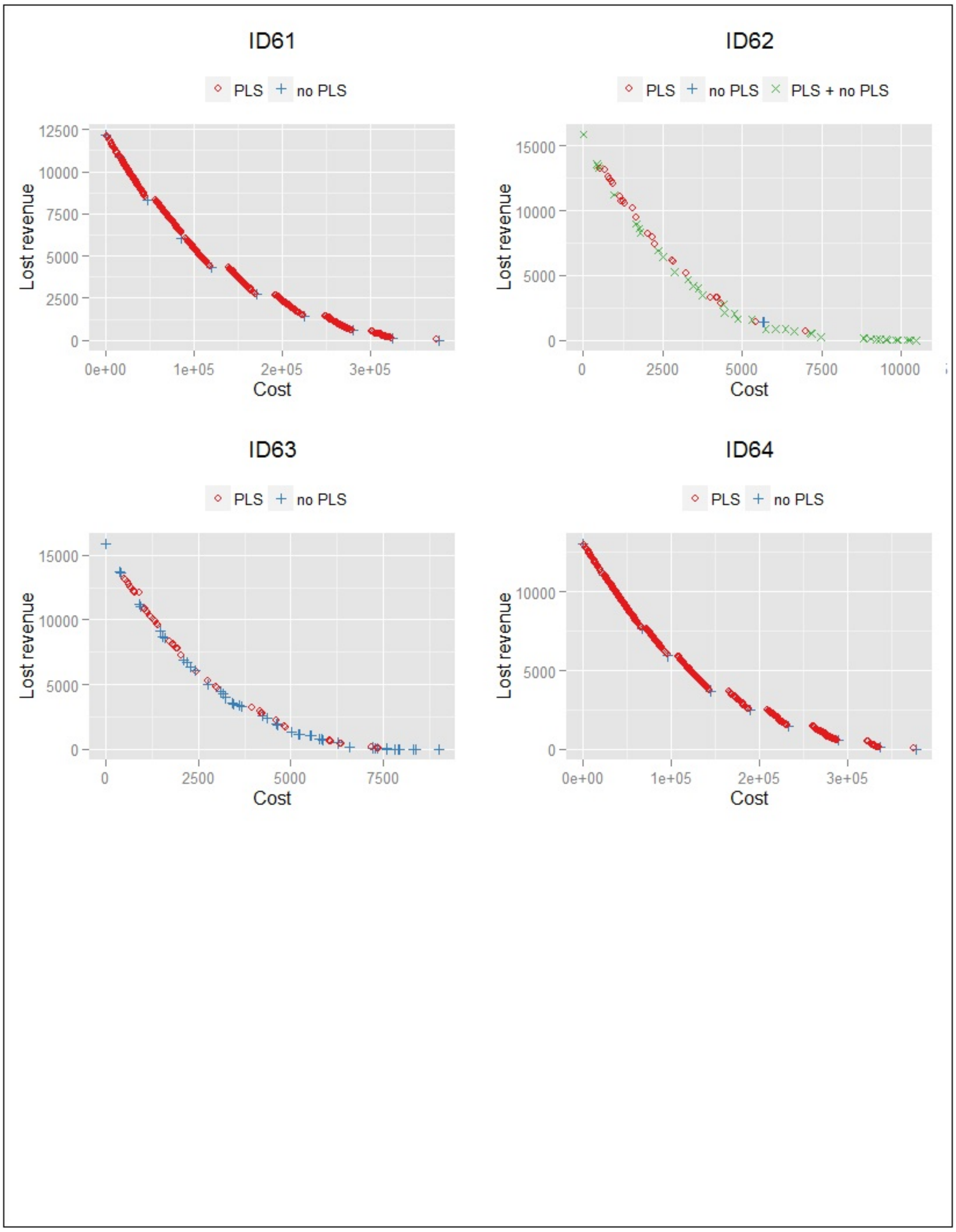

Figure B.11: Pareto Fronts of best no PLS and best PLS solution.

Bibliography

[1] E. Angel, E. Bampis, and L. Gourvès. A dynasearch neighborhood for the bicriteria traveling salesman problem. *Lecture Notes in Economics and Mathematical Systems*, 535:153–176, 2004.

[2] M. Basseur. Design of cooperative algorithms for multi-objective optimization: application to the flowshop scheduling problem. *4OR*, 4(3):255–258, 2006.

[3] M. Basseur, E.-G. Talbi, A. Nebro, and E. Alba. Metaheuristis for multiobjective combinatorial optimization problems: Review and recent issues. *Rapport de recherche*, 5978:39, 2006.

[4] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6:126–140, 1994.

[5] J. D. Beltrán, J. E.Calderón, R. J. Cabrera, J. A. Moreno-Pérez, and J. M. Moreno-Vega. Grasp-vns hybrid for the strip packing problem. *Hybrid metaheuristics*, pages 79–90, 2004.

[6] J.-F. Berube, M. Gendreau, and J.-Y. Potvin. An exact ε-constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits. *European Journal of Operational Research*, 194:39–50, 2009.

[7] C. A. C. Coello and M. S. Lechuga. Mopso : A proposal for multiple objective particle swarm optimization. *Evolutionary Computation*, 2:1051–1056, 2002.

[8] G. B. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 4:183–195, 1955.

[9] G. B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29:767–778, 1961.

[10] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Lecture Notes in Computer Science*, 1917/2000:849–858, 2000.

[11] F. Eisenbrand, F. Grandoni, T. Rothvoß, and G. Schäfer. Approximating connected facility location problems via random facility sampling and core detouring. *SODA '08 Proceedings of the nineteenth annual ACM-SIAM Symposium on Discrete algorithms*, pages 1174–1183, 2008.

[12] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6:109–133, 1995.

[13] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3:1–16, 1995.

[14] F. Glover and E. Taillard. A user's guide to tabu search. *Annals of Operations Research*, 41:1–28, 1993.

[15] D. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[16] S. Gollowitzer, B. Gendron, and I. Ljubic. A cutting plane algorithm for the capacitated connected facility location problem. *Computational Optimization and Applications*, 2013.

[17] S. Gollowitzer and I. Ljubić. Mip models for connected facility location: A theoretical and computational study. *Computers & Operations Research*, 38:435–449, 2011.

[18] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *presented at the congress on Numerical Methods in Combinatorial Optimization*, 1986.

[19] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European journal of operational research*, pages 449–467, 2001.

[20] P. Hansen, N. Mladenović, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, pages 335–350, 2001.

[21] Y. Jin. Effectiveness of weighted aggregation of objectives for evolutionary multiobjective optimization: Methods, analysis and applications. *soft-computing.net*, 2002.

[22] Y. Jin, M. Olhofer, and B. Sendhoff. Dynamic weightedaggregation for evolutionary multi-objective optimization: Why does it work and how? *Proceedings of the Genetic and Evolutionary Computation Conference*, 2001.

[23] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6:366–422, 1960.

[24] D. R. Karger and M. Minkoff. Building steiner trees with incomplete global knowledge. *Proceedings. 41st Annual Symposium on Foundations of Computer Science*, pages 613–623, 2000.

[25] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[26] J. Kennedy and R. Eberhart. Particle swarm optimization. *Conference on Neural Networks, Proceedings, IEEE International*, 4:1942 – 1948, 1995.

[27] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Elsevier, 2001.

[28] M. Laguna, J. O. Roa, A. R. Jimenez, and F. Seco. Diversified local search for the optimal layout of beacons in an indoor positioning system. *IIE Transactions*, 41:247–259, 2009.

[29] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[30] M. Leitner, I. Ljubic, and M. Sinnl. Solving the bi-objective prize-collecting steiner tree problem with the ε-constraint method. *Proceedings of the 6th International Network Optimization Conference (INOC)*, 41:181–188, 2013.

[31] M. Leitner and G. Raidl. Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks. *Hybrid Metaheuristics*, 5296:158–174, 2008.

[32] M. Leitner and G. R. Raidl. Variable neighborhood search for a prize collecting capacity constrained connected facility location problem. *Proceedings of the 2008 International Symposium on Applications and the Internet, SAINT 2008*, pages 233–236, 2008.

[33] M. Leitner and G. R. Raidl. Combining lagrangian decomposition with very large scale neighborhood search for capacitated connected facility location. *Post-Conference Book of the Eight Metaheuristics International Conference*, 2009.

[34] M. Leitner and G. R. Raidl. Branch-and-cut-and-price for capacitated connected facility location. *Journal of Mathematical Modelling and Algorithms*, 2011.

[35] M. Leitner and G. R. Raidl. Variable neighborhood search for capacitated connected facility location. *Extended Abstracts of the Thirteenth International Conference on Computer Aided Systems Theory*, pages 261–263, 2011.

[36] I. Ljubić. A hybrid vns for connected facility location. *Hybrid Metaheuristics*, 4771:157–169, 2007.

[37] T. Lust and J. Teghem. Two-phase pareto local search for the biobjective traveling salesman problem. *J Heuristics*, pages 475–510, 2010.

[38] R. Marler and J. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, pages 369–395, 2004.

[39] S. Martins, M. Resende, C. Ribeiro, and P. Pardalos. A parallel grasp for the steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, 17:267–283, 2000.

[40] O. Mersmann. emoa: Evolutionary multiobjective optimization algorithms, September 2012.

[41] J. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of Applied Optimization*, pages 65–77, 2002.

[42] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.

[43] L. Paquete, M. Chiarandini, and T. Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. *Lecture Notes in Economics and Mathematical Systems*, 535:177–199, 2004.

[44] L. Paquete and T. Stützle. A two-phase local search for the biobjective traveling salesman problem. *Proceeding EMO'03 Proceedings of the 2nd international conference on Evolutionary multi-criterion optimization*, pages 479–493, 2003.

[45] K. Parsopoulos and M. Vrahatis. Particle swarm optimization method in multiobjective problems. *SAC '02 Proceedings of the 2002 ACM symposium on Applied computing*, pages 603–607, 2002.

[46] C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. *Approximation Algorithms for Combinatorial Optimization*, 2462:256–270, 2002.

[47] M. N. Thapa. *Linear Programming 2: Theory and Extensions. Vol. 2.* Springer, 2003.

[48] A. Tomazic and I. Ljubic. A grasp algorithm for the connected facility location problem. *Applications and the Internet, 2008. SAINT 2008. International Symposium on*, pages 257–260, 2008.

[49] M. Verhoeven and M. Severens. Parallel local search for steiner trees in graphs. *Annals of Operations Research*, pages 185–202, 1999.

[50] J. von Neumann. Duality theory. *privately circulated notes - Princeton*, 1947.

[51] H. Zhang. Multiple particle swarm optimizers with inertia weight for multi-objective optimization. *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 1, 2012.

[52] E. Zitzler, D. Brockhoff, and L. Thiele. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. *Evolutionary Multi-Criterion Optimization*, pages 862–876, 2007.

[53] E. Zitzler, J. Knowles, and L. Thiele. Quality assessment of pareto set approximations. *Multiobjective Optimization*, pages 373–404, 2008.

[54] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. *PPSN-V*, pages 292–301, 1998.

[55] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3:257–271, 1999.

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