FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Complete Solution Archives for Evolutionary Combinatorial Optimization

Application to a Competitive Facility Location and a Stochastic Vehicle Routing Problem

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Benjamin Biesinger
Registration Number 0927842

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther R. Raidl

The dissertation has been reviewed by:

Günther R. Raidl	Christian Blum	Ulrich Pferschy

Vienna, 20th April, 2016

Benjamin Biesinger

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Benjamin Biesinger
Stättermayergasse 8/21-22, 1150 Wien

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.

Wien, 20. April 2016

Benjamin Biesinger

Acknowledgements

First and foremost I want to thank Günther Raidl, my excellent supervisor, who provided me with countless invaluable advice, supported me in any aspect, and always guided me in the right direction when I was facing difficulties during my research. I further want to thank Christian Blum, who happily agreed to be the second reviewer of this thesis.

I also want to express my gratitude to all of my current and former colleagues of the Algorithms and Complexity Group of the Vienna University of Technology for fruitful discussions, sharing ideas and thoughts on work-related and work-unrelated topics, and helpful comments on my own research. Especially, I want to thank Bin Hu who constantly helped me overcome any challenges during my work and who is a co-author of many of my publications. For financial support I want to thank the Vienna Scientific Fund (FWF), who provided funding within the project P24660-N23.

I owe my gratitude to my family, especially my parents, who always supported me in my course of education. My biggest thanks, however, goes to Sandra, love of my life and best friend. Thank you for your patience, support, understanding, and encouragement over the last few years and for teaching me the important things in life (apart from algorithms, obviously).

Kurzfassung

Hybride Metaheuristiken wurden in den letzten Jahrzehnten intensiv erforscht um schwierige kombinatorische Optimierungsprobleme zu lösen. In dieser Dissertation werden solche Hybridisierungen von Metaheuristiken mit auf *Tree-Search* basierenden Methoden untersucht, um Schwächen beider einzelnen Verfahren auszugleichen. Auf der einen Seite kommt es, insbesondere bei evolutionären Algorithmen, auf Grund der fehlenden Information zum bisherigen Suchverlauf oft zu unnötigen Re-evaluierungen, einem Verlust der Diversität und vorzeitiger Konvergenz. Auf der anderen Seite haben *Tree-Search* Methoden wie *Branch-and-Bound* häufig eine hohe Laufzeit und skalieren schlecht mit der Instanzgröße. Der Fokus dieser Arbeit liegt in der Hybridisierung dieser Methoden durch vollständige Trie-basierte Lösungsarchive innerhalb metaheuristischer Frameworks. Ein solches Lösungsarchiv speichert alle generierten Lösungskandidaten in einer effizienten baumbasierten Datenstruktur und vermeidet dadurch Duplikate. Bei jedem Auftreten einer Duplikatlösung wird diese in eine garantiert neue, üblicherweise ähnliche Lösung direkt vom Archiv konvertiert. Wendet man dieses Lösungsarchiv innerhalb einer Metaheuristik an, wird diese dadurch im Prinzip zu einem vollständigen, exakten Suchalgorithmus, der eine optimale Lösung bei genügend langer Laufzeit garantiert findet. Obwohl dieser Fall normalerweise nur bei kleineren Instanzen auftritt, kann das Archiv die Performance der Metaheuristik verbessern, selbst wenn der Algorithmus vorzeitig abgebrochen wird. In dieser Dissertation werden solche Lösungsarchive detailliert untersucht, mit fortgeschrittenen Verfahren erweitert und auf zwei praxisrelevante Problemstellungen angewandt.

Die erste betrachtete Problemstellung ist das *Competitive Facility Location Problem*, in dem zwei nicht kooperative Unternehmen, ein *Leader* und ein *Follower*, durch Auswählen von Filialstandorten um Marktanteile konkurrieren. Wir betrachten sechs verschiedene Szenarien für das Kundenverhalten und die Art des Bedarfs um die Marktanteile für den Leader und den Follower zu berechnen und präsentieren mathematische Modelle für jedes dieser Szenarien. Wir stellen einen heuristischen Ansatz vor, der auf einem fortgeschrittenen evolutionärem Algorithmus und einem Lösungsarchiv mit randomisierter Baumstruktur basiert. Der Algorithmus nutzt eine eingebettete lokale Suche und Tabusuche, die mit dem Lösungsarchiv auf vier verschiedene Arten kombiniert werden. Die hohe Laufzeit der Lösungsevaluierung wird durch ein multi-level Evaluierungsschema reduziert, welches einen Greedy-Algorithmus und ein *Mixed Integer Linear Programming* Modell kombiniert einsetzt. Da dieses Problem sowohl eine kompakte Lösungsrepräsentierung besitzt, da nur die Standorte des Leaders gespeichert werden, als auch eine teure Evaluierungsfunktion hat, die aus dem Finden optimaler Standorte für den Follower besteht, konnte mit dem Lösungsarchiv eine substantielle Verbesserung der finalen Lösungsgüte erreicht werden.

Die zweite Problemstellung ist das *Generalized Vehicle Routing Problem with Stochastic Demands and Preventive Restocking*, welches eine Kombination aus zwei Generalisierungen des klassischen Routenplanungsproblems ist. Ziel dieses Problems ist, Routen durch eine Menge von Standorten (Knoten) zu finden, die in disjunkte Cluster eingeteilt sind, wobei genau ein Knoten von jedem Cluster zu besuchen ist. Da die Kapazität eines Fahrzeugs beschränkt ist und daher der (stochastische) Bedarf der Cluster nicht immer erfüllt werden kann, müssen zusätzliche Wege zum Auffüllen des Fahrzeugs geplant werden. Um die optimalen Positionen für so ein Auffüllen in der Tour zu finden, die von der Realisierung der Zufallsvariablen und der derzeitigen Ladung abhängt, kann ein exakter aber zeitaufwändiger Algorithmus basierend auf dynamischer Programmierung eingesetzt werden. Für dieses Problem werden ein exakter und zwei metaheuristische Lösungsansätze entwickelt und in dieser Arbeit präsentiert. Der exakte Algorithmus basiert auf einem *Mixed Integer Linear Programming* Modell für das generalisierte *Traveling Salesman Problem*, welches via *Branch-and-Cut* gelöst wird und die dynamische Programmierung für das Finden optimaler Auffüllpositionen benutzt. Die erste Metaheuristik ist eine *General Variable Neighborhood Search* mit drei Nachbarschaftsstrukturen. Um die Laufzeit der Lösungsevaluierungen zu reduzieren, wird ein Multi-level Evaluierungsschema verwendet, welches die dynamische Programmierung benutzt und iterativ mit immer größerer Genauigkeit die exakte Lösungsqualität approximiert. In diesem Evaluierungsschema wird die Kapazität des Fahrzeugs und die Wahrscheinlichkeitsverteilungen des Bedarfs in den Clustern herab skaliert. Die zweite Metaheuristik ist ein genetischer Algorithmus, der ein vollständiges Trie-basiertes Lösungsarchiv benutzt. Das Archiv wird mit einer *Bounding* Erweiterung versehen, die Teile des Suchbereichs wegschneidet, welche garantiert keine optimale Lösung beinhalten. Empirische Resultate zeigen, dass der exakte Algorithmus nur kleinere Instanzen lösen kann, aber beide Metaheuristiken gut für größere Instanzen eingesetzt werden können. Das Lösungsarchiv stellte sich auch für dieses Problem als wichtiger Teil des genetischen Algorithmus heraus und gemeinsam mit der *Bounding* Erweiterung war es möglich, optimale oder nahezu optimale Lösungen für viele Benchmark Instanzen zu finden.

Die Resultate der entwickelten Algorithmen für die vorgestellten Probleme zeigen insgesamt, dass vollständige Trie-basierte Lösungsarchive in der Lage sind, die Performance von evolutionären Algorithmen für kombinatorische Optimierungsprobleme mit einer kompakten Lösungsrepräsentierung und zeitaufwändiger Evaluierungsfunktion signifikant zu steigern. Erweiterungen für Lösungsarchive, die deren Baumstruktur ausnutzen, können zu substantiellen Verbesserungen der Metaheuristik führen. Diese Dissertation zeigt, dass die Kombination aus evolutionären Algorithmen und Lösungsarchiven zu neuen state-of-the-art Lösungsverfahren in dem Gebiet der Standort- und Routenplanung führen können.

Abstract

Hybrid metaheuristics for solving hard combinatorial optimization problems have been intensively studied over the last few decades. This thesis considers such a hybridization of metaheuristics and tree search methods to overcome some weaknesses of each individual method. On the one hand, especially in evolutionary algorithms the lack of information on the search history usually leads to unnecessary re-evaluations, a loss of diversity, and premature convergence. On the other hand, tree search methods like branch-and-bound frequently have a high run-time requirement and scale not so well with the instance size. The focus of this thesis lies in the hybridization of these methods using complete trie-based solution archives within a metaheuristic framework. Such a solution archive stores all generated solution candidates in an efficient tree data structure and thereby avoids duplicates. Whenever a potential duplicate solution is identified it is converted into a guaranteed new, usually similar solution directly by the archive. Applying this archive to a metaheuristic turns it, in principle, into a complete exact search algorithm which finds an optimal solution given enough time. Although this is usually only possible for smaller instances, even when prematurely terminated, using the archive can improve the performance of the metaheuristic. In this thesis such solution archives are investigated in detail, extended with more advanced techniques, and applied to two practical combinatorial optimization problems with real-world applications.

The first considered problem is the competitive facility location problem, in which two non-cooperating companies, a leader and a follower, compete for market share by choosing locations for opening stores. We consider six different customer behavior scenarios and demand models to compute the market share for the leader and the follower and present mathematical models for each of them. We approach this problem heuristically with an advanced evolutionary algorithm using a solution archive with a randomized trie structure. The algorithm employs an embedded local and tabu search procedure which is combined with the solution archive in four different ways. The substantial time consumption of the solution evaluation is reduced by utilizing a multi-level evaluation scheme using a greedy algorithm and a mixed integer programming formulation in a combined way. As this problem comprises both, a compact solution representation by only storing the locations for the leader and an expensive evaluation function consisting of computing optimal locations for the follower, using a solution archive results in a substantial improvement of the final solution quality.

The second problem is the generalized vehicle routing problem with stochastic demands and preventive restocking which is a combination of two generalizations of the classical vehicle routing problem. The aim of this problem is to find routes through a set of nodes, which are partitioned into disjoint clusters and exactly one node of each cluster has to be visited. The capacity of the vehicle is limited, and therefore the (stochastic) demands of the clusters cannot always be satisfied within a route and restocking trips must be planned. Determining optimal restocking points depends on the realizations of the random variables and the actual load of the vehicle, and can be computed using an exact but time-consuming dynamic programming algorithm. For this problem an exact solution algorithm and two metaheuristics are developed and presented in this thesis. The exact algorithm is based on a mixed integer linear programming model for the generalized traveling salesman problem and solved via branch-and-cut, which uses the dynamic programming algorithm for computing the restocking points as sub-procedure in order to separate cuts. In the first metaheuristic a general variable neighborhood search with three neighborhood structures is used. For decreasing the run-time of the solution evaluations a multi-level evaluation scheme is developed, which uses the dynamic programming algorithm and iteratively approximates the actual solution quality with increasing accuracy by scaling down the vehicle capacity and the probability distributions of the cluster demands. The second metaheuristic is a genetic algorithm using a complete trie-based solution archive. This archive is further extended with a bounding procedure to cut off areas of the solution space that evidently cannot contain optimal solutions. Computational results show that while the exact algorithm is only able to solve smaller instances, both metaheuristics can be used well for larger instances. The solution archive turned out to be, also for this problem, an important component of the genetic algorithm and together with the bounding procedure the approach was able to find optimal or near-optimal results for many benchmark instances.

The overall results of the computational tests of the developed algorithms for these problems show that complete trie-based solution archives are able to significantly boost the performance of evolutionary algorithms for combinatorial optimization problems with a compact solution representation and a time-consuming evaluation function. When properly designed, extensions to the solution archive exploiting their tree structure can lead to significant improvements of the metaheuristic. This thesis shows that the combination of evolutionary algorithms with solution archives can lead to new state-of-the-art algorithms in the area of location and routing problems.

Contents

CHAPTER 1

Introduction

Efficient and sustainable utilization of the available resources is of paramount importance to the economical and social success of a company in modern society. In many industry sectors decision makers face the challenge of planning a set of actions leading to a particular outcome which should be beneficial to the company's success. This need of making good decisions gives rise to challenging optimization problems which can be computationally tackled. Especially in the areas of transportation, telecommunication, scheduling, network design, location planning, and many more, such problems arise naturally. General sample questions in these fields are the following:

- What are the most resource-efficient routes for my fleet of vehicles to distribute my goods?
- From several possible locations to open new stores / warehouses / distribution centers, which ones should I choose?
- How can I make a schedule for public transportation / patients / students / machines in order to efficiently satisfy all required conditions?

In many cases such problems can be modeled as *combinatorial optimization problems* (COPs) for which in Section 2.1 a formal definition is given. There is a large history of modeling and solving COPs. While easier variants of COPs can be solved in polynomial time, most practically relevant problems are not always efficiently solvable anymore under the assumption that P≠NP (in Section 2.2 more information about the complexity classes P and NP is provided). For a formal definition of above optimization problems, they are usually modeled as a graph $G = (V, E)$, where V denotes the set of nodes representing customers, possible locations, activities, etc. and edges $E \subseteq (V \times V)$ corresponding to connections between the nodes.

Let us consider a well-known COP which deals with a problem in the domain of location science as motivating example: the *p-Median Problem* (PMP), which was introduced by Hakimi [67]. In the PMP we are given a graph $G = (I, D)$ consisting of a set I of locations and a distance matrix $D = (d_{ij})$ containing a distance (or cost) for each pair of locations (i, j) with $i, j \in I$. The aim of the problem is to find a subset $S \subseteq I$ of exactly p locations such that the total distance between each $i \in I$ and its closest $j \in S$ is minimized:

$$\min \sum_{i \in I} \min_{j \in S} d_{ij}$$

The PMP was shown to be NP-hard. It models fundamental aspects of several real-world applications, e.g., choosing the locations for industrial plants, warehouses, and public facilities [97].

Although problems like the PMP are NP-hard and therefore in general cannot be solved in polynomial time (under the assumption that P≠NP), there exist several algorithms which are in practice sometimes able to find an optimal solution in reasonable time. These approaches include, for example, bounded enumeration, branch-and-bound, constraint programming, and dynamic programming. In Section 2.3 an overview of some of these exact methods is given. As these methods often do not scale well enough with the problem size, optimality is frequently traded for shorter running times by turning to incomplete approximate solution methods. These methods include heuristics, metaheuristics, and approximation algorithms. This thesis focuses on solving COPs with metaheuristics and therefore in Section 2.4 an overview on a selection of important types of metaheuristics is given.

A common property of metaheuristics is their lack of a long-term memory to keep track of their search history. This implies that with a high probability at some point during the search the algorithm comes to a solution candidate it has already considered before. For several types of COPs and algorithms this might be problematic since cycling among a set of solutions is possible and evaluating the quality of an already assessed solution candidate can be (unnecessarily) time-consuming. Raidl and Hu [112] performed pioneer work on complete trie-based solution archives which are data structures that store all visited solution candidates in a compact way and upon duplicate detection solutions are transformed into typically similar but guaranteed not yet visited solutions. The storing of visited solutions and the duplicate checking mechanism can also be achieved by using simpler caching approaches based on hashing. The considered trie-based solution archives, however, go further and extend the duplicate checks with a non-trivial and problem dependent transformation method that efficiently converts a found duplicate into a new solution. This is achieved by using a trie for the solution archive, which is a special tree data structure commonly used for language dictionaries. The construction of this trie depends on the solution representation used for the problem. This combination of a metaheuristic and the tree structure of the solution archive can be further exploited by concepts known from tree search methods, e.g., computing bounds on sub-trees to cut off areas which evidently do not contain an optimal solution. The main challenges of applying solution archives to a COP lie in the determination of a suitable compact solution representation, the design of the conversion operator, and to ensure that only feasible solution candidates are generated by the transformation method. The aim of this thesis is to investigate the effectiveness of such solution archives in more detail, find new application areas, and to extend the basic idea with various more advanced concepts.

In order to evaluate the effectiveness of solution archives several problems with real-world applications have been chosen. The first problem that we consider is the competitive facility location problem (CFL), which is the main topic of Chapter 4. In the CFL there are two non-cooperating companies entering a market sequentially and are competing for market share. The first decision maker, referred to as the leader, wants to maximize his market share knowing that a so-called follower will enter the same market. Thus, for evaluating a leader's candidate solution, a corresponding follower's subproblem needs to be solved, and the overall problem therefore is a *bi-level optimization problem*. This thesis considers several customer behavior scenarios combined with two different demand models. This problem is chosen because evaluating a candidate leader's solution is time-consuming and therefore re-evaluating the same solution is expensive and should be avoided. We use an evolutionary algorithm with solution archive for solving this problem heuristically. This algorithm is based on a genetic algorithm with tabu search as local improvement procedure. Different evaluation procedures, a greedy algorithm and approaches based mixed integer linear programming models, are combined in a unified approach to a multi-level evaluation scheme which decreases the overall time spent for solution evaluations. The employed solution archive is a binary trie with randomized insertion order based on the chosen locations of the leader. The conversion method changes the detected duplicate solution at at least two positions: one facility must be closed and re-opened again at another position. It is ensured that the converted solution is not too far off the original solution if possible, by preferring the values of the original solutions' variables. The evolutionary algorithm is tested both on Euclidean and non-Euclidean instances from the literature. Especially on the Euclidean instances and binary or proportional customer behavior with essential demands the developed algorithm is able to exceed previous state-of-the-art heuristic approaches in solution quality and running time in most cases. For the other considered scenarios no computational results have been published in the literature but several configurations of the evolutionary algorithm are compared. The results showed that the configurations using the solution archive provided significantly better results than the configurations without on the majority of the test instances. Finally, for a better illustration of the results and to show the practical applicability of this approach, a case study using real data of the registration districts of Vienna, Austria was conducted.

The second COP considered in this thesis is the generalized vehicle routing problem with stochastic demands (GVRPSD), which is treated in Chapter 5. The GVRPSD is a generalization of the stochastic variant of the well-known vehicle routing problem (VRP). The aim of this problem is to find a set of routes so that all customers are served. The demands of these customers, however, are not known beforehand and therefore restocking trips back to a central depot may be necessary. Computing these restocking points depends on the current load of the vehicle and is a time-consuming procedure and can be done via an exact dynamic programming algorithm. Thus, avoiding duplicate solutions by a solution archive appears highly promising. As the GVRPSD has not been considered yet in the literature, first, an exact algorithm is developed to obtain optimal solutions to at least smaller instances. This algorithm is based on a mixed integer linear programming model for the generalized traveling salesman problem and the dynamic programming algorithm. The model is iteratively solved within a branch-and-cut framework after introducing new inequalities derived by the results of the dynamic programming algorithm. The introduced inequalities reflect the additional restocking costs which are introduced by the unplanned return trips to the depot. As the results of this method showed that such an exact approach is only able to solve small instances, in a next step metaheuristic algorithms are developed. First, a variable neighborhood search is proposed which uses three neighborhood structures for permutation encodings. The main feature of this algorithm is another multi-level evaluation scheme which iteratively estimates the true objective value by scaling down the vehicle capacity and the probability distributions of the stochastic demand. We show that the resulting value of each of the levels is a lower bound to the value of the previous level which can lead to an early termination of the evaluation procedure and thereby reducing its running time. This multi-level evaluation scheme is also used in our second metaheuristic for the GVRPSD, which is an evolutionary algorithm with solution archive. This algorithm employs an embedded local improvement procedure which uses the neighborhood structures from the variable neighborhood search as well as a new one based on conversions in the solution archive. Another feature of this algorithm is the bounding extension of the solution archive. Lower bounds are computed for partial solutions so that subtries which cannot contain an optimal solution are pruned. The computational results showed that with such a bounding extension optimal solutions for smaller instances could be found and that the resulting solution quality also increased for larger instances. Also for this problem the configurations using the solution archive produced significantly better results than the others on most of the test instances.

1.1 Overview of the Thesis

In Chapter 2 an overview of complexity theory and exact and (meta-)heuristic solution methods for COPs is given. The focus lies on (mixed) integer programming, extensions for the exact methods, and on a selection of popular metaheuristics which are used to solve the problems considered later in this thesis. The last part of this chapter presents hybrid metaheuristics which deal with an efficient combination of metaheuristics and exact methods.

Chapter 3 is devoted to complete trie-based solution archives. In this chapter, first, a literature overview about existing duplicate elimination techniques is given, followed by a description of how the trie is structured and the trie insertion and conversion procedures. Furthermore, the integration into metaheuristics, especially into evolutionary algorithms, is illustrated and possible extensions are presented.

In the next two chapters the considered problems and the proposed solution approaches are shown. Therefore, first a formal problem definition is given and previous and related work is described. Then, the solution algorithms are shown and explained in detail. At the end of each chapter they are experimentally evaluated with respect to solution quality, run-time consumption, and other properties, and compared to the results from the literature as far as available. Finally, for each problem, conclusions are drawn and an outlook for future work is given.

Chapter 4 is dedicated to competitive facility location problems. Several types of customer behavior and demand models with corresponding integer programming formulations are presented. Parts of this chapter have been published in:

> B. Biesinger, B. Hu, and G. R. Raidl. Models and algorithms for competitive facility location problems with different customer behavior. *Annals of Mathematics and Artificial Intelligence*, 76(1):93–119, 2015

An earlier version of the solution algorithm which is a hybrid evolutionary algorithm with solution archive has been published in:

> B. Biesinger, B. Hu, and G. R. Raidl. A hybrid genetic algorithm with solution archive for the discrete $(r|p)$-centroid problem. *Journal of Heuristics*, 21(3):391–431, 2015

This algorithm was adapted to a different type of customer behavior and has been published in:

> B. Biesinger, B. Hu, and G. R. Raidl. An evolutionary algorithm for the leader-follower facility location problem with proportional customer behavior. In *Conference Proceedings of Learning and Intelligent Optimization Conference (LION 8)*, volume 8426 of *LNCS*, pages 203–217. Springer, 2014

A case study using the evolutionary algorithm on the data of Vienna, Austria will appear in the form of an invited book chapter in:

> B. Biesinger, B. Hu, and G. R. Raidl. A memetic algorithm for competitive facility location problems. In Natalie Jane de Vries and Pablo Moscato, editors, *Business and Consumer Analytics: New Directions (Vol1)*, pages 1–23. 2016. To appear

Furthermore, a presentation with preliminary results was given:

> B. Biesinger. A hybrid evolutionary algorithm for the discrete $(r|p)$-centroid problem. Austrian Workshop on Metaheuristics 9, Vienna, Austria, 2013

Next, Chapter 5 is dedicated to the generalized vehicle routing problem with stochastic demands. First, the problem is introduced and an exact algorithm based on integer programming is described. This work has been published in:

B. Biesinger, B. Hu, and G. R. Raidl. An integer L-shaped method for the generalized vehicle routing problem with stochastic demands. In *7th International Network Optimization Conference, INOC*, 2015. To appear

Then, two metaheuristics for this problem are proposed. The first metaheuristic is a variable neighborhood search which uses a multi-level technique for faster solution evaluation. This work has been published in:

B. Biesinger, B. Hu, and G. R. Raidl. A variable neighborhood search for the generalized vehicle routing problem with stochastic demands. In Gabriela Ochoa and Francisco Chicano, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2015*, volume 9026 of *LNCS*, pages 48–60. Springer, 2015

The second metaheuristic is a genetic algorithm in combination with a solution archive and has been submitted to:

B. Biesinger, B. Hu, and G. R. Raidl. A genetic algorithm in combination with a solution archive for solving the generalized vehicle routing problem with stochastic demands. 2016. submitted to a journal

Finally, Chapter 6 draws general conclusions on solution archives and points out possible future research directions.

All of the above publications and this thesis are supported by the Austrian Science Fund (FWF) under grant P24660-N23 and we thank the FWF for all received funding.

CHAPTER 2

Methodology

This chapter first gives an overview on some basic concepts from complexity theory related to combinatorial optimization and then presents the concepts and solution methodologies for combinatorial optimization problems which are used throughout the thesis. These methods can be classified into exact approaches, from which we will consider branch-and-bound and (mixed) integer linear programming and (meta-)heuristic methods which solve COPs only approximately. As there is also a huge variety of different (meta-)heuristics described in the literature we only focus on these relevant to the further parts of this thesis, which are construction heuristics, local search, variable neighborhood descent, variable neighborhood search, tabu search, and genetic algorithms.

2.1 Combinatorial Optimization Problems

Before starting to discuss solution methods, the considered problems are formally introduced. Let us start with *combinatorial optimization problems* in general, for which the definitions are based on Aarts and Lenstra [1]:

Definition 1. *A* combinatorial optimization problem *is specified by a set of problem instances and is either a* minimization *or a* maximization *problem.*

Definition 2. *An* instance *of a combinatorial optimization problem is a pair* (S, f)*, where the* solution set S *is the finite set of feasible solutions and the* cost function f *is a mapping* $f : S \mapsto \mathbb{R}$*. The problem is to find a* globally optimal *solution, i.e., an* $i^* \in S$ *such that*

$$f(i^*) \leq f(i) \quad \forall i \in S \tag{2.1}$$

for minimization problems and

$$f(i^*) \geq f(i) \quad \forall i \in S \tag{2.2}$$

for maximization problems. Furthermore, $f^* = f(i^*)$ *denotes the optimal cost, and* $S^* = \{i \in S \mid f(i) = f^*\}$ *denotes the set of optimal solutions.*

2.2 Computational Complexity

In the introduction we already mentioned hard, especially NP-hard, problems but what exactly does *hard* mean in this context? As this thesis considers only NP-hard problems whose underlying decision problems lie either in the complexity class NP or above, we will give a brief introduction to the basics of complexity theory, the complexity classes P and NP, as well as touch the topic of problems beyond NP. In the field of complexity theory usually primarily decision problems are considered as opposed to optimization problems. However, if we, for example, have a global upper bound to the optimal value of a minimization problem, then we can apply a binary search in the value solution space (let B be the current value) and repeatedly ask the question "Is there a solution to the given optimization problem with a value no larger than B?". Thereby, we can transform most optimization problems in a straightforward way into a series of decision problems.

To answer the question of hard problems it is necessary to give a formal definition of an algorithm and an associated computation model. For that reason we will use models from the literature and the remainder of this section is therefore based on the book *Computers and Intractability* by Garey and Johnson [56]. The simple formal model on which this section is based is a *deterministic one-tape Turing machine* (DTM), which consists of a *finite state control*, a *read-write head*, and an infinitely long *tape*, which is partitioned into *tape squares*. Then we can define a program for a DTM as follows [56].

Definition 3. *A* program *for a DTM specifies the following information:*

(a) A finite set Γ *of tape* symbols*, including a subset* $\Sigma \subset \Gamma$ *of* input symbols *and a distinguished* blank symbol $b \in \Gamma - \Sigma$*;*

(b) a finite set Q *of* states*, including a distinguished* start state q_0 *and two distinguished* stop states q_Y *and* q_N*;*

(c) a transition function $\delta : (Q - \{q_\mathrm{Y}, q_\mathrm{N}\}) \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$.

The input of such a program is a string $x \in \Sigma^*$, where Σ^* is the set of all finite strings using symbols from Σ. In the beginning of the execution of the program this input is written on the tape starting at position 0. The program starts in state q_0 and proceeds by following the statements of the transition function until either state q_Y or q_N is reached which corresponds to a *yes*-answer or a *no*-answer to the decision problem, respectively. We remark that in general the program does not have to stop at all but this does not pose a problem here.

The input string x corresponds to a specific encoding scheme e of an instance of a decision problem, where an encoding scheme e describes each instance of a decision problem by a string of symbols over a fixed alphabet Σ. Before we can define the complexity class P we have to establish the connection between a decision problem and a *language*. We associate a language $L[\Pi, e]$ with the decision problem Π and a proper encoding e as follows [56]:

$$L[\Pi, e] = \left\{ x \in \Sigma^* : \begin{array}{l} \Sigma \text{ is the alphabet used by } e \text{, and } x \text{ is the encoding under } e \\ \text{of an instance } I \in Y_\Pi \end{array} \right\},$$

where Y_Π is the set of *yes*-instances of decision problem Π. Then, we say that a program M with input alphabet Σ *accepts* $x \in \Sigma^*$ if and only if M halts in state q_Y. The language L_M that is *recognized* by program M is given by $L_M = \{x \in \Sigma^* : M \text{ accepts } x\}$ [56]. Finally, we can formally define the complexity class P as follows:

Definition 4. *Complexity class* P
P $= \{L :$ *there is a polynomial time DTM program M for which* $L = L_M\}$

Going further to nondeterministic algorithms and the complexity class NP we first give an intuitive explanation of this class. We can informally classify a problem to belong to the complexity class NP if we can devise a *guess-and-check* algorithm. First, we nondeterministically *guess* an arbitrary solution candidate S. When speaking about languages, each possible string from Γ^* can be guessed since we are doing this nondeterministically. After the guessing phase we *check* if the guessed string is actually a *yes*-instance. Now, if the string is polynomially bounded in the size of Γ and the checking algorithm runs in polynomial time we say that the decision program lies in the complexity class NP. More formally, we extend the model of a DTM to a *nondeterministic one-tape Turing machine* (NDTM) by adding a *guessing module* which has its own *write-only head.* A program of a NDTM is defined the same way as for a DTM but the computation has an additional guessing phase before it continues working like the DTM. In the guessing phase the write-only head writes an arbitrary string from Γ^* on the tape. Then, the checking phase starts with the same rules as for the DTM. In contrast to the DTM from before we say that an NDTM program M *accepts* a given input string x if at least one of all (infinitely many) possible guessed strings will result in a halt in the state q_Y. We define the complexity class NP as follows:

Definition 5. *Complexity class* NP
NP $= \{L :$ *there is a polynomial time NDTM program M for which* $L = L_M\}$

Having defined the most important complexity classes P and NP, we continue by showing how to identify NP-hard problems. Therefore, we first have to introduce the notion of *polynomial reductions* between two languages. Formally we define a *polynomial reduction* from a language $L_1 \subseteq \Sigma_1^*$ to another language $L_2 \subseteq \Sigma_2^*$ (we write $L_1 \propto L_2$) to be a function $f : \Sigma_1^* \to \Sigma_2^*$ that satisfies the following two conditions [56]:

1. There is a polynomial DTM program that computes f.
2. For all $x \in \Sigma_1^*, x \in L_1$ if and only if $f(x) \in L_2$.

Informally spoken, the reduction must be computable in polynomial time and after the transformation the problem must be solvable with any algorithm that solves the corresponding problem of L_2. Then, we say that a language L is *NP-hard* if for all other languages $L' \in NP$, $L' \propto L$ holds. Correspondingly, a language L is *NP-complete* if $L \in NP$ and L is NP-hard. Proving that a problem Π is NP-complete can be done methodologically by first devising a guess-and-check algorithm to prove that $\Pi \in NP$ and then taking any problem which is known to be NP-hard and find a polynomial reduction to Π which shows that it is also NP-hard.

Now that we have defined that NP-complete problems are in some sense the *hardest* problems in NP one could raise the question what happens if we leave the class NP and take a look at even higher levels of complexity. Therefore, we first introduce the notion of an *oracle* and an *oracle Turing machine.* Intuitively, an oracle is an algorithm (or program, subroutine, ...) which solves a specific problem in zero time. Such an oracle can be used to further specify the complexity of a given problem in the following way: Suppose, we know that a subproblem Π' of the input problem Π has a known complexity (and can possibly be solved easily). What is the remaining complexity of problem Π? To comply with our notion of Turing machines, we define an *oracle Turing machine* (OTM) as an extension of a DTM. An OTM has an additional tape and additional states: a query state q_c and two answer states of the oracle q^c_{Y}, and q^c_{N}. The computation of an OTM program works similarly to that of a DTM, except that if the current state is q_c, then it consults an oracle and gets an answer from the oracle (state q^c_{Y} or q^c_{N}) in one step. By using this definition we introduce a notation for new complexity classes. We write $C_1^{C_2}$ for problems which can be decided by an OTM within the time bound given by complexity class C_1 and an oracle for any problem in the complexity class C_2. On basis of this definition Meyer and Stockmeyer [96] observed that such a structure can be extended indefinitely and thereby introduced the *polynomial hierarchy*, which is defined as follows [56]:

Definition 6. *Polynomial Hierarchy*

$$\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathrm{P}$$

and for all $k \geq 0$

$$\begin{aligned}
\Delta_{k+1}^p &= \mathrm{P}^{\Sigma_k^p} \\
\Sigma_{k+1}^p &= \mathrm{NP}^{\Sigma_k^p} \\
\Pi_{k+1}^p &= co\text{-}\Sigma_{k+1}^p
\end{aligned}$$

Although we so far did not mention *co*-classes here, our focus lies on the class Σ_k^p and therefore we refer the interested reader to the literature (e.g., [56, 103]). An easy example of a set of problems in Σ_k^p is based on *quantified boolean formulas* (QBFs). Suppose we are given a well-formed boolean expression ϕ with boolean variables partitioned into i disjoint sets $X_1, \ldots, X_i$. Then, the *QBF problem with i alternating quantifiers* asks the question if *there is* a truth assignment for the variables in X_1 such that *for all* truth assignments for X_2 *there exists* a truth assignment for $X_3, \ldots$ such that ϕ is satisfied.

Such an alternating behavior is characteristic for problems of the polynomial hierarchy. Although it is not known yet if any of the relations $\Delta_k^p = \Sigma_k^p = \Pi_k^p = NP = P$, $\forall k > 1$ hold, it is assumed that those problems are considerably harder to solve than any problem of the class NP. Chapter 4 considers a COP of practical relevance that is Σ_2^p-hard and in Chapter 5 solution methods for an NP-hard problem are developed.

2.3 Exact Methods

Many exact solution methods for COPs are based on integer linear programming (ILP), which is in principle a modeling technique. In this section we will define and briefly discuss ILPs and show how they can be solved. This section is based on the books *Introduction to Linear Optimization* by Bertsimas and Tsitsiklis [12] and *Integer Programming* by Wolsey [128].

2.3.1 (Mixed) Integer Linear Programming

First, we start by giving the definition of a linear program (LP). Suppose that we have a vector of n continuous decision variables $x = (x_1, \dots, x_n)$ and an associated cost vector $c = (c_1, \dots, c_n)$. Furthermore, we are given m constraints via an $m \times n$ matrix A and an m-dimensional column vector b. A linear program in general form is stated as follows:

Definition 7. *Linear Program*

$$\begin{aligned} \min \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \mathbb{R}^n \end{aligned}$$

We assume in Definition 7 that a minimization problem is considered and that all the constraints have the same structure, i.e., are expressed as greater-than inequalities. This is, however, not a restriction because of the following transformation rules:

$$\begin{aligned} &\max \mathbf{c}^{\mathrm{T}}\mathbf{x} \Leftrightarrow \min -\mathbf{c}^{\mathrm{T}}\mathbf{x} \\ &\mathbf{a_i}^{\mathrm{T}}\mathbf{x} = b_i \Leftrightarrow \mathbf{a_i}^{\mathrm{T}}\mathbf{x} \leq b_i \wedge \mathbf{a_i}^{\mathrm{T}}\mathbf{x} \geq b_i \\ &\mathbf{a_i}^{\mathrm{T}}\mathbf{x} \leq b_i \Leftrightarrow -\mathbf{a_i}^{\mathrm{T}}\mathbf{x} \geq -b_i \end{aligned}$$

For showing how to solve such a linear program we first define the notion of a *polyhedron* [128], *active constraints*, a *basic solution*, and a *basic feasible solution* [12].

Definition 8. *A subset of $\mathbb{R}^n$ described by a finite set of linear constraints $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ is a* polyhedron.

Definition 9. *If a vector x^* satisfies $a_i^{\mathrm{T}} x^* = b_i$ for some $i = 1, \dots, m$, we say that the corresponding constraint is* active *at x^*.*

Definition 10. *Consider a polyhedron P defined by linear equality and inequality constraints, and let x^* be an element of $\mathbb{R}^n$.*

(a) The vector x^* is a basic solution *if:*

(i) All equality constraints are active;

(ii) Out of the constraints that are active at x^*, there are n of them that are linearly independent.

(b) If x^* is a basic solution that satisfies all the constraints, we call it a basic feasible solution.

There are several solution algorithm for LPs described in the literature and the most practically important one is the *simplex* method. As a detailed discussion of it is out of scope of this thesis, we only sketch its principles here, a more detailed description can be found in Bertsimas and Tsitsiklis [12, Chapter 3]. The working principle of the simplex algorithm is based on the fact that if an LP has an optimal solution then there exists an optimal basic feasible solution [12]. The algorithm starts at an arbitrary basic feasible solution and moves to another basic feasible solution by exchanging one active variable with another in a direction which reduces the costs. After a finite number of such steps there is no direction available which reduces the costs and at that point we know that the current basic feasible solution is optimal. Although the simplex algorithm has an exponential worst case run-time, in practice it is most often the fastest solution method.

Apart from the simplex method there are two other noteworthy solution algorithms for LPs:

- *Ellipsoid method* [123, 130] – A rather theoretical algorithm which is not practically efficient but showed that linear programs are efficiently, i.e., in polynomial time solvable.

- Interior point methods [78] – These methods have a practical relevance as they are frequently competitive so the simplex method, and even able to outperform it on certain kind of problems. They are called interior point methods because they find an optimal solution while moving in the interior of the feasible set, in contrast to the simplex method which moves on the borders. A detailed description on interior point methods and several variants are described in Bertsimas and Tsitsiklis [12, Chapter 9].

Being able to solve LPs is the basis of solving ILPs which are the basis for modeling discrete problems such as COPs involving integral decision variables. In the following we will briefly discuss ILPs which is succeeded by a description of a solution method in Section 2.3.2. An ILP is defined as follows:

Definition 11. *Integer Linear Program*

$$\begin{aligned} \min \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n \end{aligned}$$

In Definition 11 we see that the essential difference of an LP and an ILP is the *integrality condition* of the decision variables. Unfortunately, this integrality condition makes the problem NP-hard [79] and in general also much more difficult to solve in practice.

Two variants of ILPs are mixed integer linear programs (MILPs) in which only some of the decision variables need to be integral and 0-1 or binary integer programs (BIPs) in which all of the decision variables need to be either 0 or 1.

2.3.2 (LP-based) Branch-and-Bound

As we saw in the previous section, LPs can be solved efficiently in polynomial time. As this is not the case anymore for ILPs and MILPs another solution algorithms are required. The most common solution technique for solving ILPs is LP-based branch-and-bound (B&B) which follows the principle of divide and conquer and is based on the following observation [128]:

Proposition 1. *We are given the problem*

$$z = \min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} : \mathbf{x} \in S\}$$

Let $S = S_1 \cup \cdots \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} : \mathbf{x} \in S_k\}$ for $k = 1, \ldots, K$. Then $z = \min_k z^k$.

We conclude from Proposition 1 that we do not need to solve the problem in its entirety but it is sufficient to find a suitable decomposition and solve all the resulting subproblems. These subproblems can be decomposed again and this procedure can be recursively repeated until the resulting problems cannot be split further and a complete enumeration tree is created. Completely enumerating all possible solutions to a given problem is usually not a practical approach, so a bounding procedure is added to prematurely discard subproblems that cannot yield a better solution than already known. Suppose that we have a procedure which computes lower / upper bounds to the optimal value of the subproblems. Note that for minimization problems the objective value of every feasible solution is always an upper bound. We use the following proposition to restrict the number of problems we need to solve [128]:

Proposition 2. *Let $S = S_1 \cup \cdots \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} : \mathbf{x} \in S_k\}$ for $k = 1, \ldots, K$, $\overline{z}^k$ be an upper bound on z^k and $\underline{z}^k$ be a lower bound on z^k. Then $\overline{z} = \min_k \overline{z}^k$ is an upper bound on z and $\underline{z} = \min_k \underline{z}^k$ is a lower bound on z.*

Using Proposition 2 three cases can be identified in which the subtree of the given subproblem $z^k = \min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} : \mathbf{x} \in S_k\}$ does not have to be examined any further:

(i) *Prune by optimality*: If $\overline{z}^k = \underline{z}^k$ the exact value of z^k is known and the problem is solved.

(ii) *Prune by bound*: If $\underline{z}^k \geq \overline{z}$ the optimal solution cannot be in S_k.

(iii) *Prune by infeasibility*: If $S_k = \emptyset$ no solution lies in S_k.

Based on the above considerations we can devise a BNB algorithm which is shown in Algorithm 2.1. The most common way for computing bounds of ILPs within a branch-and-bound algorithm is to solve the linear programming (LP) relaxation of the given problem:

Definition 12. *Given the Integer Linear Program* $\min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} : \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^n\}$*, the* LP relaxation *is the solution to the simplified problem* $\min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} : \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n\}$.

Algorithm 2.1: LP-based Branch-and-Bound

```
Input: ILP $\min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} : \mathbf{x} \in S\}$
Output: Optimal solution $x^*$
1  $L = \{S\}$;
2  $x^* = $ NULL;
3  $\overline{z} = \infty$;
4  while $L \neq \emptyset$
5      Choose next problem $S_k \in L$;
6      $L = L \setminus S_k$;
7      Solve the LP relaxation of problem $S_k$;
8      let $\underline{z}^k$ be the resulting LP value;
9      let $x^k(LP)$ be the LP solution;
10     If $S_k = \emptyset$ then
11         prune by infeasibility;
12     Else if $\underline{z}^k \geq \overline{z}$ then
13         prune by bound;
14     Else if $x^k(LP) \in S_k$ then
15         $\overline{z} = \underline{z}^k$;
16         $x^* = x^k(LP)$;
17         prune by optimality;
18     Else
19         Generate new subproblems $S_k^1$ and $S_k^2$;
20         $L = L \cup \{S_k^1, S_k^2\}$;
21 return $x^*$;
```

In Algorithm 2.1 first the variables are initialized and the original problem is added to the list of open problems L. Then, until L is empty a subproblem S_k is chosen from L and the LP relaxation of S_k is solved. Based on this value, the node is either pruned or two new subproblems based on S_k are generated and inserted into L. At the end the best found solution x^* is returned, which is an optimal solution to the overall problem.

There are, however, still two decisions to be made:

- How to generate new subproblems?
- How to choose the next subproblem?

For choosing the next subproblem there are two basic possibilities: A *depth-first search* strategy descends further into the tree to hopefully find a feasible solution and thus a good upper bound soon. Contrary, a *best-node first* strategy first examines the node with the best lower bound to minimize the number of node evaluations. Combination of these strategies or more advanced node selection strategies are also possible also described in the literature, see, e.g., [55].

By using the LP-relaxation for computing lower bounds there is a natural and common way to generate two new subproblems. First, we choose a variable x_i with a fractional value x_i^k in the LP solution. Then, in one subproblem the constraint $x_i \leq \lfloor x_i^k \rfloor$ and in the other subproblem the constraint $x_i \geq \lceil x_i^k \rceil$ is added.

Cutting Plane Methods and Branch-and-Cut

LP-based branch-and-bound strongly relies on good bounds and therefore there are some methods to strengthen the bound obtained from the LP relaxation. It is also possible that a reasonable ILP formulation has an exponentially large number of constraints which cannot be completely enumerated. In both cases a *cutting plane* approach can be used in order to iteratively add constraints, resolve the problem, and thereby strengthen the obtained bound. These steps can be repeated until an optimal solution is found. The process of finding suitable constraints to add in this way on the fly is known as the *separation problem* and is often a non-trivial task. One must find at least one constraint which is valid for the problem but violated by the current (LP) solution. As the cutting plane method would in many cases add an exponential number of constraints it is a common approach to incorporate such cutting plane methods within a branch-and-bound algorithm. This method is known as *branch-and-cut* (B&C). Such algorithms apply a cutting plane approach in every node of the branch-and-bound tree in addition to solving the LP relaxation. Therefore, usually stronger bounds are obtained and less nodes have to be examined. A similar approach is used in Section 5.4, where the exact L-shaped method, which is a kind of a B&C algorithm for stochastic problems, is used to solve the generalized vehicle routing problem with stochastic demands.

Column Generation and Branch-and-Price

A kind of dual approach for tightening the LP relaxation is *column generation* (CG). In contrast to the cutting plane method, variables are iteratively added to the model instead of constraints. Especially when the ILP formulation has a large (exponential) number of variables this approach can be viable. The working principle of CG is that initially the model contains only a small set of variables. Then, variables which may improve the LP relaxation value are iteratively added to the model, which is then resolved. Finding such variables is called the *pricing problem* and, similar to the B&C algorithm, it can often be a difficult task to solve on its own that has to be performed for each problem individually. For more information about column generation the reader is referred to the book by Desaulniers et al. [41].

Column generation can also be used within a branch-and-bound framework in a similar fashion as the cutting plane method, which is then called *branch-and-price* (B&P). In a B&P algorithm the LP relaxation is solved in each node using the CG method.

Finally, branch-and-cut-and-price approaches combine the cutting plane method and column generation with B&B.

2.4 Heuristic and Metaheuristic Methods

In contrast to the exact methods from the previous section, which are in principle guaranteed to find an optimal solution to COPs, providing one exists, in many cases it is sufficient to provide only a near optimal or high-quality solution. Especially when the problem instances are large, those exact algorithms may not find the optimal solution, or, even worse, are not able to find any feasible solution, in a reasonable amount of time. In practical applications time is often crucial and therefore solution quality is traded for shorter running time. In those scenarios heuristic solution methods and especially metaheuristics come into play as they are known to be frequently able to provide near optimal solutions relatively quickly. Parts of this section are based on the *Handbook of Metaheuristics* edited by Gendreau and Potvin [60] and for an overview of different metaheuristics we refer to, e.g., [59, 28]. First, we describe construction heuristics and local search methods.

2.4.1 Construction Heuristics

Construction heuristics are typically relatively fast and intuitive. The methods range from simple ones, which construct solutions by iteratively adding arbitrary or randomly selected solution components to more complex algorithms that use more sophisticated heuristic information for the construction. A common approach are *greedy* algorithms which iteratively build a solution by always adding a locally best component according to some selected criterion.

Algorithm 2.2 illustrates a greedy construction heuristic (GCH) in pseudocode. The GCH starts with an empty solution S and iteratively adds solution components to it. Therefore, for each feasible component i its incremental costs are computed. In each iteration a cheapest component is chosen and added to S until the whole solution is constructed.

Algorithm 2.2: Greedy Construction Heuristic

```
  Input: COP $\Pi$
  Output: Feasible solution $S$
1 $S = \emptyset$;
2 while further components can / need to be added to $S$
3     Let $C$ be the set of feasible components to extend $S$;
4     Determine cost $c_i$ for adding component $i$, $\forall i \in C$;
5     Find component $s = \text{argmin}_{s \in C} c_s$;
6     $S = S \cup s$;
7 return $S$;
```

Let us consider two examples of GCHs for the traveling salesman problem (TSP). In the TSP we are given a set of cities $V = \{v_1, \dots, v_n\}$, a starting city $v_s \in V$, and a distance function $d : V \times V \mapsto \mathbb{R}$. The aim is to find a tour through all cities of minimum length starting and ending at v_s. The *nearest neighbor heuristic* starts at v_s and iteratively appends a not yet added $v_i \in V$ to the tour with minimum distance to the last added city. In contrast, the *insertion heuristic* iteratively adds cities to the initially empty tour by choosing a city and an insertion position so that the additional traveled distance is increased less.

2.4.2 Local Search

Solutions obtained from simple construction heuristics are often not good enough. It is therefore natural to try to improve such solutions. Local Search provides a systematic way to do this. It is based on the notion of a *neighborhood structure*, which defines a *neighborhood* for each solution x in the search space. A neighborhood is a set of solution candidates, which are in some sense "near" to x. More formally, suppose we have the set of feasible solutions S to a given problem.

Definition 13. *Neighborhood Structure*
A neighborhood structure *is a function $N : S \to 2^S$, that assigns to each solution candidate $x \in S$ a set of* neighbors *$N(x) \subseteq S$. The function $N(x)$ is called the* neighborhood *of solution candidate x.*

Usually, neighborhood structures are not defined explicitly but by a description of valid *moves* from a given solution. Such a move defines the structure of an allowed change of the solution. A prominent example of a neighborhood structure for the TSP (as defined in Section 2.4.1) is *2-opt*. A 2-opt move deletes two edges $(v_i, v_j), (v'_i, v'_j) \in V \times V$ of an existing solution and inserts them again by connecting v_i with v'_i and v_j with v'_j. This basically changes the visit sequence of the cities between i and j' and removes edge crossings on Euclidean instances. Another well-known neighborhood structure, which is applicable for a solution representation using binary strings, is the *flip* neighborhood structure. A move in the flip neighborhood changes the value of a variable from zero to one or the other way round, i.e., it *flips* a bit.

The defined neighborhood is then searched for a solution candidate x' with a better objective value than the starting solution x. If there is such a solution candidate the next iteration of the LS is started with x' as the new incumbent solution. The procedure is repeated as long as an improving solution candidate is found or another stopping criterion, e.g., a time limit is satisfied.

Algorithm 2.3: Local search

```
  Input: Initial solution $x$
  Output: Possibly improved solution
1 repeat
2   choose an $x' \in N(x)$;
3   if $x'$ has a better objective value than $x$ then
4     $x = x'$;
5 until stopping criterion satisfied;
6 return $x$;
```

Algorithm 2.3 shows a general LS procedure in pseudocode. In line 2 of the algorithm a neighbor of solution x' is chosen but it is not specified which one. There are these commonly applied strategies:

- *Random improvement*: A neighbor is chosen randomly, which is computationally inexpensive but the resulting solution candidate can frequently be worse than the incumbent.
- *Next improvement*: A systematic search of $N(x)$ is performed and the first improving solution is chosen. With this method it is guaranteed to find a better solution if there exists one but it is not guaranteed that this is an overall best neighbor. Compared to random improvement this is a more expensive step function but probably it likely has a better outcome.
- *Best improvement*: With the best improvement step function the whole neighborhood is searched and a solution with the best objective value always is returned.

If the LS terminates with a solution x and there does not exist any neighboring solution $x' \in N(x)$ with a better objective value, then the solution x is said to be *locally optimal* with respect to the neighborhood $N(x)$. Figure 2.1 shows that such a local optimum is not necessarily a *global optimum.* In the next sections we will see how to escape from such a local optimum in order to find the global optimum more likely.

2.4.3 Metaheuristics

Construction heuristics and local search methods suffer from the already mentioned drawback of locality. They are often trapped in local optima with no possibility to escape

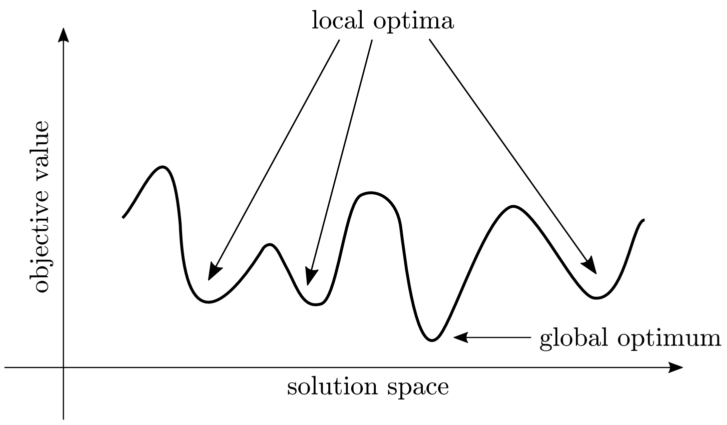


Figure 2.1: Local vs. global optimum

them. This is where metaheuristic come into play which introduce a way of getting out of local optima. So, we identify two important principles every metaheuristic should follow:

- *Exploration*: The metaheuristic search should cover a large part of the search space to identify promising areas. Thus, a method to diversify the search to escape local optima has to be employed.
- *Exploitation*: When promising areas in the search space are identified they should be more intensively investigated for the best solution within this area, which is often achieved by local search procedures, see Section 2.4.2.

In any metaheuristic it is usually important to find a right balance between exploration and exploitation. For a formal definition of the term metaheuristic, several variants can be found in the literature. Bianchi et al. [15] state them as follows:

Definition 14. *Metaheuristic*
A metaheuristic *is a higher-level procedure or heuristic designed to find, generate, or select a heuristic that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity.*

In the next sections we present several metaheuristics which are well-known in the literature, e.g., variable neighborhood search, tabu search, and genetic algorithms. The focus lies on those which are later used to solve the problems introduced in Chapter 4 and 5.

2.4.4 Variable Neighborhood Descent

The first step towards escaping local optima is based on the observation that if a solution x is locally optimal for some neighborhood $N_1(x)$ this does not necessarily mean that x is locally optimal for another neighborhood $N_2(x)$. A *variable neighborhood descent* [69]

(VND) exploits this observation and uses different neighborhood structures for its search. Let us suppose we have $k_{\max}$ neighborhood structures which are denoted by $N_1, \ldots, N_{k_{\max}}$ and we use the *next* or *best improvement* step function. The pseudocode of a VND is shown in Algorithm 2.4.

Algorithm 2.4: Variable neighborhood descent

```
   Input: Initial solution $x$
   Output: Locally optimal solution with respect to all neighborhoods $N_1, \ldots, N_{k_{\max}}$
1  $k = 1$;
2  repeat
3      choose $x' \in N_k(x)$;
4      if $x'$ is better than $x$ then
5          $x = x'$;
6          $k = 1$;
7      else
8          $k = k + 1$;
9  until $k == k_{\max}$;
10 return $x$;
```

At the beginning, the first neighborhood of the starting solution is searched. If the obtained neighbor has a better objective value than the original solution the neighborhood structure stays the same and the search continues from the new solution candidate. However, if the solution is already locally optimal for the neighborhood then the structure is changed to the next one. This process is repeated until no improvement was found in neighborhood $N_{k_{\max}}(x)$. Then we know that the solution x is locally optimal for all neighborhoods $N_1, \ldots, N_{k_{\max}}(x)$. The described strategy follows a *sequential* ordering of the neighborhood structures and determining such an ordering is also a design decision. There is, however, an alternative *nested* structure, which, e.g., performs a VND using the first two neighborhood structures for each solution of a third neighborhood structure [69].

2.4.5 Variable Neighborhood Search

Variable neighborhood search (VNS), introduced by Hansen and Mladenović [98], is related to VND but does not systematically search the neighborhoods. Instead, it typically applies larger neighborhoods and the random neighbor step function, whose application is called *shaking* in this context. Each solution obtained from shaking undergoes usually some local improvement, which can be an embedded LS or even a VND. In the latter case, the whole approach is also called a general VNS (GVNS). Let $N^s_1, \ldots, N^s_{l_{\max}}$ be the sequence of shaking neighborhood structures to be used.

Algorithm 2.5 shows the VNS in pseudocode. In line 4 a random neighbor is chosen from the current shaking neighborhood, which corresponds to the exploration step. The obtained solution is locally improved (emphasis on exploitation). Based on the objective value of the resulting solution the shaking neighborhood structure either stays the same

Algorithm 2.5: Variable neighborhood search

```
   Input: Initial solution $x$
   Output: Best found solution
 1 repeat
 2   $l = 1$;
 3   repeat
 4     choose a random neighbor $x' \in N_l^s(x)$;
 5     $x''$ = possibly locally improve $x'$
 6     if $x''$ has a better objective value than $x$ then
 7       $x = x''$;
 8       $l = 1$;
 9     else
10       $l = l + 1$;
11   until $l == l_{\max}$;
12 until stopping criterion satisfied;
13 return $x$;
```

or changes. This process is repeated until a prespecified stopping criterion is satisfied which is in many cases a time limit, a maximum number of iterations, or a convergence criterion. The shaking neighborhood structures are often ordered in increasing size so that in the later iterations also solutions are considered which are “farther” away in some sense but there exist also adaptive approaches [69].

In the literature there are several further variants of VNS described, e.g., skewed VNS, variable neighborhood decomposition search. In a skewed VNS the algorithm sometimes accepts a slightly worse solution in order to reduce the shaking to solutions which are far away which would lead to a degeneration. A variable neighborhood decomposition search splits up the VNS in a two-level approach, in which first a set of attributes of a solution is fixed and then a local search is performed on the remaining attributes. For more information about VNS see Hansen et al. [69].

2.4.6 Tabu Search

Another class of local search based metaheuristics is tabu search (TS). TS is based on a local search and a memory storing some history of the search, which is used as a diversification mechanism to avoid getting stuck in poor local optima. In the basic TS the memory is only short-term and usually stores specific changes, so called *tabu attributes* to a solution candidate derived from performed moves from solutions to their neighbors in a *tabu list*. Moves matching attributes in the tabu list are forbidden for a prespecified number of iterations. The tabu attributes are typically chosen so that the reversal of performed moves is prohibited in the near future. This number of iterations is called the *tabu tenure*. Examples of tabu attributes can be shown by taking the neighborhood structures introduced in Section 2.4.2. For 2-opt a reasonable tabu attribute would be the newly added edges; for the next few iterations it is prohibited to remove them again. A possible tabu attribute for the flip neighborhood structure is the flipped bit, which is not allowed to flip again for the next few iterations. A different but rather unusual approach is to store whole solutions, which are prohibited in the next iterations. Gendreau and Potvin [61] claim that this approach consumes a lot of storage and is expensive but we will see in Chapter 3 a more general, efficient approach how to forbid the creation of already considered solution candidates by using solution archives.

As the main reason for forbidding moves is to avoid cycling between (a possibly small set) of solution candidates it could happen that the tabu list is too strict and potentially good solution candidates are missed. Therefore, one could use an *aspiration criterion* to accept a move even though it is tabu. A common aspiration criterion is that a solution candidate is accepted if it would improve the best solution found so far.

Algorithm 2.6: Tabu search

```
   Input: Initial solution $x$
   Output: Best found solution $x^*$
1  $x^* = x$;
2  repeat
3      get set of valid neighbors $N'(x) \subseteq N(x)$;
4      choose best $x' \in N'(x)$;
5      if $x'$ has a better objective value than $x^*$ then
6          $x^* = x'$;
7      $x = x'$;
8      update tabu list;
9  until stopping criterion satisfied;
10 return $x^*$;
```

A pseudocode of a TS shown in Algorithm 2.6. In line 3 of the algorithm the set of valid neighboring solution candidates based on the neighborhood structure N, the tabu list, and the aspiration criterion is built. Then, the best solution of this set is always accepted for the next iteration even though it might be worse than the original solution. This avoids getting trapped in a local optimum and makes it possible to explore other parts of the search space.

A common challenge of designing an efficient TS lies in the exploration of the search space. As this method heavily relies on local search technique it also suffers from the same drawback as the local search, the locality. Gendreau and Potvin [61] give two suggestions to overcome this problem: *Restart diversification* forces rarely used components and restarts the search at that point and *continuous diversification* alters the objective function to prefer the use of moves involving such rarely used components.

Another critical decision when designing a TS is the choice of an appropriate tabu tenure. If it is too short the TS might not be able to reliably escape local optima and cycling would be possible but if it is too long the search is too restricted and would forbid too many possible neighbors. A method to overcome the problem of choosing an appropriate tabu tenure beforehand is a *reactive tabu search* [9], in which the tabu tenure is automatically adapted by reacting to the occurrence of cycles.

2.4.7 Genetic Algorithms

Compared to the previous metaheuristic, which all work on a single solution there also exist population based metaheuristics. The most prominent representative of this category are *genetic algorithms* (GAs). Originally introduced by Holland [71] in 1975 GAs evolved and grew in popularity over the last decades. They are inspired by principles of Darwin's theory of evolution [36] and imitate several concepts of evolution like survival of the fittest, sexual reproduction, and mutation in an abstract way.

A GA works on a *population* of solution candidates (in this context also called *chromosomes* or *individuals*) which are encoded in a specific representation. In the literature the representation of the variables of these solution candidates is referred to as *genotype*, which decodes to the *phenotype*, which is the natural expression of a solution [113]. A *fitness* value is associated to each chromosome which is computed by the specified *fitness function.* Usually, a higher fitness represents a better solution candidate and this fitness has a direct relation to the objective function of the problem to be solved. The aim of the GA is to find an individual with the highest possible fitness, which is then decoded into the actual solution to the COP. In some cases it is beneficial to reduce the search space of the GA by choosing a suitable representation and to shift a part of the complexity into this decoding procedure. Then, we generally speak of an *incomplete* solution representation. Algorithm 2.7 shows a canonical GA in pseudocode.

Algorithm 2.7: Genetic Algorithm

```
   Output: Best found solution
1  $t = 0$;
2  $P(t)$ = initialize();
3  evaluate($P(t)$);
4  repeat
5      $t = t + 1$;
6      $Q_s(t)$ = select($P(t-1)$);
7      $Q_r(t)$ = recombine($Q_s(t)$);
8      $Q_m(t)$ = mutate($Q_r(t)$);
9      $P(t)$ = replace($P(t-1), Q_m(t)$);
10     evaluate($P(t)$);
11 until stopping criterion satisfied;
12 return $x \in P(t)$ with the best fitness;
```

First, the population has to be initialized, which is often done by some simple random solution construction but also more elaborate initialization procedures are used. When designing a suitable method one should strive for diversity in the initial population in order to avoid converging too quickly into suboptimal areas of the search space. It is also possible to construct the initial population more intelligently by construction heuristics, as described in Section 2.4.1 if properly randomized. In line 3 of the algorithm the whole population is evaluated, which means that the fitness function is computed for each generated individual to assess its quality. Then, the main loop of the GA starts with the four steps *selection*, *recombination* (also called *crossover*), *mutation*, and *replacement*.

In the selection process individuals are selected that will be recombined. Fitter individuals should be selected with higher probability and therefore a typical selection is some sort of *fitness proportional selection*, where the basic variant is the *roulette-wheel selection* (RWS). In an RWS each individual is selected with a probability which is proportional to its fitness. This method suffers, however, from a high variability and in practice often *stochastic universal sampling* (SUS) is used [113]. Similar to the one-arm roulette-wheel from the RWS, which points to exactly one individual, in the SUS method an equally spaced m-arm spinner is used, which points at m individuals simultaneously. Both methods, however, have the problem of finding an appropriate measure of fitness respecting the scale of fitness values and also possibly negative fitness values. This problem can be overcome by using some scaling mechanism [63]. More robust alternatives to the classical fitness proportional selection are rank-based methods such as *tournament selection* which chooses a certain number of individuals uniformly at random and selects the best individual among them.

For the recombination of two (or more, in some cases) individuals many different methods are possible which are usually designed to incorporate problem specific knowledge. The main idea is to create new solutions from properties appearing in the parent solutions. There exist, however, several standard operators which will be briefly surveyed here. Given a binary string of length n as solution representation, then we choose a random point $1 \leq l \leq n$ in this string to perform a *one-point crossover*. We adopt the values of the variables on positions $1, \dots, l$ from the first parent and the other values from the second parent to create a new offspring. Performing this procedure the other way round a second offspring is generated immediately. Other possibilities for crossover operators are, for example, *uniform crossover*, in which it is randomly decided for each variable individually from which parent it inherits its value or *k-point crossover*. Specifically for permutation representations there exist other crossover operators like *partially mapped crossover*, *cyclic crossover*, see [101] for a study on different permutation crossover operators.

The mutation operator is used in a GA to (re-)introduce "genetic material", i.e., variable values that do not appear in the current population. A mutation changes one or more variables to another, usually randomly chosen, value and is typically not performed for each individual in each generation but only with a specific *mutation probability*. Staying at the previous example of the binary string, a mutation would, e.g., flip one bit at a randomly chosen location.

In the original GA as proposed by Holland [71], a new population is generated in each iteration based on selection, recombination, and mutation which then replaces the previous population. A reasonable extension, referred to as *elitism*, was introduced by De Jong [40]. A small set of the best individuals of one population always is directly adopted by the next population. This idea can even be extended to a *steady-state genetic algorithm* which produces only exactly one new individual per iteration which then replaces the individual with the worst fitness of the current population. In this step often a diversity management strategy is used to maintain (or even increase) the diversity of the population. A common method is to forbid duplicate solutions within the same population as this could lead to premature convergence. In this thesis the focus lies also on the avoidance of duplicate solutions but during the whole search process instead of only the current population. In Chapter 3 the concept of complete solution archives is introduced and it is also explained how it can be incorporated in metaheuristics in general and GAs in particular.

As a last note on GAs it should be mentioned that they often suffer from the lack of a proper exploitation technique, which is the reason why they are often combined with local search as subprocedure to intensify the search. This combination is frequently called *memetic algorithm* and was introduced by Moscato [99]

2.5 Hybrid Metaheuristics

In the previous two chapters rather "pure" exact and (meta-)heuristic solution approaches were considered. In practice, however, *hybrid* techniques, which combine different kinds of methods, in particular also exact and / or heuristic algorithms, are often among the most successful approaches for many problems. This chapter therefore gives a brief overview on some possibilities to combine exact and heuristic approaches in a meaningful way to benefit from synergy and exploit the individual advantages. As the hybridization possibilities are huge it would be out of scope of this thesis to describe them here in detail. For detailed surveys on different hybrid metaheuristics we refer to [111, 110, 26, 126] and to the recent book by Blum and Raidl [27] on which the remainder of this section is based.

2.5.1 Guidance for (Meta-)heuristics by Solving Problem Relaxations

In the context of mixed integer linear programming, problem relaxations can be used as guidance or a starting point for heuristics and metaheuristics. When solving the LP relaxation of a given formulation, the information of the non-integral variables or other dual information can be used within a heuristic. Chu and Beasley [35] and Puchinger et al. [109] make use of so-called *pseudo-utility ratios* for the primal variables for multidimensional knapsack problems and use them to guide repair and local improvement methods.

As we have seen in Section 2.4, for metaheuristics it is often crucial to start their search at a good initial solution. Especially for evolutionary methods like GAs a set of diverse and high-quality solutions is important for their performance. Several initialization algorithms can be used to create the initial population, which could have a different focus, e.g., part of the population is initialized randomly for a set of diverse individuals. For high-quality initial solutions exact algorithms, which solve a relaxed version of the original problem could be employed. The solution to this relaxed problem then can then be repaired and integrated into the population. Such a repair method can be, e.g., a rounding scheme which is applied to the solution of an LP relaxation. In one of the problems considered in this thesis such intelligent initialization methods based on solving an ILP model is used and explained in detail in Section 5.5.2.

2.5.2 Exact Methods Integrated in Metaheuristics

Exact methods can also be used as subordinate procedure for the search itself as embedded improvement method. A prominent example of such an approach is the exploration of a large neighborhood in the framework of a *(Very) Large Neighborhood Search* ((V)LNS) [122, 3] using exact approaches. In the LNS the neighborhood of a solution candidate is not exhaustively searched by complete enumeration like in the standard local search but more intelligently. Ahuja et al. [3] and Pisinger and Ropke [105] classify VLNS algorithms into three classes:

- Variable-Depth Methods, which are based on iteratively largen the neighborhood to escape local optima.
- Network flow-based improvement methods, which use neighborhood structures that can effectively searched by network flow algorithms.
- Efficiently solvable special cases, which are based on intelligently chosen neighborhood structures so that the remaining search space of the neighborhood can be searched efficiently, e.g., in polynomial time.

Such LNS approaches can be used in combination with constraint programming [122] and it is also possible to use other exact algorithms to search the large neighborhood, especially ILP methods. Moreover, adaptive LNS (ALNS) approaches have been described in the literature which use multiple destroy operators, which define the neighborhood, and repair mechanism, which define the way the neighborhood is searched. Especially for vehicle routing problems ALNS algorithms are successful, e.g., for the pickup and delivery problem with time windows [117].

2.5.3 Metaheuristics Integrated in Exact Methods

We have seen in the last section that exact methods can be used within metaheuristic but we can also consider the other way round. Within a branch-and-bound framework heuristics and metaheuristics can be used for initially finding good primal bounds and can also be used to solve subproblems occurring in the inner nodes. More complex applications of metaheuristics and especially local search methods within branch-and-bound are, e.g., *local branching* [52]. As the name indicates, local branching is a binary branching strategy in which the remaining search space is divided in the subspace spanned by the k-opt neighborhood of the incumbent solution and in the subspace containing the remaining solutions. Then, the MIP solver is forced to first search the solution space given by the k-opt neighborhood.

Also the proposed complete solution archive is a tree-based exact method which is combined with a metaheuristic to improve the performance of the overall algorithm. There, the metaheuristic guides the search and the decisions within the search tree, see Chapter 3 for further details.

2.5.4 Exact Methods for Decoding Indirect or Incomplete Solution Representations

As already pointed out in Section 2.7 in some cases, especially for GAs, it can make sense to encode the actual solution candidates of a problem using an indirect or incomplete solution representation. This reduces the search space of the metaheuristic and, if properly designed, makes the operators which work on solutions easier to design. For obtaining and evaluating the actual solution a decoding procedure has to be developed, which can be done using an exact algorithm. A prominent example of indirect solution representation are permutations, especially in the context of scheduling or packing problems [80]. The decoder then reconstructs a solution to the problem by using a construction algorithm which follows the order of the elements in the permutation. In this thesis such decoding methods based on solving ILPs as well as by using a dynamic programming algorithm are developed for both considered problems, see Chapter 4 and 5. Note that such an incomplete solution representation is especially useful when using complete solution archives, as it provides a compact encoding as well as frequently a time-consuming solution evaluation.

2.5.5 Hybridization Based on Problem Instance Reduction

Another hybridization is based on the observation that general MIP solvers are very effective for solving instances up to a certain size. When the problem instances can be reduced in such a way that their optimal solution is also an optimal or at least a high quality solution to the original problem, such MIP solvers can be used also for large instances. This problem instance reduction is achieved by considering only a subset of all solution components. An example of such a hybridization is the *Construct, Merge, Solve & Adapt* (CMSA) algorithm [25]. The CMSA algorithm first generates a number of feasible solutions to the original problem. Then all components of these generated solutions are merged into the incumbent solution, which is then solved exactly using a MIP solver. After a solution to the reduced problem has been found, some solution components of the incumbent solution are deleted by an aging mechanism to keep the size of the reduced instance small. Such an approach can in principle be applied to any problem for which there exists a way of probabilistically generating feasible solutions and a method for exactly solving smaller instances is known.

2.5.6 Parallel, Non-independent Construction of Solutions Within Metaheuristics

Dual bounds and parallel extensions of partial solutions for the solution construction can also be combined into a hybrid approach. The construction of solutions can be seen as an exploration of the search space in form of a tree in which the inner nodes represent partial solutions. A greedy algorithm, for example, always chooses the best child node with respect to a greedy criterion. Other methods expand the partial solution in a randomized way, e.g., by assigning a probability to each possibility using heuristic information. These expansions can be extended when additionally dual bounds are considered like in branch-and-bound algorithms. A systematic way to construct solutions in a parallel and non-independent way using dual bounds is a *Beam-ACO* [24], which combines the branch-and-bound derivate *beam search* [102] with an *ant colony optimization* [42], which is a metaheuristic based on iteratively constructing solutions.

2.5.7 Other Possibilities for Hybridization

In the literature there are many more important classes of hybridizations of metaheuristics and exact algorithms and some of them are described in the following.

Originally, the first combination of metaheuristics with other optimization techniques was the hybridization with another metaheuristic [27]. A typical hybridization are *memetic algorithms* [99], which combine genetic algorithms with local search techniques and have already been mentioned at the end of Section 2.4.7. It is also possible to integrate population based methods within the framework of local search techniques, e.g., in *population-based iterated local search* algorithms [124] and *population-based iterated greedy algorithms* [31].

In the area of mathematical programming, the special structure of practical problems can be exploited by devising a decomposition approach, which can provide a fruitful basis for hybrid metaheuristics. Prominent examples of such decompositions are *lagrangian decomposition*, *Danzig-Wolfe decomposition*, and *Benders decomposition* and we refer to Boschetti and Maniezzo [29, 30] and the survey by Raidl [111] for more information on such hybridizations.

CHAPTER 3

Complete Trie-Based Solution Archives

Before we proceed with the two problem classes which are considered in this thesis, first a general overview of the main principle of complete trie-based solution archives (SAs) is given. Under the term *complete solution archive* we understand here a data structure that stores all generated candidate solutions in a compact way with the ability to efficiently search for already contained solutions and in particular to convert them into guaranteed new ones. Especially evolutionary algorithms can benefit from such an archive because the on-the-fly conversion of already visited solution candidates increases diversity in the population, reduces the danger of premature convergence, and re-evaluations of already generated solution candidates are avoided completely. In principle, such a SA is able to turn a metaheuristic into a complete optimization approach always yielding a guaranteed optimal solution in bounded (but not necessarily practically acceptable) time after considering all solutions in the search space. In practice, however, the algorithm is typically terminated earlier and yields only a heuristic solution. Furthermore, one can also see the archive-enhanced metaheuristic as a hybridization with tree-search, and concepts from there, like pruning subspaces based on bounds, can be adopted.

In this chapter we are going to give a literature survey about other known techniques to avoid duplicate solutions during a heuristic search and introduce the basic general concept of SAs. The general trie structure is described as well as the fundamental solution insertion and conversion methods, although these are mostly problem-dependent. Finally, at the end of this chapter, the actual integration of SAs into metaheuristics, in particular GAs, is presented.

3.1 Duplicate Detection Strategies

The importance of duplicate detection, especially in evolutionary algorithms, has been pointed out already in the early 1980s by Mauldin [95], who demonstrated that maintaining diversity in each population can significantly improve the performance of genetic search. This result is not obvious at all since the removal of duplicate solutions may prevent elitism, which may be more important than the loss of diversity. This is, however, in general not the case and Ronald [116] argued that duplicate removal is not at odds with the basic mechanisms of genetic algorithms. He also introduced hash tagging as duplicate detection technique to avoid duplicates within the current population. Kratica [82] also described an approach using hash tables and Louis and Li [89] suggest the use of a binary search tree. In contrast to simple hashing-based approaches, in which duplicates are only discarded, there exist a few works where an archive is not just used to recognize duplicates, but more importantly to also efficiently convert them into similar not yet considered solutions. The first non-revisiting GA following such an approach was developed by Yuen and Chow [131] who implemented a complete SA based on a *k*-*d*-tree for solving continuous optimization problems. When a duplicate solution is detected a backtracking to a preceding node is performed and the solution is mutated to a not yet considered value with minimum (Euclidean) distance to the original solution. The authors also showed that the pruning of subtrees containing only solutions that have already been visited is isomorphic to a parameter-less self adaption mutation operator. The results of this work showed that the GA using this archive produced solutions of significantly higher quality on their six continuous benchmark functions.

Raidl and Hu [112] adapted and extended this idea for discrete search spaces and introduced complete SAs for GAs based on a so-called trie data structure. They compared the impact of using such a SA on the results of a developed genetic algorithm for several binary benchmark problems like Royal Road functions and NK landscapes. It turned out that the quality of solutions increased in most cases when using the solution archive while maintaining the same time limits. Ruthmair and Raidl [118] and Hu and Raidl [73] implemented SAs for more relevant, complex problems, the rooted delay-constrained minimum spanning tree (RDCMST) problem and the generalized minimum spanning tree (GMST) problem, respectively. For the RDCMST a solution representation was chosen which stores for each node of the tree a delay value and a decoding algorithm is used which constructs a solution based on these values. Upon a duplicate detection the conversion method of the SA changes one or more of these values to another not yet considered composition of delay values. For the GMST two different solution representations were used, the *spanned nodes representation* and the *global structure representation* which complement each other. For each solution representation an own SA was used an upon duplicate detection a conversion procedure is carried out in turn by both SAs and re-checked in the opposite trie until a solution has been derived which is new in both SAs. The used SAs for the GMST are extended in a follow-up work [74] with bounding extensions used to prune subspaces which improved the results significantly. From the results presented in these articles one can conclude that the use of such complete SAs

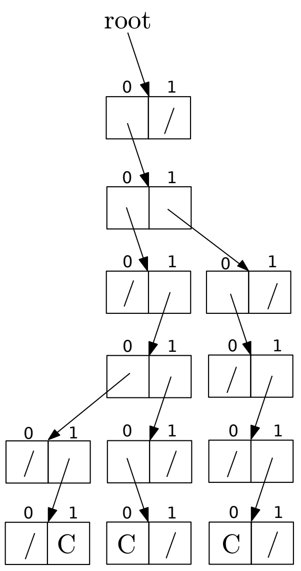


Figure 3.1: Example of a binary trie-based solution archive

are especially beneficial if the problem has a *compact solution representation* and the *solution evaluation is costly.* This thesis builds upon these results, extends the basic idea of SAs, and fruitfully applies the concept to new application domains.

3.2 Trie Structure

The underlying data structure of complete SAs, as considered in this thesis, is an indexed trie, which is a tree data structure often applied in dictionary applications [65]. On some memory-intensive applications also a linked trie could be used to trade memory consumption for run-time. For the performance of a SA it is important that inserting, searching and converting a solution can be performed efficiently. A trie has an exceptionally good performance for this purpose because basically all of these operations can be implemented in $\mathcal{O}(m)$ time (as we will see in Section 3.3 and 3.4), where m is the length of the solution representation. For scalability it is especially important that the run-time of these operations does not strongly increase with the number of solutions the solution archive contains.

Let us assume we are given a problem where solutions can be encoded as binary strings of length m and the feasible search space is $\{0,1\}^m$, i.e., the domain of each decision variable $x_1,\ldots,x_m$ is $\mathcal{A}=\{0,1\}$. Our indexed trie T has a maximum height of m and on each level $l=1,\ldots,m$ there exist at most 2^{l-1} trie nodes. Each trie node q at level l corresponds to variable x_l and has the same structure consisting of two entries q.next[0] and q.next[1]. Such an entry can be either a pointer to a successor node at the next level or is set to one of the two special flags *null* or *complete.* Each node of the trie refers to a part of the search space, and the root node at level 1 corresponds to the

```
Algorithm 3.1: insert($x, l, q$)
Global Variable:
$devpoints = \emptyset$;          // Set of feasible deviation positions for conversion
Input : leader solution $x$, level $l$, node $q$
Output: boolean value whether or not $x$ is already contained in the archive
1  $alreadyContained = false$;
2  if $l \leq m \wedge q \neq complete$ then
3      if $q$.next$[1 - x_l] \neq complete$ then
4          $devpoints = devpoints \cup \{(l, p)\}$
5      end
6      if $q$.next$[x_l] == null$ then
7          $q$.next$[x_l]$ = new trienode($null, null$);
8      end
9      $alreadyContained$ = insert($x, l + 1, q$.next$[x_l]$);
10 end
11 if $q == complete$ then
12     $alreadyContained = true$;
13 end
14 else if $l > m$ then
15     $q = complete$;
16 end
   // Pruning
17 else if $q$.next$[x_l] = complete \wedge q$.next$[1 - x_l] = complete$ then
18     $q = complete$;
19 end
20 return $alreadyContained$;
```

whole search space $\{0,1\}^m$. The entries q.next[0] and q.next[1] of a trie node q at level l split the solution space into two subspaces with $x_l = 0$ and $x_l = 1$, respectively. In both subspaces all elements from x_1 to x_{l-1} are fixed according to the path from the root to node q. Note that such a trie is related to an explicitly stored branch-and-bound tree.

Figure 3.1 shows an example of a trie-based solution archive with three solutions $(0,0,1,0,1,1)$, $(0,0,1,1,0,0)$, and $(0,1,0,1,1,0)$. Arrows are pointers to successor nodes, a *null* flag is denoted by a slash, and a *complete* flag is shown as a C.

3.3 Insertion into the Solution Archive

Knowing the structure of the trie, inserting a solution candidate is straightforward and shown in Algorithm 3.1 in pseudocode. For inserting a solution $x = (x_1, \ldots, x_m)$ into the trie we start at the root node with the first element x_1 of the solution vector. The recursive algorithm is therefore initially called with parameters $(x, 1, root)$. On each level

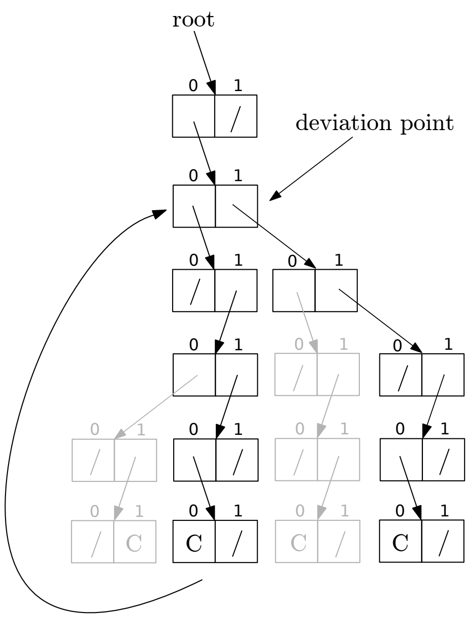


Figure 3.2: Example of a solution conversion in a binary trie-based solution archive

$i = 1, \ldots, m-1$ of the trie we follow the pointer indexed by x_i. Furthermore, we store all potential nodes for a possibly needed succeeding conversion in the set *devpoints*. At the lowest level $m-1$, a *complete* flag is stored to finally represent the solution. Intermediate nodes are always only created when needed. A pruning procedure ensures that each subtrie contains at least one not yet visited solution candidate. Whenever all entries of a trie node are *complete*, this node is deleted and the corresponding entry in the parent node is set to *complete*. This is performed iteratively until the whole trie consists of just one *complete* node and the whole solution space has been exhaustively searched or a node with a not yet *complete* entry is encountered. Pruning also ensures that the memory consumption is not unnecessarily high. The worst case run-time complexity of Algorithm 3.1 is $\mathcal{O}(m)$ because l is initially set to one, in every recursive call l is increased by one, and the algorithm terminates once $l > m$.

3.4 Conversion within the Solution Archive

Whenever a *complete* entry is encountered during the insertion of a solution candidate we know that this solution has already been inserted and is contained in the SA. This duplicate solution is then converted into a new solution candidate directly by the SA. Algorithm 3.2 shows such a conversion in pseudocode. For converting such a duplicate solution $(x_1, \ldots, x_m)$ into a usually similar but not yet discovered one, we first choose a position where we will alter the solution. All possible positions were recognized during

Algorithm 3.2: convert($x, devpoints$)

```
Input : duplicate leader solution $x$, feasible deviation positions $devpoints$
Output: converted not yet considered solution $x$
q = random entry from devpoints
$l$ = level of the trie node $q$
$x_l = 1 - x_l$;
while $q.next[x_l] \neq null$ do
    if $q.next[x_l] == complete$ then
        $x_l = 1 - x_l$;
    end
    if $q.next[x_l] == null$ then
        break;
    end
    $q = q.next[x_l]$;
    $l = l + 1$;
end
insert($x$,$l$,$q$);
return $x$;
```

the insertion procedure and stored in the set *devpoints*. From these possibilities, one deviation position is then selected uniformly at random. If there does not exist a feasible deviation position anymore (i.e., the root node is *complete*), we know that the whole search space has been covered and we can stop the whole optimization with the so far best solution being a global optimum. Otherwise, we change the element at the deviation position to another randomly chosen feasible value for which the corresponding entry is not *complete* (in the case of binary tries we flip the bit). There are two possible cases depending on the pointer at the deviation position:

- If it is a *null*-pointer, we know that the corresponding subspace has not been explored yet, which means that any feasible solution from this point on is a new one. Therefore, we can just insert the remaining solution without the need of further changes.

- If the pointer at the deviation position points towards a successive trie node, we visit this node and consider its entries. If all of them but one are *complete*, we have no choice but to follow the not complete one. Otherwise, we prefer the pointer corresponding to the original solution's variable value, i.e., at level j we follow the entry indexed by x_j, and repeat this process until we end up in a *null* entry. From there on we proceed analogously to the first case and insert the remaining elements of the unchanged solution unchanged. This procedure is guaranteed to terminate with a feasible solution because the pruning ensures that there must be at least one *null* pointer in each subtrie.

Also for the conversion method (Algorithm 3.2), the worst case run-time complexity is clearly $\mathcal{O}(m)$ because l is increased in each iteration of the while-loop, it is guaranteed that eventually the while-loop terminates, and we have already shown that the insertion method has a $\mathcal{O}(m)$ worst case run-time complexity.

Figure 3.2 shows a possible trie conversion when inserting the solution $(0,0,1,1,0,0)$ into the SA from Figure 3.1. As deviation position level 2 is chosen, where x_2 is changed from 0 to 1. Then, it is clear that we can insert the remaining elements unchanged because on the next node we follow an entry which was *null* before. This shows that with such a conversion procedure we can efficiently convert a duplicate solution into a new and usually similar solution. In this case we transformed the duplicate $(0,0,1,1,0,0)$ into the new unexplored solution $(0,1,1,1,0,0)$.

3.5 Incorporation into a Metaheuristic

The incorporation of such a solution archive into a metaheuristic is best explained in the context of a GA. In contrast to a standard GA as described in Section 2.4.7 at the end of each iteration after the mutation step the newly generated solution is transferred to the SA. In the SA, it is checked whether it is a new solution and is converted if not. This possibly converted solution is then transferred back to the GA and replaces the original solution in the population, after which the GA proceeds as usual.

A SA can also be used in combination with local search based approaches. In these cases two reasonable SA integration strategies are possible. First, after each move the created solution is transferred to the SA and the same procedure as for the GA is applied. This would, however, change the structure of the neighborhood and could possibly alter the local optimum. Another strategy is to use the archive within the local search only for consultation whether the solution has already been considered. This would not lead to any changes of the neighborhood structure except its size but could, thus, lead to a speed-up of the search process. Also, a combination of the two strategies is reasonable, especially in later iterations of the search the first method could be employed to modify the neighborhood and thereby escaping local optima.

3.6 Extensions

Frequently, practical relevant problems have a more complex solution representation and the structure of this basic binary trie must be altered. Especially when using representations in which not every possible solution is feasible, the trie structure gets more complicated. In general, for constrained problems each variable from the solution vector $(x_1,\ldots,x_m)$ can have its own domain $\mathcal{A}_l$, for $l=1,\ldots,m$, which may further depend on the already fixed values of other variables. In such a case, the corresponding trie node q at level l now has $|\mathcal{A}_l|$ entries $q.\text{next}[0],\ldots,q.\text{next}[|\mathcal{A}_l|-1]$. In the applications of SAs considered in this thesis we show how to extend the structure of a trie to a binary solution representation with constraints in Section 4.6.3 and to permutation encodings in Section 5.6.4. This further leads to problem-specific conversion methods, which have to take care to produce only feasible solutions and therefore an efficient realization might not always be possible.

Another possible extension to the basic scheme is the randomization of the insertion order of the variables. In the description of the insertion above we inserted a solution $x_1, \dots, x_m$ by starting at x_1 and using the order of the solution representation. Instead, we could decide at each level anew with which yet not inserted variable we want to continue. This reduces the bias towards the elements with higher indices of the conversion method and we will see in Section 4.6.6 how such a randomization can be applied.

Finally, a bounding extension to a SA can be applied to cut off subtrees with evidently cannot contain an optimal solution. Therefore, we need a procedure to compute lower or upper bounds on partial solutions for minimization and maximization problems, respectively. These bounds can then be stored in the corresponding trie nodes. Directly after the computation of such a bound and at each subsequent visit it is checked if this bound is larger (minimization) or smaller (maximization) than the objective value of the best solution found so far. If this is the case the corresponding subspace can be pruned. In Section 5.6.4 such a bounding extension is applied to a practical relevant COP.

CHAPTER 4

Competitive Facility Location Problems

In this chapter competitive facility location problems (CFLPs) are considered which are the first type of problems considered in this thesis. CFLPs arise in the context of two non-cooperating companies, a leader and a follower, competing for market share from a given set of customers. We assume that the firms place a given number of facilities on locations taken from a discrete set of possible points. For this bi-level optimization problem we consider six different customer behavior scenarios from the literature: binary, proportional and partially binary, each combined with essential and unessential demand. The decision making for the leader and the follower depends on these scenarios. We present mixed integer linear programming models for the follower problem of each scenario and use them in combination with an evolutionary algorithm to optimize the location selection for the leader. A complete solution archive as presented in Chapter 3 is used to detect already visited candidate solutions and convert them efficiently into similar, not yet considered ones. As the solution evaluation entails the solution of the follower's problem, which is an NP-hard optimization problem itself, it is a time-consuming procedure. Therefore, the application of a solution archive seems promising and we will see the results of the application in this chapter. Also, different solution evaluation methods are combined into a multi-level-evaluation scheme.

First, an introduction to CFLPs and a classification of several variants is given in Section 4.1. Then, in Section 4.2 we give a formal problem definition and explain all the customer behavior scenarios and both demand models in detai. After discussing the related and previous work in Section 4.3, we present mixed integer linear programming models for all variants in Section 4.4. The used solution representation and different evaluation methods are proposed in Section 4.5. The novel evolutionary algorithm (EA), which incorporates a solution archive in order to store and transform already visited solutions, as well as a local improvement component is introduced in Section 4.6. Different concepts of how the local search method can benefit from the solution archive are investigated in Section 4.7. An extension to the GA which intends to lower the effort for solution evaluation by applying a multi-level strategy is presented in Section 4.8. Finally, at the end of this chapter we discuss computational results and compare the new method to other state-of-the-art approaches from literature if available in Section 4.9. This section further includes a case study of Vienna, Austria, in which we assume the registration districts to be possible locations and the demand of the customers is proportional to the population of the respective district.

4.1 Introduction

In the considered CFLPs two decision makers, a leader and a follower, compete for market share. They choose given numbers of facility locations from a finite set of possible positions in order to satisfy client demands, whereas the leader starts to place all of his facilities, then the follower places his facilities. In this thesis different scenarios are considered which vary in the way customers satisfy their demands from the set of open facilities. This classification is taken from Suárez-Vega et al. [125]:

Customer behavior

- Binary: The demand of each customer is fulfilled by the nearest facility only.
- Proportional: Each customer splits his demand over all open facilities proportional to an attractiveness value, which depends on the distances to the facilities.
- Partially binary: This is similar to the proportional behavior but the demand is split only between the nearest leader and nearest follower facility, again, proportional to an attractiveness value depending on the distance.

Demand model

- Essential demand: The customers satisfy all of their demand.
- Unessential demand: The customers do not satisfy all of their demand but only a proportion depending on the attractiveness of the serving facility.

Combining the three customer behaviors and the two demand models results in six different scenarios. Since demand corresponds to the buying power of the customers the market share (or turnover) of the competing firms increases with the amount of fulfilled demand. Therefore, in order to obtain an accurate revenue value for the leader, the subproblem of finding an optimal set of facility locations for the follower for a given set of leader locations has to be solved. This makes the problem a Σ_2^P-hard bi-level optimization problem [100]. In this work we model the decision problem of the leader who wants to maximize his market share knowing that a follower will enter the market subsequently under a given customer behavior scenario. We propose mathematical models as well as a hybrid metaheuristic based on an evolutionary algorithm to approximately solve all variants of this problem in a practically efficient way.

Our evolutionary algorithm (EA) searches for the best possible facility locations for the leader so that his turnover is maximized. It is assumed that the follower will place his facilities optimally, i.e., aiming at maximizing his revenue or minimizing the leader's revenue. For the problem of finding the optimal locations for the follower, MIP models for different customer behaviors are presented. These models can then be solved either exactly using a general purpose MIP solver like CPLEX or approximated by solving their linear programming (LP) relaxation or by a greedy algorithm. As a result, we obtain a multi-level evaluation scheme which reduces the number of accurate, hence more time-consuming, evaluations which can be applied when the LP relaxation value of the model is good enough. The EA is further enhanced with a solution archive which is a special data structure that stores all generated candidate solutions and converts duplicate solutions into guaranteed not yet considered ones. A local search procedure, combined with the archive into a tabu search variant, further improves promising solutions of the EA and thus turns it into a powerful hybrid approach.

4.2 Problem Definition

In the following we will formally define the competitive facility location problem with different customer behavior scenarios. Given are the numbers $p \geq 1$ and $r \geq 1$ of facilities to be opened by the leader and follower, respectively, and a weighted complete bipartite graph $G = (I, J, E)$ where $I = \{1, \ldots, m\}$ represents the set of potential facility locations, $J = \{1, \ldots, n\}$ represents the set of customers, and $E = I \times J$, is the set of edges indicating corresponding assignments. Let $w_j > 0, \forall j \in J$, be the demand of each customer, which corresponds to the (maximal) turnover to be earned by the serving facilities, and $d_{ij} \geq 0, \forall (i, j) \in E$, be the distances between customers and potential facility locations. The goal for the leader is to choose exactly p locations from I for opening facilities in order to maximize his turnover under the assumption that the follower in turn chooses r locations for his facilities optimally maximizing his turnover.

The turnover distribution of the customers differ in the six scenarios defined before and in the following we will give a formal description of the turnover computation of all scenarios. In the following let (X, Y) be a candidate solution to our competitive facility location problem, where $X \subseteq I,\ |X| = p$, is the set of locations chosen by the leader and $Y \subseteq I,\ |Y| = r$, is the associated set of follower locations. Note that X and Y do not have to be disjunct in general. Further, let $D(j, V) = \min\{d_{ji} \mid i \in V\},\ \forall j \in J,\ V \subseteq I$ be the minimum distance from customer j to all facility locations in set V. Following Kochetov et al. [81] we define the attractiveness of a facility location to a customer by $v_{ij} = \frac{a_{ij}}{(d_{ij}+1)^{\beta}}$ and define analogous to the minimum distance the maximum attractiveness from customer j to all facility locations in the set V as $A(j, V) = \max\{v_{ji} \mid i \in V\},\ \forall j \in J,\ V \subseteq I$. In this work we set $\beta = 1$ and $a_{ij} = 1\ \forall i \in I,\ j \in J$. For the attractiveness one is added to the original distances d_{ij} just to avoid numerical problems with zero distances which might occur when considering the same locations for facilities and customers.

In the next sections we follow the classification of the different customer behavior scenarios [125] and give definitions of the turnover computation of each of these scenarios.

4.2.1 Binary Essential

Each customer $j \in J$ chooses the closest facility, hence the owner of this closest facility gains the complete turnover w_j. The leader is preferred in case of equal distances so the follower never places a facility at a location occupied by the leader and therefore we can assume that $X \cap Y = \emptyset$ for this scenario. The set of customers which are served by one of the follower's facilities is $U^{\mathrm{f}} = \{j \in J \mid D(j, Y) < D(j, X)\}$ and the customers served by the leader is given by $U^{\mathrm{l}} = J \setminus U^{\mathrm{f}}$. Consequently, the total turnover of the follower is $p^{\mathrm{f}} = \sum_{j \in U^{\mathrm{f}}} w_j$ and the total turnover of the leader $p^{\mathrm{l}} = \sum_{j \in J} w_j - p^{\mathrm{f}}$. Note that this problem is also known as $(r|p)$-*centroid problem* [68].

4.2.2 Proportional Essential

Each customer j splits all of his demand over all opened facilities proportional to their attractiveness. Let $x_i = 1$ if $i \in X$ and $x_i = 0$ otherwise, and $y_i = 1$ if $i \in Y$ and $y_i = 0$ otherwise, $\forall i \in I$. Then, the turnover of the follower is

$$p^{\mathrm{f}} = \sum_{j \in J} w_j \frac{\sum\limits_{i \in I} v_{ij} y_i}{\sum\limits_{i \in I} v_{ij} x_i + \sum\limits_{i \in I} v_{ij} y_i}$$

and the turnover of the leader is

$$p^{\mathrm{l}} = \sum_{j \in J} w_j - p^{\mathrm{f}}.$$

4.2.3 Partially Binary Essential

Each customer j splits all of his demand over the nearest leader and the nearest follower facility proportional to their attractiveness. Let $v_j^L = A(j, X)$, i.e., the highest attraction value from any leader facility to customer j and $v_j^F = A(j, Y)$. Then, the turnover of the follower is

$$p^{\mathrm{f}} = \sum_{j \in J} w_j \frac{v_j^F}{v_j^F + v_j^L}$$

and the turnover of the leader is

$$p^{\mathrm{l}} = \sum_{j \in J} w_j - p^{\mathrm{f}}.$$

4.2.4 Unessential Demand

In the unessential demand scenarios we need a function which describes how much the demand of a customer decreases with the distance to the nearest facility. We define this demand reduction function as $f(d) = \frac{1}{(d+1)^\gamma}$. Parameter γ controls the decrease of demand, in our work we assume $\gamma = 1$. Further, we note that when the demand is unessential $\sum_{j \in J} w_j \geq p^\mathrm{l} + p^\mathrm{f}$, i.e., the total demand satisfied by the leader and the follower is no longer necessarily equal to the total demand of all customers. In the following we present the profit computation for the unessential scenarios under the different customer choice rules:

- Binary Unessential

$$p^\mathrm{f} = \sum_{j \in U^\mathrm{f}} w_j f(D(j,Y)) \quad \text{and} \quad p^\mathrm{l} = \sum_{j \in U^\mathrm{l}} w_j f(D(j,X))$$

- Proportional Unessential

$$p^\mathrm{f} = \sum_{j \in J} w_j \frac{\sum\limits_{i \in I} v_{ij} f(d_{ij}) y_i}{\sum\limits_{i \in I} v_{ij} x_i + \sum\limits_{i \in I} v_{ij} y_i} \quad \text{and}$$

$$p^\mathrm{l} = \sum_{j \in J} w_j \frac{\sum\limits_{i \in I} v_{ij} f(d_{ij}) x_i}{\sum\limits_{i \in I} v_{ij} x_i + \sum\limits_{i \in I} v_{ij} y_i}$$

- Partially Binary Unessential

$$p^\mathrm{f} = \sum_{j \in J} w_j \frac{v_j^F}{v_j^F + v_j^L} f(D(j,Y)) \quad \text{and} \quad p^\mathrm{l} = \sum_{j \in J} w_j \frac{v_j^L}{v_j^F + v_j^L} f(D(j,X))$$

4.3 Related Work

There is a huge number of articles in the literature dealing with location problems. While we focus here on competitive variants we refer to a recently published book by [88] and two recent review articles by [50] and [7] which give an overview on recent developments in general location science.

4.3.1 Competitive Facility Location Problems – Variants

Competitive facility location problems have been introduced as early as the late 1920s by Hotelling [72]. He originally considers two salespersons competing for market share while placing one facility each on a line. From then on many researchers within the operations research community started to embed competition in their location models, especially Drezner [46] and Hakimi [68] did important early work in this field. There are several review articles handling various topics in the competitive facility location domain and we refer to [48, 83, 8, 49] and [47].

Several location models have been described in the literature and whereas in the original work by Hotelling [72] the demand of each customer in the market is satisfied by the nearest facility only, Huff [75] introduced a new notion of attraction. So, instead of being served by only one facility, each customer splits its demand over all facilities in the market based on an attraction factor. This factor determines a patronising probability of each facility. A frequent form of an attraction function on a network is, as introduced in the previous section, $v_{ij} = \frac{a_{ij}}{(d_{ij})^{\beta}}$, where i is a customer, j is a facility, a_{ij} is the attractiveness of facility j to customer i, d_{ij} is the distance, and β determines the influence of the distance to the overall attractiveness. An extension to the basic Huff model is the Pareto-Huff Model [104], in which the attractiveness of facilities with the same or worse quality than a nearer facility is set to zero.

Another choice in the area of CFLPs is whether to allow the placement only on a set of predetermined points or anywhere on the plane. The solution approaches of these continuous or discrete models are usually significantly different and most articles focus on one of these two models. In [43, 44, 45, 32, 51, 125, 13] the authors focused on continuous location models and [68, 121, 85, 115, 4, 5, 6] concentrate on the discrete variant. Here we only consider the discrete variant to model the CFLP as a combinatorial optimization problem.

In CFLPs it further can be distinguished between the case where a competitor already exists in the market and the case where he has not yet entered the market, i.e., competition with foresight. In the static competition model the competition is assumed to be known and fixed and is therefore a simpler and easier model [106]. Nevertheless, this is an important case to consider since more complex competition models often use methodologies for the static competition model as a basis. When the competitor only enters the market after the first decision maker fixed its preferred locations, we obtain the leader-follower model. This form of sequential model is a Stackelberg-type model, in which the evaluation of the choice of locations for the leader includes the solution of finding the optimal locations of the follower based on the leader's locations, which is often a non-trivial optimization problem itself. Hakimi [68] was among the first to consider the discrete variant with binary customer behavior and essential demand and called the resulting problem $(r|p)$-centroid problem.

In practical applications the actual point of the location may not be the only decision to make but also the specifics of the facility, e.g., quality or size. The optimal design of the facilities then depends on their location and influences the attractiveness to customers. Also, different opening costs for each possible location and a limited budget could be considered in the model. Such location and design problems are approached, e.g., in [84, 119] and [120].

The complexity of the discrete CFLP model with the different customer behavior scenarios which is considered in this work is shown to be Σ_2^P-hard by Noltemeier et al. [100]. Even if the locations of the leader are fixed, the remaining problem of finding the optimal locations for the follower, which is also called the $(r|X_p)$-medianoid problem, is proven to be NP-hard by Hakimi [68]. These complexity results are strengthened by Davydov et al. [39], who looked at special network structures and showed that the $(r|p)$-centroid problem remains Σ_2^P-hard for the special case of Euclidean distances.

4.3.2 Competitive Facility Location Problems – Solution approaches

As the presented CFLP models are hard to solve exactly, most solution methods for discrete CFLPs are metaheuristics. Laporte and Benati [85] developed a tabu search heuristic for the $(r|p)$-centroid problem. They use an embedded second-level tabu search for solving the $(r|X_p)$-medianoid problem. The final solution quality is thus only approximated as the $(r|X_p)$-medianoid problem is not solved exactly.

Alekseeva et al. [6] present several approaches for the discrete $(r|p)$-centroid problem including an exact procedure. The first method is a hybrid memetic algorithm (HMA) which uses a probabilistic tabu search as local improvement procedure. It employs rather simple genetic operators and the tabu search utilizes a probabilistic swap neighborhood structure, which is well known from the p-median problem; see the review article by Mladenović et al. [97] for an overview of this problem. A neighborhood of this structure contains elements only with a given probability to speed up the search. They use the linear programming relaxation of a mixed integer linear programming model for the solution evaluation which will also be used here and is described in Section 4.4.1. The authors observe that this approach outperforms several simpler heuristics including an alternating heuristic originally proposed for a continuous variant of the problem [13]. In [4] results for the tabu search alone are presented which are similar to the results of the HMA. They further describe an exact method based on a single level binary integer program with exponentially many constraints and variables. For solving this model they present an algorithm similar to a column generation approach where new sets of locations for the follower are iteratively added to the model which is then solved again. The optimal value of this model defines an upper bound and by solving the follower's problem using solutions of the model a lower bound is obtained. If the bounds coincide the optimum has been found. The HMA is applied for finding the initial family of follower solutions. Using this method the authors are able to optimally solve instances with up to 100 customers and $p = r = 5$.

Campos-Rodríguez et al. [32] studied particle swarm optimization methods for the continuous $(r|p)$-centroid problem, where the facilities can be placed anywhere on the Euclidean plane, as well as for the discrete variant [33]. A jumping particle swarm optimization is used with two swarms, one for the leader and one for the follower. The particles jump from one solution to another in dependence of its own best position so far, the best position of its neighborhood and the best position obtained by all particles so far, i.e, the best global position. In the experiments this algorithm was able to solve instances with 25 customers, $p = 3$ and $r = 2$ to optimality.

Davydov [37] describes another tabu search for the RPCP. He uses a probabilistic swap neighborhood structure similar to the one developed by [6]. For the solution evaluation the follower problem is approximately solved by Lagrangian relaxation. The method is tested on the instances from Alekseeva et al. [6] and additionally on some non-Euclidean instances. For many of the instances optimal solutions are obtained.

A recent article by Davydov et al. [38] proposes two metaheuristics which are both based on the swap neighborhood structure. The first one uses a variable neighborhood search (VNS) with a disjoint partitioning of the swap neighborhood structure into three sub-neighborhoods *Fswap*, *Nswap*, and *Cswap*. A neighbor from the *Fswap* neighborhood is determined by closing one leader facility and opening a facility at a location chosen by the follower. The *Nswap* neighborhood consists of solution candidates that are generated by closing one leader facility and opening a facility at a location in its vicinity. *Cswap* consists of all solutions that are in the swap neighborhood but not already in *Fswap* or *Nswap*. The second method, which is called STS, uses the probabilistic swap neighborhood which has been proposed by Alekseeva et al. [6]. The STS uses the same neighborhood partioning as the VNS. Additionally, a tabu list is maintained to remove elements of the neighborhood which consist of pairs for leader facilities that have been closed and opened during the last few iterations. Both methods use the same model for the solution evaluation as Alekseeva et al. [6]. The authors were able to find good solutions for many instances faster than the tabu search by Davydov [37]. Moreover, they tested their algorithms on non-euclidean instances, on which both methods, VNS and STS, showed similar performance.

Roboredo and Pessoa [115] developed an exact branch-and-cut algorithm for the discrete RPCP. They use a single-level integer programming model which is similar to the model by Alekseeva et al. [6] but with only a polynomial number of variables. It consists of exponentially many constraints, one for each follower strategy, i.e., for each set of possible facility locations of the follower. An important reason for the success of their method is the introduction of strengthening inequalities by lifting the exponentially many constraints. Due to the assumption that the customers are conservative the lower bound on the leader's solution becomes zero if the follower chooses the same facility location. Therefore, for each facility location an alternative location is given which is chosen if the position has already been used by the leader. These cuts are separated either by a greedy heuristic or by solving a mixed integer programming model. For most of the benchmark instances the authors report better results than Alekseeva et al. [6], i.e., they found optimal solutions in less time. Instances with 100 customers and up to $r = p = 15$ facilities could be solved to optimality. The authors also present promising results for $r = p = 20$ but are not able to prove their optimality within the given time limit of 10 hours.

Alekseeva and Kochetov [4] give an overview of recent research regarding the discrete $(r|p)$-centroid problem. They also improve their iterative exact method by using a model with only a polynomial number of variables and by using the strengthening inequalities introduced by Roboredo and Pessoa [115]. This improved iterative approach is able to find optimal solutions for instances with up to 100 customers and $r = p = 15$. Especially for the instances with $r = p \in \{5, 10\}$ optimal solutions are found significantly faster than by the branch-and-cut algorithm from Roboredo and Pessoa [115].

For the other customer behavior scenarios not many algorithms are described in the literature. An exception is the work by Kochetov et al. [81], who propose an algorithm for the CFLP with proportional customer behavior. The algorithm's principle is similar to the alternating heuristic for the RPCP by Bhadury et al. [13].

Serra and Revelle [121] propose a heuristic approach for a variant of the discrete RPCP which is based on repeatedly solving a maximum capture (MAXCAP) problem.

The MAXCAP problem is similar to the $(r|X_p)$-medianoid problem with the difference that it is possible to place a facility on one of the leader's locations with the result that the captured demand is equally shared between the two players. The algorithm is basically a local search using the swap neighborhood structure and candidate solutions are evaluated by solving the MAXCAP problem by means of integer programming or by using a local search heuristic for larger instances.

4.4 Mathematical Models

In this section we present mathematical models for CFLPs with different customer behavior scenarios. In case of binary choice we adopt the linear model from Alekseeva et al. [5]. Finding linear models for the partially binary and proportional case is not straightforward because we have to model a ratio of demand fulfilled by the leader and the follower, respectively. In these cases we present linear transformations which are based on the transformation performed by Kochetov et al. [81] for the proportional essential scenario. All models use two types of binary decision variables:

$$x_i = \begin{cases} 1 & \text{if the leader opens a facility at location } i \\ 0 & \text{else} \end{cases} \qquad \forall i \in I$$

and

$$y_i = \begin{cases} 1 & \text{if the follower opens a facility at location } i \\ 0 & \text{else} \end{cases} \qquad \forall i \in I.$$

4.4.1 Binary Essential

The following bi-level MIP model has been introduced in [5]. It uses an additional type of binary decision variables:

$$u_j = \begin{cases} 1 & \text{if customer } j \text{ is served by the leader} \\ 0 & \text{if customer } j \text{ is served by the follower} \end{cases} \qquad \forall j \in J.$$

We define the set of facilities that allow the follower to capture customer j if the leader uses solution x ($x = (x_i)_{i \in I}$):

$$I_j(x) = \{i \in I \mid d_{ij} < \min_{l \in I | x_l = 1} d_{lj}\} \quad \forall j \in J$$

Then we can define the upper level problem, denoted as leader's problem, as follows:

$$\max \sum_{j \in J} w_j u_j^* \tag{4.1}$$
$$\text{s.t.} \sum_{i \in I} x_i = p \tag{4.2}$$
$$x_i \in \{0,1\} \qquad \forall i \in I \tag{4.3}$$

where $(u_1^*, \dots, u_m^*)$ is an optimal solution to the lower level problem, denoted as follower's problem:

$$\max \sum_{j \in J} w_j (1 - u_j) \tag{4.4}$$
$$\text{s.t.} \sum_{i \in I} y_i = r \tag{4.5}$$
$$1 - u_j \leq \sum_{i \in I_j(x)} y_i \qquad \forall j \in J \tag{4.6}$$
$$x_i + y_i \leq 1 \qquad \forall i \in I \tag{4.7}$$
$$u_j \geq 0 \qquad \forall j \in J \tag{4.8}$$
$$y_i \in \{0,1\} \qquad \forall i \in I, \forall j \in J \tag{4.9}$$

The objective function for the leader's problem (4.1) maximizes the leader's turnover. Equation (4.2) ensures that the leader places exactly p facilities. The objective function for the follower's problem (4.4) maximizes the follower's turnover. Similarly as in the leader's problem, (4.5) ensures that the follower places exactly r facilities. Inequalities (4.6) together with the objective function ensure the u_j variables to be set correctly, i.e., decide for each customer $j \in J$ from which competitor he is served. Inequalities (4.7) guarantee that the follower does not choose a location where the leader has already opened a facility. Note that all x_i variables are considered as constants here. Variables u_j are not restricted to binary values because in an optimal solution they will become 0 or 1 anyway.

4.4.2 Proportional Essential

For the proportional essential scenario we start with a non-linear bi-level model which is derived from Kochetov et al. [81]. The upper level problem (leader's problem) is:

$$\max \sum_{j \in J} w_j \frac{\sum_{i \in I} v_{ij} x_i}{\sum_{i \in I} v_{ij} x_i + \sum_{i \in I} v_{ij} y_i^*} \tag{4.10}$$
$$\text{s.t.} \sum_{i \in I} x_i = p \tag{4.11}$$
$$x_i \in \{0,1\} \qquad \forall i \in I \tag{4.12}$$

where $(y_1^*, \dots, y_m^*)$ is an optimal solution to the lower level problem (follower's problem):

$$\max \sum_{j \in J} w_j \frac{\sum_{i \in I} v_{ij} y_i}{\sum_{i \in I} v_{ij} x_i + \sum_{i \in I} v_{ij} y_i} \tag{4.13}$$

$$\text{s.t.} \sum_{i \in I} y_i = r \tag{4.14}$$

$$y_i \in \{0, 1\} \qquad \forall i \in I \tag{4.15}$$

The objective functions (4.10) and (4.13) maximize the sums of the fulfilled demand by the leader and the follower, respectively, considering the splitting over the facilities inversely proportional to their distances. Constraint (4.11) ensures that the leader opens exactly p facilities and, similarly, constraint (4.14) guarantees that the follower places exactly r facilities. Note that the follower in principle is allowed to open facilities at the same locations as the leader. All of the x_i variables are considered as constants in the follower's problem.

In order to be able to solve the follower's problem more efficiently Kochetov et al. [81] suggested a linear transformation of this model, which works as follows. First, two new kinds of variables are introduced:

$$z_j = \frac{1}{\sum_{i \in I} v_{ij} x_i + \sum_{i \in I} v_{ij} y_i} \qquad \forall j \in J \tag{4.16}$$

and

$$y_{ij} = w_j z_j v_{ij} y_i \qquad \forall i \in I, j \in J. \tag{4.17}$$

Variables y_{ij} have the intuitive meaning that they are the demand of customer j that is supplied by the follower facility at location i and the z_j variables are basically the denominator of the fractional objective function for a fixed j. It is obvious that if we are able to model the non-linear equation (4.17) in a linear way such that equation (4.16) is valid we get a model that is equivalent to (4.13–4.15). This is realized by the following mixed integer linear program:

$$\max \sum_{j \in J} \sum_{i \in I} y_{ij} \tag{4.18}$$

$$\text{s.t.} \quad (4.14), (4.15) \text{ and}$$

$$\sum_{i \in I} y_{ij} + w_j z_j \sum_{i \in I} v_{ij} x_i \leq w_j \qquad \forall j \in J \tag{4.19}$$

$$y_{ij} \leq w_j y_i \qquad \forall i \in I, j \in J \tag{4.20}$$

$$y_{ij} \leq w_j v_{ij} z_j \leq y_{ij} + W(1 - y_i) \qquad \forall i \in I, j \in J \tag{4.21}$$

$$y_{ij} \geq 0, z_j \geq 0 \qquad \forall i \in I, j \in J \tag{4.22}$$

Objective function (4.18) maximizes the turnover obtained by the follower. Constraints (4.19) set the variables y_{ij} by restricting them to not exceed the total demand of customer

j minus the demand captured by the leader. The fact that a facility location i can only get some turnover from customer j when the follower opens a facility there is ensured by constraints (4.20). Finally, equations (4.17) are fulfilled because of constraints (4.21).

Constant W is chosen large enough, so that an optimal solution to this model satisfies equations (4.16), i.e., $W = \max_{j \in J}(w_j) \cdot \max_{i \in I, j \in J}(v_{ij}) \cdot \max_{j \in J}(z_j)$, where $\max_{j \in J}(z_j) \leq \max_{j \in J}(1/\sum_{i \in I} v_{ij}x_i)$ because of constraints (4.19). Due to constraints (4.21) with its W, the LP relaxation of this model unfortunately is in general relatively weak, therefore finding an optimal follower solution by this model using a general purpose mixed integer programming solver like CPLEX is time-consuming even for small instances. Nevertheless, this model is still easier to solve than the non-linear model (4.13–4.15).

4.4.3 Partially Binary Essential

The model for the partially binary essential scenario is similar to the model for the proportional case. The difference is that for each customer we only have to model the ratio of the nearest leader and the nearest follower facility, which results in the following non-linear bi-level model:

$$\max \sum_{j \in J} w_j \frac{v_j^L}{v_j^L + v_j^{F^*}} \tag{4.23}$$

$$\text{s.t.} \sum_{i \in I} x_i = p \tag{4.24}$$

$$v_j^L = \max_{i \in I}(v_{ij}x_i) \qquad \forall j \in J \tag{4.25}$$

$$x_i \in \{0,1\} \qquad \forall i \in I \tag{4.26}$$

where $(v_1^{F^*}, \ldots, v_m^{F^*})$ is an optimal solution to the lower level problem:

$$\max \sum_{j \in J} w_j \frac{v_j^F}{v_j^L + v_j^F} \tag{4.27}$$

$$\text{s.t.} \sum_{i \in I} y_i = r \tag{4.28}$$

$$v_j^F = \max_{i \in I}(v_{ij}y_i) \qquad \forall j \in J \tag{4.29}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{4.30}$$

The objective functions (4.23) and (4.27) maximize the sums of the fulfilled demand by the leader and the follower, respectively, considering the splitting over their nearest facilities. Constraint (4.24) ensures that the leader opens exactly p facilities and, similarly, constraint (4.28) guarantees that the follower places exactly r facilities. The highest attraction values for each customer j, expressed by variables v_j^L and v_j^F, $\forall j \in J$ are set by the non-linear constraints (4.25) and (4.29).

Again, we propose a linear transformation of the follower model similar to the proportional case. We introduce three new kinds of variables:

$$z_j = \frac{1}{v_j^L + v_j^F} \qquad \forall j \in J \tag{4.31}$$

$$\hat{y}_{ij} = \begin{cases} 1 & \text{if } i \text{ is the nearest follower facility to customer } j \\ 0 & \text{else} \end{cases}$$

and

$$y_{ij} = w_j z_j v_{ij} \hat{y}_{ij} \qquad \forall i \in I, j \in J. \tag{4.32}$$

Once more, variables y_{ij} are set to the amount of demand a (possible) follower facility at location i supplies to customer j and z_j is the denominator of the objective function Note that exactly one facility satisfies a certain amount of demand of a customer and therefore for a fixed j exactly one y_{ij} variable has a non-zero value. The linearized model is presented next.

$$\begin{aligned}
\max\ & \sum_{j \in J} \sum_{i \in I} y_{ij} && && (4.33)\\
\text{s.t. } & \sum_{i \in I} y_i = r && && (4.34)\\
& \sum_{i \in I} y_{ij} + w_j z_j v_j^L \leq w_j && \forall j \in J && (4.35)\\
& y_{ij} \leq w_j \hat{y}_{ij} && \forall i \in I, j \in J && (4.36)\\
& y_{ij} \leq w_j v_{ij} z_j \leq y_{ij} + W(1 - \hat{y}_{ij}) && \forall i \in I, j \in J && (4.37)\\
& \hat{y}_{ij} \leq y_i && \forall i \in I, j \in J && (4.38)\\
& \sum_{i \in I} \hat{y}_{ij} = 1 && \forall j \in J && (4.39)\\
& y_i \geq 0, y_{ij} \geq 0, z_j \geq 0 && \forall i \in I, j \in J && (4.40)\\
& \hat{y}_{ij} \in \{0, 1\} && \forall i \in I, j \in J && (4.41)
\end{aligned}$$

Objective function (4.33) maximizes the turnover obtained by the follower. Constraints (4.35) set the variables y_{ij} by restricting them to not exceed the total demand of customer j minus the demand captured by the leader. The fact that a facility location i can only get some turnover from customer j when there is the nearest open follower facility is ensured by constraints (4.36). Equations (4.32) are fulfilled because of constraints (4.37). Constraints (4.38) and (4.39) guarantee that there is exactly one nearest follower facility to each customer and that this location has to be chosen by the follower.

4.4.4 Unessential Cases

When the demand of customers is unessential, two different goals for the follower are possible. He can either aim to minimize the leader's profit (LMIN) or to maximize his profit (FMAX). Depending on the goal the follower might choose different locations for his facilities. In this section we will discuss the changes to the models introduced before that are needed to consider unessential demand.

4.4.5 Binary Unessential

In the LMIN scenario only a change in the objective function is needed because the distance from each customer to the nearest leader facility is known beforehand. The new objective function for the follower's problem is the following:

$$\min \sum_{j \in J} w_j z_j f(D(j, X))$$

If the follower uses the FMAX strategy new variables have to be introduced to indicate which location i hosts a follower facility that is nearer to a customer j than any other open (leader or follower) facility. This is modelled by decision variables $\hat{y}_{ij}$ which are defined similarly as before:

$$\hat{y}_{ij} = \begin{cases} 1 & \text{if } i \text{ is the nearest follower facility to customer } j \\ & \text{and nearer than all leader facilities} \\ 0 & \text{else} \end{cases}$$

The complete model for the follower problem is as follows:

$$\max \sum_{j \in J} w_j \sum_{i \in I} \hat{y}_{ij} f(d_{ij}) \tag{4.42}$$

$$\text{s.t.} \sum_{i \in I} y_i = r \tag{4.43}$$

$$1 - z_j \leq \sum_{i \in I_j(x)} y_i \qquad \forall j \in J \tag{4.44}$$

$$x_i + y_i \leq 1 \qquad \forall i \in I \tag{4.45}$$

$$\hat{y}_{ij} \leq y_i \qquad \forall i \in I, \forall j \in J \tag{4.46}$$

$$\hat{y}_{ij} \leq 1 - z_j \qquad \forall i \in I, \forall j \in J \tag{4.47}$$

$$\sum_{i \in I} \hat{y}_{ij} \leq 1 \qquad \forall j \in J \tag{4.48}$$

$$z_j \geq 0 \qquad \forall j \in J \tag{4.49}$$

$$y_i \in \{0, 1\} \qquad \forall i \in I, \forall j \in J \tag{4.50}$$

$$\hat{y}_{ij} \in \{0, 1\} \qquad \forall i \in I, \forall j \in J \tag{4.51}$$

In this model there are three new types of constraints to set the $\hat{y}_{ij}$ variables correctly. Constraints (4.46) ensure that if one of these variables is set to one then there must be a follower facility on this location. Furthermore, a $\hat{y}_{ij}$ variable is only set to 1 iff customer j is served by the follower, which is ensured by constraints (4.47). Of course, only one follower facility can be the nearest to a customer, so constraints (4.48) are introduced. The change in the objective function models the unessential demand by reducing the turnover gained by each customer by our demand reduction function f.

4.4.6 Proportional Unessential

In the proportional customer behavior scenario for both LMIN and FMAX a change in the objective function is needed and for LMIN additionally a change of constraints (4.19):

$$\begin{aligned}
\text{LMIN:}\quad & \min \sum_{j\in J} w_j z_j \sum_{i\in I} v_{ij} x_i f(d_{ij}) \\
& \sum_{i\in I} y_{ij} + w_j z_j \sum_{i\in I} v_{ij} x_i = w_j && \forall j \in J \\
\text{FMAX:}\quad & \max \sum_{j\in J} \sum_{i\in I} y_{ij} f(d_{ij})
\end{aligned}$$

4.4.7 Partially Binary Unessential

Also for the partially binary case, the objective function changes and for LMIN the constraints (4.35) as well:

$$\begin{aligned}
\text{LMIN:}\quad & \min \sum_{j\in J} w_j z_j v_j^L f(\frac{1}{v_j^L} - 1) \\
& \sum_{i\in I} y_{ij} + w_j z_j v_j^L = w_j && \forall j \in J \\
\text{FMAX:}\quad & \max \sum_{j\in J} \sum_{i\in I} y_{ij} f(d_{ij})
\end{aligned}$$

4.5 Solution Representation and Evaluation

We use the binary vector $x = (x_1, \ldots, x_m)$ as incomplete solution representation where the value of variable x_i, $1 \le i \le m$ indicate whether a facility is opened at location i. An actual solution to the problem also includes the locations for the follower and to get a better notion of feasible solutions we extend the problem definition of Section 4.2 by the following further definitions which are adopted from [4].

Definition 15. *Semi-feasible Solution*
The tuple (X, Y) is called a semi-feasible solution *to the competitive facility location problem iff $X \subseteq I$ with $|X| = p$, $Y \subseteq I$ with $|Y| = r$ and, for binary customer behavior, $X \cap Y = \emptyset$.*

Let $p^{\mathrm{l}}(X, Y)$ be the turnover of the leader and $p^{\mathrm{f}}(X, Y)$ be the turnover of the follower where X is the set of facility locations chosen by the leader and Y is the set of facility locations chosen by the follower. Then we define a feasible solution and an optimal solution as follows.

Definition 16. *Feasible Solution*
A semi-feasible solution (X, Y^*) *is called a* feasible solution *to the discrete* $(r|p)$*-centroid problem iff* $p^{\mathrm{f}}(X, Y^*) \geq p^{\mathrm{f}}(X, Y)$ *for each possible set of follower locations* Y.

Definition 17. *Optimal Solution*
A feasible solution (X^*, Y^*) *is called an* optimal solution *to the discrete* $(r|p)$*-centroid problem iff* $p^{\mathrm{l}}(X^*, Y^*) \geq p^{\mathrm{l}}(X, Y)$ *for each feasible solution* (X, Y).

It is easy to find a semi-feasible solution but already NP-hard to find a feasible solution because an optimal follower solution has to be found. This means that the solution evaluation of an arbitrary leader solution might be quite time-consuming. For practice there are several possibilities how to evaluate such a leader solution X. In our metaheuristic approach we consider the following natural ways of evaluating a leader solution x; two of them use the models introduced in Section 4.4.

4.5.1 Exact evaluation

In the exact evaluation we solve the follower's problem of the corresponding scenario (see Section 4.4) exactly using a MIP solver, e.g., CPLEX.

4.5.2 Linear Programming (LP) evaluation

In the LP evaluation we solve the LP relaxation of the follower's problem using CPLEX. This will in general yield not even semi-feasible solutions because of fractional values of some variables. For intermediate solution candidates we might, however, only be interested in an approximate objective value of a leader's solution for which purpose this method may be sufficient. This approximation yields a lower bound of the real objective value of x.

4.5.3 Greedy evaluation

To yield semi-feasible solutions and therefore an upper bound to the objective value of x greedy evaluation algorithms are used for solving the follower's problem. They perform by iteratively selecting a locally best possible position for opening a facility, until all r follower facilities are placed. A currently best possible location is determined by computing the turnover of the follower for each possible additional location depending on the specific consumer behavior using the corresponding functions defined in Section 4.2:

$$\text{Binary Essential:} \quad p^{\mathrm{f}}_{\mathrm{BE}}(y) = \sum_{j \in U^{\mathrm{f}}(y)} w_j \tag{4.52}$$

$$\text{Proportional Essential:} \quad p^{\mathrm{f}}_{\mathrm{PE}}(y) = \sum_{j \in J} w_j \frac{\sum\limits_{i \in I} v_{ij} y_i}{\sum\limits_{i \in I} v_{ij} x_i + \sum\limits_{i \in I} v_{ij} y_i} \tag{4.53}$$

$$\text{Partially Binary Essential:} \quad p^{\mathrm{f}}_{\mathrm{PBE}}(y) = \sum_{j \in J} w_j \frac{v^F_j(y)}{v^F_j(y) + v^L_j} \tag{4.54}$$

Here, $y = (y_1, \dots, y_m)$ is the partial solution vector of the follower containing all so far opened facilities plus the candidate location. A location with the highest turnover is chosen; ties are broken randomly. The final value obtained from this procedure is a lower bound to the follower's problem and therefore $\sum_{j \in J} w_j - p^{\mathrm{f}}(y)$ is an upper bound to the objective value of the leader's solution. For the binary essential case we do not have to recompute the whole function each time we place a new facility. Whenever a new facility captures facilities from the leader, they are removed from the set of customers and therefore do no longer increase the turnover of the follower. Then we only compute the turnover gain for each placed facility separately and in the end take the sum. When the demand is unessential the greedy criteria can be adapted analogously. However, the upper bound to the leader's problem has to be computed using the functions for the turnover computation for the leader defined in Section 4.2.

In Section 4.9.1 we will observe that among our evaluation algorithms for binary customer behavior the LP evaluation usually offers the best compromise in terms of speed and evaluation precision. However, by applying the different solution evaluation methods in a joined way within a multi-level evaluation scheme described in Section 4.8, we will be able to significantly improve the performance. For proportional and partially binary customer behavior even the LP evaluation is too time-consuming and therefore the greedy evaluation is used for intermediate evaluations. After the termination of the EA the final best solution is evaluated exactly for all scenarios to obtain a feasible solution.

4.6 Evolutionary Algorithm

This section describes the developed evolutionary algorithm which is used for all considered scenarios. The framework is a steady-state GA with an embedded local improvement. It uses simple genetic operators, which are explained in Section 4.6.1. The local improvement procedure is based on the swap neighborhood structure and is addressed in Section 4.6.2. Most importantly, the GA utilizes a complete solution archive for duplicate detection and conversion, which is detailed in Section 4.6.3.

As mentioned in the previous section we use the leader's incidence vector x as solution representation for the GA. The initial population is generated by choosing p locations uniformly at random to ensure high diversity in the beginning. Then, in each GA iteration one new solution is derived and always replaces the worst solution of the current population. Selecting parents for crossover is performed by binary tournament selection with replacement. Mutation is applied to offsprings with a certain probability in each iteration.

4.6.1 Variation Operators

We use the following variation operators within the GA:

Crossover Operator Suppose that we have two candidate solutions $X^1 \subset I$ and $X^2 \subset I$. An offspring X' is derived from its parents X^1 and X^2 by adopting all locations that are opened in both, i.e., all locations from $S = X^1 \cap X^2$ and then choosing the remaining $p - |X^1 \cap X^2|$ locations from $(X^1 \cup X^2) \setminus S$, i.e., the set of locations that are opened in exactly one of the parents, uniformly at random.

Mutation Operator Mutation is based on the swap neighborhood structure, which is also known from the p-median problem [97]. A swap move closes a facility and re-opens it at a different, so far unoccupied position. Our mutation applies μ random swap moves, where μ is determined anew at each GA-iteration by a random sample from a Poisson distribution with mean value one so that each position is mutated independently with probability $\frac{1}{p}$.

4.6.2 Local Search

Each new candidate solution derived in the GA via recombination and mutation whose objective value is at most $\alpha\%$ off the so far best solution value further undergoes a local improvement, with $\alpha \in \{1, 5\}$ in our experiments presented here. Local search (LS) is applied with the swap neighborhood structure already used for mutation. The best improvement step function is used, so all neighbors of a solution that are reachable via one swap move are evaluated and a best one is selected for the next iteration. This procedure terminates with a local optimal solution when no superior neighbor can be found.

4.6.3 Solution Archive

After each iteration of the genetic algorithm the newly created offspring is inserted into the archive. If this solution is already contained in the archive, the solution conversion is automatically performed and this adapted and guaranteed new solution is integrated in the population of the GA. The conversion operation can therefore also be considered as "intelligent mutation". As suggested in Chapter 3 the used data structure for the solution archive is a binary trie. Like in the example of Section 3.2, the maximum height of the solution archive is m and the domain of each variable is $\mathcal{A} = \{0, 1\}$ and therefore each trie node q has two entries q.next[0] and q.next[1], see Figure 4.1.

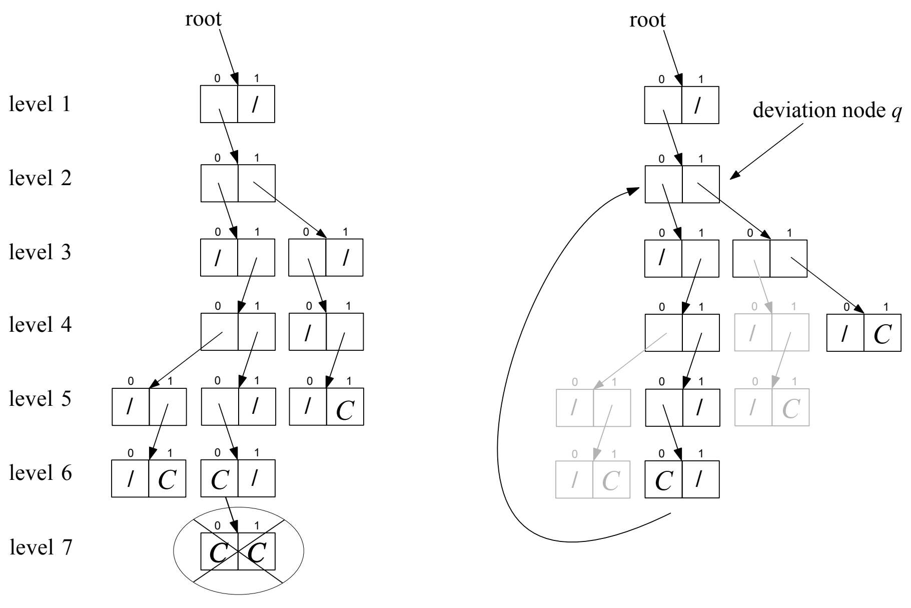


Figure 4.1: Solution archive with some inserted solutions on the lefthand side and a conversion of $(0, 0, 1, 1, 0, 0, 1)$ into the new solution $(0, 1, 1, 1, 0, 0, 0)$ on the righthand side.

4.6.4 Insertion

Algorithm 4.1 shows how to insert a new candidate solution $x = (x_1, \ldots, x_m)$ into the trie. The biggest difference to the generic method from Chapter 3 is that we can stop the insertion procedure when all chosen locations have been inserted, i.e., the values of the remaining decision variables are all zero. Still, the whole insertion procedure will be described in detail here. Initially, the recursive insertion method is called with parameters $(x, 1, root, 0)$. We start at the root node at level 1 with the first element x_1. At each level $l = 1, \ldots, m$ of the trie we follow the pointer indexed by x_l. When the p-th facility has been encountered, i.e., $openFacs = p$, at some node q the procedure stops and we set q.next[1] to *complete*. We further check at each insertion of a "one" at trie node q if enough facilities would still fit if instead a zero would be chosen. If this is not the case, q.next[0] is set to *complete* to indicate that there is no valid candidate solution in this subtrie. A set of feasible deviation positions, *devpoints*, is computed during the insertion and needed for the potentially following conversion. This set is cleared at the beginning of each solution insertion and contains all trie nodes visited during insertion where both entries are not *complete*. When we encounter a *complete*-pointer we know that this solution is already contained in the trie and it must be converted.

If we are finished with the insertion and the solution is not a duplicate, we prune the trie if possible to reduce its memory consumption. Pruning is performed as described in Chapter 3 by checking all trie nodes that have been visited during insertion bottom up if both entries of a trie node q are set to *complete*. If q.next[0] = q.next[1] = *complete*

Algorithm 4.1: insert($x, l, q, openFacs$)

```
Global Variable:
$devpoints = \emptyset$;          // Set of feasible deviation positions for conversion
Input: leader solution $x$, level $l$, node $q$,
       int $openFacs$; // Number of facilities opened until level $l$
Output: boolean value whether or not $x$ is already contained in the archive
1  $alreadyContained = false$;
2  if $l \leq m \wedge q \neq complete \wedge openFacs < p$ then
3      if $x_l == 1$ then
4          if $m - l < p - openFacs$ then
5              $q$.next[0] = $complete$;
6          end
7          $openFacs = openFacs + 1$;
8      end
9      if $q$.next[$1 - x_l$] $\neq complete$ then
10         $devpoints = devpoints \cup \{(l, p)\}$
11     end
12     if $q$.next[$x_l$] == $null$ then
13         $q$.next[$x_l$] = new trienode($null, null$);
14     end
15     $alreadyContained$ = insert($x, l + 1, q$.next[$x_l$], $openFacs$);
16 end
17 if $q == complete$ then
18     $alreadyContained = true$;
19 end
20 else if $l > m$ then
21     $q = complete$;
22 end
   // Pruning
23 else if $q$.next[$x_l$] = $complete \wedge q$.next[$1 - x_l$] = $complete$ then
24     $q = complete$;
25 end
26 return $alreadyContained$;
```

we prune this trie node by setting the corresponding entry of the preceding trie node to *complete*. On the left-hand side of Figure 4.1 an example of a trie containing the three solutions $(0, 0, 1, 1, 1, 0, 0, 1)$, $(0, 1, 0, 1, 1, 0, 0)$, and $(0, 0, 1, 0, 1, 1, 0)$ is given. The crossed out node at level 7 is a demonstration of setting a "zero" entry to *complete* because no feasible solution fits in this subtrie anymore and of the pruning that followed.

Note that no explicit look-up procedure is needed because the insertion method sketched in Algorithm 4.1 integrates the functionality to check whether or not a candidate solution is already contained.

4.6.5 Conversion

Algorithm 4.2: convert(x, *devpoints*)

```
Input : duplicate leader solution $x$, feasible deviation positions $devpoints$
Output: converted not yet considered solution $x$
q = random entry from devpoints
$l$ = level of the trie node $q$
$x_l = 1 - x_l$;
while $q.next[x_l] \neq null$ do
    if $q.next[x_l] == complete$ then
        $x_l = 1 - x_l$;
    end
    if $q.next[x_l] == null$ then
        break;
    end
    $q = q.next[x_l]$;
    $l = l + 1$;
end
$openFacs$ = number of facilities opened in $x$
$k = p - openFacs$;
if $k > 0$ then
    open $k$ facilities among $x_{l+1}, \ldots, x_m$ randomly
end
else if $k < 0$ then
    close $|k|$ facilities among $x_{l+1}, \ldots, x_m$ randomly
end
insert($x$,$l$,$q$,$openFacs$);
return $x$;
```

When the insertion procedure detects a solution which is already contained in the archive, a conversion into a new solution is performed. A pseudocode of this procedure is given in Algorithm 4.2. In contrast to the general method described in Chapter 4 we have to take care that the conversion produces only feasible solutions, i.e., solutions with exactly p open facilities. Therefore, we have to apply at least two changes: open a facility and close another one. For the first change, let *devpoints* denote the set of feasible deviation points computed during insertion. A trie node q at level l is chosen from this set uniformly at random. Should this set be empty, we know that the whole search space has been covered and we can stop the optimization process with the so far best solution being a proven optimum. Otherwise we set the l-th element of the solution vector to $1 - x_l$, which corresponds to opening or closing a facility at position l. Now we have to apply a second (inverse) change at a later position in order to have exactly p facilities opened. We go down the subtrie level by level using the following strategy. For each trie node q' at level l' we prefer to follow the original solution, i.e., the pointer q'.next[$x_{l'}$]. If it is complete, we have no choice but to use the pointer q'.next[$1 - x_{l'}$] instead (which corresponds to adding further modifications to the solution vector). As soon as we reach a *null*-pointer at a trie node q' at level l', we know that the corresponding subspace has not been explored yet, i.e., any feasible solution from this point on is a new one. Therefore, we apply the remaining necessary changes to get a feasible solution. If the number of opened facilities in x exceeds p, we close the appropriate number of facilities randomly among $\{x_{l'+1}, \ldots, x_m\}$. Otherwise, if this number is smaller than p, we open the appropriate number of facilities analogously. Finally, this new solution is inserted by applying Algorithm 4.1 starting from trie node q' at level l'.

On the righthand side of Figure 4.1 an example of a solution conversion is shown. The duplicate solution $x = (0, 0, 1, 1, 0, 0, 1)$ is inserted into the trie and subsequently converted. Node q on level 2 is chosen as the deviation point for the first change and we set $x_2 = 1$, resulting in solution $(0, 1, 1, 1, 0, 0, 1)$. Since the alternative entry at q.next[1] points to another trie node, this path is followed until a *null*-pointer is reached at level 3. Then we close the facility at the randomly chosen position 7 to get the valid solution $(0, 1, 1, 1, 0, 0, 0)$.

4.6.6 Randomization of the Trie

The above conversion procedure can only change values of solution elements with a greater index than the level of the deviation position. This induces an undesirable bias towards elements on positions with higher indices being changed more likely. In order to counter this problem, a technique called trie randomization is employed, which has first been suggested by [112]. For each search path of the trie we use a different ordering of the solution variables, i.e., a trie node on level l does not necessarily correspond to element x_l of the solution vector. Instead, the index of the element related to a trie node q is chosen randomly from the indices not already used in the path from the root to node q. In our case this is achieved by additionally storing the corresponding variable index at each trie node. Another possibility is to compute the next index by a deterministic pseudo random function taking the path from the root to node q as input. This method saves memory but needs more computational effort and is applied in [112]. Figure 4.2 shows an example of a randomized trie. Although this technique cannot avoid biasing completely, the negative effect is substantially reduced.

4.7 Local and Tabu Search with Solution Archive

There exist several options for possibly utilizing the archive not just within the GA but also the embedded LS, based on the original swap neighborhood structure. The idea of the *reduced* and the *conversion* neighborhood has already been introduced in Section 3.5 and is practically applied here.

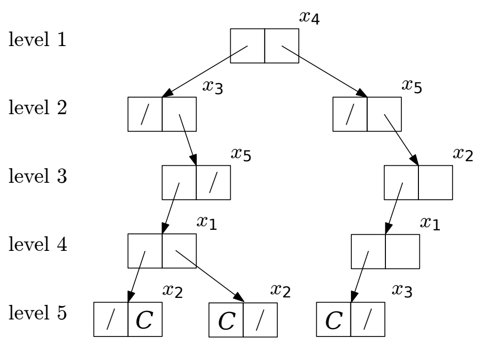


Figure 4.2: Candidate solutions (0,1,1,0,0), (1,0,1,0,0), and (0,0,0,1,1) in a randomized trie, where the variables are randomly associated with the levels.

4.7.1 Complete Neighborhood

The simplest way to perform LS is just to use the complete neighborhood as introduced in Section 4.6.2 without considering the solution archive. This method will find the best solution within the swap neighborhood but there is no benefit from the solution archive. We have to re-evaluate already visited solutions within the LS. However, all generated solutions during the LS are inserted into the solution archive so that the variation operators of the GA are still guaranteed to produce only not yet considered solution candidates.

4.7.2 Reduced Neighborhood

The second option is to skip already visited solutions in the neighborhood search. After each swap it is checked if the new solution is already contained in the solution archive. If this is the case the evaluation of this solution is skipped and the LS continues with the next swap. Otherwise this solution is inserted into the solution archive. The advantage of this method is that re-evaluations of already generated solutions are completely avoided and the neighborhoods are usually much smaller, resulting in a lower run-time. A downside is, however, that due to the reduced neighborhoods LS may terminate with worse solutions that are not local optimal with respect to the original neighborhood anymore.

4.7.3 Conversion Neighborhood

Another possibility for a combination of the local search and the solution archive is to perform a conversion whenever an already visited solution is generated by the local search. This implies that the size of this neighborhood is the same as the complete neighborhood but instead of re-evaluating duplicates, solutions that are farther away are considered to possibly find a better solution.

4.7.4 Tabu Search

The fourth method we consider uses a tabu search instead of a local search where the tabu list is realized by the solution archive. This means in particular that the search is not stopped when a neighborhood does not contain a better solution but a best neighbor solution that has not been visited, even when worse than the current solution, is always accepted and the search continues. In this way, the algorithm might escape local optima. This strategy can be combined with either of the latter two methods. Unlike the LS, since there is no predefined end of the tabu search, an explicit termination criterion is needed, e.g., a time limit or a number of iterations without improvement. As final solution, the best one encountered during the whole tabu search is returned.

4.8 Multi-level Solution Evaluation Scheme

In this section we want to exploit several relationships between the solution values of the different evaluation methods which are described in Section 4.5. Suppose that $p^{\mathrm{f}}_{\mathrm{LP}}(x)$ is the objective value of the follower's problem obtained by LP evaluation for a given leader solution x, $p^{\mathrm{f}}_{\mathrm{exact}}(x)$ is the objective value obtained by exact (MIP-based) evaluation and $p^{\mathrm{f}}_{\mathrm{greedy}}(x)$ is the objective value of the follower's problem when using the greedy evaluation. Then $p^{\mathrm{f}}_{\mathrm{LP}}(x)$ is obviously an upper bound and $p^{\mathrm{f}}_{\mathrm{greedy}}(x)$ a lower bound to $p^{\mathrm{f}}_{\mathrm{exact}}(x)$, i.e., the following relations hold:

$$p^{\mathrm{f}}_{\mathrm{greedy}}(x) \leq p^{\mathrm{f}}_{\mathrm{exact}}(x) \leq p^{\mathrm{f}}_{\mathrm{LP}}(x) \tag{4.55}$$

Since we compute the turnover of the leader by subtracting the turnover of the follower from the total demand for all customers, i.e.,

$$\begin{aligned} p^{\mathrm{l}}_{\mathrm{LP}}(x) &= \sum_{j \in J} w_j - p^{\mathrm{f}}_{\mathrm{LP}}(x), \\ p^{\mathrm{l}}_{\mathrm{exact}}(x) &= \sum_{j \in J} w_j - p^{\mathrm{f}}_{\mathrm{exact}}(x), \\ p^{\mathrm{l}}_{\mathrm{greedy}}(x) &= \sum_{j \in J} w_j - p^{\mathrm{f}}_{\mathrm{greedy}}(x), \end{aligned}$$

we obtain:

$$p^{\mathrm{l}}_{\mathrm{LP}}(x) \leq p^{\mathrm{l}}_{\mathrm{exact}}(x) \leq p^{\mathrm{l}}_{\mathrm{greedy}}(x) \tag{4.56}$$

4.8.1 Basic Multi-Level Solution Evaluation Scheme

Based on inequalities (4.56) we devise a multi-level solution evaluation scheme. Suppose that $p^{\mathrm{l}}_{\mathrm{LP}}(\hat{x})$ is the value of the leader's turnover obtained by LP evaluation of the best solution found so far $\hat{x}$. For each generated solution candidate x we evaluate it using greedy evaluation yielding a maximum achievable turnover of $p^{\mathrm{l}}_{\mathrm{greedy}}(x)$. Then we distinguish two cases:

- $p^1_{\text{greedy}}(x) \leq p^1_{\text{LP}}(\hat{x})$: This implies that $p^l_{\text{exact}}(x) \leq p^l_{\text{exact}}(\hat{x})$ and therefore x cannot be better than the so far best solution. So we do not put more effort in evaluating x more accurately.

- $p^1_{\text{greedy}}(x) > p^1_{\text{LP}}(\hat{x})$: We do not know if $p^l_{\text{exact}}(x) > p^l_{\text{exact}}(\hat{x})$ and therefore have to evaluate x more accurately. We do this by performing a more accurate (i.e., LP or exact) evaluation after the initial greedy evaluation to get a better estimate of the quality of x.

Preliminary tests showed that during an average run of our algorithm we can avoid the more accurate and thus more time-consuming solution evaluation for over 95% of the solution candidates for binary customer behavior. Therefore, it is likely that this method will reduce the overall optimization time of our algorithm in comparison to always performing an accurate evaluation. In Section 4.9.1 we will show that this multi-level solution evalution scheme is able to improve the results for binary essential customer behavior significantly in terms of running time and final solution quality.

4.8.2 Multi-Level Solution Evaluation Scheme and Local Search

For intermediate local search a modification of the multi-level evaluation scheme is needed. Suppose that $\hat{x}$ is the so far best candidate solution with an objective value of $p^1_{\text{LP}}(\hat{x})$ which is obtained by LP evaluation. Furthermore let x' be the starting solution of the local search which has an objective value of $p^1_{\text{LP}}(x') \leq p^1_{\text{LP}}(\hat{x})$ also obtained by LP evaluation. Then we encounter a problem if the objective value $p^1_{\text{greedy}}(x)$ of a neighboring candidate solution x, which initially is obtained by greedy evaluation lies between $p^1_{\text{LP}}(x')$ and $p^1_{\text{LP}}(\hat{x})$, i.e.,

$$p^1_{\text{LP}}(x') < p^1_{\text{greedy}}(x) \leq p^1_{\text{LP}}(\hat{x}).$$

Since $p^1_{\text{greedy}}(x)$ is smaller than the best LP solution value found so far, x is not evaluated more accurately. It is, however, greater than the LP solution value of the starting solution of the LS so a move toward this solution is performed. This could lead to undesirable behavior because in fact we do not know if solution x is superior to solution x' and the LS would most likely perform moves towards a solution with a good greedy value instead of a solution with a good LP or exact value.

To avoid this problem we compare the solution value obtained by the initial greedy evaluation to the best LP solution value found so far in this local search call instead of the global best LP solution value for determining whether or not the solution shall be evaluated more accurately. This implies that in each iteration of the local search we start with a candidate solution that is evaluated using LP evaluation. This results in a local search towards the candidate solution with the best LP value at the cost of additional LP evaluations.

4.9 Computational Results

In this section we present computational results of our algorithmic approach applied to different customer behavior scenarios and demand models. We consider separate sets of instances for the binary and for the proportional and partially binary case because the binary essential case has a significantly lower complexity and we want to maintain comparability to algorithms from the literature. We used instances generated in [18] for the binary case and newly created instances [17] for the proportional and partially binary case. Both instance sets are based on instances from the discrete problem library[1] and can be found online[2]. In all instances each customer location corresponds to a possible facility location, i.e., $I = J$ and the other properties are shown in Table 4.1.

Table 4.1: Properties of the benchmark instances.

	Binary	**Proportional and partially binary**
Number of locations	essential: 200, unessential: 100, chosen randomly on an Euclidean plane of size 7000×7000	100, chosen randomly on an Euclidean plane of size 100×100
Customer demands	chosen uniformly at random from the set $\{1, \ldots, 200\}$	chosen uniformly at random from the set $\{1, \ldots, 10\}$
r, p	$r = p \in \{10, 15, 20\}$	$r \in \{2, \ldots, 5\}, p \in \{2, \ldots, 10\}$

The parameter settings for the EA were determined in preliminary tests and are similar for all scenarios. The population size is 100 and the EA is terminated after 3000 iterations without improvement or after 300 seconds except for the binary case, where we have a fixed time limit of 600 seconds. The termination condition for the tabu search-based local search is set to five iterations without improvement. Local search/tabu search is called for each candidate solution whose objective value lies within 1% (for the binary case 5%) of the best solution found so far. After the EA finishes. the final best solution is evaluated exactly by solving the corresponding MIP from Section 4.4 and using the best greedy solution as starting solution with CPLEX 12.5. In preliminary tests it turned out that for the binary behavior the exact evaluation of one candidate solution needs less than one second. so in these test cases we evaluated the whole population after the last iteration exactly and took the best solution candidate among them as our final solution. All tests were performed on a single core of an Intel Xeon Quadcore with 2.54 GHz. In the next sections each customer behavior scenario with essential demand is analyzed and discussed.

At the end of the following tables for essential demands we give a quick overview over all instances on the geometric mean. the number of instances where the corresponding

[1] `http://www.math.nsc.ru/AP/benchmarks/english.html`

[2] `https://www.ac.tuwien.ac.at/research/problem-instances/#Competitive_Facility_Location_Problems`

configuration performed best and the number of instances where the algorithm performed best and better than all others.

4.9.1 Binary Essential

During the computational study for the binary essential scenario we tested most components of the EA individually to study their impact on the solution quality and / or run-time. To keep clarity in the following tables for some tests only a representative selection of the whole instances set is chosen and the full result tables are given in Appendix A.1. In most of the following sections there is a second table after the main results table. These tables display the results of pairwise Wilcoxon rank sum tests of the different configurations with error levels of 5%. The value in the cell at line i and row j gives the number of instances for which configuration i yields significantly better results than configuration j. The rightmost column lists the sums over all numbers in the corresponding rows. For all main result tables the following holds: Instances *Code111* to *Code1011* are the instances with $n = 100$ by Alekseeva et al. [5] and instances *Code123* to *Code1023* are the uniform instances also with $n = 100$ by Davydov et al. [38]. The other instance names contain either 150 or 200 which stands for the number of customers. The number right after *rp* corresponds to the number of facilities to place for both the leader and the follower. In the first row the name of the algorithm is listed. The second row describes the columns. where $\overline{obj}$ stands for the average of the final leader objective value over 30 runs. *sd* is the corresponding standard deviation and t^* is the median time needed for finding the best solution in seconds. All runs are terminated after 600 seconds to ensure comparability. We start by analyzing the impact of the solution evaluation method on the final objective value and run-time. Then. the GA with and without the local search and the solution archive is tested and the results are compared. This is followed an investigation of the influence of the different variants of the neighborhood structures in combination with the SA and the multi-level evaluation scheme on the final results. This section ends with a comparison of the developed EA to algorithms from the literature on Euclidean instances and instances with uniformly distributed distances. The best configuration found during these computational tests was also used for the remaining customer behavior scenarios and demand models.

Solution Evaluation on Euclidean Instances

In the following tests we compare three types of solution evaluation schemes according to Section 4.5: greedy evaluation. LP evaluation and exact evaluation. The aim of these tests is to find out which run-time / solution accuracy tradeoff is suitable for this problem.

Table 4.2 and 4.3 show the results. As we can see, although each greedy evaluation is 4 to 5 times faster than the LP evaluation, the results for the greedy evaluation are rather poor because the solution with the highest greedy value often does not correspond to an optimal solution according to the exact evaluation. In contrast, the results for evaluating solutions using the LP evaluation are similiar to those obtained by using the

Table 4.2: Results of different solution evaluation methods using the standard configuration.

	greedy			LP			exact		
Instance	$\overline{obj}$	*sd*	$t^*[s]$	$\overline{obj}$	*sd*	$t^*[s]$	$\overline{obj}$	*sd*	$t^*[s]$
Code111w_rp10	4359.00	0.00	14.80	**4361.00**	0.00	130.30	**4361.00**	0.00	70.60
Code111w_rp15	4547.11	6.01	20.10	**4596.00**	0.00	64.10	**4596.00**	0.00	55.60
Code111w_rp20	**4508.50**	6.09	253.30	4505.47	11.22	343.30	4502.90	11.26	217.70
Code1_150w_rp10	7132.20	130.36	250.20	7138.37	112.88	88.60	**7167.43**	51.47	94.00
Code1_150w_rp15	7008.63	54.17	138.40	7077.97	35.79	341.20	**7088.83**	43.99	398.50
Code1_150w_rp20	7070.67	52.46	314.20	7198.27	19.01	380.20	**7198.53**	22.50	370.60
Code1_200w_rp10	9349.60	69.78	406.60	9476.17	107.30	200.60	**9478.50**	92.39	369.70
Code1_200w_rp15	9814.13	185.24	351.30	**10001.40**	92.78	394.40	10000.30	82.92	475.60
Code1_200w_rp20	9615.13	135.94	411.90	**9753.07**	77.54	572.80	9697.53	85.19	586.90
Code211w_rp10	5309.47	2.92	26.50	**5310.00**	0.00	43.50	**5310.00**	0.00	46.10
Code211w_rp15	**5373.00**	0.00	97.10	**5373.00**	0.00	111.80	**5373.00**	0.00	95.70
Code211w_rp20	**5431.57**	2.37	284.50	5404.43	29.69	291.60	5405.63	31.19	365.20
Code2_150w_rp10	7181.53	52.91	332.10	7247.47	53.43	292.50	**7253.30**	71.29	291.00
Code2_150w_rp15	7590.23	92.37	154.60	**7743.20**	4.70	281.40	7742.00	5.57	358.40
Code2_150w_rp20	7673.90	83.24	255.00	**7772.13**	40.91	349.50	7755.50	46.49	347.80
Code2_200w_rp10	9032.00	71.74	221.20	9231.63	75.55	249.80	**9254.53**	62.00	448.00
Code2_200w_rp15	9274.23	153.66	312.40	**9539.27**	70.94	516.40	9505.43	109.72	438.40
Code2_200w_rp20	9381.90	138.87	475.30	**9579.83**	118.18	508.00	9570.30	110.98	548.80
Code311w_rp10	4392.47	42.88	17.80	**4483.00**	0.00	25.80	**4483.00**	0.00	24.50
Code311w_rp15	4782.10	18.23	221.60	**4800.00**	0.00	73.60	**4800.00**	0.00	63.20
Code311w_rp20	4853.47	8.76	100.50	**4892.80**	0.61	297.90	4892.67	0.76	250.80
Code3_150w_rp10	7240.20	75.69	362.80	7286.93	16.36	310.70	**7291.87**	13.30	369.20
Code3_150w_rp15	7499.30	44.12	161.60	7589.00	18.83	285.80	**7589.27**	18.01	200.50
Code3_150w_rp20	7520.40	63.06	303.20	**7624.43**	34.03	309.10	7624.37	34.20	411.20
Code3_200w_rp10	9224.03	52.98	202.70	**9300.23**	70.30	291.70	9287.13	74.85	362.90
Code3_200w_rp15	9145.17	210.97	378.30	9304.57	71.44	386.40	**9308.37**	70.27	459.10
Code3_200w_rp20	8902.30	210.90	468.70	**9197.97**	155.51	516.80	9145.73	107.33	574.10
geometric mean	6907.43			**6995.12**			6993.29		
#best results	11			**53**			48		
#unique best res.	6			**35**			31		

Table 4.3: Results of Wilcoxon Rank Sum tests with error levels of 5% for the different solution evaluation methods.

	greedy	LP	exact	Σ
greedy	–	5	5	10
LP	75	–	6	**81**
exact	73	4	–	77

exact evaluation. In many cases the root LP relaxation of the follower's problem is already integral and no branching has to be performed, hence the similar results. Therefore, for the remaining tests we primarily use the LP evaluation method.

Table 4.4: Results of different configurations for the GA: the pure genetic algorithm (GA), GA with local search (GA + LS), GA with solution archive (GA + solA) and GA with solution archive and local search (GA + LS + solA).

	GA			GA + LS			GA + solA			GA + LS + solA		
Instance	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code111w_rp10	4331.80	12.32	338.20	4348.97	25.00	113.00	4334.20	10.75	402.60	**4361.00**	0.00	14.70
Code111w_rp15	4572.37	21.18	461.10	4586.43	16.65	38.50	4582.67	6.63	412.80	**4596.00**	0.00	16.10
Code111w_rp20	4452.23	17.50	538.40	4474.97	23.52	162.50	4464.17	21.42	521.40	**4505.47**	11.22	209.50
Code1_150w_rp10	6503.63	123.24	538.20	**7163.57**	54.75	113.40	6606.03	120.76	530.10	7138.37	112.88	29.20
Code1_150w_rp15	6823.13	63.36	549.60	7021.47	75.64	149.30	6876.50	49.83	546.80	**7077.97**	35.79	133.00
Code1_150w_rp20	6900.63	109.91	550.20	7163.03	58.69	187.30	6980.63	105.85	560.30	**7198.27**	19.01	241.40
Code1_200w_rp10	8810.10	201.26	531.40	9443.60	105.48	246.70	8926.30	189.46	509.10	**9476.17**	107.30	243.10
Code1_200w_rp15	9079.43	265.11	572.00	9956.77	112.63	349.90	9212.53	227.11	553.60	**10001.40**	92.78	297.10
Code1_200w_rp20	8899.63	157.41	562.00	9683.20	119.85	479.60	8996.93	219.23	560.80	**9753.07**	77.54	460.50
Code211w_rp10	5289.47	25.60	521.20	**5310.00**	0.00	51.20	5291.13	28.09	473.40	**5310.00**	0.00	8.10
Code211w_rp15	5279.47	35.12	546.90	5362.33	19.73	42.40	5294.43	31.31	493.60	**5373.00**	0.00	23.10
Code211w_rp20	5250.03	52.47	556.90	5351.80	55.16	135.10	5268.93	47.01	546.80	**5404.43**	29.69	82.40
Code2_150w_rp10	6962.67	46.48	519.90	7229.90	89.61	233.40	6969.40	43.43	512.40	**7247.47**	53.43	242.70
Code2_150w_rp15	7423.00	94.91	537.40	7702.23	103.32	148.00	7496.67	66.09	516.30	**7743.20**	4.70	96.90
Code2_150w_rp20	7439.37	65.12	540.10	7713.43	70.96	208.40	7500.33	60.97	551.80	**7772.13**	40.91	211.50
Code2_200w_rp10	8609.17	151.04	524.40	9181.07	127.89	228.80	8717.20	131.21	529.70	**9231.63**	75.55	130.10
Code2_200w_rp15	8639.43	159.89	523.30	9482.17	119.41	342.10	8678.77	144.79	520.70	**9539.27**	70.94	392.00
Code2_200w_rp20	8626.47	119.66	557.10	9508.30	119.17	521.50	8726.30	112.43	547.80	**9579.83**	118.18	421.30
Code311w_rp10	4472.50	34.65	331.60	**4483.00**	0.00	21.20	**4483.00**	0.00	336.20	**4483.00**	0.00	8.10
Code311w_rp15	4775.93	19.79	473.20	4785.40	22.85	70.40	4781.13	21.89	534.60	**4800.00**	0.00	13.30
Code311w_rp20	4835.77	15.62	534.70	4879.17	14.61	116.60	4849.67	11.70	495.30	**4892.80**	0.61	65.20
Code3_150w_rp10	6975.17	75.29	522.30	7252.50	69.75	136.20	7004.47	71.96	493.00	**7286.93**	16.36	35.40
Code3_150w_rp15	7333.73	124.31	551.10	7554.00	47.95	131.20	7391.33	95.72	523.60	**7589.00**	18.83	142.50
Code3_150w_rp20	7358.33	70.60	543.60	7601.30	45.29	214.80	7406.17	50.70	559.40	**7624.43**	34.03	274.00
Code3_200w_rp10	8832.40	181.41	553.50	9229.40	86.17	239.00	8856.23	185.09	535.80	**9300.23**	70.30	227.00
Code3_200w_rp15	8232.10	269.27	563.40	9265.43	98.36	333.10	8497.40	203.87	562.00	**9304.57**	71.44	281.90
Code3_200w_rp20	8091.83	188.37	564.80	9150.37	174.73	509.90	8251.03	134.42	550.60	**9197.97**	155.51	426.90
geometric mean		6643.72			6964.10			6691.70			**6995.12**	
#best results		1			10			3			**85**	
#unique best res.		0			5			0			**80**	

Genetic Algorithm on Euclidean Instances

Now, we analyze different configurations of the GA. The GA was tested with and without the local search and with and without the solution archive. The aim was to see the impact of the different techniques on the solution quality and speed. Table 4.4 and 4.5 show the computational results. We can make several interesting observations: As expected, the GA alone performs not very well, neither regarding solution quality nor convergence speed, but its performance is substantially improved by executing intermediate local searches. By adding the solution archive (solA) to the pure GA we were also able to significantly improve the results. The benefit of the local search seems to be greater than the benefit of the solution archive because the relative difference of the geometric mean of GA + LS and the GA is about 5% while the difference of GA + SA and GA is only about 0.7%. Adding both, LS and solA, to the GA clearly further improves the performance. For this combined approval not only the solution quality is the best among the configurations but these solutions in most of the cases are also found faster.

Table 4.5: Results of Wilcoxon Rank Sum tests with error levels of 5% for the different configurations of the GA.

	GA	GA+LS	GA+solA	GA+LS+solA	Σ
GA	–	0	0	0	0
GA + LS	85	–	84	0	169
GA + SA	56	0	–	0	56
GA + LS + SA	87	60	87	–	**234**

Neighborhoods of the Local Search and Tabu Search on Euclidean Instances

Table 4.6 and 4.7 show the results of using the different strategies for utilizing the solA within LS and tabu search (TS), respectively (c.f. Section 4.7). As expected the complete neighborhood strategy performed worst because of the overhead of re-evaluating already visited solutions but on some of the smaller test instances it is able to produce equally good results. Among all tested LS neighborhoods, reduced neighborhood yields the best results, so it is chosen for all further tests. While on the smaller test instances with 100 customers the conversion and the complete neighborhood can keep up with the reduced neighborhood in terms of mean objective value, on larger instances the performance gap increases. The differences in the objective value of the conversion neighborhood and the reduced neighborhood are small and the conversion neighborhood even finds the best solution in less time for some instances, e.g., *Code111w_rp10* and *Code211w_rp10*. However, this difference vanishes when considering larger instances, where the reduced neighborhood consistently finds better solutions. Apparently, for these instances conversion moves were too rarely able to improve the starting solution. The largest improvement of the overall results could be achieved by using a tabu search with the reduced neighborhood. In none of our benchmark instances any other configuration was able to find solutions with a statistical significant better mean objective value.

Table 4.6: Results of using different neighborhood (NB) structures for intermediate local / tabu search (TS).

	complete NB			reduced NB			conversion NB			TS with reduced NB		
Instance	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code111w_rp10	**4361.00**	0.00	133.80	**4361.00**	0.00	130.30	**4361.00**	0.00	72.20	**4361.00**	0.00	47.00
Code111w_rp15	**4596.00**	0.00	44.90	**4596.00**	0.00	64.10	**4596.00**	0.00	79.40	**4596.00**	0.00	55.10
Code111w_rp20	4488.20	22.42	299.50	4505.47	11.22	343.30	4497.93	14.69	343.60	**4506.23**	8.39	268.70
Code1_150w_rp10	7157.07	89.96	113.60	7138.37	112.88	88.60	7171.30	47.65	137.40	**7180.00**	0.00	118.00
Code1_150w_rp15	7055.70	39.71	240.40	7077.97	35.79	341.20	7070.77	45.79	339.40	**7143.30**	31.63	337.00
Code1_150w_rp20	7187.87	22.59	289.30	7198.27	19.01	380.20	7190.87	24.84	400.70	**7221.50**	27.75	455.20
Code1_200w_rp10	9471.80	85.42	286.70	9476.17	107.30	200.60	9508.00	67.29	400.10	**9532.50**	56.10	350.30
Code1_200w_rp15	9959.17	133.66	483.60	**10001.40**	92.78	394.40	9988.87	100.51	461.90	9986.13	99.43	500.60
Code1_200w_rp20	9706.37	70.27	530.90	9753.07	77.54	572.80	9720.67	74.29	521.80	**9760.10**	69.67	600.00
Code211w_rp10	**5310.00**	0.00	85.50	**5310.00**	0.00	43.50	**5310.00**	0.00	29.00	**5310.00**	0.00	33.10
Code211w_rp15	5367.60	8.39	62.40	**5373.00**	0.00	111.80	**5373.00**	0.00	100.40	**5373.00**	0.00	162.80
Code211w_rp20	5386.77	37.42	291.80	5404.43	29.69	291.60	5404.13	30.62	439.50	**5427.37**	9.90	200.00
Code2_150w_rp10	7234.77	55.54	303.70	7247.47	53.43	292.50	7265.53	58.72	216.00	**7290.17**	48.81	401.10
Code2_150w_rp15	7738.40	13.75	306.10	**7743.20**	4.70	281.40	7742.30	5.53	367.50	7741.60	6.29	342.30
Code2_150w_rp20	7737.80	51.87	360.50	7772.13	40.91	349.50	7771.40	39.29	390.40	**7774.13**	59.61	317.60
Code2_200w_rp10	9214.43	102.90	266.60	9231.63	75.55	249.80	9230.23	94.53	395.30	**9252.07**	89.10	376.50
Code2_200w_rp15	9525.37	92.86	419.00	9539.27	70.94	516.40	9518.10	93.23	439.20	**9561.30**	83.08	591.40
Code2_200w_rp20	9542.93	106.14	549.80	9579.83	118.18	508.00	9549.43	101.89	549.80	**9619.70**	67.33	600.00
Code311w_rp10	**4483.00**	0.00	25.40	**4483.00**	0.00	25.80	**4483.00**	0.00	23.80	**4483.00**	0.00	31.10
Code311w_rp15	**4800.00**	0.00	85.90	**4800.00**	0.00	73.60	**4800.00**	0.00	50.80	**4800.00**	0.00	49.20
Code311w_rp20	4890.63	3.95	183.30	**4892.80**	0.61	297.90	4892.67	0.76	264.90	4892.73	0.69	213.90
Code3_150w_rp10	7285.67	15.76	113.40	7286.93	16.36	310.70	7293.10	12.10	314.30	**7299.00**	0.00	130.90
Code3_150w_rp15	7567.03	34.68	193.00	7589.00	18.83	285.80	**7589.90**	18.86	286.60	7583.77	20.38	192.40
Code3_150w_rp20	7605.33	47.90	298.50	7624.43	34.03	309.10	**7627.93**	26.49	331.80	7620.67	46.21	429.70
Code3_200w_rp10	9265.97	102.41	247.40	9300.23	70.30	291.70	9275.87	85.79	320.20	**9339.97**	76.46	375.40
Code3_200w_rp15	9279.60	91.05	383.20	9304.57	71.44	386.40	9301.60	68.80	438.00	**9317.83**	66.40	496.10
Code3_200w_rp20	9149.43	118.80	518.40	9197.97	155.51	516.80	9147.50	115.48	517.80	**9220.97**	105.33	600.00
geometric mean	6983.65			6995.12			6992.38			**7006.53**		
#best results	13			27			22			**74**		
#unique best results	0			9			7			**55**		

Table 4.7: Results of Wilcoxon Rank Sum tests with error levels of 5% for the different local search neighborhood structures and tabu search.

	complete NB	reduced NB	conversion NB	TS with reduced NB	Σ
complete NB	–	1	0	0	1
reduced NB	31	–	5	0	36
conversion NB	23	3	–	0	26
TS with reduced NB	54	21	32	–	**107**

Multi-Level Evaluation Scheme on Euclidean Instances

The computational results for testing the multi-level evaluation scheme (ML-ES) confirms the hypothesis that it is able to speed up the algorithm significantly. We further tested if the local search using the local best LP solution (improved LS) as described in Section 4.8.2 actually improves the solution quality. Finally we investigated the tabu search approach (improved TS), which is explained in Section 4.7.4 in combination with the reduced NB. For the TS we also used the adaptation for the improved LS in a straightforward way and set a termination criterion of five iterations without improvement.

Table 4.8 and 4.9 show the results of these tests. We observe that the multi-level evaluation scheme is able to improve the solution quality for some instances, especially the larger ones with 200 customers. The largest improvement could be made in the time needed for finding the best solution. It is in general much lower than when using only the simple LP evaluation, e.g., for instance *Code111w_rp10* the time could be decreased by about 90%. With the improved local search the mean solution quality gets better in 65 of the (mostly larger) instances, while it is equal on most of the other ones. Our best setup turned out to be GA + solA + ML-ES + improved TS, when we switched from a local search to a tabu search. We have a low standard deviation of the results and achieved a better mean objective value than the local search in 47 instances. The improvements are again mostly on the larger instances with 150 and 200 customers because, as we see in Section 4.9.1, we could find optimal solutions for many of the instances with 100 customers. In total, our best configurations of the multi-level evaluation scheme was able to produce statistically better results in 63 out of 90 instances.

Comparison to Results from the Literature on Euclidean Instances

In this section we compare the results of our best configuration to the state-of-the-art in the literature. Since the metaheuristic approaches of Alekseeva et al. [6], Alekseeva and Kochetov [4], and Davydov et al. [38] are the best heuristic approaches so far we compare with them. For this purpose both the probabilistic tabu search (TS_{Al}) [6] and the hybrid memetic algorithm (HMA) [4] were re-implemented in C++. Our re-implementations were verified to exhibit nearly equal performance as published in the respective original papers. The results of the STS are taken from [38], where they presented the objective values of single runs (obj_1). Although they developed two metaheuristics here we only compare with STS because the other one, the VNS, performed worse on Euclidean

Table 4.8: Results for the multi-level evaluation scheme and the local/tabu search improvement compared to the standard LP solution evaluation.

	GA + solA + LP + LS			GA + solA + ML-ES + LS			GA + solA + ML-ES + improved LS			GA + solA + ML-ES + improved TS		
Instance	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code111w_rp10	**4361.00**	0.00	130.30	**4361.00**	0.00	13.00	**4361.00**	0.00	12.70	**4361.00**	0.00	14.70
Code111w_rp15	**4596.00**	0.00	64.10	**4596.00**	0.00	22.20	**4596.00**	0.00	12.20	**4596.00**	0.00	16.10
Code111w_rp20	4505.47	11.22	343.30	4507.77	4.15	181.80	4511.47	1.38	231.30	**4511.87**	0.73	209.50
Code1_150w_rp10	7138.37	112.88	88.60	**7180.00**	0.00	23.50	**7180.00**	0.00	33.70	**7180.00**	0.00	29.20
Code1_150w_rp15	7077.97	35.79	341.20	7080.23	45.92	110.30	7130.97	36.67	291.60	**7153.93**	0.25	133.00
Code1_150w_rp20	7198.27	19.01	380.20	7198.70	23.39	264.90	7208.13	18.97	282.50	**7247.27**	7.97	241.40
Code1_200w_rp10	9476.17	107.30	200.60	9515.23	54.65	201.80	9558.83	37.26	298.50	**9594.00**	10.37	243.10
Code1_200w_rp15	10001.40	92.78	394.40	10026.23	67.17	134.20	10077.10	42.96	289.20	**10095.00**	37.02	297.10
Code1_200w_rp20	9753.07	77.54	572.80	9742.10	93.20	225.20	9807.57	74.67	460.80	**9831.97**	56.35	460.50
Code211w_rp10	**5310.00**	0.00	43.50	**5310.00**	0.00	8.60	**5310.00**	0.00	8.30	**5310.00**	0.00	8.10
Code211w_rp15	**5373.00**	0.00	111.80	**5373.00**	0.00	37.00	**5373.00**	0.00	17.80	**5373.00**	0.00	23.10
Code211w_rp20	5404.43	29.69	291.60	5427.80	10.53	252.60	**5432.00**	0.00	165.70	5431.57	2.37	82.40
Code2_150w_rp10	7247.47	53.43	292.50	7276.70	61.74	98.60	7328.97	23.91	166.70	**7337.00**	0.00	242.70
Code2_150w_rp15	7743.20	4.70	281.40	7735.93	9.15	60.30	7744.00	3.81	102.90	**7745.00**	0.00	96.90
Code2_150w_rp20	7772.13	40.91	349.50	7759.60	40.59	181.30	7789.27	28.71	177.00	**7802.03**	15.79	211.50
Code2_200w_rp10	9231.63	75.55	249.80	9238.23	78.81	58.30	9307.13	53.11	236.50	**9321.13**	26.28	130.10
Code2_200w_rp15	9539.27	70.94	516.40	9471.07	78.21	119.40	9593.53	53.40	345.90	**9626.67**	17.34	392.00
Code2_200w_rp20	9579.83	118.18	508.00	9599.40	88.02	241.50	9643.80	80.47	402.70	**9666.37**	52.72	421.30
Code311w_rp10	**4483.00**	0.00	25.80	**4483.00**	0.00	5.90	**4483.00**	0.00	7.90	**4483.00**	0.00	8.10
Code311w_rp15	**4800.00**	0.00	73.60	**4800.00**	0.00	13.50	**4800.00**	0.00	19.80	**4800.00**	0.00	13.30
Code311w_rp20	4892.80	0.61	297.90	4889.33	6.19	100.10	**4893.00**	0.00	103.30	**4893.00**	0.00	65.20
Code3_150w_rp10	7286.93	16.36	310.70	7292.50	13.38	62.90	7296.83	8.33	64.10	**7299.00**	0.00	35.40
Code3_150w_rp15	7589.00	18.83	285.80	7580.97	23.60	45.60	7597.77	13.80	207.80	**7603.10**	2.75	142.50
Code3_150w_rp20	7624.43	34.03	309.10	7605.77	60.47	111.10	7636.47	15.28	289.20	**7646.87**	4.32	274.00
Code3_200w_rp10	9300.23	70.30	291.70	9320.53	62.72	164.60	9358.97	48.66	283.30	**9374.30**	28.15	227.00
Code3_200w_rp15	9304.57	71.44	386.40	9310.33	71.22	192.00	9353.33	39.29	388.40	**9365.97**	17.19	281.90
Code3_200w_rp20	9197.97	155.51	516.80	9265.93	107.10	252.20	9285.73	81.47	416.80	**9296.67**	70.96	426.90
geometric mean		6995.12			7000.54			7019.37			**7027.16**	
#best results		19			24			43			**85**	
#unique best results		0			0			5			**47**	

Table 4.9: Results of Wilcoxon Rank Sum tests with error levels of 5% for the multi-level evaluation scheme configurations.

	GA+solA+ LP+LS	GA+solA+ ML-ES+LS	GA + solA + ML-ES + imp. LS	GA + solA + ML-ES + imp. TS	Σ
GA + solA + LP + LS	–	10	0	0	10
GA + solA + ML-ES + LS	17	–	1	2	18
GA + solA + ML-ES + imp. LS	60	55	–	1	115
GA + solA + ML-ES + imp. TS	63	59	29	–	**151**

instances.

Table 4.10 shows the results of their approaches compared to our algorithm (GA + solA + ML-ES + imp. TS) with $n = 100$. Additionally, Tables 4.11 and 4.12 show the results of it compared to TS_{Al} and HMA with $n = 150$ and $n = 200$. It can be seen that especially for larger instances our algorithm achieves the best results among all three tested algorithms. For the instances with 100 customers the algorithm described in this work gets better or equally good results than TS_{Al} and HMA in all but one instances, although the differences in the mean objective value is rather small. Compared to STS both algorithms get better objective values on 3 instances and equally good solutions on the remaining 24 instances. However, the time needed to find these solutions is much lower for most instances when using our algorithm. The differences in the objective value become larger when considering larger instances. On all instances with $n = 200$ the presented EA obtains better results than HMA and on 24 out of 30 instances it also gets better mean objective values than TS_{Al}.

We observe that because of the time-consuming local searches in the creation of the initial population the HMA was not able to finish the initialization within the timelimit for some instances, so we made further tests with an increased timelimit of 1800 seconds. The results of these tests can be found in Table 4.14 for $n = 100$ and Table 4.15 for $n = 150$ and $n = 200$. In Table 4.14 we also show the results of the modified iterative exact method (MEM) by Alekseeva and Kochetov [4] and the results of the branch-and-cut by Roboredo and Pesso [115]. For more than 100 customers no results of exact methods are published in the literature. From Table 4.14 we conclude that our algorithm is able to find optimal solutions to all but one instance, with $n = 100$ but in much less time than the exact algorithms. Table 4.15 shows that the described approach is still superior and exceeds the HMA in most instances. The HMA can compete with the GA on some of the instances with $n = 150$ and even gets better mean objective values for 3 instances, e.g., *Code5_150w_rp15*. However, the differences are rather small and for $n = 200$ the GA is better in 28 out of 30 instances with a much lower standard deviation on most instances.

It is interesting that although our algorithm did not find the optimal solution for instance *Code511* with $r = p = 15$ it always terminated with the same suboptimal solution. This is due to its solution evaluation method because even though the optimal

Table 4.10: Comparison to results from the literature with a runtime of 600 seconds and $n = 100$. The Tabu Search (TS_{Al}), the Hybrid Memetic Algorithm (HMA), and the STS approach by Alekseeva et al. [6], Alekseeva and Kochetov [4], and Davydov et al. [38], respectively, compared to our best configuration GA + solA + ML-ES + improved TS.

	TS_{Al}			HMA			STS		GA + solA + ML-ES+imp.TS		
Instance	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	obj_1	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code111w_rp10	**4361.00**	0.00	118.0	**4361.00**	0.00	92.3	**4361.00**	63.7	**4361.00**	0.00	14.7
Code111w_rp15	**4596.00**	0.00	38.6	**4596.00**	0.00	106.8	**4596.00**	173.3	**4596.00**	0.00	16.1
Code111w_rp20	4506.87	6.96	92.1	4510.60	2.03	393.6	4484.00	118.1	**4511.87**	0.73	209.5
Code211w_rp10	**5310.00**	0.00	11.9	**5310.00**	0.00	26.6	**5310.00**	23.5	**5310.00**	0.00	8.1
Code211w_rp15	**5373.00**	0.00	115.5	**5373.00**	0.00	121.9	**5373.00**	88.9	**5373.00**	0.00	23.1
Code211w_rp20	5428.13	6.01	167.1	5430.67	3.40	287.2	**5432.00**	289.2	5431.57	2.37	82.4
Code311w_rp10	**4483.00**	0.00	11.6	**4483.00**	0.00	45.1	**4483.00**	33.8	**4483.00**	0.00	8.1
Code311w_rp15	**4800.00**	0.00	71.3	4799.77	1.28	122.5	**4800.00**	91.1	**4800.00**	0.00	13.3
Code311w_rp20	4892.73	0.69	94.7	4892.60	0.81	297.8	**4893.00**	211.3	**4893.00**	0.00	65.2
Code411w_rp10	**4994.00**	0.00	11.7	**4994.00**	0.00	24.0	**4994.00**	19.3	**4994.00**	0.00	7.9
Code411w_rp15	5063.20	2.07	139.0	5063.80	1.10	201.0	**5064.00**	121.3	**5064.00**	0.00	48.8
Code411w_rp20	**5209.00**	0.00	105.6	5208.93	0.25	275.5	**5209.00**	288.9	**5209.00**	0.00	39.6
Code511w_rp10	**4906.00**	0.00	38.8	**4906.00**	0.00	81.1	**4906.00**	27.2	**4906.00**	0.00	8.9
Code511w_rp15	5123.00	0.00	104.1	5127.00	4.07	263.1	**5131.00**	216.2	5123.00	0.00	63.4
Code511w_rp20	5327.30	13.81	231.5	5329.93	7.26	219.5	**5334.00**	133.2	**5334.00**	0.00	76.0
Code611w_rp10	**4595.00**	0.00	82.7	**4595.00**	0.00	93.3	**4595.00**	44.5	**4595.00**	0.00	17.7
Code611w_rp15	**4881.00**	0.00	47.3	**4881.00**	0.00	67.0	**4881.00**	114.8	**4881.00**	0.00	15.7
Code611w_rp20	4951.73	1.46	137.0	4951.20	2.44	225.5	4944.00	198.1	**4952.00**	0.00	96.0
Code711w_rp10	**5586.00**	0.00	48.9	**5586.00**	0.00	78.9	**5586.00**	101.0	**5586.00**	0.00	8.7
Code711w_rp15	**5827.00**	0.00	155.5	5826.27	4.02	160.9	**5827.00**	210.9	**5827.00**	0.00	31.6
Code711w_rp20	5884.37	15.92	53.7	5892.30	2.74	216.4	**5893.00**	254.3	**5893.00**	0.00	29.8
Code811w_rp10	**4609.00**	0.00	70.5	**4609.00**	0.00	169.6	**4609.00**	27.2	**4609.00**	0.00	21.6
Code811w_rp15	4674.47	1.38	158.4	4674.87	0.73	233.9	**4675.00**	123.4	**4675.00**	0.00	41.6
Code811w_rp20	4857.63	2.01	154.6	4854.60	6.59	271.3	**4858.00**	118.8	**4858.00**	0.00	24.4
Code911w_rp10	**5302.00**	0.00	28.8	**5302.00**	0.00	37.2	**5302.00**	19.2	**5302.00**	0.00	7.5
Code911w_rp15	5157.63	1.13	204.7	5156.90	2.01	284.7	**5158.00**	157.8	5157.93	0.25	220.6
Code911w_rp20	5458.67	1.03	178.7	5457.50	1.78	208.9	5455.00	202.2	**5459.00**	0.00	92.5
Code1011w_rp10	5003.67	7.30	66.7	5004.10	4.93	104.4	**5005.00**	103.5	**5005.00**	0.00	18.2
Code1011w_rp15	5194.47	2.29	233.2	5194.23	3.52	223.6	**5195.00**	48.2	**5195.00**	0.00	29.2
Code1011w_rp20	**5399.00**	0.00	49.0	**5399.00**	0.00	261.8	**5399.00**	184.4	**5399.00**	0.00	60.8
geometric mean	5043.99			5044.47			5043.74		**5044.90**		
#best results	16			13			**27**		**27**		
#un. best res.	0			0			**3**		**3**		

Table 4.11: Comparison to results from the literature with a runtime of 600 seconds and $n = 150$. The Tabu Search (TS_{Al}) and the Hybrid Memetic Algorithm (HMA) by Alekseeva et al. [6] and Alekseeva and Kochetov [4], respectively, compared to our best configuration GA + solA + ML-ES + improved TS.

	TS_{Al}			HMA			GA + solA + ML-ES + improved TS		
Instance	$\overline{obj}$	*sd*	$t^*[s]$	$\overline{obj}$	*sd*	$t^*[s]$	$\overline{obj}$	*sd*	$t^*[s]$
Code1_150w_rp10	**7180.00**	0.00	40.10	**7180.00**	0.00	118.50	**7180.00**	0.00	29.20
Code1_150w_rp15	7152.23	5.02	207.00	7132.60	32.10	428.80	**7153.93**	0.25	133.00
Code1_150w_rp20	**7247.77**	7.45	368.60	7211.07	26.63	353.10	7247.27	7.97	241.40
Code2_150w_rp10	7321.07	23.97	146.70	7325.57	16.59	334.60	**7337.00**	0.00	242.70
Code2_150w_rp15	7736.87	8.36	174.60	7732.87	11.63	335.90	**7745.00**	0.00	96.90
Code2_150w_rp20	7796.43	14.76	386.30	7770.07	25.08	510.10	**7802.03**	15.79	211.50
Code3_150w_rp10	**7299.00**	0.00	103.90	**7299.00**	0.00	267.90	**7299.00**	0.00	35.40
Code3_150w_rp15	7596.47	12.06	252.60	7593.07	16.98	231.70	**7603.10**	2.75	142.50
Code3_150w_rp20	7610.47	63.78	217.80	7630.60	14.08	242.60	**7646.87**	4.32	274.00
Code4_150w_rp10	7306.17	39.97	157.00	7307.63	19.12	298.00	**7318.00**	0.00	38.30
Code4_150w_rp15	7406.73	7.08	180.10	7392.30	18.53	259.10	**7409.00**	0.00	71.30
Code4_150w_rp20	7926.00	5.19	227.30	7917.87	10.70	244.00	**7927.50**	2.74	251.30
Code5_150w_rp10	6972.50	5.19	173.70	6968.90	8.56	202.40	**6975.00**	0.00	32.90
Code5_150w_rp15	**7154.77**	19.25	370.60	7135.10	26.72	459.90	7139.97	26.56	214.60
Code5_150w_rp20	7322.50	6.30	272.60	7316.13	13.54	457.00	**7326.50**	3.29	227.30
Code6_150w_rp10	7047.27	7.02	182.50	7043.60	10.88	323.50	**7050.00**	0.00	36.90
Code6_150w_rp15	7184.83	4.49	183.80	7172.50	16.84	409.60	**7186.00**	0.00	71.60
Code6_150w_rp20	7378.10	14.45	338.40	7333.67	39.97	500.50	**7386.00**	0.00	133.90
Code7_150w_rp10	6247.10	3.59	264.10	**6248.17**	2.57	397.90	6248.10	0.55	190.10
Code7_150w_rp15	6839.60	2.19	107.20	6834.33	9.36	175.60	**6840.00**	0.00	82.90
Code7_150w_rp20	7284.37	18.24	129.50	7275.30	20.64	341.10	**7290.83**	14.02	203.10
Code8_150w_rp10	**7732.00**	0.00	159.20	**7732.00**	0.00	232.20	**7732.00**	0.00	28.70
Code8_150w_rp15	7658.23	7.54	237.50	7650.80	20.36	443.60	**7662.00**	0.00	103.10
Code8_150w_rp20	**7848.80**	8.38	164.50	7836.40	18.38	428.80	7846.73	11.06	188.20
Code9_150w_rp10	**6855.00**	0.00	182.20	6853.47	5.84	309.40	**6855.00**	0.00	55.50
Code9_150w_rp15	6881.30	5.52	297.90	6878.13	7.77	350.70	**6883.40**	0.93	148.40
Code9_150w_rp20	**7177.90**	19.76	230.80	7145.17	35.49	453.60	7160.40	41.30	299.90
Code10_150w_rp10	**6715.00**	0.00	88.80	**6715.00**	0.00	209.40	**6715.00**	0.00	30.20
Code10_150w_rp15	7009.07	13.99	231.00	7008.07	16.87	405.10	**7014.00**	0.00	104.30
Code10_150w_rp20	7201.07	13.43	320.00	7181.53	24.32	489.10	**7203.40**	10.21	175.30
geometric mean	7260.27			7251.42			**7263.31**		
#best results	9			5			**25**		
#unique best res.	4			1			**20**		

Table 4.12: Comparison to results from the literature with a runtime of 600 seconds and $n = 200$. The Tabu Search (TS_{Al}) and the Hybrid Memetic Algorithm (HMA) by Alekseeva et al. [6] and Alekseeva and Kochetov [4], respectively, compared to our best configuration GA + solA + ML-ES + improved TS.

	TS_{Al}			HMA			GA + solA + ML-ES + improved TS		
Instance	$\overline{obj}$	*sd*	$t^*[s]$	$\overline{obj}$	*sd*	$t^*[s]$	$\overline{obj}$	*sd*	$t^*[s]$
Code1_200w_rp10	9545.43	35.14	402.30	9505.07	57.16	348.50	**9594.00**	10.37	243.10
Code1_200w_rp15	10076.73	49.31	337.60	10051.83	59.42	348.30	**10095.00**	37.02	297.10
Code1_200w_rp20	**9837.17**	53.95	405.50	9767.93	58.96	365.60	9831.97	56.35	460.50
Code2_200w_rp10	**9324.50**	50.20	279.80	9217.80	58.07	455.30	9321.13	26.28	130.10
Code2_200w_rp15	9578.77	46.03	370.60	9514.93	51.54	286.20	**9626.67**	17.34	392.00
Code2_200w_rp20	**9667.17**	32.12	386.70	9602.20	38.63	258.70	9666.37	52.72	421.30
Code3_200w_rp10	9367.07	32.45	237.10	9329.37	53.93	350.40	**9374.30**	28.15	227.00
Code3_200w_rp15	9355.93	18.85	323.80	9310.30	44.48	317.70	**9365.97**	17.19	281.90
Code3_200w_rp20	9286.17	67.10	358.20	9253.50	63.57	313.00	**9296.67**	70.96	426.90
Code4_200w_rp10	8882.03	18.31	259.60	8877.13	22.02	460.40	**8888.47**	14.39	115.60
Code4_200w_rp15	9169.93	18.46	279.30	9116.27	68.57	365.90	**9179.03**	32.68	241.30
Code4_200w_rp20	**9439.13**	34.47	393.30	9402.23	55.74	248.50	9404.70	89.41	388.50
Code5_200w_rp10	9227.30	48.62	294.60	9240.40	52.15	304.80	**9273.10**	27.45	268.20
Code5_200w_rp15	9242.57	64.44	382.50	9237.70	41.65	325.90	**9252.03**	42.10	320.90
Code5_200w_rp20	9498.80	38.81	364.40	9422.63	52.81	379.40	**9512.10**	42.91	345.90
Code6_200w_rp10	9825.20	35.02	402.10	9808.13	39.34	330.50	**9850.53**	5.58	197.50
Code6_200w_rp15	10119.03	52.39	401.30	10095.73	41.17	269.30	**10148.23**	27.71	326.70
Code6_200w_rp20	**10283.10**	83.09	438.90	10210.53	59.37	270.30	10261.53	91.67	452.50
Code7_200w_rp10	9225.70	42.60	356.90	9183.77	55.95	382.20	**9270.30**	20.44	222.80
Code7_200w_rp15	9556.13	39.65	424.40	9496.63	59.54	267.30	**9580.30**	35.03	283.90
Code7_200w_rp20	9902.20	43.20	430.40	9860.03	52.13	221.80	**9943.10**	33.88	361.90
Code8_200w_rp10	9088.17	9.62	269.40	9046.43	34.70	345.10	**9092.57**	2.37	170.60
Code8_200w_rp15	9047.13	47.40	413.80	8987.20	41.46	244.00	**9063.10**	41.76	357.90
Code8_200w_rp20	9329.67	29.32	368.20	9248.07	59.96	302.60	**9342.90**	23.35	484.30
Code9_200w_rp10	9009.53	3.68	381.70	8950.47	59.78	324.80	**9011.40**	8.76	182.90
Code9_200w_rp15	9124.70	66.93	341.00	9086.47	65.56	297.60	**9168.20**	23.40	335.40
Code9_200w_rp20	9438.00	17.91	383.40	9404.67	42.67	300.90	**9452.57**	16.55	416.80
Code10_200w_rp10	9382.67	25.28	388.00	9365.40	46.44	498.60	**9411.00**	0.00	151.70
Code10_200w_rp15	9290.80	49.24	408.10	9240.83	57.79	252.70	**9312.40**	51.91	434.30
Code10_200w_rp20	**9741.20**	35.77	467.60	9683.63	50.92	328.30	9688.73	74.95	460.40
geometric mean	9456.10			9411.33			**9470.05**		
#best results	6			0			**24**		
#unique best res.	6			0			**24**		

Table 4.13: Results of Wilcoxon Rank Sum tests with error levels of 5% for the algorithms of the literature and the GA.

	TS	HMA	GA + solA + ML-ES + improved TS	Σ
TS	–	45	3	57
HMA	4	–	1	7
GA + solA + ML-ES + improved TS	38	56	–	**123**

value of the LP relaxation and the optimal value to the follower problem often coincide, it is not the case here. During our runs the algorithm might have visited the optimal solution but it was not able to identify it because its objective value is only approximated by the LP evaluation and was therefore discarded later.

Uniform Instances

Next, we tested our algorithm on the Uniform instances and compared the results with the VNS and STS by Davydov et al. [38], who tested their algorithms on a Pentium Intel Core Dual with 2.66 GHz. They published only the result of one single run (obj_1). Since we perform 30 runs, it is not straightforward to compare these approaches. Therefore we list for our algorithm the average objective value ($\overline{obj}$) and the objective value of the best run (obj^*).

When using the configuration that performed best in the Euclidean instances (GA + solA + ML-ES + imp. TS) we see in Table 4.17 that for the instances where $r = p = 7$ the algorithm quickly converges to a non-optimal solution. We observed that on Uniform instances the case occurs more frequently where a good LP value does not necessarily lead to a good (or optimal) solution. To further investigate this issue we modified the ML-ES to solve the follower's problem exactly instead of only using the LP evaluation (GA + solA + ML-ES(EE) + imp. TS). Indeed, although the runtime increases, with this modification the GA was able to find the same solutions as STS for all but one instances with $r = p = 7$. Compared to the VNS, better solutions were found for five of ten instances but more time was required. STS performed excellent on these instances and was able to find equally good or better solutions in less time. However, the best of the 30 runs for GA + solA + ML-ES(EE) + imp. TS identifies a better or equally good solution for each instance.

When considering the instances where $w_j = 1,\ \forall j \in J$ and $r = p \in \{25, 30\}$ the modification of ML-ES is apparently not beneficial because the runtime increases and the average objective value is often equal but sometimes even worse. The results in Table 4.17 confirm the observation of Davydov et al. [38] that when r and p increase at least for these instances the problem gets easier since GA + solA + ML-ES + imp. TS obtains equally good results as the VNS with a low standard deviation. For instances with $r = p = 25$ we observe the largest deviation of the objective values. While the results of the GA are often within 3% of the VNS results in 9 out of 10 instances, they are generally worse and in one case it is equally good. However, also for these cases the best run for each instance found a solution that is equally good or better.

From these results we conclude that especially for the Uniform instances the search order of the neighborhoods is important to speed up the algorithm. It seems that it is often unnecessary to search through the whole swap neighborhood but to consider promising moves first. An interesting extension to the presented algorithm would be to incorporate the VNS into a hybrid GA, e.g., we could replace the swap neighborhood for the TS with the neighborhoods of the VNS. The combination of the strengths of both the special neighborhood structure of the VNS and STS with the hybrid GA could potentially lead to an algorithm that performs well on both Euclidean and Uniform instances.

Table 4.14: Comparison of the results from instances with $n = 100$ of the so far best exact methods MEM by Kochetov et al. [81] and B&C by Roboredo and Pessoa [115], the so far best heuristic methods HMA and STS by Alekseeva and Kochetov [4] and Davydov et al. [38], respectively, and our GA + solA + ML-ES + imp. TS with a runtime of 1800 seconds.

	B & C		MEM		HMA			STS		GA+solA+ML-ES+imp. TS		
Instance	$\overline{obj}$	$t^*[s]$	$\overline{obj}$	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	obj_1	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code111w_rp10	**4361.00**	10217.0	**4361.00**	3600.0	**4361.00**	0.00	97.4	**4361.00**	63.7	**4361.00**	0.00	11.2
Code111w_rp15	**4596.00**	9752.0	**4596.00**	4320.0	**4596.00**	0.00	118.4	**4596.00**	173.3	**4596.00**	0.00	16.3
Code111w_rp20	**4512.00**	>36000	**4512.00**[a]	60.0	4511.47	1.38	492.9	4484.00	118.1	**4512.00**	0.00	159.4
Code211w_rp10	**5310.00**	9488.8	**5310.00**	2520.0	**5310.00**	0.00	34.6	**5310.00**	23.5	**5310.00**	0.00	8.1
Code211w_rp15	**5373.00**	80956.4	**5373.00**	230700.0	**5373.00**	0.00	116.1	**5373.00**	88.9	**5373.00**	0.00	18.0
Code211w_rp20	**5432.00**	>36000	**5432.00**[a]	11100.0	**5432.00**	0.00	284.0	**5432.00**	289.2	**5432.00**	0.00	75.4
Code311w_rp10	**4483.00**	19071.3	**4483.00**	8760.0	**4483.00**	0.00	38.8	**4483.00**	33.8	**4483.00**	0.00	8.2
Code311w_rp15	**4800.00**	27707.3	**4800.00**	23700.0	**4800.00**	0.00	120.5	**4800.00**	91.1	**4800.00**	0.00	18.9
Code311w_rp20	**4893.00**	>36000	**4893.00**[a]	14880.0	4892.93	0.37	297.7	**4893.00**	211.3	**4893.00**	0.00	85.4
Code411w_rp10	**4994.00**	13743.9	**4994.00**	1980.0	**4994.00**	0.00	34.1	**4994.00**	19.3	**4994.00**	0.00	8.0
Code411w_rp15	**5064.00**	84140.1	**5064.00**	73380.0	**5064.00**	0.00	206.9	**5064.00**	121.3	**5064.00**	0.00	54.5
Code411w_rp20	**5209.00**	>36000	**5209.00**[a]	300.0	**5209.00**	0.00	259.6	**5209.00**	288.9	**5209.00**	0.00	46.8
Code511w_rp10	**4906.00**	80413.8	**4906.00**	23940.0	**4906.00**	0.00	75.1	**4906.00**	27.2	**4906.00**	0.00	11.4
Code511w_rp15	**5131.00**	79099.6	**5131.00**	127200.0	5130.67	1.49	579.1	**5131.00**	216.2	5123.00	0.00	58.2
Code511w_rp20	**5334.00**	>36000	**5334.00**[a]	6600.0	**5334.00**	0.00	274.4	**5334.00**	133.2	**5334.00**	0.00	60.9
Code611w_rp10	**4595.00**	51583.2	**4595.00**	8580.0	**4595.00**	0.00	154.6	**4595.00**	44.5	**4595.00**	0.00	21.6
Code611w_rp15	**4881.00**	28342.7	**4881.00**	137580.0	**4881.00**	0.00	52.5	**4881.00**	114.8	**4881.00**	0.00	17.2
Code611w_rp20	**4952.00**	>36000	**4952.00**[a]	11400.0	**4952.00**	0.00	281.6	4944.00	198.1	**4952.00**	0.00	58.1
Code711w_rp10	**5586.00**	20352.7	**5586.00**	4380.0	**5586.00**	0.00	108.3	**5586.00**	101.0	**5586.00**	0.00	9.2
Code711w_rp15	**5827.00**	48600.5	**5827.00**	79200.0	**5827.00**	0.00	172.6	**5827.00**	210.9	**5827.00**	0.00	33.8
Code711w_rp20	**5893.00**	>36000	**5893.00**[a]	5820.0	**5893.00**	0.00	168.7	**5893.00**	254.3	**5893.00**	0.00	21.5
Code811w_rp10	**4609.00**	26808.0	**4609.00**	9120.0	**4609.00**	0.00	139.2	**4609.00**	27.2	**4609.00**	0.00	18.8
Code811w_rp15	**4675.00**	115183.5	**4675.00**	274200.0	**4675.00**	0.00	266.8	**4675.00**	123.4	**4675.00**	0.00	52.1
Code811w_rp20	**4858.00**	>36000	**4858.00**[a]	34200.0	**4858.00**	0.00	370.3	**4858.00**	118.8	**4858.00**	0.00	22.9
Code911w_rp10	**5302.00**	2377.9	**5302.00**	360.0	**5302.00**	0.00	30.6	**5302.00**	19.2	**5302.00**	0.00	7.5
Code911w_rp15	**5158.00**	>36000	**5158.00**	>36000	**5158.00**	0.00	338.8	**5158.00**	157.8	**5158.00**	0.00	240.1
Code911w_rp20	**5459.00**	>36000	**5459.00**[a]	9900.0	5458.90	0.55	350.6	5455.00	202.2	**5459.00**	0.00	146.2
Code1011w_rp10	**5005.00**	33765.1	**5005.00**	5820.0	**5005.00**	0.00	120.0	**5005.00**	103.5	**5005.00**	0.00	15.0
Code1011w_rp15	**5195.00**	72034.4	**5195.00**	>36000	**5195.00**	0.00	223.8	**5195.00**	48.2	**5195.00**	0.00	28.2
Code1011w_rp20	**5399.00**	>36000	**5399.00**[a]	7800.0	**5399.00**	0.00	188.4	**5399.00**	184.4	**5399.00**	0.00	28.0
geometric mean	**5045.18**		**5045.18**		5045.15			5043.74		5044.92		
#best results	**30**		**30**		26			27		29		
#unique best results	0		0		0			0		0		

[a] time needed for finding solutions that are within 5% of the optimum, i.e., the optimality is not proven

Table 4.15: Comparison of the results from instances with $n = 150$ and $n = 200$ of HMA and our GA + solA + ML-ES + imp.TS with a runtime of 1800 seconds.

	HMA		GA + solA + ML-ES+imp.TS			HMA		GA + solA + ML-ES+imp.TS	
Instance	$\overline{obj}$	*sd*	$\overline{obj}$	*sd*	***Instance***	$\overline{obj}$	*sd*	$\overline{obj}$	*sd*
Code1_150w_rp10	**7180.00**	0.00	**7180.00**	0.00	Code1_200w_rp10	9575.27	24.62	**9598.00**	0.00
Code1_150w_rp15	7153.90	0.31	**7154.00**	0.00	Code1_200w_rp15	10107.73	31.02	**10130.00**	0.00
Code1_150w_rp20	7249.70	0.47	**7249.90**	0.31	Code1_200w_rp20	9858.93	31.22	**9894.33**	24.19
Code2_150w_rp10	**7337.00**	0.00	**7337.00**	0.00	Code2_200w_rp10	9325.33	40.56	**9333.80**	35.42
Code2_150w_rp15	7744.10	3.45	**7745.00**	0.00	Code2_200w_rp15	9615.30	23.20	**9633.80**	6.57
Code2_150w_rp20	7804.90	8.76	**7809.40**	3.29	Code2_200w_rp20	9685.57	27.10	**9702.27**	20.60
Code3_150w_rp10	**7299.00**	0.00	**7299.00**	0.00	Code3_200w_rp10	9378.57	10.85	**9382.00**	0.00
Code3_150w_rp15	7603.40	2.28	**7604.00**	0.00	Code3_200w_rp15	9362.63	11.78	**9371.00**	0.00
Code3_150w_rp20	**7648.00**	0.00	7647.47	2.92	Code3_200w_rp20	9334.60	39.89	**9352.00**	58.98
Code4_150w_rp10	**7318.00**	0.00	**7318.00**	0.00	Code4_200w_rp10	8893.73	9.19	**8897.00**	0.00
Code4_150w_rp15	**7409.00**	0.00	**7409.00**	0.00	Code4_200w_rp15	9174.73	14.64	**9185.00**	0.00
Code4_150w_rp20	7927.00	3.81	**7928.00**	0.00	Code4_200w_rp20	9458.97	23.79	**9473.83**	6.39
Code5_150w_rp10	**6975.00**	0.00	**6975.00**	0.00	Code5_200w_rp10	9266.47	28.84	**9281.27**	9.49
Code5_150w_rp15	**7162.13**	8.99	7158.93	15.76	Code5_200w_rp15	**9286.63**	30.70	9259.57	52.77
Code5_150w_rp20	7324.87	4.01	**7326.77**	2.87	Code5_200w_rp20	9541.27	23.04	**9556.00**	0.00
Code6_150w_rp10	7049.67	1.83	**7050.00**	0.00	Code6_200w_rp10	9847.47	8.89	**9852.00**	0.00
Code6_150w_rp15	**7186.00**	0.00	**7186.00**	0.00	Code6_200w_rp15	10141.63	18.54	**10155.27**	9.03
Code6_150w_rp20	7383.83	6.61	**7386.00**	0.00	Code6_200w_rp20	10333.63	30.55	**10350.83**	39.81
Code7_150w_rp10	**6250.30**	1.29	6248.00	0.00	Code7_200w_rp10	9252.13	33.59	**9277.00**	0.00
Code7_150w_rp15	**6840.00**	0.00	**6840.00**	0.00	Code7_200w_rp15	9570.43	16.92	**9588.00**	0.00
Code7_150w_rp20	7295.77	6.76	**7297.00**	0.00	Code7_200w_rp20	9930.83	31.85	**9952.33**	17.29
Code8_150w_rp10	**7732.00**	0.00	**7732.00**	0.00	Code8_200w_rp10	9092.77	4.93	**9093.00**	0.00
Code8_150w_rp15	**7662.00**	0.00	**7662.00**	0.00	Code8_200w_rp15	9069.63	21.61	**9085.80**	21.12
Code8_150w_rp20	7850.77	1.28	**7851.00**	0.00	Code8_200w_rp20	9326.77	30.95	**9348.77**	18.64
Code9_150w_rp10	**6855.00**	0.00	**6855.00**	0.00	Code9_200w_rp10	9012.60	1.04	**9013.00**	0.00
Code9_150w_rp15	6883.30	1.21	**6883.80**	0.61	Code9_200w_rp15	9160.20	20.13	**9174.07**	10.59
Code9_150w_rp20	7185.30	4.67	**7187.00**	3.81	Code9_200w_rp20	9450.97	13.26	**9462.67**	1.83
Code10_150w_rp10	**6715.00**	0.00	**6715.00**	0.00	Code10_200w_rp10	9406.97	4.80	**9411.00**	0.00
Code10_150w_rp15	**7014.00**	0.00	**7014.00**	0.00	Code10_200w_rp15	9323.43	34.26	**9353.80**	25.02
Code10_150w_rp20	7205.13	2.73	**7206.00**	0.00	Code10_200w_rp20	**9751.03**	18.42	9742.53	30.86
geometric mean	7265.37		**7265.68**			9478.43		**9490.76**	
#best results	16		**27**			2		**28**	
#unique best res.	3		**14**			2		**28**	

Table 4.16: Results of Wilcoxon Rank Sum tests with error levels of 5% for HMA and the GA with longer runtime for all 90 Euclidean test instances.

	HMA	GA + solA + ML-ES + improved TS	Σ
HMA	–	3	3
GA + solA + ML-ES + improved TS	**31**	–	**31**

Distribution of Facilities

After the computational tests with binary essential customer behavior we want to investigate how solutions to the problem may look like and what properties they might have. Although we cannot make general statements that apply to all problem instances, we show in Figure 4.3 the graphical representation of optimal solutions for one instance with 100 customers (Code311w) with different r and p values. The circles are customer locations, the filled points stand for locations chosen by the leader and the rectangles represent facilities of the follower. The size of the symbols depends on the demand of the corresponding location, i.e., the larger the symbol the higher the demand.

In Figure 4.3 it seems that the leader tends to choose locations that are more or less evenly spreaded across the whole region with a focus on the more crowded areas. The follower then appears to prefer locations in the vicinity of a leader's facility. Another interesting observation is that while some locations are picked each time by the leader for different r and p values, some other locations are not always chosen. Visualizations on other instances reveal similar patterns, but as said before it is hard to draw precise general conclusions.

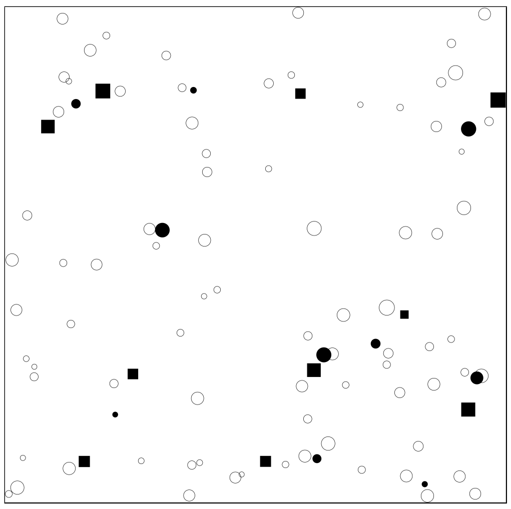
4.9.2 Proportional Essential

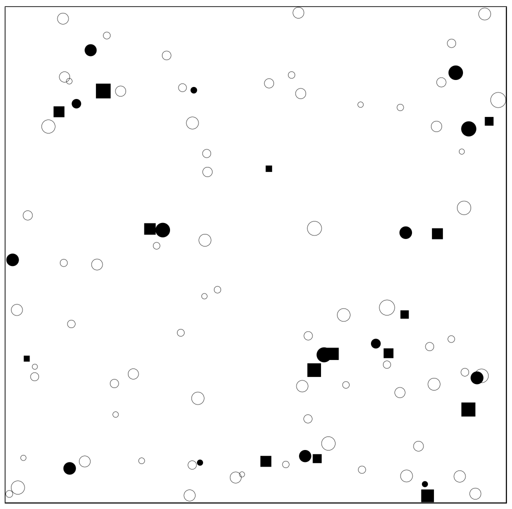
For evaluating the EA with the other customer behavior scenarios we use the configuration determined by previous tests as basis. We choose to evaluate the impact of the solution archive on the results in Table 4.18 as well as investigating the performance compared to the alternating heuristic (AH) by Kochetov et al. [81]. Their AH is based on a starting solution for the leader to find the optimal facility locations for the follower which are computed using the linear MIP model for the follower. This follower solution is subsequently chosen as leader solution and the optimal follower solution is found again. This procedure is repeated until a solution is obtained which has already been generated. Since the repeated exact computation of the optimal follower's locations is time-consuming we modified their approach by using our greedy algorithm instead of the MIP as described in Section 4.5 for finding the locations for the follower. We analyze the following configurations:

- The EA variant where the final best solution is not evaluated with the MIP. This

Table 4.17: Comparison of results from Uniform instances of VNS and STS by Davydov et al. [38] with our algorithm GA+solA+ML-ES+imp. TS and our modification ML-ES(EE).

	VNS		STS		GA+solA+ML-ES+imp. TS				GA+solA+ML-ES(EE)+imp. TS			
Instance	obj_1	$t^*[s]$	obj_1	$t^*[s]$	obj^*	$\overline{obj}$	sd	$t^*[s]$	obj^*	$\overline{obj}$	sd	$t^*[s]$
123Comp-Unif_rp7	5009.00	304.17	5009.00	65.09	4904.00	4904.00	0.00	54.10	5009.00	5009.00	0.00	504.00
223Comp-Unif_rp7	5459.00	182.91	5459.00	63.18	5459.00	5459.00	0.00	38.40	5459.00	5459.00	0.00	245.50
323Comp-Unif_rp7	5009.00	145.01	5019.00	54.69	5003.00	5003.00	0.00	53.10	5019.00	5019.00	0.00	445.40
423Comp-Unif_rp7	4908.00	296.63	4908.00	145.22	4846.00	4846.00	0.00	58.80	4908.00	4908.00	0.00	641.80
523Comp-Unif_rp7	5198.00	292.05	5208.00	22.63	5206.00	5206.00	0.00	57.90	5208.00	5208.00	0.00	374.10
623Comp-Unif_rp7	5032.00	296.52	5032.00	197.08	5032.00	5032.00	0.00	51.00	5032.00	5028.50	19.17	386.70
723Comp-Unif_rp7	5055.00	286.04	5055.00	62.23	4962.00	4962.00	0.00	46.80	5055.00	5055.00	0.00	446.10
823Comp-Unif_rp7	4860.00	295.77	4951.00	74.49	4847.00	4847.00	0.00	83.30	4951.00	4951.00	0.00	441.30
923Comp-Unif_rp7	5060.00	217.70	5127.00	111.27	5127.00	5127.00	0.00	80.20	5127.00	5127.00	0.00	825.00
1023Comp-Unif_rp7	5067.00	322.48	5084.00	278.18	5000.00	5000.00	0.00	55.20	5084.00	5084.00	0.00	510.70
$w_j = 1.\ \forall j \in J$:												
123Comp-Unif_rp25	62.00	16.12	62.00	30.27	62.00	61.20	1.27	89.90	62.00	61.30	1.09	154.90
223Comp-Unif_rp25	62.00	157.40	62.00	38.84	61.00	60.70	0.47	96.40	62.00	60.77	0.57	160.30
323Comp-Unif_rp25	61.00	34.60	61.00	47.48	61.00	60.20	1.30	106.60	61.00	59.50	1.83	172.70
423Comp-Unif_rp25	59.00	25.59	59.00	17.26	59.00	58.60	0.89	112.90	59.00	58.80	0.48	155.00
523Comp-Unif_rp25	63.00	70.91	63.00	41.00	63.00	61.90	0.92	95.00	63.00	61.90	0.84	164.80
623Comp-Unif_rp25	62.00	16.39	61.00	35.61	62.00	60.13	1.48	91.00	62.00	60.07	1.26	148.00
723Comp-Unif_rp25	66.00	15.69	66.00	19.42	66.00	65.87	0.73	110.40	66.00	65.87	0.73	172.90
823Comp-Unif_rp25	60.00	32.98	60.00	19.38	60.00	58.20	1.16	91.60	60.00	58.43	1.07	145.00
923Comp-Unif_rp25	63.00	18.96	63.00	17.43	63.00	61.70	1.18	98.60	63.00	61.43	1.38	164.60
1023Comp-Unif_rp25	65.00	15.49	65.00	21.58	65.00	65.00	0.00	91.90	65.00	65.00	0.00	153.70
123Comp-Unif_rp30	70.00	15.73	69.00	50.97	70.00	70.00	0.00	116.50	70.00	70.00	0.00	182.30
223Comp-Unif_rp30	65.00	15.81	65.00	67.87	65.00	65.00	0.00	103.40	65.00	65.00	0.00	167.30
323Comp-Unif_rp30	65.00	15.82	65.00	86.92	65.00	65.00	0.00	114.10	65.00	65.00	0.00	174.40
423Comp-Unif_rp30	65.00	16.09	65.00	91.13	65.00	65.00	0.00	119.40	65.00	65.00	0.00	197.00
523Comp-Unif_rp30	69.00	15.58	69.00	76.31	69.00	69.00	0.00	114.80	69.00	69.00	0.00	185.20
623Comp-Unif_rp30	69.00	15.94	69.00	78.44	69.00	69.00	0.00	111.70	69.00	69.00	0.00	179.90
723Comp-Unif_rp30	72.00	15.64	72.00	74.20	72.00	72.00	0.00	113.40	72.00	72.00	0.00	182.80
823Comp-Unif_rp30	62.00	15.64	62.00	80.56	62.00	62.00	0.00	91.30	62.00	61.77	0.73	146.20
923Comp-Unif_rp30	68.00	15.42	68.00	59.36	68.00	68.00	0.00	107.50	68.00	68.00	0.00	170.00
1023Comp-Unif_rp30	73.00	15.42	71.00	126.19	73.00	73.00	0.00	115.20	73.00	73.00	0.00	176.30
geometric mean	277.42		277.24		276.76	275.48			277.78	276.24		


(a) $r = p = 10$


(b) $r = p = 15$

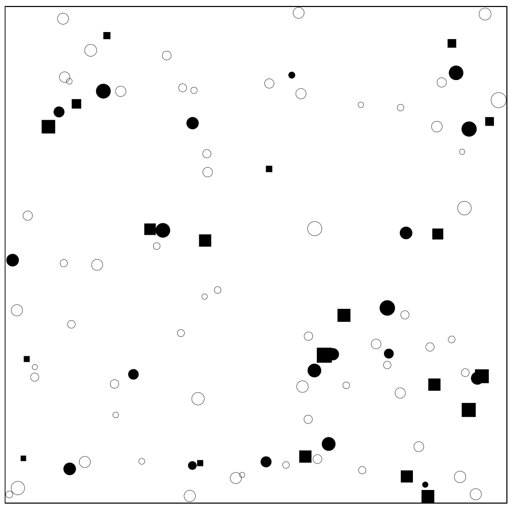

(c) $r = p = 20$

Figure 4.3: Optimal Solutions for instance Code311w with different r and p values.

means that the corresponding objective values are not exact, but only approximate values from the greedy evaluation method.

- A modified version of the Alternating Heuristic (MAH) by Kochetov et al. [81], where each solution candidate is approximated by our greedy algorithm instead of evaluated exactly.
- The EA variant (EA + MIP) that does not employ the archive and utilizes the basic local search only; the final best solution is evaluated with MIP.
- The EA variant (EA + SA + MIP) that uses the solution archive and the tabu search as local improvement method; the final best solution is evaluated with MIP.

In this table, again, $\overline{obj}$ stands for the average of the objective values over 30 runs with their standard deviation in column sd . The time needed until termination is given in column $t[s]$. Since MAH is a deterministic algorithm only one run is performed.

In Table 4.18 the numerical values are given. Numbers in parenthesis mean that evaluating the best solution candidate of the EA needed more than the time limit of 3600 seconds and so the objective values are determined by the greedy algorithm only. Therefore they are only approximations and not directly comparable to exact objective values. So in the summary of the EA configuration these values are not considered for comparison. The best value in each row is marked bold. When some values of a row are obtained by greedy evaluation and some other values in the same row are exact solution qualities, only the exact values are compared to each other, e.g., in the row with $r = p = 3$.

In some cases of the EA + MIP variant not all 30 runs terminated within the time limit so only the average over the finished runs is given, e.g., the row with $r = 3$ and $p = 5$. We observe that even for small p values of 4 and 5 we were not able to evaluate even one solution candidate in the given time limit. Another interesting point is that evaluating the candidate solution exactly via the MIP is the most time-consuming part of the algorithm; for $r = 3$ and $p = 8$ it needed over 90% of the overall time but it decreases when p increases. The run-time of all configurations that incorporate the exact evaluation increases steadily with r because of the growing complexity of the MIP.

On some instances MAH finds a solution in less time than our algorithms and especially when the exact evaluation is too time consuming it is very fast. The quality of the solutions is similar to our EA approach when we do not use the SA, but by incorporating the solution archive we boosted the performance of our algorithm so that the final solution quality is in all but 4 of the tested instances better than the quality of the solutions produced by MAH and in 3 of the 4 cases it is equal. For some of the smaller instances EA + SA + MIP has a very small standard deviation, which underlines the robustness of our algorithm.

Compared to binary customer behavior, the proportional scenario is much harder to solve and we can only approximate the value of solution candidates for instances that are only half the size.

Table 4.18: Results of proportional customer behavior with essential demand.

		EA			EA + MIP			MAH		EA + SA + MIP		
r	p	$\overline{obj}$	sd	$t[s]$	$\overline{obj}$	sd	$t[s]$	$\overline{obj}$	$t[s]$	$\overline{obj}$	sd	$t[s]$
2	2	(280.002)	0.13	106	278.671	0.15	677	277.942	667	**278.736**	0.00	600
2	3	(338.170)	0.31	157	336.587	0.19	684	334.233	535	**337.228**	0.00	625
2	4	(374.754)	0.71	154	373.455	0.48	623	373.665	503	**374.425**	0.00	674
2	5	(399.834)	0.50	200	398.642	0.90	493	399.208	260	**401.781**	0.00	505
2	6	(419.360)	0.46	241	418.779	0.67	525	419.920	275	**421.091**	0.15	586
2	7	(434.640)	0.62	223	434.388	0.74	394	431.803	272	**436.123**	0.00	440
2	8	(447.131)	0.42	202	446.710	0.46	322	446.474	158	**448.192**	0.18	440
2	9	(456.992)	0.45	300	456.615	0.63	419	455.788	166	**458.905**	0.37	529
2	10	(465.178)	0.54	300	464.620	0.53	401	463.211	173	**467.055**	0.16	416
3	2	(223.059)	0.18	144	(223.059)	0.18	144	(223.153)	<1	**(223.194)**	0.00	27
3	3	(281.283)	0.31	174	(281.283)	0.31	174	276.818	5959	**279.000**	0.00	6397
3	4	(321.185)	0.86	201	(321.185)	0.86	201	319.427	4128	**319.819**	0.00	3956
3	5	(349.644)	0.54	300	347.429*	0.48	3892	349.471	3867	**349.793**	0.00	2703
3	6	(372.924)	0.68	300	371.900*	0.98	3896	372.760	3453	**373.836**	0.12	2777
3	7	(391.264)	0.65	300	390.753*	0.74	3493	391.314	2086	**391.894**	0.39	2658
3	8	(406.302)	0.52	300	405.907*	0.77	3124	407.623	1721	**407.765**	0.08	3148
3	9	(418.553)	0.37	300	418.051*	0.53	2795	419.985	1709	**420.305**	0.18	2424
3	10	(429.040)	0.56	300	428.357	0.53	2370	430.465	1299	**431.578**	0.33	2670
4	2	(183.188)	0.11	246	(183.188)	0.11	246	**(183.223)**	<1	**(183.223)**	0.00	38
4	3	(238.953)	0.33	226	(238.953)	0.33	226	(239.527)	<1	**(239.628)**	0.00	83
4	4	(279.021)	0.56	298	(279.021)	0.56	298	(280.336)	<1	**(280.549)**	0.08	126
4	5	(310.562)	0.70	300	(310.562)	0.70	300	**(313.041)**	<1	**(313.041)**	0.00	157
4	6	(335.415)	0.65	300	(335.415)	0.65	300	(337.158)	<1	**(337.540)**	0.12	242
4	7	(355.659)	0.52	300	(355.659)	0.52	300	(356.575)	<1	**(358.233)**	0.18	267
4	8	(372.334)	0.67	300	(372.334)	0.67	300	(374.436)	<1	**(375.031)**	0.04	300
4	9	(386.207)	0.81	300	(386.207)	0.81	300	(387.975)	<1	**(389.837)**	0.12	300
4	10	(398.011)	0.74	300	(398.011)	0.74	300	(400.421)	1	**(401.428)**	0.13	300
5	2	(156.357)	0.15	199	(156.357)	0.15	199	**(156.538)**	<1	**(156.538)**	0.00	44
5	3	(207.548)	0.18	293	(207.548)	0.18	293	(207.682)	<1	**(208.025)**	0.00	112
5	4	(247.295)	0.76	300	(247.295)	0.76	300	(244.959)	<1	**(248.663)**	0.06	212
5	5	(278.806)	0.60	300	(278.806)	0.60	300	(279.889)	<1	**(281.522)**	0.00	194
5	6	(304.283)	0.54	300	(304.283)	0.54	300	(305.488)	<1	**(307.129)**	0.13	300
5	7	(325.520)	0.81	300	(325.520)	0.81	300	(327.357)	<1	**(328.314)**	0.05	300
5	8	(343.534)	0.61	300	(343.534)	0.61	300	(345.947)	<1	**(346.254)**	0.12	300
5	9	(358.373)	1.02	300	(358.373)	1.02	300	(360.572)	<1	**(362.159)**	0.31	300
5	10	(371.213)	0.74	300	(371.213)	0.74	300	**(374.737)**	1	(374.556)	0.24	300
geo. mean		(330.38)			330.06			330.55		**331.67**		
#best res.		-			0			1		**35**		
#u. best res.		-			0			1		**32**		

*Not all of the 30 runs completed in the time limit

4.9.3 Partially Binary Essential

In the next computational tests we analyzed the partially binary essential customer behavior. Since there is, to the best of our knowledge, no algorithm with numerical results described in the literature we only compare different configurations of our EA. Similarly to the proportional case we compare our EA without exact evaluation, the EA with exact evaluation in the end (EA + MIP) and the EA with solution archive and exact evaluation (EA + SA + MIP). Table 4.19 shows our numerical results, column names have the same meaning as before.

Table 4.19: Results of partially binary customer behavior with essential demand.

		EA			EA + MIP			EA + SA + MIP		
r	p	$\overline{obj}$	sd	$t[s]$	$\overline{obj}$	sd	$t[s]$	$\overline{obj}$	sd	$t[s]$
2	2	(283.753)	0.29	26	278.450	1.15	549	**278.931**	0.00	529
2	3	(315.105)	0.47	33	309.243	0.49	539	**310.013**	0.00	515
2	4	(337.476)	1.13	32	330.013	1.24	432	**332.359**	0.00	376
2	5	(349.361)	0.35	38	343.743	0.46	417	**345.116**	0.32	444
2	6	(359.113)	0.53	36	354.270	0.76	436	**357.640**	0.45	437
2	7	(368.140)	0.64	40	363.248	0.70	404	**366.883**	2.55	412
2	8	(376.035)	0.71	33	370.994	1.18	378	**376.136**	2.12	382
2	9	(383.784)	0.84	47	378.761	1.91	382	**385.354**	1.08	316
2	10	(391.553)	1.31	44	384.370	1.72	378	**388.068**	0.47	303
3	2	(259.461)	0.25	38	247.791	0.34	590	**247.946**	0.00	606
3	3	(289.450)	0.74	42	277.505	1.26	451	**279.000**	0.00	432
3	4	(311.032)	1.50	43	299.228	1.47	380	**302.217**	0.00	354
3	5	(323.333)	0.93	39	312.901	1.92	362	**313.582**	0.43	362
3	6	(334.559)	0.65	42	324.425	0.88	393	**325.250**	0.97	386
3	7	(343.815)	0.66	49	333.255	1.39	354	**335.827**	1.37	348
3	8	(352.919)	0.83	57	341.766	0.91	331	**347.421**	2.07	344
3	9	(360.388)	1.14	72	349.304	1.94	333	**356.983**	2.23	320
3	10	(367.969)	1.45	64	355.705	1.97	305	**363.047**	1.97	348
4	2	(239.204)	0.54	58	225.354	0.57	559	**225.640**	0.00	560
4	3	(269.482)	0.64	52	253.806	1.15	429	**255.072**	0.00	410
4	4	(290.283)	1.68	45	274.913	1.94	331	**279.000**	0.00	330
4	5	(303.248)	1.61	51	288.922	1.95	349	**291.000**	0.62	330
4	6	(315.374)	0.65	56	301.074	0.58	331	**303.139**	0.78	343
4	7	(324.823)	0.61	83	310.542	0.83	325	**315.167**	0.33	298
4	8	(333.640)	0.86	78	319.463	1.15	317	**327.670**	0.00	302
4	9	(341.007)	0.87	80	327.074	3.05	299	**335.919**	0.13	318
4	10	(348.310)	1.02	97	335.461	1.64	299	**343.982**	0.58	304
5	2	(220.928)	0.72	58	211.955	0.94	667	**212.604**	0.00	626
5	3	(250.491)	0.99	52	240.746	1.07	515	**242.035**	0.00	427
5	4	(272.251)	2.35	45	262.368	2.43	379	**265.917**	0.00	365
5	5	(285.997)	1.32	51	276.552	1.71	403	**278.193**	0.00	401
5	6	(297.032)	0.58	56	287.690	0.77	402	**290.754**	0.88	400
5	7	(306.395)	0.75	83	297.093	0.92	356	**301.843**	0.49	340
5	8	(315.239)	0.74	78	306.201	1.21	370	**314.168**	0.00	340
5	9	(323.263)	0.98	80	314.983	1.40	353	**323.154**	0.00	364
5	10	(330.717)	2.00	99	322.066	2.12	315	**330.684**	0.00	311
geo. mean		(316.00)			305.65			**309.24**		
#best res.		-			0			**36**		
# u. best res.		-			0			**36**		

First, we observe that for all our tested instance we were able to evaluate the best solution candidate exactly, even for the cases which were not possible for the proportional customer behavior. Also, the time needed for this evaluation is much less and at most about 10 minutes for the hardest instance (in the case of $r = 5$ and $p = 2$). The deviation of the greedy objective value and the exact objective value is around 3% on average which shows that our greedy solution evaluation method is relatively accurate. In this customer behavior scenario the benefits of using a solution archive are even more obvious than in the other scenarios as EA + SA + MIP performed better in all our tested instances. Second, we see again that for a fixed r the time needed for solving the model decreases with increasing p because the solution space is getting smaller. For many of the instances we obtained a very low standard deviation which, again, shows the robustness of our approach.

Compared to the other customer behavior scenarios the complexity of partially binary behavior lies in between the binary and the proportional choice rule, where binary is the easiest to solve and proportional by far the hardest. We also see that the leader is preferred in proportional scenarios as for a fixed r and p the turnover is higher than in the partially binary case in most of the instances but especially for a large p and small r, i.e., when he is able to place more facilities than the follower. For example, the turnover for the leader when $r = 3$ and $p = 10$ is in the proportional case nearly 16% higher than when the customers use the partially binary choice.

4.9.4 Unessential Demands

We performed computational tests for all customer behavior scenarios with unessential demands. Like in the partially binary customer behavior also for unessential demands there are no numerical results available in the literature. We tested the two different follower strategies LMIN and FMAX and compared them to each other. In the following tables in addition to the average leader objective value ($\overline{obj}^{\,l}$) over 30 runs we also present the average turnover obtained by the follower for the corresponding best leader solution found ($\overline{obj}^{\,f}$). For both values the standard deviations are given as well (sd). Usually only a fraction of the total demand of all customers can be satisfied when the demand is unessential and these (average) fractions are the values in column market saturation (sat.).

The first interesting observation is that the turnover of the follower and the market saturation in the FMAX strategy is higher than in the LMIN strategy in all our test cases, which corresponds to our intuition. Also, in the proportional and the partially binary case the turnover of the leader is always lower when the follower uses the LMIN strategy. It is quite surprising that this is not always the case for the binary customer behavior, see Table 4.20. In most of the instances with $r = p = 20$ the leader objective value is higher for the LMIN strategy. This can be explained by the observation that the model for the follower is usually easier to solve in the LMIN case. Therefore the algorithm is able to converge faster and the leader can frequently obtain better facility locations. To confirm this assumption we increased the time limit on instances with $r = p = 20$ to 1800 seconds and report the results in Table 4.21. There we see that now the leader's profit of the LMIN strategy is consistently lower than with the FMAX strategy.

Table 4.20: Results of binary customer behavior with unessential demand.

			LMIN					FMAX				
Instance	r	p	$\overline{obj}^{\,l}$	sd	$\overline{obj}^{\,f}$	sd	sat.	$\overline{obj}^{\,l}$	sd	$\overline{obj}^{\,f}$	sd	sat.
Code1	10	10	1846.37	0.00	699.52	0.00	29.30%	1849.32	0.00	1539.73	0.00	39.00%
Code2	10	10	1927.16	0.00	988.90	0.00	27.72%	1929.70	0.00	1793.24	0.00	35.39%
Code3	10	10	1850.92	0.00	1164.79	0.00	32.25%	1855.38	0.01	1564.22	0.01	36.57%
Code4	10	10	1914.31	0.00	1313.22	0.00	32.51%	1918.90	0.00	1674.72	0.00	36.20%
Code5	10	10	1909.94	0.00	1189.45	0.00	30.26%	1912.81	0.00	1718.66	0.00	35.45%
Code6	10	10	1892.39	0.00	1000.04	0.00	30.39%	1898.67	0.00	1753.45	0.00	38.37%
Code7	10	10	1928.64	0.00	895.01	0.00	25.21%	1937.19	0.00	1819.43	0.00	33.54%
Code8	10	10	1889.22	0.00	995.05	0.00	30.18%	1893.17	0.00	1698.59	0.00	37.58%
Code9	10	10	1937.16	0.00	1001.99	0.00	28.27%	1943.06	0.00	1756.90	0.00	35.59%
Code10	10	10	1926.28	0.00	1165.26	0.00	30.23%	1931.94	0.00	1793.41	0.00	36.43%
Code1	15	15	2626.28	0.00	1270.15	0.00	44.84%	2630.42	0.41	2065.77	0.41	54.05%
Code2	15	15	2835.00	0.00	1148.96	0.00	37.87%	2835.93	4.31	2492.44	4.31	50.65%
Code3	15	15	2673.87	0.00	1602.77	0.00	45.73%	2675.32	5.63	2118.62	5.63	51.27%
Code4	15	15	2786.36	0.00	1576.55	0.00	43.95%	2791.15	0.00	2231.67	0.00	50.60%
Code5	15	15	2788.79	0.00	1433.94	0.00	41.23%	2790.91	3.26	2346.67	3.26	50.16%
Code6	15	15	2780.75	0.00	1635.72	0.00	46.40%	2786.19	0.26	2398.07	0.26	54.47%
Code7	15	15	2850.97	0.89	1453.86	10.88	38.44%	2857.61	4.30	2563.34	4.30	48.41%
Code8	15	15	2766.82	0.00	1429.97	0.00	43.91%	2769.97	2.24	2239.73	2.24	52.42%
Code9	15	15	2827.96	0.00	1553.82	0.00	42.15%	2832.91	1.53	2418.83	1.54	50.52%
Code10	15	15	2835.45	0.00	1686.00	0.00	44.22%	2841.64	1.69	2466.41	1.69	51.91%
Code1	20	20	3156.98	64.77	1549.71	132.88	54.17%	3153.85	51.35	2654.00	40.05	66.84%
Code2	20	20	3476.13	65.08	1813.75	212.00	50.28%	3515.71	54.99	3224.91	47.04	64.07%
Code3	20	20	3267.67	48.30	1806.11	131.34	54.26%	3227.89	64.64	2754.22	57.30	63.97%
Code4	20	20	3386.16	85.40	1985.39	132.71	54.11%	3375.96	62.21	2817.20	60.38	62.39%
Code5	20	20	3418.97	59.38	2011.84	134.34	53.02%	3402.50	60.47	2987.49	56.47	62.38%
Code6	20	20	3436.56	82.77	1807.69	152.24	55.10%	3356.40	80.46	2963.24	66.61	66.40%
Code7	20	20	3537.77	61.17	1978.53	155.31	49.26%	3548.15	63.04	3316.42	45.49	61.30%
Code8	20	20	3378.65	60.82	1714.22	133.31	53.29%	3355.12	58.21	2851.08	54.63	64.94%
Code9	20	20	3540.28	59.59	1981.12	148.48	53.11%	3494.47	51.85	3124.28	36.54	63.67%
Code10	20	20	3482.95	78.33	2137.71	134.22	54.96%	3463.84	81.00	3135.54	65.65	64.54%

The model for the follower's problem for the proportional unessential customer behavior, especially FMAX, is still hard to solve so we again only compute greedy values for some of the instances, denoted by parentheses in Table 4.22. For these instances we do not state the market saturation and the objective values and standard deviations of the follower because we do not have exact results. However, compared to essential demands we were able to solve the follower's model for more instances and therefore get accurate results in more cases.

In Table 4.23 we see the results of partially binary customer behavior with unessential demands. The results show that for this scenario the results are very stable because the objective values have a very low standard deviation in many instances. The most interesting observation in this table is that, in contrast to the essential cases, in most instances the leader objective value (and the market saturation) is higher than in the proportional scenario. The reason for this behavior is that in the partially binary scenario more demand is satisfied by nearer facilities and therefore the total satisfied demand also

Table 4.21: Results of binary customer behavior with unessential demand for instances with $r = p = 20$ with an increased time limit of 1800s.

			LMIN					FMAX				
Instance	r	p	$\overline{obj}^{\,l}$	sd	$\overline{obj}^{\,f}$	sd	sat.	$\overline{obj}^{\,l}$	sd	$\overline{obj}^{\,f}$	sd	sat.
Code1	20	20	3379.95	0.00	1607.54	0.00	57.40%	3382.53	0.00	2442.62	0.00	67.04%
Code2	20	20	3714.01	0.00	1407.34	0.00	48.68%	3717.09	0.00	3034.83	0.00	64.18%
Code3	20	20	3409.31	0.00	1749.61	0.00	55.17%	3412.34	0.00	2578.23	0.00	64.06%
Code4	20	20	3582.97	0.00	1849.46	0.00	54.72%	3587.58	0.00	2611.09	0.00	62.44%
Code5	20	20	3621.48	0.00	2053.21	0.00	55.40%	3626.47	0.00	2773.11	0.00	62.48%
Code6	20	20	3639.92	0.00	1687.07	0.00	55.97%	3643.62	0.00	2706.55	0.00	66.72%
Code7	20	20	3739.66	0.00	1956.10	0.00	50.86%	3750.13	0.00	3132.62	0.00	61.46%
Code8	20	20	3581.40	0.00	1517.67	0.00	53.35%	3586.28	0.00	2635.38	0.00	65.10%
Code9	20	20	3689.02	0.00	1983.98	0.00	54.57%	3693.16	0.00	2949.16	0.00	63.89%
Code10	20	20	3712.01	0.00	1744.49	0.00	53.36%	3716.74	0.00	2908.06	0.00	64.78%

increases.

From the results we conclude that in general the FMAX strategy is better because significantly more demand can be satisfied and the follower increases his profit, too. However, if the follower wants to lower the turnover of the leader by all means the LMIN strategy might be useful. Compared to the essential demand cases we showed that while the complexity of the models for the follower's problem of the binary behavior increases, the complexity of the other two scenarios decreases and we got accurate results for more instances.

Table 4.22: Results of proportional customer behavior with unessential demand.

		LMIN					FMAX				
r	p	$\overline{obj}^{\,l}$	sd	$\overline{obj}^{\,f}$	sd	sat.	$\overline{obj}^{\,l}$	sd	$\overline{obj}^{\,f}$	sd	sat.
2	2	19.818	0.00	19.818	0.00	7.10%	30.460	0.00	29.331	0.00	10.72%
2	3	32.045	0.00	18.100	0.00	8.99%	42.091	0.00	26.735	0.00	12.33%
2	4	42.586	0.00	16.047	0.00	10.51%	52.266	0.00	25.160	0.00	13.88%
2	5	52.538	0.00	15.569	0.00	12.21%	61.266	0.00	23.528	0.00	15.20%
2	6	61.468	0.00	14.990	0.00	13.70%	70.285	0.00	22.218	0.00	16.58%
2	7	69.757	0.00	14.551	0.00	15.11%	78.613	0.00	20.634	0.00	17.79%
2	8	77.767	0.00	14.143	0.00	16.47%	85.893	0.00	19.610	0.00	18.91%
2	9	84.962	0.00	13.604	0.00	17.66%	91.963	0.00	19.118	0.00	19.91%
2	10	91.552	0.00	13.027	0.00	18.74%	98.666	0.03	18.557	0.07	21.01%
3	2	17.133	0.00	23.343	0.00	7.25%	28.049	0.00	40.476	0.00	12.28%
3	3	25.262	0.00	25.262	0.00	9.05%	39.386	0.00	38.039	0.00	13.88%
3	4	35.786	0.02	23.610	0.32	10.64%	49.724	0.03	35.434	0.03	15.26%
3	5	45.888	0.00	22.267	0.00	12.21%	59.172	0.00	33.332	0.00	16.58%
3	6	54.939	0.00	21.571	0.00	13.71%	67.564	0.00	31.682	0.00	17.79%
3	7	63.250	0.00	21.026	0.00	15.10%	74.954	0.00	29.902	0.00	18.79%
3	8	71.341	0.08	20.260	0.10	16.42%	82.241	0.00	28.839	0.00	19.91%
3	9	78.551	0.20	19.595	0.08	17.59%	89.195	0.07	28.040	0.13	21.01%
3	10	85.244	0.19	18.998	0.08	18.68%	95.494	0.39	27.384	0.21	22.02%
4	2	15.120	0.00	33.323	0.00	8.68%	(26.501)	0.00	–	–	–
4	3	22.935	0.00	28.380	0.00	9.20%	(37.586)	0.00	–	–	–
4	4	29.895	0.00	29.895	0.00	10.72%	47.448	0.00	45.056	0.00	16.58%
4	5	39.870	0.00	28.417	0.00	12.24%	56.606	0.00	42.641	0.00	17.79%
4	6	49.085	0.00	27.466	0.00	13.72%	64.863	0.13	40.647	0.14	18.91%
4	7	57.464	0.03	26.822	0.03	15.10%	72.384	0.03	39.126	0.16	19.98%
4	8	65.234	0.34	25.972	0.25	16.35%	79.469	0.25	37.534	0.30	20.97%
4	9	72.388	0.54	25.180	0.23	17.49%	86.197	0.53	36.634	0.27	22.01%
4	10	78.244	0.84	24.138	0.38	18.35%	91.158	1.06	35.784	0.33	22.75%
5	2	(13.856)	0.00	–	–	–	(25.474)	0.00	–	–	–
5	3	20.973	0.00	35.673	0.00	10.15%	(36.173)	0.00	–	–	–
5	4	27.804	0.00	32.685	0.00	10.84%	(45.780)	0.00	–	–	–
5	5	34.449	0.00	34.449	0.00	12.35%	(54.150)	0.00	–	–	–
5	6	43.715	0.01	33.071	0.02	13.76%	62.393*	0.05	49.228*	0.11	20.00%
5	7	52.077	0.13	32.331	0.17	15.13%	70.206	0.25	47.418	0.26	21.08%
5	8	59.373	0.77	31.206	0.36	16.23%	76.770	0.51	46.073	0.41	22.01%
5	9	65.956	1.14	30.357	0.30	17.26%	82.630	0.68	44.856	0.36	22.85%
5	10	70.972	2.02	29.086	0.38	17.93%	86.950	1.85	43.642	0.38	23.40%

*Not all of the 30 runs completed in the time limit

Table 4.23: Results of partially binary customer behavior with unessential demand.

		LMIN					FMAX				
r	p	$\overline{obj}^{\,\mathrm{l}}$	sd	$\overline{obj}^{\,\mathrm{f}}$	sd	sat.	$\overline{obj}^{\,\mathrm{l}}$	sd	$\overline{obj}^{\,\mathrm{f}}$	sd	sat.
2	2	21.288	0.00	21.288	0.00	7.63%	34.168	0.00	31.450	0.00	11.76%
2	3	35.324	0.00	20.960	0.00	10.09%	47.015	0.00	30.165	0.00	13.83%
2	4	48.015	0.00	21.055	0.00	12.38%	59.276	0.00	29.247	0.00	15.86%
2	5	59.988	0.00	19.512	0.00	14.25%	71.004	0.00	27.864	0.00	17.72%
2	6	71.623	0.00	19.022	0.00	16.24%	82.540	0.00	27.079	0.00	19.64%
2	7	82.551	0.00	19.783	0.00	18.34%	93.484	0.00	25.939	0.00	21.40%
2	8	93.423	0.00	19.487	0.00	20.23%	103.987	0.30	25.514	0.65	23.21%
2	9	103.254	0.00	20.031	0.00	22.09%	114.234	0.00	25.246	0.00	25.00%
2	10	113.116	0.00	20.735	0.00	23.99%	122.132	1.36	24.473	0.38	26.27%
3	2	20.501	0.00	22.075	0.00	7.63%	32.744	0.00	44.798	0.00	13.90%
3	3	28.658	0.00	28.658	0.00	10.27%	45.339	0.00	43.072	0.00	15.84%
3	4	41.349	0.00	28.753	0.00	12.56%	57.582	0.00	41.117	0.00	17.69%
3	5	53.268	0.00	29.170	0.00	14.77%	69.171	0.00	39.802	0.00	19.53%
3	6	64.903	0.00	28.679	0.00	16.77%	80.790	0.00	38.398	0.00	21.36%
3	7	75.591	0.00	29.186	0.00	18.78%	91.440	0.00	37.178	0.00	23.05%
3	8	86.463	0.00	28.890	0.00	20.67%	101.820	0.00	36.985	0.00	24.88%
3	9	96.482	0.00	27.898	0.00	22.29%	111.967	0.02	36.416	0.20	26.59%
3	10	106.419	0.00	28.221	0.00	24.13%	122.981	0.31	35.902	0.04	28.47%
4	2	19.856	0.00	22.720	0.00	7.63%	31.007	0.00	57.367	0.00	15.84%
4	3	27.934	0.00	29.378	0.00	10.27%	44.102	0.00	55.005	0.00	17.76%
4	4	35.294	0.00	35.294	0.00	12.65%	56.457	0.00	53.602	0.00	19.72%
4	5	47.213	0.00	35.711	0.00	14.86%	67.858	0.00	51.352	0.00	21.36%
4	6	58.848	0.00	35.220	0.00	16.86%	79.037	0.00	49.204	0.00	22.98%
4	7	69.629	0.00	35.515	0.00	18.84%	89.714	0.00	48.228	0.00	24.72%
4	8	79.507	0.00	35.948	0.00	20.69%	100.461	0.00	47.720	0.00	26.56%
4	9	90.337	0.00	34.987	0.00	22.46%	110.602	0.03	46.858	0.16	28.22%
4	10	100.274	0.00	35.311	0.00	24.30%	120.672	0.07	46.275	0.33	29.92%
5	2	19.241	0.00	20.779	0.00	7.17%	30.831	0.00	68.490	0.00	17.80%
5	3	27.318	0.00	27.420	0.00	9.81%	43.493	0.00	66.877	0.00	19.78%
5	4	34.570	0.00	36.014	0.00	12.65%	55.232	0.00	64.410	0.00	21.44%
5	5	41.462	0.00	41.462	0.00	14.86%	66.663	0.00	61.991	0.00	23.06%
5	6	53.097	0.00	40.971	0.00	16.86%	77.940	0.00	59.831	0.00	24.69%
5	7	63.878	0.00	41.266	0.00	18.84%	88.616	0.00	58.855	0.00	26.43%
5	8	74.554	0.00	40.796	0.00	20.67%	98.942	0.03	58.530	0.12	28.22%
5	9	84.363	0.00	41.222	0.00	22.51%	109.581	0.00	57.526	0.00	29.95%
5	10	94.297	0.00	41.528	0.00	24.34%	119.869	0.00	56.054	0.00	31.53%

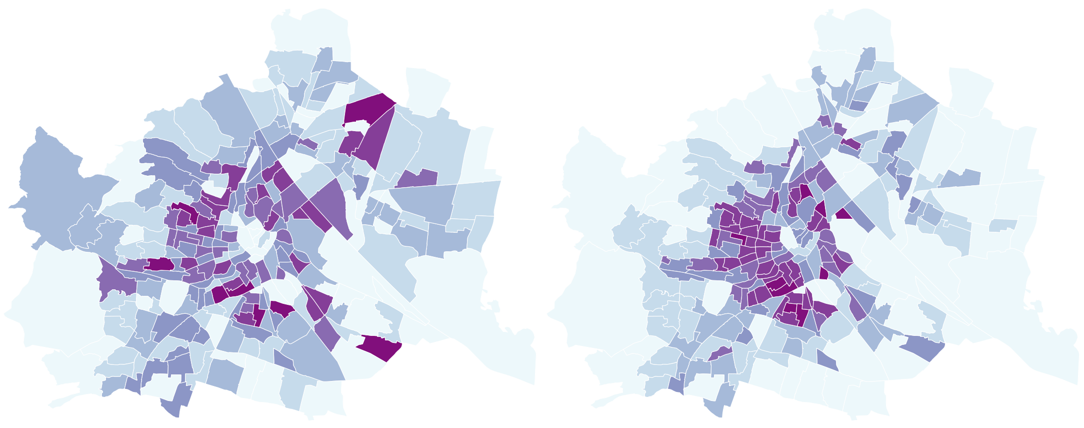


Figure 4.4: Registration districts of Vienna: absolute population and density.

4.9.5 Case Study of Vienna, Austria

For evaluating the algorithm under realistic conditions, we further perform a case study on Vienna, Austria using real world demographic data from Stadt Wien[3] from 2014. In our hypothetical scenario, a hypermarket chain wants to access the Viennese market, with the knowledge that a rival company has the same intention in the near future.

Vienna has a total population of 1,716,635 (year 2014) and consists of 23 districts. In order to achieve a reasonable level of detail for the strategic planning, we consider the further subdivision into 250 registration districts as planning cells. We assume that the demand of each such district is proportional to its population. In average these cells have a size of around 1.5 km^2 where those in the inner districts are smaller (0.2 km^2 – 0.8 km^2) and the peripheral ones are larger (up to 26 km^2). Though the latter cells would be too large for a detailed planning, they are rather unproblematic since they cover large weakly populated areas and therefore are less interesting for the case study anyway.

Figure 4.4 shows the 250 registration districts of Vienna and the population distribution. The left figure shows the absolute population and the right figure shows the density. Cells with darker and more purple colors indicate a higher population or a higher density.

The leader has to decide in which cells he opens hypermarket stores. Recall that the attractiveness of a store is determined by the distance to the potential customers, i.e., $v_{ij} = \frac{a_{ij}}{(d_{ij})^\beta}$ with $a_{ij} = 10,000$ for $i \in I, j \in J$ and $\beta = 2$. For the unessential cases we set the parameter b of the demand reduction function to 10,000. We distinguish between two situations for approximating the distance:

- On the cell where a store is opened, we approximate the cell's shape by a circle and take half of its radius as average distance for the inhabitants. This includes the assumption that within the cell, the store is located at a promising location.

[3]https://www.data.gv.at/auftritte/?organisation=stadt-wien

- On the other cells, we use the distances between the geometric centers[4] to that of the store's cell as approximation.

For the computational experiments we investigate all six scenarios for the customer behavior: binary essential, binary unessential, proportional essential, proportional unessential, partially binary. While some of these scenarios are more suited than others for the hypermarket scenario, we want to provide evaluations for all of them. Also, in a real world scenario, a combination of these elementary behaviors is most likely the happen.

In our case study we assume that the leader knows that the follower has a budget to open five stores in Vienna, i.e., $r = 5$. To compete over the market share, the leader considers different number of stores and their optimal placement.

For obtaining the computational results, we used the same parameter setting as described in Section 4.9, except the time limit, which is increased to 1,800 CPU seconds for the evolutionary algorithm and 86,400 CPU seconds for solving the final model because the instance is larger.

Tables 4.24, 4.25, and 4.26 show the numerical results of the tests for fixed $r = 5$ and different values for p for all scenarios. The values in the tables under column *ms* are the average market shares over 30 runs for the leader and the follower, respectively, expressed relative to the total demand of the market (Vienna's population). Next to the market shares also the corresponding standard deviations (sd) and the median run-times in seconds (t) are given. The results are shown for each scenario separately and Table 4.24 shows the results for binary customer behavior, Table 4.25 for proportional customer behavior, and Table 4.26 for the partially binary customer behavior. The values for the proportional essential case are listed in parentheses indicating that CPLEX could not solve the model within the time limit and the values being therefore only upper bounds on the leader's market share. Bold values indicate the values for p for which the leader could achieve a greater market share than the follower. We have to keep in mind, however, that the presented solution method solves the problem only heuristically and therefore the values for the leader's market share represent only lower bounds to the actual optimal value.

For the essential demand model we see that the leader's market share is naturally steadily increasing with increasing values of p. We also see that if $r = p$ in this demand model the follower has always a greater market share. It is clear that for proportional and partially binary customer behavior the leader can have at most 50% market share because the follower can choose the same locations as the leader and the demand then always splits equally between them. For the binary customer behavior, however, the follower's advantage of knowing the locations of the leader outweighs the leader's advantage of choosing his locations first.

When looking at the results for the unessential demand model we observe that the total market share of the leader and the follower is much lower than for the essential demand model and always less than 3% for each individual player. This is due to the assumption that attractiveness of a facility decreases quadratically with respect to the

[4] `http://www.wu.ac.at/inst/iir/datarchive/dist_zbez.html`

Table 4.24: Results for *binary* customer behavior, $r = 5$, and different values for p.

	Leader									Follower	
	Essential			Unessential LMIN			Unessential FMAX			LMIN	FMAX
p	ms [%]	sd [%]	t [s]	ms [%]	sd [%]	t [s]	ms [%]	sd [%]	t [s]	ms [%]	ms [%]
2	5.35	0.00	289	0.43	0.04	1800	0.56	0.05	1839	1.09	1.68
3	23.17	0.00	223	0.59	0.06	1800	0.75	0.06	1859	1.00	1.67
4	37.37	0.00	111	0.76	0.07	1800	0.91	0.08	1884	1.13	1.64
5	46.10	0.25	133	0.92	0.11	1800	1.04	0.08	1925	1.12	1.65
6	**53.97**	0.11	121	1.06	0.08	1800	1.21	0.09	1939	1.09	1.63
7	**59.77**	0.90	123	**1.19**	0.10	1800	1.34	0.08	1964	1.13	1.61
8	**63.08**	0.52	136	**1.33**	0.08	1800	1.48	0.09	1963	1.11	1.59
9	**66.24**	0.55	185	**1.49**	0.11	1800	**1.61**	0.10	1986	1.13	1.59
10	**69.10**	0.61	216	**1.60**	0.10	1800	**1.76**	0.12	1996	1.13	1.58

Table 4.25: Results for *proportional* customer behavior, $r = 5$, and different values for p.

	Leader									Follower	
	Essential			Unessential LMIN			Unessential FMAX			LMIN	FMAX
p	ms [%]	sd [%]	t [s]	ms [%]	sd [%]	t [s]	ms [%]	sd [%]	t [s]	ms [%]	ms [%]
2	(27.54)	0.21	88200	0.31	0.00	12463	0.67	0.00	3435	0.85	1.47
3	(37.05)	0.21	88200	0.43	0.02	4241	0.87	0.05	3899	0.80	1.44
4	(43.92)	0.63	88200	0.55	0.03	3302	1.04	0.07	3866	0.73	1.41
5	(49.23)	0.58	88200	**0.67**	0.04	3170	1.21	0.07	3758	0.67	1.38
6	**(53.51)**	0.75	88200	**0.82**	0.05	3122	1.36	0.09	3988	0.67	1.37
7	**(57.25)**	0.58	88200	**0.96**	0.08	3292	**1.48**	0.10	4040	0.68	1.35
8	**(60.37)**	0.69	88200	**1.09**	0.08	3246	**1.59**	0.08	3800	0.66	1.35
9	**(62.57)**	0.59	88200	**1.22**	0.09	3075	**1.69**	0.10	4138	0.65	1.32
10	**(64.51)**	0.50	88200	**1.33**	0.10	2979	**1.84**	0.12	4292	0.65	1.29

Table 4.26: Results for *partially binary* customer behavior, $r = 5$, and different values for p.

	Leader									Follower	
	Essential			Unessential LMIN			Unessential FMAX			LMIN	FMAX
p	ms [%]	sd [%]	t [s]	ms [%]	sd [%]	t [s]	ms [%]	sd [%]	t [s]	ms [%]	ms [%]
2	33.88	0.00	33527	0.36	0.00	6318	0.71	0.00	2271	0.35	1.53
3	40.91	0.00	22428	0.54	0.00	6059	1.03	0.00	2683	0.55	1.44
4	45.32	0.36	19564	**0.72**	0.00	4108	1.32	0.00	2567	0.70	1.37
5	49.26	0.60	17914	**0.89**	0.00	4829	**1.56**	0.00	2688	0.89	1.34
6	**52.35**	0.44	15280	**1.16**	0.00	3365	**1.82**	0.01	2689	0.88	1.31
7	**55.18**	0.49	14930	**1.41**	0.00	3260	**2.04**	0.02	2937	0.88	1.28
8	**57.34**	0.61	12975	**1.64**	0.02	3304	**2.24**	0.04	2972	0.87	1.26
9	**59.25**	0.60	12546	**1.83**	0.04	3319	**2.45**	0.04	2932	0.88	1.26
10	**61.14**	0.58	11514	**1.99**	0.04	3251	**2.61**	0.07	2656	0.84	1.27

distance. In the LMIN scenarios the follower can always achieve his goal to lower the market share of the leader, as it is always lower than in the FMAX scenarios for the same value of p. However, the follower's market share is also always significantly lower in the LMIN cases and therefore this behavior is not beneficial for the overall penetration of the market.

The tables further show that for both LMIN and FMAX in the partially binary behavior scenario the leader needs the least number of facilities to have a greater market share than the follower and in the binary behavior he needs the most. Especially in the partially binary LMIN case the leader needs only four stores, which means that the advantage of being the first in the market is larger in this case.

Regarding the variance of the results over the 30 runs we observed that the standard deviation increases with increasing p but is always less than 0.91 for the essential demand model and less than 0.13 for unessential demand. In 17 cases for low values of p the standard deviation was even zero which means that the algorithm stopped with a solution of the same quality in each run. Overall, the MA therefore is quite robust.

The run-time of the tests differ in each scenario, and in all but the binary cases, in which solving the model needs less than one second, the run-time of exactly solving the follower's subproblem model for the final solution is significantly high and in the most difficult cases even dominates the total run-time of the MA. In general the essential models are harder to solve than the unessential models. We conclude from Table 4.25 and 4.26 that the models for the proportional cases are the hardest to solve followed by the models for the partially binary cases. Also, for proportional and partially binary customer behavior the LMIN scenario needs more time to solve than the FMAX scenario. While the run-times of the FMAX scenarios do not differ much for different values of p, in the LMIN cases the run-time tends to decrease with increasing p.

To get a better understanding of the found solutions, Figures 4.5, 4.6 and 4.7 show best found solutions for $r = p = 5$ within a map of Vienna for different customer behaviors and demand models. For the unessential demand always the FMAX variant is illustrated. In these figures the leader is marked with blue color and the follower with red and the actual chosen locations are the dark districts. If the leader and the follower choose the same planning cell for their locations, it is marked with purple. The color of the remaining districts indicate the amount of demand that is captured by the leader and the follower, respectively. The more blue an area is, the more demand is satisfied by the leader. The same holds for the follower with red. Purple areas are served by both the leader and the follower.

In Figure 4.5 left we see that in case of a binary behavior, each cell is either served by the leader or the follower. All cells are roughly equally divided between both competitors. We also see that in the lower left area, the follower is able to significantly reduce the influence of a leader's facility by placing two facilities next to it. In the unessential case on the right, there are large white areas where demand is hardly fulfilled because of their large distances to the facilities.

In Figures 4.6 and 4.7 we see that basically all districts are served by both leader and follower facilities. In the essential model, especially for the proportional behavior in

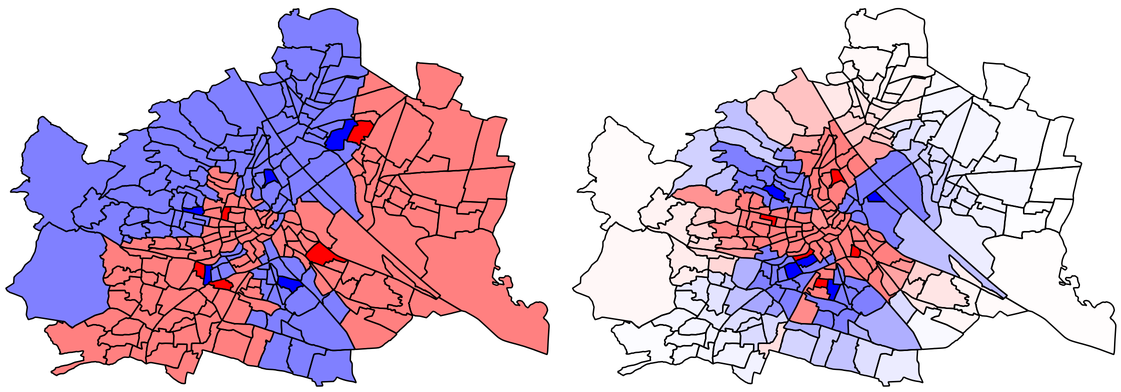


Figure 4.5: Best found solution for *binary* customer behavior and essential (left) and unessential (right) demand models with $r = p = 5$.

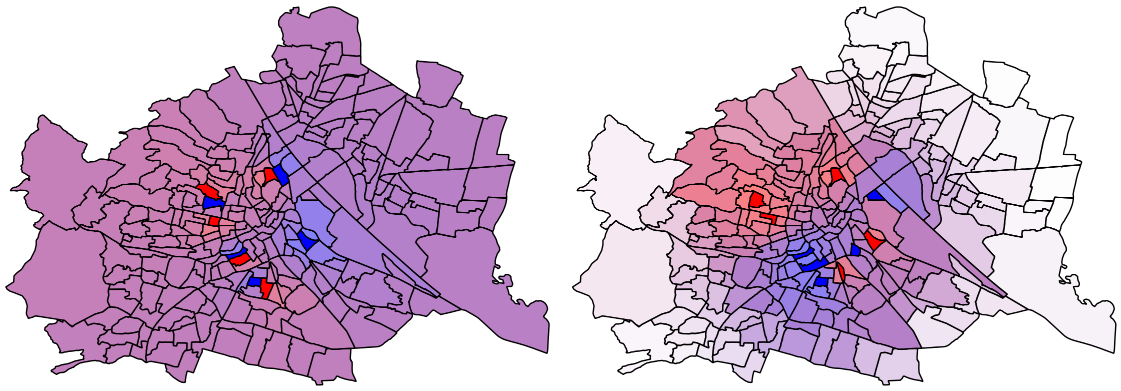


Figure 4.6: Best found solution for *proportional* customer behavior and essential (left) and unessential (right) demand models with $r = p = 5$.

Figure 4.6, the facilities are placed very centrally since each customer has a probability to visit all facilities. For the partially binary behavior in Figure 4.7, the facilities are more spread out, but some profitable districts are occupied by both competitors.

Overall we observe that in the unessential demand models the facility locations tend to be more central than in the essential demand models, which corresponds to our intuition of choosing the profitable districts first. We also see in all six figures that some locations are often chosen regardless of the considered scenario. For essential demand two locations were chosen for two different customer behavior models and for unessential demand four locations are chosen twice. These indicate promising, robust choices which could also be used for real-world applications, where the actual customer behavior is not so easy to determine. For practical applications of these results one also has to decide on the demand model one wants to assume, which highly depends on the offered goods. While some goods like basic foods are obviously essential, other goods are not

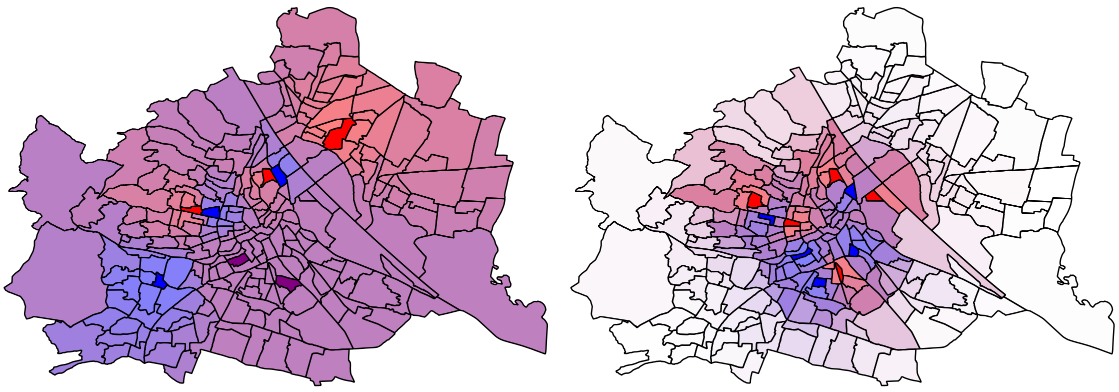


Figure 4.7: Best found solution for *partially binary* customer behavior and essential (left) and unessential (right) demand models with $r = p = 5$.

so easily categorized. Especially in our case, as hypermarkets offer a variety of essential and unessential goods, the demand model is not totally accurate. Nevertheless, the practitioner could simulate all kinds of different customer behavior and demand models and make a decision based on the results of all these scenarios.

4.10 Conclusions

This chapter presented the application of a complete trie-based solution archive to a genetic algorithm to solve variants of competitive facility location problems. The employed local search procedure led to a significant efficiency gain. Several ways of combining the local search with the solution archive were investigated, and the reduced neighborhood in combination with a tabu search was identified to work best in practice. Different solution evaluation methods were considered for each customer scenario and for the binary cases an effective way to combine them was found, which led to the multi-level evaluation scheme. For all customer behavior scenarios and demand models bi-level mixed integer programming models are presented.

Extensive tests showed that especially the solution archive had a significant positive effect on the quality of the final solutions. In combination with the other developed advanced techniques the overall algorithm showed to perform well, especially on Euclidean instances. For many of the commonly used instances it is able to exceed previous state-of-the-art heuristic approaches and also scales well to larger instances that cannot be solved with today's exact methods anymore. The conducted case study showed the practical applicability of this approach and several promising locations could be identified, some of them turned out to be robust choices regardless of the considered scenario.

Future research directions could be the development of a better approximation of the leader's objective value, e.g., by extending our greedy algorithms with a local search. When using a more elaborate solution evaluation we have obviously a tradeoff between accuracy and run-time. This could be especially useful for the customer scenarios other than binary and non-Euclidean instances. It would also be interesting to extend our models for different customer behavior to more realistic scenarios by taking opening costs and design of facilities into account to be able to maximize not only the turnover of the leader but also the profit.

CHAPTER 5

Generalized Vehicle Routing Problem with Stochastic Demands

The second type of problem considered in this thesis is the generalized vehicle routing problem with stochastic demands (GVRPSD). This NP-hard problem combines the clustering aspect of the generalized vehicle routing problem with the uncertainty aspect of the vehicle routing problem with stochastic demands. We consider the preventive restocking strategy which is substantially harder than the standard restocking strategy used by the majority of the published articles for the stochastic vehicle routing problem. Using this strategy the vehicle can make a return trip to the depot even before an actual stockout occurs and therefore save travel time. As this particular problem has not been covered in the literature yet, first an exact MILP-based solution algorithm is presented, which is based on the integer L-shaped method which is itself based on decomposition and branch-and-cut. This decomposition is also further used for two metaheuristics, which have also been developed to tackle larger instances. While the upper level problem is used as search space for the exact and heuristic algorithms, the lower level problem of computing the exact restocking costs is based on dynamic programming. The first metaheuristic is a VNS using three different neighborhood structures and a multi-level evaluation scheme (ML-ES) to reduce the overall time needed for solution evaluations. This ML-ES is also used within the second metaheuristic, an evolutionary algorithm, which is enhanced by a complete trie-based solution archive based on the methods described in Chapter 3. The tree structure of the SA is further exploited to compute lower bounds on the nodes to cut off parts of the solution space which evidently do not contain good solutions. Using this bounding procedure leads to optimal solutions for smaller instances and could also improve the results for medium to large instances.

First, an introduction to the problem is given in Section 5.1 and a formal problem definition is introduced in Section 5.2. Then, Section 5.3 gives an overview of the related literature. The algorithms are described in Sections 5.4, 5.5, and 5.6, starting from the exact algorithm, after which the VNS is explained and finally the GA is presented. Computational results and comparisons of all the developed methods are shown in Section 5.7 and conclusions are drawn in Section 5.8.

5.1 Introduction

Vehicle Routing Problems (VRPs) are among the most important and widely studied transportation and logistics problems in the field of combinatorial optimization. In the classical variants a set of delivery or pick-up routes for a capacity constrained fleet of vehicles starting from a central depot has to be designed in order to satisfy customers' demands. Here we consider two generalizations of this basic problem:

- In some applications specific delivery locations are not of importance but requested goods can be brought to any delivery points in the surrounding areas of the customers. In practice, the redistribution within each area is then carried out by the customers. Practical examples of this generalization are disaster relief operations to distribute medical staff or equipment to damaged sites [2] and the distribution of goods over sea to a number of customers in an archipelago, where each island has several ports from which the actual point of delivery can be chosen [62]. Ghiani and Improta [62] originally introduced this VRP variant and named it Generalized Vehicle Routing Problem (GVRP).
- The actual demand of the customers may not be precisely known in advance, resulting in the vehicle routing problem with stochastic demands (VRPSD). This situation can occur in urban waste collection, where garbage trucks need to collect the waste from certain collection points to deliver it to a central landfill site [129], or in the delivery of petrol to petrol stations [14]. In practical applications the demands are usually not uniformly random but specific probability distributions can be deduced from historical data.

The generalized vehicle routing problem with stochastic demands considers both above extensions at once. A cluster of delivery points is given for each customer, as well as a stochastic demand, which is modeled by a random variable with a certain probability distribution. The aim of this problem is to plan so-called *a-priori* routes with minimum expected length or costs, respectively.

An important characteristic of stochastic routing problems is that the planned routes may not be followed as planned. Since the demand of the visited clusters may be higher than expected the vehicle may be depleted before the tour is finished. Then, a recourse action must be executed in order to satisfy the remaining demand of the tour. The most widely used recourse action in the literature, which we call *standard* restocking henceforth, sends the vehicle back to the depot whenever it is not able to service a current customer, e.g., [11, 86, 58, 70, 114]. However, this strategy is sub-optimal with respect to the expected length of the routes as shown by Yang et al. [129]. A recourse action which may result in shorter routes is the *preventive* restocking strategy which allows return trips to the depot before the vehicle is fully depleted. Although expected costs can frequently be significantly lower by employing such a strategy, the computational overhead for computing them is substantial. A dynamic programming algorithm can be used for this purpose. In this work we consider such a preventive restocking policy and a relatively efficient computation of the expected costs is explained in Section 5.4.2.

5.2 Problem Definition

The GVRPSD is defined on a complete undirected graph $G = (V, E)$ with node set V and edge set E. The nodes are partitioned into disjoint clusters $C = \{C_0, C_1, \ldots, C_m\}$, $C_i \subseteq V$, $\forall i = 0, \ldots, m$, such that $C_0 \cup C_1 \cup \cdots \cup C_m = V$. Each edge $(i, j) \in E$ has a distance or cost value $d_{ij} \geq 0$. Node v_0 represents the depot node and is the only node contained in C_0. Each other cluster C_j, $j = 1, \ldots, m$ has an associated stochastic demand ξ_j which is modeled as a random variable with a known discrete probability distribution and has r possible values $\xi_j^1, \ldots, \xi_j^r$. Thus, we know for each cluster C_j the probability mass function given by values p_{jk} for all $k = 0, \ldots, Q$ denoting the probability that cluster j has an actual demand of k. Furthermore, we are given one vehicle with a limited capacity Q. Situations where the demand exceeds the vehicle capacity are not considered, so we assume that $p_{jk} = 0$, $\forall j = 1, \ldots, m$, $\forall k > Q$. The goal is to find a tour starting from the depot which visits one node from each cluster exactly once and returns to the depot with minimum expected costs. During the route the clusters' actual demands, which depend on the realization of the random variables ξ_j, get revealed upon arrival and the load of the vehicle reduces by exactly these amounts. Intermediate visits of the depot are always allowed and become necessary when the vehicle cannot satisfy the demand of a cluster. Note that without further restrictions the planning of only one tour is sufficient because by employing the preventive restocking strategy the capacity constraints cannot be violated as the restocking trips are dynamically planned.

Figure 5.1 shows an example of a solution candidate for a small instance. In this example the vehicle capacity $Q = 10$, and all clusters except C_1 have a constant demand of 6 for cluster C_4 and 1 for the other clusters C_2, C_3, and C_5. Depending on the realization of ξ_1 a tour without an intermediate return to the depot could be planned (if $\xi_1 = 1$) or a restocking has to be performed (if $\xi_1 = 5$). However, as the actual demand only becomes known upon arrival at C_1 a restocking trip back to the depot would be needed with a high probability of 0.9. Therefore, as we use the *preventive* restocking strategy an anticipatory restocking trip from v_1 back to the depot v_0 is beneficial because its cost is significantly lower than the cost of the likely needed restocking trip from v_2.

5.3 Related Work

As the generalized vehicle routing problem with stochastic demands is a new variant of a VRP, there is not much specific literature available yet. When each generalization

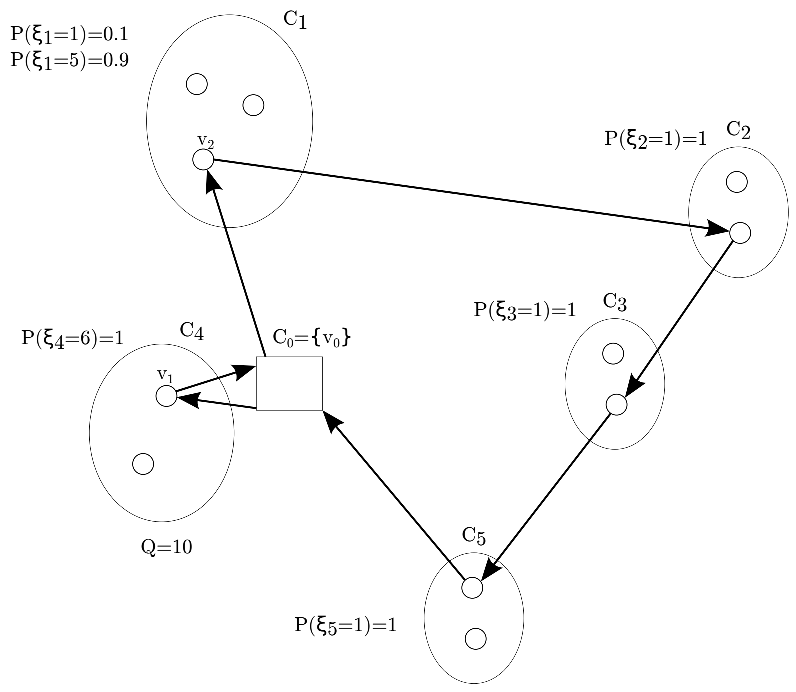


Figure 5.1: Example of a solution for an instance of the GVRPSD.

is considered separately, the literature for the GVRP and the VRPSD is richer. Since the introduction of the GVRP by Ghiani and Improta [62], several exact and heuristic methods have been proposed for solving the problem. Exact methods include the compact mixed integer programming formulations by Kara and Bektaş [77], with which they solved instances with up to 50 nodes and 25 clusters. More elaborate exact methods include branch-and-cut algorithms by Bektaş et al. [10] and Hà et al. [66], with the latter being based on a two-commodity flow model, and a column generation approach by Afsar et al. [2]. The latter presents also two heuristics based on a route-first, cluster-second approach, in which the split procedure is executed using an iterated local search. Other heuristic methods for the GVRP include a genetic algorithm [108], a variable neighborhood search [107], and a hybrid metaheuristic combining a greedy randomized adaptive search procedure with an evolutionary local search [66]. The largest instance which is tackled by all of these algorithms contains 262 vertices and 131 clusters.

In the area of vehicle routing problems with stochastic demands most works use the standard restocking approach. There is much literature for exact methods, e.g., [57, 70, 87, 34, 76] and heuristic approaches, e.g., [58, 114, 64]. Current state-of-the-art exact solution approaches based on the integer L-shaped method are able to solve instances with up to 100 customers and 2 vehicles [87] or instances with up to 4 vehicles but only 60 customers [76]. Christiansen and Lysgaard [34] complement the L-shaped method by introducing a branch-and-price algorithm for solving the VRPSD and are able to solve instances with up to 60 customers and 16 vehicles with tighter capacity constraints.

The situation changes when preventive restocking is considered. To the best of our knowledge there is no exact algorithm described for the VRPSD with preventive restocking. However, the L-shaped method for the GVRPSD with preventive restocking presented in Section 5.4 can also be used to solve the non-generalized version. Several authors developed metaheuristics. Yang et al. [129] were the first to introduce the preventive restocking strategy and a dynamic programming (DP) procedure to compute the expected costs of a tour using this strategy. The authors also describe two heuristics for solving a variant of the VRPSD in which the maximum planned expected route length is limited. Based on that DP [14] developed several metaheuristics and approximative algorithms for move evaluations in the Or-opt local search neighborhood structure. Marinakis et al. presented a particle swarm optimization (PSO) algorithm [91], extended it with a combinatorial expanding neighborhood topology (CENTPSO) [92] and a memetic differential evolution algorithm [94] for solving the VRPSD with preventive restocking. The latest heuristic solution method is a glowworm swarm optimization which makes use of path relinking and a variable neighborhood search [90] which, together with a hybrid clonal selection algorithm [93] and the CENTPSO, constitute the current state-of-the-art algorithms.

5.4 An Integer L-shaped Method for the GVRPSD

First, we approach the GVRPSD by solving it exactly using the integer L-shaped method and a formulation for the GTSP by Fischetti et al. [53]. This method has been introduced by Van Slyke and Wets [127] and is a well-known algorithm in the field of stochastic programming. It is based on a Benders decomposition approach in which subproblems compute the restocking costs [86] for which optimality cuts are then iteratively added to the initially relaxed master problem. The L-shaped method has already been successfully applied to several stochastic optimization problems, e.g,. stochastic vehicle routing problems [86, 70, 76].

5.4.1 Mathematical Model

To model the GVRPSD we define binary decision variables $x_e, \forall e \in E$, which are set to one if edge e is used in the solution and zero otherwise. We further use binary variables $y_v, \forall v \in V$ to determine which nodes are visited, i.e., y_v is set to one if v is visited and zero otherwise. Finally, variable θ denotes a lower bound on the restocking costs. In a preprocessing step a global lower bound L of the expected restocking costs is computed as follows. We calculate the rounded expected number of restocks $E[nr] = \left\lfloor \frac{\sum_{k=1}^{m}(E[\xi_k])}{Q} \right\rfloor$ and for each pair of nodes (i,j) the costs $s_{ij} = d_{(i,0)} + d_{(0,j)} - d_{(i,j)}$ for a preventive restock between them. Then, L is the sum of the $E[nr]$ smallest s_{ij} values. Function $\delta(S)$ denotes the edge-set of the cut $(S, V \setminus S)$, i.e., $\delta(S) = \{(i,j) \in E \mid i \in S, j \notin S\}, \forall S \subseteq V$. Our model for the master problem works as follows.

$$\min \sum_{e\in E} d_e x_e + \theta \tag{5.1}$$

$$\text{s.t.} \quad \sum_{e\in\delta(\{v\})} x_e = 2y_v \qquad \text{for} \quad v \in V \tag{5.2}$$

$$\sum_{v\in C_j} y_v = 1 \qquad \text{for} \quad j = 1,\dots,m \tag{5.3}$$

$$\sum_{e\in\delta(S)} x_e \geq 2 \qquad \forall S \subset C,\ 2 \leq |S| \leq m-2,\ C_0 \in S \tag{5.4}$$

$$\theta \geq L \tag{5.5}$$

$$x_e \in \{0,1\} \qquad \text{for} \quad e \in E \tag{5.6}$$

$$y_v \in \{0,1\} \qquad \text{for} \quad v \in V \tag{5.7}$$

$$(\theta_\pi - L)\left[\left(\sum_{(i,l)\in E_\pi} x_{il} - (m-2)\right)/2\right] + L \leq \theta \qquad \forall \pi \in \Pi \tag{5.8}$$

Objective function (5.1) minimizes the sum of the total travel costs and the additional restocking costs. Equations (5.2) ensure that each node which is used in the solution has exactly two outgoing edges. Constraints (5.3) guarantee that exactly one node from each cluster is visited. Eliminating subtours is done by classical cut-set inequalities (5.4). They are dynamically added within a branch-and-cut framework. The separation procedure is based on m maximum flow computations. Inequality (5.5) sets the value of the lower bound of the restocking costs to be at least the global lower bound L. Inequalities (5.8) are the specific optimality cuts setting the restocking costs accordingly. Set Π consists of all possible cluster permutations. The edge set E_π of a cluster permutation $\pi \in \Pi$ consists of all the edges between all consecutive clusters without cluster C_0.

The value of θ_π represents the recourse costs for permutation π and is computed in the subproblem via a dynamic programming procedure, which is described in the next section. This value is equal to θ for exactly the route corresponding to the cluster permutation π, because then $\sum_{(i,j)\in E_\pi} x_{ij} = m$. In all other feasible routes at least two edge variables in E_π are zero and therefore the lefthand side of the inequality does not exceed L.

5.4.2 Dynamic Programming for Computing the Restocking Costs

In this section the computation of the restocking costs using the preventive restocking strategy is described. As mentioned before a dynamic programming procedure is used to compute these additional costs exactly. A similar algorithm has already been used for the VRPSD [129, 14]. In the DP we define a recursive function $f_{ij}(q)$ for all $q = 0,\dots,Q,\ j = 0,\dots,m,\ i = 0,\dots,|\pi(j)|-1$ to be the expected remaining cost of the tour after servicing the i-th node of the cluster $\pi(j)$ with respect to the remaining capacity q. The auxiliary function $b_j(l)$ returns the l-th node of cluster at position j. To compute the total cost $c^*(\pi)$ of a cluster permutation π the following DP recursion is used:

$$f_{ij}(q) = \min\{f^p_{ij}(q), f^r_{ij}(q)\}$$

$$\begin{aligned} f^p_{ij}(q) = \min_{l=0,\dots,|\pi(j+1)|}\{d_{b_j(i),b_{j+1}(l)} \quad &+ \sum_{k=0}^{q} f_{l,j+1}(q-k)p_{j+1,k} \\ &+ \sum_{k=q+1}^{Q} 2d_{b_{j+1}(l),0} + f_{l,j+1}(q+Q-k)p_{j+1,k}\} \\ f^r_{ij}(q) = d_{b_j(i),0} + \min_{l=0,\dots,|\pi(j+1)|}\{d_{0,b_{j+1}(l)} \quad &+ \sum_{k=0}^{Q} f_{l,j+1}(Q-k)p_{j+1,k}\} \end{aligned}$$

$$\forall q = 0,\dots,Q,\; j = 0,\dots,m,\; i = 0,\dots,|\pi(j)|-1$$

and the boundary condition

$$f_{im}(q) = d_{b_m(i),0}, \quad \forall q = 0,\dots,Q,\; i = 0,\dots,|\pi(m)|-1$$

This DP computes for each node i and each vehicle load q if it is in the expected case more cost efficient to proceed directly to the next cluster with costs $f^p_{ij}(q)$ or to make a preventive restock with costs $f^r_{ij}(q)$. The total cost $c^*(\pi)$ is given by $f_{0,0}(Q)$.

5.4.3 Specific and General Optimality Cuts

When considering restocking costs the direction of the route is important so the actual cost of an undirected route stated as permutation π is given by

$$c^*(\pi^*) = \min\{c^*(\pi), c^*(\text{reverse}(\pi)\},$$

where reverse(π) represents the reversed order of the elements in π. Let $\hat{c}(\pi)$ be the travel costs of permutation π from solving the model without considering the restocking costs. Then the exact restocking costs θ_π are $c^*(\pi^*) - \hat{c}(\pi)$.

Each time the master problem is solved, the DP algorithm from Section 5.4.2 is executed and the respective specific optimality cut (5.8) is added to the model if not yet included. The procedure iterates until no further cuts are separated and thus an optimum overall solution has been found. A weakness of these specific optimality cuts is that they only apply to exactly one solution but in the next section we show how to generalize these cuts in order to increase the lower bound on the restocking costs more generally.

5.4.4 General Optimality Cuts

We generalize the specific optimality cuts from Section 5.4.3 by computing lower bounds on the restocking costs for partial routes. Therefore, we consider for a given permutation $\pi = \langle C_{\pi(0)}, C_{\pi(1)}, \dots, C_{\pi(m-1)}\rangle$, $m-2$ partial segments $\pi^h = \langle C_{\pi(h)}, C_{\pi(h+1)}, \dots, C_{\pi(m-1)}\rangle$,

$\forall h = 1, \dots, m-2$. Each segment implicitly starts and ends at the depot and we use a slightly modified version of the DP for computing a lower bound on the restocking costs θ_{π^h} for all solutions using this segment. For all θ_{π^h} values which are larger than zero the following general optimality cut is added to the model.

$$(\theta_{\pi^h} - L)\left[\left(\sum_{(i,l)\in E_{\pi^h}} x_{il} - (m-h-3)\right)/2\right] + L \leq \theta$$

Adding such a general optimality cut increases the lower bounds on the restocking costs for all cluster permutations π which contain π^h as partial segment. In a naive approach $2(m-2)$ executions of the DP algorithm would be needed to compute the restocking costs of all these segments. However, by using an incremented evaluation technique we can store the values of $f_{ij}(q)$ and use them for the next DP computation. More specifically, we start with the last segment π^{m-2} and calculate $f_{i,m-2}(q),\ \forall i \in \pi(m-2),\ q = 0, \dots, Q$. This value is used for the next iteration where segment π^{h-3} is considered and we only have to compute $f_{i,m-3}(q), \forall i \in \pi(m-3), q = 0, \dots, Q$ and use the previous values of $f_{i,m-2}(q)$. This method saves many unnecessary computations so that we can add up to $m-2$ general optimality cuts without having to execute the whole DP algorithm each time.

The computational results of this integer L-shaped algorithm and the differences when only specific or both types of cuts are used will be shown in the computational study described in Section 5.7.

5.5 A Variable Neighborhood Search for the GVRPSD

Our first proposed metaheuristic for the GVRPSD, a VNS, follows the general variable neighborhood search scheme as described in [69]. The underlying variable neighborhood descent (VND) considers three neighborhood structures which are well-known for the TSP: *1-shift*, *2-opt*, and *Or-opt*. As a shaking procedure for diversification we perform k^2 moves in the k-th neighborhood with $k = 1, \dots, 3$.

Following the decomposition approach used by the exact algorithm described in Section 5.4, it is sufficient to plan one giant tour through all clusters. Therefore, we use a solution representation based on the sequence of clusters that are visited and the tour, and use the DP from Section 5.4.2 to compute the objective value of such a sequence. However, for our VNS such an expensive solution evaluation is inconvenient for larger instances with a large vehicle capacity. In Section 5.5.1 we describe a method to potentially reduce the run-time of the solution evaluation within the VNS framework, which can also be applied to other metaheuristics, e.g., our genetic algorithm in Section 5.6.

5.5.1 Multi-Level Evaluation Scheme

In this section we describe a multi-level evaluation scheme to iteratively estimate the exact objective value of a solution candidate with increasing accuracy until we either know that it cannot be better than the best solution found so far or we know its exact value. The basic idea is to scale down both the vehicle capacity and the probability distribution of the demand for each cluster accordingly. Since the time needed for the solution evaluation is quadratically dependent on Q, a large performance gain in terms of run-time is expected when Q is decreased.

In our ML-ES there are $\log_2 Q$ levels of approximation, where level 0 is the exact evaluation and $\log_2 Q$ is the roughest approximation level. Starting with level 0, increasing the level by one means to scale down the vehicle capacity Q and all demand distributions p_{jk} by a factor of two. We introduce a new vehicle capacity Q^i and new probabilities p^i_{jk} subject to level i, which are defined in the following way:

$$Q^0 = Q \tag{5.9}$$

$$p^0_{jk} = p_{jk} \qquad \forall j = 1, \dots, |C|,\ k = 0, \dots, Q \tag{5.10}$$

$$Q^i = \left\lceil \frac{Q^{i-1}}{2} \right\rceil \qquad \forall i = 1, \dots, \lceil \log_2 Q \rceil \tag{5.11}$$

$$p^i_{jk} = p^{i-1}_{j,2k} + p^{i-1}_{j,2k+1} \qquad \forall j = 1, \dots, |C|,\ k = 0, \dots, Q^i,\quad \forall i = 1, \dots, \lceil \log_2 Q \rceil \tag{5.12}$$

Figure 5.2 shows exemplarily for one cluster how the probability distribution for the demand changes at each level.

Not only is level $i \geq 1$ an approximation, but its objective value is also a lower bound for the objective value of the preceding level $i-1$, which we will show next.

Lemma 1. *With increasing level i the ratio of the scaled expected demand of each cluster to the vehicle capacity Q^i is non-increasing.*

Proof. We have to show for each cluster j and each demand $0 \leq k \leq Q$ that

$$\frac{\sum_{k=0}^{Q^i} k p^i_{jk}}{Q^i} \leq \frac{\sum_{k=0}^{Q^{i-1}} k p^{i-1}_{jk}}{Q^{i-1}}, \quad \forall i = 1, \dots, \lceil \log_2 Q \rceil$$

is valid. Suppose to the contrary that for one cluster j and one demand k the following holds:

$$\frac{k p^i_{jk}}{Q^i} = \frac{k(p^{i-1}_{j,2k} + p^{i-1}_{j,2k+1})}{\frac{Q^{i-1}}{2}} > \frac{2k p^{i-1}_{j,2k} + (2k+1) p^{i-1}_{j,2k+1}}{Q^{i-1}} = \frac{k p^{i-1}_{jk}}{Q^{i-1}}$$

$$2k p^{i-1}_{j,2k} + 2k p^{i-1}_{j,2k+1} > 2k p^{i-1}_{j,2k} + 2k p^{i-1}_{j,2k+1} + p^{i-1}_{j,2k+1}$$

$$0 > p^{i-1}_{j,2k+1}$$

Obviously, this is a contradiction because all probabilities must be non-negative. Therefore, as no $\frac{k p^i_{jk}}{Q^i}$ can be larger than $\frac{k p^{i-1}_{jk}}{Q^{i-1}}$ for any cluster j this also holds for the sum over all demands, which proves the Lemma. $\square$

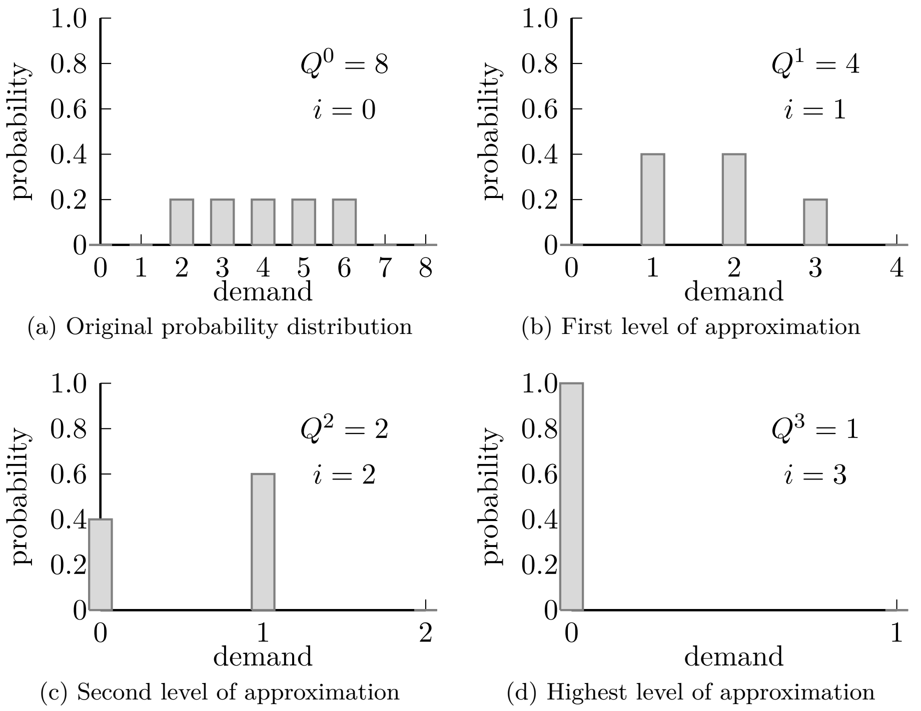


Figure 5.2: An exemplary demand probability distribution and its different levels of approximation.

Theorem 1. *Let $c^i(t)$ be the objective value of a tour t on approximation level i. For each tour t it holds that $c^i(t) \leq c^{i-1}(t), \forall i = 1, \ldots, \lceil \log_2 Q \rceil$.*

Proof. Due to Lemma 1 it follows that the total expected relative demand of all clusters on level i is smaller or equal to that of level $i - 1$. So we can possibly service more customers before a restocking is needed and therefore the resulting objective $c^i(t)$ value is a lower bound to the exact objective value $c^0(t)$ and to the objective value at the preceding level $c^{i-1}(t)$. □

Algorithm 5.1 describes our ML-ES in pseudocode, where $DP(t, i)$ executes the DP described above with the scaled vehicle capacity and probability distributions according to (5.9–5.12). Algorithm 5.1 returns either the exact objective value of t if $DP(t, 0)$ is executed or a lower bound to the exact value otherwise. In the latter case the solution candidate can immediately be discarded because we know that it cannot be better than the best solution found so far.

5.5.2 Initial Solution

For finding an initial solution for the VNS two types of construction heuristics are considered. The first, *farthest insertion*, is well-known for the classical traveling salesman

Algorithm 5.1: ML-ES(t, $bestObj$)

```
Input : tour $t$, objective value of best solution found so far $bestObj$
Output: exact or approximate objective value
1 $obj = 0$;
2 $i = \lceil \log_2 Q \rceil$;
3 while $obj < bestObj \wedge i \geq 0$ do
4     $obj = DP(t, i)$;
5     $i = i - 1$;
6 end
7 return $obj$;
```

problem and suited for Euclidean instances only. It builds iteratively a tour by starting at the depot cluster and inserting the cluster which is farthest away from the last inserted cluster at the best possible position. For that purpose we have to define distances between clusters, which is done by computing the geometric centers of clusters by taking the average of the x- and y-coordinates of its nodes. Then the distance between two clusters is the Euclidean distance between their centers.

An alternative but much more time-consuming method for finding a starting solution is solving the GTSP relaxation of the problem. From the solution of the GTSP relaxation we extract the cluster sequence which is then our initial solution. The GTSP is solved exactly by using a branch-and-cut algorithm with CPLEX and the E-GTSP formulation described by Fischetti et al. [54], which is also used as basis for the developed exact method 5.4.

5.5.3 Neighborhood Structures

Three types of neighborhood structures are used in the VND part, which are searched with a best improvement step function in the order they are described here.

- **1-shift:** A cluster is shifted to another position of the tour.
- **2-opt:** A subsequence of the tour is inverted.
- **Or-opt:** First two, then three consecutive clusters are shifted to another position of the tour. Note that Or-opt usually starts by shifting only one cluster in the tour but we covered this case by our first neighborhood structure and omit it here.

Like the results for the integer L-shaped method, the computational results and of the VNS and appropriate comparisons are shown in Section 5.7.

5.6 A Genetic Algorithm in Combination with a Solution Archive for Solving the GVRPSD

The second developed metaheuristic for solving the GVRPSD is a genetic algorithm with solution archive (GASA). The overall algorithmic framework, which also uses a VND procedure similar to the one described in Section 5.5, is depicted in Algorithm 5.2.

Algorithm 5.2: Genetic algorithm with solution archive (GASA)

```
1  begin
2     Initialize Population;
3     while unconsidered solutions remaining according to the solution archive
4        Select parent solutions $x_{P_1}$ and $x_{P_2}$;
5        Derive child $x_C$ from $x_{P_1}$ and $x_{P_2}$ using a crossover operator;
6        Perform mutation of $x_C$ with probability $p_{\text{mut}}$;
7        if $f(x_C) < \alpha f(x_{\text{best}})$ then
8           Improve $x_C$ by executing VND($x_C$);
9        else
10          if $x_C$ is not yet contained in the solution archive then
11             Insert $x_C$ into the solution archive;
12          else
13             Convert $x_C$;
14       Delete the worst individual of the population;
15       Add $x_C$ to the population;
16    end while
17    Return best found solution;
18 end
```

The genetic algorithm is a steady-state GA, which replaces in each iteration the worst solution of the current population with the newly created solution. The solution representation and evaluation function is the same as for the VNS described in Section 5.5 and ML-ES is also applied here. Within the GA a VND procedure is executed for promising solution candidates whose objective value, which is computed by the function $f(\cdot)$ (described in Section 5.4.2), is close to the best solution found so far, where closeness is defined by the parameter α. After the genetic operators produced the new solution candidate, it is either inserted into the solution archive or converted if the archive already contains the new solution. This step is skipped when VND was performed on this solution because then the insertion / conversion procedure is carried out within the VND. The algorithm terminates after a specific time limit T_{max}.

The individual components of GASA are described in the following section. Sections 5.6.1 and 5.6.2 address the framework of the GA which includes the initial population generation method and its operators. The VND is presented in Section 5.6.3 and the solution archive with its bounding extension is explained in Section 5.6.4.

5.6.1 Initial Population

The choice of the generation method for the initial population of this GA is important and the aim here is to include both diverse and high quality individuals. Therefore, three different methods for solution initialization are applied with specific particular purpose:

1. **High quality**
   To get one initial solution candidate of typically relatively high quality we solve the generalized travelling salesman problem (GTSP) with the given instance ignoring the demands as underlying graph. As for the VNS, the MILP model by [54], which is based on an undirected cut-set formulation, is solved with a branch-and-cut algorithm based on CPLEX. After an optimal solution to this model is obtained a VND is performed starting from this solution to obtain a typically even better initial solution candidate; see Section 5.6.3 for a detailed description of the VND. Note that due to the relatively high computational effort for solving the ILP, this solution generation method is aborted after 120 seconds with the best solution found so far. If no solution could be obtained within that time a randomly generated solution is used instead.

2. **Medium quality / medium diversity**
   For Euclidean instances the next $\lfloor \frac{P_{\text{size}}-1}{2} \rfloor$ initial solutions are generated by using a *farthest insertion* heuristic based on cluster distances as one variant described for the VNS in Section 5.5. We compute the distances between every pair of two clusters by taking the Euclidean distances between their geometric centers, which are obtained by taking the arithmetic mean of the Euclidean coordinates of their nodes. Then a starting cluster is chosen at random and the other clusters are iteratively inserted at the best possible position of the current tour by always taking the farthest, not yet inserted cluster from the last inserted one. Ties are broken randomly.

3. **Low quality / high diversity**
   The remaining $\lceil \frac{P_{\text{size}}-1}{2} \rceil$, or $P_{\text{size}} - 1$ for non-Euclidean instances, individuals are generated uniformly at random.

5.6.2 Genetic Operators

For selecting the crossover candidates a tournament selection is employed. The GA uses a cyclic crossover operator to generate one child solution out of two parent solutions (A and B). This operator takes a randomly chosen sub-tour of the parent A and successively appends clusters from parent B starting from the last node of the sub-tour, skipping any already considered clusters. For diversification a *swap*-mutation operator is developed which swaps two randomly chosen cluster positions. This move is repeated for *nMut* times, where *nMut* is a parameter of the algorithm.

5.6.3 Variable Neighborhood Descent

To intensify the search a variable neighborhood descent (VND) algorithm with four different neighborhood structures is used, where the first three are taken from the VNS of Section 5.5:

$\mathcal{N}_1$ *1-shift*: One cluster is shifted to another position.

$\mathcal{N}_2$ *2-opt*: A sequence of clusters between two positions is inversed.

$\mathcal{N}_3$ *Or-opt*: Two or three consecutive clusters are shifted to another position.

$\mathcal{N}_4$ *SAConv*: One solution conversion based on the solution archive is performed.

The VND is executed during the GA for each solution candidate x whose objective value is at most α times larger than the best solution found so far where α is an exogenous parameter. The fourth neighborhood structure is based on the solution archive, which is described in detail in Section 5.6.4. Having defined the neighborhood structures, the complete VND with the solution archive is shown in Algorithm 5.3 as pseudocode.

Algorithm 5.3: Variable neighborhood descent with solution archive

```
Input: Initial solution $x$
Output: Local optimal solution
begin
    $l \leftarrow 1$;
    repeat
        $x^* = x$;
        $f^* = f(x)$;
        for all $x' \in \mathcal{N}_l(x)$
            if $x'$ is already contained in the solution archive then
                if $SA_{\text{conv}}$ then Convert $x'$;
                else continue;
            else
                Insert $x'$ into the solution archive;
            if $f(x') < f^*$ then
                $f^* = f(x')$;
                $x^* = x'$;
        if $f^* < f(x)$ then
            $x \leftarrow x^*$;
            $l \leftarrow 1$;
        else
            $l \leftarrow l + 1$;
    until $l > l_{\max}$;
    return $x$;
end
```

The VND systematically searches the given neighborhoods and basically follows the standard procedure as described in Section 2.4.4 using a best improvement step function for $\mathcal{N}_1$ to $\mathcal{N}_3$. However, before each neighboring solution is evaluated it is checked if it is already contained in the solution archive. Depending on a binary parameter SA_{conv} this solution is either converted into a new solution or its evaluation is skipped and the search continues as if it is already contained in the archive. This parameter determines, similar to the local search with solution archive as described for the CFLP in Section 4.7, if the *reduced* or the *converted* neighborhood structure is used.

5.6.4 Solution Archive

An important part of the GA is the employed solution archive (SA). As shown in Algorithm 5.2 and 5.3 the SA is used in all parts of the algorithm and is attached to the GA after mutation and after each neighborhood move in the VND. It is expected that the performance of the GA can be significantly increased by employing a SA because we use a compact representation and the solution evaluations are costly. The next sections describe the particularities of the complete trie-based solution archive employed to the GVRPSD.

Trie Structure

As described in the general section about solution archives (see Chapter 3) the underlying data structure of the solution archive is an indexed trie, but for this problem the trie nodes may have more than two children.

In Figure 5.3 the trie structure for the proposed solution archive is shown. Each level i represents a position in the permutation based solution representation whereas each trie node of a level corresponds to a specific variable assignment of the first i positions. The size of each trie node is decreasing with increasing depth and has m possible child nodes on the first level. Each node q on level l has the same structure consisting of $m - l + 1$ entries $q[0], \ldots, q[m - l + 1]$. Like in the general SA and the SA for the CFLP, each entry can either be a pointer to another trie node on level $l + 1$ (denoted by an arrow), a *null*-pointer (denoted by a slash), or a *complete*-pointer (denoted by a C). Now let $(i_1, \ldots, i_m)$ be the cluster permutation representing a solution candidate which should be inserted. Then, each variable i_l is related to a node q of level l in the trie. This node q splits the solution space into $m - l + 1$ parts, where in all subspaces the variables i_1 to i_{l-1} are fixed according to the path from the root node to q. Figure 5.3 shows two already inserted solution candidates $(i_1, \ldots, i_m)$ and $(j_1, \ldots, j_m)$. Here we see, that the decision of the node on the first level i_1 or j_1, respectively, fixes the first variable so that on the second level only $m - 1$ decisions remain. This number of decisions, which corresponds to deciding which cluster is visited next in the sequence, decreases on each level so that on level m the last decision, i.e., the last remaining cluster, is already fixed and a *complete*-pointer is set to the associated entry.

This structure is further exploited in Section 5.6.4 when lower bounds on partial solutions are computed to cut off subtries which evidently cannot contain good solution

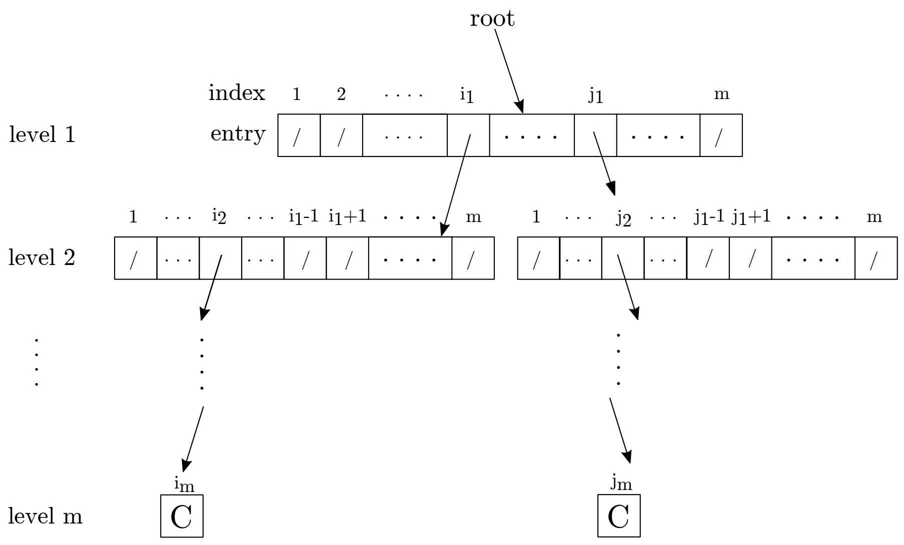

root
index 1 2 .... i1 j1 m
level 1
entry / / .... / .... \ .... /
1 ··· i2 ··· i1-1 i1+1 .... m
1 ··· j2 ··· j1-1 j1+1 .... m
level 2
im
jm
level m
C
C

Figure 5.3: Solution archive with two solution candidates $(i_1, i_2, \ldots, i_m)$ and $(j_1, j_2, \ldots, j_m)$.

candidates.

Solution Conversion

The basic structure of the solution conversion is similar to the generic version described in Chapter 3. Whenever the insertion procedure detects a duplicate solution such a conversion is performed. Assume that the solution $x = (x_1, \ldots, x_m)$ is inserted and on level $l \in \{1, \ldots, m\}$ a *complete*-pointer is encountered. Let $P = \{q_1, \ldots, q_l\}$ be the trie nodes visited during the insertion. Then, a conversion is performed by choosing a conversion node $q' \in P$ randomly which has at least one other entry whose value is not a *complete*-pointer. If there is no such node we know that the whole solution space has been covered and we can stop the optimization with the so far best solution candidate being a proven optimum. Otherwise we pick a non-*complete* entry $q'[k]$, $k \in \{0, \ldots, m - l + 1\}$ uniformly at random and swap its index x'_l with x_l in the solution. If the value at $q'[k]$ was a *null*-pointer we know that this new solution obtained by the swap has not been considered so far and therefore we insert it from node q on, which completes the conversion. Otherwise if the value at $q'[k]$ was a pointer to another trie node we could end up in a *complete*-pointer again. Then, analogously, another swap is performed. This procedure is repeated until level m is reached, at which point a guaranteed new and usually similar solution after at most $m - l$ swaps has been derived.

An example of a solution conversion for an instance with five clusters is shown in Figure 5.4. The sequence of visited trie nodes is denoted by the enumeration of the arcs starting from the root node and ending at the node where the conversion ends. The solution archive contains already two solutions $s_1 = (C_4, C_1, C_2, C_3, C_5)$ and

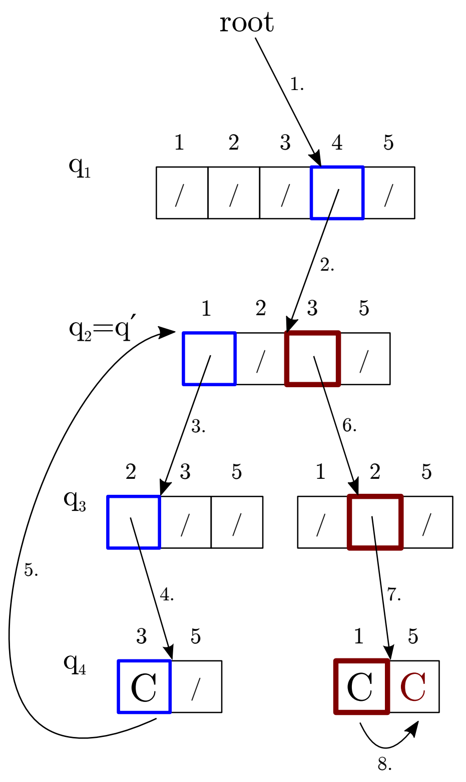


Figure 5.4: Example of a conversion operation in the solution archive transforming the duplicate solution $(C_4, C_1, C_2, C_3, C_5)$ into the new solution candidate $(C_4, C_3, C_2, C_5, C_1)$.

$s_2 = (C_4, C_3, C_2, C_1, C_5)$ before the duplicate s_1 is inserted again into the archive by following the first four arcs. On node q_4 the duplicate is detected and consequently a conversion is performed. The node q_2 is chosen for conversion among all the visited nodes $\{q_1, \ldots, q_4\}$. On that level a swap of C_1 and C_3 is performed leading to the intermediate solution s_2. However, while inserting the remaining solution it is observed that it is still not a new solution yet, so another conversion on level 4 has to be performed leading to the final converted solution candidate $(C_4, C_3, C_2, C_5, C_1)$.

Computing Lower Bounds for Partial Solutions

As an additional feature of the solution archive a bounding extension is added, which is similar to the one described by [74] for the generalized minimum spanning tree problem. It is based on one of the basic ideas of a tree search like branch-and-bound: as mentioned in Section 5.6.4 each node of the trie represents a subspace of all solutions. If meaningful lower bounds for the objective values of the solutions associated with trie nodes can be computed, some of these nodes can likely be pruned in a branch-and-bound manner.

Before we compute lower bounds on trie nodes, we reverse the order of the variables as they are considered in the trie, i.e., for a given solution $x = (x_1, \ldots, x_m)$ the variable order is $x_m, x_{m-1}, \dots, x_1$. This order is beneficial for the bound computation as we will see next. To compute a lower bound for a particular subtrie represented by entry k of a trie node q on level l, we partition the set of clusters into three disjoint subsets C_0, C^f, and C^o as shown in Figure 5.5. C^f denotes the set of fixed clusters, which is given by the fixed part of the solution $(x_m, \dots, x_{m-l+1})$. Assume that the last fixed cluster of C^f is cluster C_l. $C^o = C \setminus (C^f \cup C_0)$ is the set of open clusters, for which the sequence of visit is still unknown. For these clusters four conditions are relaxed:

1. Capacity constraints of the vehicle.

2. Connectivity constraints for avoiding subtours.

3. Constraints ensuring that exactly one node from each cluster must be chosen

4. Degree constraints of the nodes (the degree of each node must be either zero or two).

In the following we use the notation $a(C^k)$, $\forall C^k \in 2^C$ to determine all inter-cluster edges of the clusters in C^k. Then, a valid lower bound on the partial solution $(x_m, \dots, x_{m-l+1})$ can be computed by summing up costs of five different components:

1. The dynamic programming algorithm for the solution evaluation is adapted to work correctly for partial solutions and is executed on the fixed set of clusters, which results in the value lb^f. Note that lb^f already contains an arc from C_0 to C_l, although C_l is definitely not the first visited cluster, so the arc weight $a^f = \max\limits_{j \in C_l} c_0 j$ is subtracted from lb^f resulting in a lower bound on C^f of $lb_1 = lb^f - a^f$.

2. The total cost of the $|C^o| - 1$ cheapest edges in C^o is denoted by lb_2. While there are methods that can produce better bounds, e.g., computing a minimum spanning tree, we choose to use this simple computation to keep the time consumption low.

3. A lower bound on the restocking costs for the clusters in C^o is computed by first taking the total sum of the expected demand $E[C^o]$ of these clusters. Then, lb_3 is given by multiplying the cheapest edge from the depot to any node in C^o, $\lfloor \frac{E[C^o]}{Q} \rfloor$ times.

4. To connect C^o to C^f the cheapest edge from C^o to C_l determines lb_4.

5. Finally, lb_5 is given by the cheapest edge from C_0 to any node in C^o.

These individual parts form the lower bound $lb = \sum\limits_{i=1}^{5} lb_i$, which is stored at the corresponding trie node. Directly after this computation or whenever this trie node is visited again, this lower bound is compared to the value of the best solution found so far, which corresponds to a global upper bound, to possibly cut off this node and the corresponding solution subspace. Figure 5.5 shows an example of the bounding procedure,

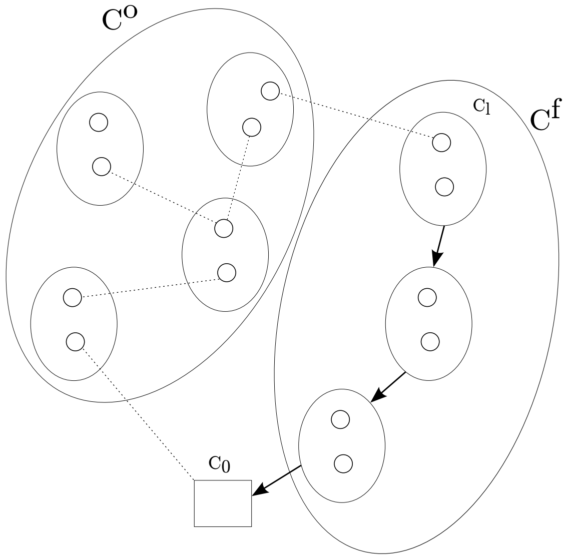


Figure 5.5: Example of the computation of a lower bound on a partial solution.

where the position of three clusters are already fixed as denoted by the arrows. The dotted lines represent the lowest-cost edges which form lb_2, lb_4, and lb_5.

To speed up the computation of lb_1 at the cost of potentially worsening the bound, any approximation level from the multi-level evaluation (see Section 5.5.1) can be chosen for the DP algorithm. In our preliminary tests it turned out that even at the highest approximation level $\lceil \log_2 Q \rceil$ the bound was reasonably good so that this level is chosen for the remaining computational tests. However, even with this speed-up computing bounds on each new trie node would be too time consuming and therefore this procedure is only applied with a certain probability whenever a trie node is accessed.

5.7 Computational Results

To evaluate the developed algorithms a computational study is performed. We rely on a set of 158 benchmark instances for the GVRPSD[1] which are based on (deterministic) instances for the generalized vehicle routing problem generated by Bektaş et al. [10]. They modified instances from the CVRP-library (`http://branchandcut.org/VRP/data/`), having 16 to 262 nodes and partitioned them into $m = \lfloor \frac{n}{\theta} \rfloor$ clusters, where $\theta = \{2, 3\}$. These instances are adapted to the GVRPSD by setting the expected demand of each cluster to the deterministic demand of the corresponding cluster. Then the clusters are divided into *low spread* and *high spread* clusters uniformly at random. The possible demand values for each cluster lie in $\pm 10\%$ of the expected demand (rounded down) for *low spread* clusters and $\pm 30\%$ (rounded up) for *high spread* clusters. Values lower than zero or larger than

[1]`https://www.ac.tuwien.ac.at/research/problem-instances/#Generalized_Vehicle_Routing_Problem_with_Stochastic_Demands`

Q are not considered. A uniform distribution is used for the set of possible demands. The algorithms are implemented in C++ using CPLEX 12.6 for the integer L-shaped method as well as for solving the GTSP in the initial solution creation phase. All runs were executed on a single core of an Intel Xeon processor with 2.54 GHz and 20 GB RAM. For the metaheuristics each run of all tested configurations was repeated 30 times.

5.7.1 Integer L-shaped Method

For evaluating the integer L-shaped method we use a selection of the smaller instances with $\theta = 3$ and additionally consider instances with a higher demand variance. In these instances the possible demand values lie in $\pm 50\%$ of the expected demand for *low spread* and $\pm 80\%$ for *high spread* clusters.

In Table 5.1 the integer L-shaped algorithm with only the specific optimality cuts is compared to the version where both specific and general optimality cuts are considered. For both configurations the final objective value *obj*, the time needed in seconds $t[s]$ and the resulting optimality gap (*gap*) are listed. Additionally, the number of specific ($\#\mathrm{O}_S$) and general ($\#\mathrm{O}_G$) optimality cuts are given.
We observe that when only the specific optimality cuts are used only 7 out of 27 instances could be solved to optimality within the CPU time limit of four hours. This number is increased to 11 when also general optimality cuts are added. Also for the unsolved instances the final optimality gap is consistently better and the number of specific optimality cuts lower when considering general optimality cuts. We further observe that a lot of optimality cuts were needed to solve the model, which indicates a potential to improve this algorithm. Apparently, the lower bounds obtained by the general optimality cuts are still too weak to be able to solve all instances in reasonable time. We notice that the complexity of the instances does not only depend on the number of clusters and nodes but even more on the number of expected restocks as no instance with $E[nr] > 2$ could be solved to optimality. The variability of the demand for each cluster, however, seems not to increase the hardness of the instance, as the results are similar as for the instances with the lower variance. Therefore, for the metaheuristic algorithms only the instances with lower demand variance are considered.

5.7.2 Variable Neighborhood Search

For the tests of the VNS we cosider the full benchmark set of 158 instances but here we show only a representative selection of 37 instances and refer to the Appendix A.2 for the full result tables.

In the following tables the results for the different configurations are given with their (average) objective values, their standard deviations (*sd*) if applicable and either the total run-time in seconds $t[s]$ for the deterministic configurations or the average time when the best solution is identified $t^*[s]$.

First we compare the results of the different starting solutions, farthest insertion (FI) and GTSP, with a subsequent VND using the neighborhood structures and the order described in Section 5.5.3. Table 5.2 shows the (deterministic) numerical results for these

Table 5.1: Results of the integer L-shaped method with and without general optimality cuts.

				L-shaped + OPT$_S$				L-shaped + OPT$_S$ + OPT$_G$				
Instance	n	m	$E[nr]$	*obj*	$t[s]$	*gap*	#O$_S$	*obj*	$t[s]$	*gap*	#O$_S$	#O$_G$
A-n32-k5-C11-V2	32	11	1.39	386.909	>14400	4.2%	877	386.909	2086	0.0%	160	631
A-n33-k5-C11-V2	33	11	1.52	318.028	231	0.0%	389	318.028	84	0.0%	76	353
A-n33-k6-C11-V2	33	11	1.91	364.589	10172	0.0%	1367	364.589	658	0.0%	78	359
A-n34-k5-C12-V2	34	12	1.66	419.124	>14400	8.1%	2752	419.124	10824	0.0%	260	1467
A-n36-k5-C12-V2	36	12	1.34	399.905	>14400	12.9%	2018	399.997	>14400	9.1%	671	2882
A-n37-k5-C13-V2	37	13	1.43	359.133	1271	0.0%	903	359.133	212	0.0%	158	594
A-n38-k5-C13-V2	38	13	1.71	371.795	>14400	4.0%	2007	371.795	339	0.0%	61	408
P-n40-k5-C14-V2	40	14	1.51	214.753	2308	0.0%	1279	214.753	399	0.0%	123	748
P-n45-k5-C15-V2	45	15	1.61	239.357	>14400	7.2%	2337	239.568	>14400	3.0%	281	1910
P-n76-k4-C26-V2	76	26	1.33	310.397	>14400	5.9%	1229	310.397	>14400	6.1%	283	2269
P-n76-k5-C26-V2	76	26	1.67	310.397	>14400	5.9%	1252	310.397	>14400	6.3%	299	3484
Instances with higher demand variance												
A-n32-k5-C11-V2	32	11	1.39	393.475	>14400	6.5%	1180	393.475	3845	0.0%	199	1209
A-n33-k5-C11-V2	33	11	1.52	322.048	571	0.0%	577	322.048	134	0.0%	108	605
A-n33-k6-C11-V2	33	11	1.91	366.445	9380	0.0%	1335	366.445	570	0.0%	84	519
A-n34-k5-C12-V2	34	12	1.66	427.409	>14400	9.7%	3673	427.409	>14400	2.6%	371	2705
P-n40-k5-C14-V2	40	14	1.51	215.798	2480	0.0%	1662	215.798	392	0.0%	131	1056
P-n45-k5-C15-V2	45	15	1.61	241.271	>14400	8.1%	2624	241.271	>14400	3.6%	257	2315
P-n76-k4-C26-V2	76	26	1.33	310.397	>14400	6.0%	1078	310.397	>14400	7.2%	161	1742
P-n76-k5-C26-V2	76	26	1.67	311.431	>14400	6.4%	1032	311.431	>14400	6.8%	255	3786
Instances with $E[nr] > 2$												
A-n45-k6-C15-V3	45	15	2.09	478.219	>14400	16.8%	2369	478.219	>14400	13.3%	503	4400
A-n45-k7-C15-V3	45	15	2.06	491.539	>14400	31.7%	3498	491.739	>14400	29.0%	871	6614
A-n46-k7-C16-V3	46	16	2.08	471.716	>14400	23.3%	2626	465.624	>14400	15.7%	511	4734
A-n48-k7-C16-V3	48	16	2.13	465.343	>14400	28.7%	2361	465.35	>14400	25.8%	892	8151
A-n53-k7-C18-V3	53	18	2.09	443.873	>14400	14.6%	1977	445.802	>14400	11.8%	539	6521
A-n54-k7-C18-V3	54	18	2.19	496.364	>14400	28.8%	1961	507.513	>14400	27.9%	661	6785
A-n55-k9-C19-V3	55	19	2.75	481.531	>14400	21.9%	1662	483.997	>14400	19.2%	359	4625
A-n60-k9-C20-V3	60	20	2.8	622.404	>14400	36.6%	1726	622.404	>14400	34.6%	440	5835
Average gap						10.7%				8.2%		

configurations. Additionally, it contains a third configuration where the ML-ES is used along with the GTSP starting solution.

The results indicate that both starting solutions produce similarly good results for the instances with up to 75 nodes. However, when considering larger instances with 76 nodes and more, FI is not competitive anymore. When starting from an inferior solution produced by FI the VND needs too much time and could not even be completed within the time limit of 10000 seconds. When comparing run-time we also see the advantage of using the GTSP over the FI; for most of the instances, especially for the larger ones, it pays off to invest more time to get a better starting solution so that the subsequent VND does not need so many iterations. We also applied the ML-ES to the GTSP + VND configuration and we observe a huge drop in run-time. It is clear that the resulting solution is the same as in the GTSP + VND configuration but the run-time could be reduced substantially. Only by using the ML-ES the VND is about 10 times faster on average with a peak speedup factor of 75 for instance P-n76-k4-C26-V2. During our tests

Table 5.2: Results for the different configurations of the VND.

				FI + VND		GTSP + VND		GTSP + VND + ML-ES	
Instance	n	m	$E[nr]$	*obj*	$t[s]$	*obj*	$t[s]$	*obj*	$t[s]$
P-n19-k2-C7-V1	19	7	0.71	**112.105**	5	**112.105**	3	**112.105**	<1
P-n20-k2-C7-V1	20	7	0.68	**117.306**	4	**117.306**	<1	**117.306**	<1
P-n21-k2-C7-V1	21	7	0.64	**117.071**	3	**117.071**	<1	**117.071**	<1
P-n22-k2-C8-V1	22	8	0.73	**111.194**	10	**111.194**	5	**111.194**	<1
B-n31-k5-C11-V2	31	11	1.38	**355.729**	25	**355.729**	16	**355.729**	5
A-n32-k5-C11-V2	32	11	1.39	**386.909**	20	388.597	10	388.597	2
A-n33-k5-C11-V2	33	11	1.52	**318.028**	17	**318.028**	15	**318.028**	3
A-n33-k6-C11-V2	33	11	1.91	**367.629**	23	**367.629**	16	**367.629**	4
A-n34-k5-C12-V2	34	12	1.66	**419.124**	29	**419.124**	25	**419.124**	4
B-n34-k5-C12-V2	34	12	1.34	**363.089**	33	**363.089**	13	**363.089**	5
B-n35-k5-C12-V2	35	12	1.54	**501.470**	32	**501.470**	14	**501.470**	6
A-n36-k5-C12-V2	36	12	1.34	404.579	30	**399.905**	23	**399.905**	7
A-n37-k5-C13-V2	37	13	1.43	**359.133**	45	**359.133**	20	**359.133**	3
A-n37-k6-C13-V2	37	13	1.95	467.266	31	**430.987**	32	**430.987**	7
A-n38-k5-C13-V2	38	13	1.71	**371.795**	57	**371.795**	20	**371.795**	2
B-n38-k6-C13-V2	38	13	1.93	**386.195**	55	389.241	27	389.241	7
A-n39-k5-C13-V2	39	13	1.48	390.400	47	**371.410**	20	**371.410**	8
A-n39-k6-C13-V2	39	13	1.83	**417.844**	43	**417.844**	40	**417.844**	8
B-n39-k5-C13-V2	39	13	1.45	**281.482**	50	**281.482**	<1	**281.482**	<1
P-n40-k5-C14-V2	40	14	1.51	**214.775**	175	**214.753**	49	**214.753**	4
B-n41-k6-C14-V2	41	14	1.82	**404.261**	93	**404.261**	34	**404.261**	11
B-n43-k6-C15-V2	43	15	1.81	394.529	74	**347.650**	33	**347.650**	6
A-n44-k6-C15-V2	44	15	2.00	**505.129**	105	508.981	51	508.981	15
B-n45-k5-C15-V2	45	15	1.51	**419.613**	116	**419.613**	59	**419.613**	8
B-n45-k6-C15-V2	45	15	1.96	**358.989**	72	**358.989**	83	**358.989**	31
P-n45-k5-C15-V2	45	15	1.61	**239.568**	172	**239.357**	94	**239.357**	6
A-n45-k6-C15-V3	45	15	2.09	**478.219**	105	**478.219**	56	**478.219**	12
A-n45-k7-C15-V3	45	15	2.06	516.508	94	**488.017**	99	**488.017**	34
A-n46-k7-C16-V3	46	16	2.08	**465.624**	209	471.980	82	471.980	16
A-n48-k7-C16-V3	48	16	2.13	474.210	150	**462.548**	95	**462.548**	35
A-n53-k7-C18-V3	53	18	2.09	450.973	268	**443.875**	97	**443.875**	16
A-n54-k7-C18-V3	54	18	2.19	507.805	201	**490.544**	134	**490.544**	41
A-n55-k9-C19-V3	55	19	2.75	475.919	292	**474.048**	114	**474.048**	15
A-n60-k9-C20-V3	60	20	2.80	**614.515**	517	620.897	361	620.897	117
P-n76-k4-C26-V2	76	26	1.33	461.753	>10000	**310.397**	4312	**310.397**	58
P-n76-k5-C26-V2	76	26	1.67	373.937	>10000	**310.397**	3748	**310.397**	56
P-n101-k4-C34-V2	101	34	1.25	992.679	>10000	**371.926**	9979	**371.926**	397

when a solution is evaluated using ML-ES the procedure could be terminated in the top 30% of the approximation levels where the acceleration is the largest.

Next we show how average results over 30 independent runs of the VNS with the GTSP starting solution and the ML-ES compares to the integer L-shaped method. Table 5.3 shows the numerical comparison with the exact method and for the exact algorithm the optimality gap (*gap*) and the time needed is stated in the table.

The high optimality gaps on the medium to large instances show that the GVRPSD is a hard problem but on the instances where the L-shaped method is able to find a proven optimal solution the VNS also finds it in substantially less time. In the extreme case of instance A-n34-k5-C12-V2 the VNS found an optimal solution in all 30 runs about 865 times faster than the exact algorithm. However, since the L-shaped method

Table 5.3: Comparison of the proposed VNS with an exact integer L-shaped method.

				L-shaped			VNS + GTSP + ML-ES		
Instance	n	m	$E[nr]$	obj	gap	$t[s]$	$\overline{obj}$	sd	$t^*[s]$
P-n19-k2-C7-V1	19	7	0.71	**112.105**	0.0%	<1	**112.105**	0.00	<1
P-n20-k2-C7-V1	20	7	0.68	**117.306**	0.0%	<1	**117.306**	0.00	<1
P-n21-k2-C7-V1	21	7	0.64	**117.071**	0.0%	<1	**117.071**	0.00	<1
P-n22-k2-C8-V1	22	8	0.73	**111.194**	0.0%	<1	**111.194**	0.00	<1
B-n31-k5-C11-V2	31	11	1.38	**355.729**	11.1%	>7200	**355.729**	0.00	5
A-n32-k5-C11-V2	32	11	1.39	**386.909**	0.0%	1331	**386.909**	0.00	25
A-n33-k5-C11-V2	33	11	1.52	**318.028**	0.0%	374	**318.028**	0.00	2
A-n33-k6-C11-V2	33	11	1.91	**364.589**	0.0%	598	**364.589**	0.00	27
A-n34-k5-C12-V2	34	12	1.66	**419.124**	0.0%	3719	**419.124**	0.00	4
B-n34-k5-C12-V2	34	12	1.34	**363.089**	18.1%	>7200	**363.089**	0.00	4
B-n35-k5-C12-V2	35	12	1.54	501.470	26.3%	>7200	**501.450**	0.11	6
A-n36-k5-C12-V2	36	12	1.34	**399.905**	9.4%	>7200	**399.905**	0.00	7
A-n37-k5-C13-V2	37	13	1.43	**359.133**	0.0%	98	**359.133**	0.00	3
A-n37-k6-C13-V2	37	13	1.95	434.865	18.5%	>7200	**430.987**	0.00	7
A-n38-k5-C13-V2	38	13	1.71	**371.795**	0.0%	588	**371.795**	0.00	2
B-n38-k6-C13-V2	38	13	1.93	**386.195**	12.2%	>7200	388.734	1.15	8
A-n39-k5-C13-V2	39	13	1.48	**371.410**	7.6%	>7200	**371.410**	0.00	8
A-n39-k6-C13-V2	39	13	1.83	**417.844**	5.6%	>7200	**417.844**	0.00	8
B-n39-k5-C13-V2	39	13	1.45	**281.482**	0.0%	282	**281.482**	0.00	<1
P-n40-k5-C14-V2	40	14	1.51	**214.753**	0.0%	392	**214.753**	0.00	4
B-n41-k6-C14-V2	41	14	1.82	408.977	16.7%	>7200	**404.261**	0.00	10
B-n43-k6-C15-V2	43	15	1.81	**347.650**	19.7%	>7200	**347.650**	0.00	6
A-n44-k6-C15-V2	44	15	2.00	509.254	16.2%	>7200	**508.981**	0.00	15
B-n45-k5-C15-V2	45	15	1.51	**419.613**	3.6%	>7200	419.096	1.05	8
B-n45-k6-C15-V2	45	15	1.96	367.730	23.6%	>7200	**358.989**	0.00	31
P-n45-k5-C15-V2	45	15	1.61	**239.357**	4.6%	>7200	**239.357**	0.00	6
A-n45-k6-C15-V3	45	15	2.09	478.265	16.2%	>7200	**478.219**	0.00	12
A-n45-k7-C15-V3	45	15	2.06	491.539	30.6%	>7200	**488.017**	0.00	34
A-n46-k7-C16-V3	46	16	2.08	**465.624**	19.0%	>7200	471.539	0.51	16
A-n48-k7-C16-V3	48	16	2.13	469.690	28.7%	>7200	**462.548**	0.00	35
A-n53-k7-C18-V3	53	18	2.09	**443.873**	13.6%	>7200	443.875	0.00	16
A-n54-k7-C18-V3	54	18	2.19	500.349	28.6%	>7200	**490.544**	0.00	41
A-n55-k9-C19-V3	55	19	2.75	483.997	21.7%	>7200	**474.048**	0.00	15
A-n60-k9-C20-V3	60	20	2.80	623.528	35.6%	>7200	**617.575**	4.98	118
P-n76-k4-C26-V2	76	26	1.33	**310.397**	6.1%	>7200	**310.397**	0.00	55
P-n76-k5-C26-V2	76	26	1.67	**310.397**	5.8%	>7200	**310.397**	0.00	53
P-n101-k4-C34-V2	101	34	1.25	**371.926**	5.7%	>7200	**371.926**	0.00	379

guarantees the optimality of the solution we cannot directly compare the run-time of these algorithms. Our tests further showed that in the VNS the ML-ES can be terminated within the top 4% of the approximation levels which is even better than for the VND.

5.7.3 Genetic Algorithm with Solution Archive

Finally, the genetic algorithm with solution archive is evaluated. Since the GASA consists of several components, each is evaluated separately. Each run of the GA is terminated after a maximum of 300 CPU seconds (T_{max}=300). Preliminary tests showed that the parameters for the basic GA were not particularly sensitive to changes, therefore the population size is fixed to 100, p_{mut} to 0.1, and $nMut$ to 10.

In the first set of experiments for the GASA the VND is examined more closely to evaluate the effectiveness of the used neighborhood structures within the GA. Then, extensive tests regarding the solution archive are performed. Therefore, the algorithm is run with and without the solution archive and results are compared. After that, the bounding extension is investigated in detail.

Variable Neighborhood Descent

In a first step tests with various values for α, which determines the frequency of VND executions, are performed. In preliminary tests it turned out that when α is higher than 0.1 the run-time spent in the VND dominates the other parts of the algorithm too much. Therefore, we consider here $\alpha \in \{0.01, 0.05, 0.1\}$. Table 5.4 shows a summary of the results grouped by the instance set. In this table as well as in other tables in this chapter $\overline{obj}$ stands for the average objective value over 30 independent runs using all instances of the respective group, $\overline{obj_g}$ is the geometric mean over these runs, and $\overline{gap}$ *to BKS* is the average percentage gap to the best known solution (BKS). The BKS is determined by taking the best objective value of each instance separately over all runs and configurations which are tested here. The row *#Best results* indicates the number of instances, for which this configuration yields the best average objective value of the configurations under comparison. The next three rows show the p-Values of one-sided paired Wilcoxon rank sum tests which were performed over all instances.

Table 5.4: Performance of the GA+VND with different values for α.

	Instances with $\theta = 2$			Instances with $\theta = 3$		
	$\alpha = 0.01$	$\alpha = 0.05$	$\alpha = 0.1$	$\alpha = 0.01$	$\alpha = 0.05$	$\alpha = 0.1$
$\overline{obj}$	696.32	695.53	695.23	492.59	490.74	490.74
$\overline{obj_g}$	540.19	539.58	539.75	399.48	399.19	398.97
$\overline{gap}$ to BKS	1.31%	1.19%	1.25%	0.77%	0.70%	0.64%
# Best results	61	65	72	67	74	83
p-Value ($< \alpha = 0.01$)	-	0.000011	<0.000001	-	0.000002	<0.000001
p-Value ($< \alpha = 0.05$)	0.999989	-	0.006113	0.999998	-	0.082960
p-Value ($< \alpha = 0.1$)	>0.999999	0.993892	-	>0.999999	0.917141	-

The results in Table 5.4 show that $\overline{obj}$, $\overline{obj_g}$, and the gap to the BKS for $\alpha = 0.05$ and $\alpha = 0.1$ are lower than for $\alpha = 0.01$. The conclusion that the configuration with $\alpha = 0.01$ is worse than the other values for α is also confirmed by the statistical tests, which showed that both $\alpha = 0.05$ and $\alpha = 0.1$ are significantly better with an error level of less than 1%. Considering the number of best results both configurations with $\alpha \in \{0.05, 0.1\}$ have similar values and the gap to the BKS is lower for $\alpha = 0.05$ when instances with $\theta = 2$ are considered but higher for the instance set with $\theta = 3$. However, the statistical test revealed that $\alpha = 0.1$ is significantly better for instances with $\theta = 2$ and therefore this value is used for the remaining tests.

To investigate the effectiveness of each of the used neighborhood structures within the VND the number of times where an improvement could be achieved was counted for each neighborhood $\mathcal{N}_1$ to $\mathcal{N}_4$. For this test we include the solution archive as it is the basis for $\mathcal{N}_4$. The number of conversions, which corresponds to the size of this neighborhood structure, is set to n^2 to have a comparable size to the other neighborhoods. Tables 5.5 and 5.6 show the number of improvements for each instance, which are referred to by their name, the number of nodes n, the number of clusters m, and the expected number of restocks $E[\text{nr}]$. Note that the instances are the same as for the other tested algorithms but abbreviations of the instance names are used.

Table 5.5: Average number of improvements in the different neighborhood structures for smaller instances.

$\theta = 2$	n	m	$E[\text{nr}]$	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$	$\theta = 3$	n	m	$E[\text{nr}]$	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$
P1	16	8	4.20	38	8	5	48	P1	16	6	3.00	3	0	0	1
P2	19	10	1.08	10	13	1	2	P2	19	7	0.71	0	0	0	0
P3	20	10	1.04	28	16	3	6	P3	20	7	0.68	0	0	0	0
P4	21	11	1.04	268	110	26	46	P4	21	7	0.64	0	0	0	0
P5	22	11	1.04	147	47	14	21	P5	22	8	0.73	0	0	0	0
P6	22	11	3.87	380	166	110	40	P6	22	8	2.82	6	3	1	2
P7	23	12	4.53	11907	2694	2339	2652	P7	23	8	2.55	34	13	6	41
B1	31	16	2.03	8779	2648	953	951	B1	31	11	1.38	8701	3000	1034	3189
A1	32	16	2.00	10100	2741	706	887	A1	32	11	1.39	117	47	14	32
A2	33	17	2.33	9333	3272	722	736	A2	33	11	1.52	43	25	2	9
A3	33	17	2.74	9436	4039	818	658	A3	33	11	1.91	98	39	9	15
A4	34	17	2.25	9099	3345	778	666	A4	34	12	1.66	793	265	79	122
B2	34	17	2.05	7243	2395	521	791	B2	34	12	1.34	748	423	59	151
B3	35	18	2.23	7421	1946	505	621	B3	35	12	1.54	4596	1630	432	973
A5	36	18	1.98	6572	2860	484	494	A5	36	12	1.34	771	223	52	125
A6	37	19	2.10	2552	1127	151	163	A6	37	13	1.43	147	57	6	16
A7	37	19	2.93	6754	2520	575	571	A7	37	13	1.95	6282	1949	719	906
A8	38	19	2.42	7011	2513	518	430	A8	38	13	1.71	257	133	25	33
B4	38	19	2.75	5291	1989	447	758	B4	38	13	1.93	8167	2799	702	1708
A9	39	20	2.47	6163	1995	494	350	A9	39	13	1.48	1478	449	101	184
A10	39	20	2.68	6002	1971	491	456	A10	39	13	1.83	1743	897	185	335
B5	39	20	2.33	5753	1660	153	508	B5	39	13	1.45	3	4	0	0
P8	40	20	2.26	5956	2021	325	284	P8	40	14	1.51	215	118	13	29
B6	41	21	2.88	5344	1922	243	339	B6	41	14	1.82	4751	1654	210	584
B7	43	22	2.78	5037	1359	350	296	B7	43	15	1.81	2355	1171	238	265
A11	44	22	2.91	4818	1526	371	367	A11	44	15	2.00	6241	1921	508	802
B8	44	22	3.16	4030	1255	303	332	B8	44	15	2.24	5579	1726	361	785
A12	45	23	3.12	4234	1528	335	275	A12	45	15	2.09	6596	2174	439	732
A13	45	23	3.16	3749	1265	401	328	A13	45	15	2.06	5538	1682	450	860
B9	45	23	2.45	5486	1344	213	211	B9	45	15	1.51	602	200	51	76
B10	45	23	3.05	3865	1361	367	338	B10	45	15	1.96	6417	2155	353	673
P9	45	23	2.35	3529	1123	190	220	P9	45	15	1.61	1202	457	73	140
A14	46	23	3.12	3856	1379	309	270	A14	46	16	2.08	5478	1816	409	635
A15	48	24	3.25	3474	1070	329	331	A15	48	16	2.13	4531	1188	352	709
B11	50	25	3.01	3260	1223	155	283	B11	50	17	2.05	3539	1195	121	301
B12	50	25	4.26	2336	282	210	274	B12	50	17	2.72	4624	1508	385	716
P10	50	25	4.77	3364	994	298	280	P10	50	17	3.32	4934	1437	336	725
P11	50	25	3.18	2952	803	169	208	P11	50	17	2.21	3689	1264	171	452
P12	50	25	3.98	3059	966	203	217	P12	50	17	2.77	4266	1164	239	586

As we see in Tables 5.5 and 5.6 the number of improvements achieved in a specific neighborhood structure is usually smaller than the number in the preceding neighborhood structure. This can be explained by the design of the employed VND: each time an improvement in any neighborhood is found, the search restarts with the first neighborhood

Table 5.6: Average number of improvements in the different neighborhood structures for larger instances.

$\theta = 2$	n	m	E[nr]	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$	$\theta = 3$	n	m	E[nr]	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$
B13	51	26	3.56	2693	866	234	239	B13	51	17	2.44	5710	1249	282	454
P13	51	26	5.04	3123	862	275	237	P13	51	17	3.31	4964	1255	350	617
B14	52	26	3.12	2799	872	161	224	B14	52	18	2.18	3622	1412	266	474
A16	53	27	3.38	2688	687	197	164	A16	53	18	2.09	4178	1176	253	348
A17	54	27	3.43	2477	812	226	227	A17	54	18	2.19	3966	939	220	356
A18	55	28	4.18	2344	639	189	147	A18	55	19	2.75	3818	1362	294	353
P14	55	28	4.72	2244	702	186	169	P14	55	19	3.21	3353	830	215	400
P15	55	28	7.76	2094	257	258	253	P15	55	19	5.27	3580	402	344	763
P16	55	28	3.19	2189	654	127	115	P16	55	19	2.17	2983	859	134	253
P17	55	28	3.39	2260	661	122	118	P17	55	19	2.31	3041	771	149	269
B15	56	28	3.19	1753	413	91	111	B15	56	19	2.20	4013	1134	123	279
B16	57	29	3.68	2394	543	119	107	B16	57	19	2.39	2724	598	104	188
B17	57	29	4.19	1831	511	144	161	B17	57	19	2.55	2928	840	256	445
A19	60	30	4.11	1700	589	173	135	A19	60	20	2.80	3010	875	229	330
P18	60	30	4.83	1748	521	143	126	P18	60	20	3.19	2728	694	179	331
P19	60	30	7.23	1684	530	195	165	P19	60	20	4.78	3306	871	284	474
A20	61	31	4.51	1528	423	143	115	A20	61	21	3.09	2801	897	202	322
A21	62	31	3.66	1605	504	127	78	A21	62	21	2.59	2586	577	185	300
A22	63	32	4.69	1372	420	137	92	A22	63	21	3.14	2493	826	219	308
A23	63	32	4.52	1384	477	135	119	A23	63	21	2.96	2437	723	205	344
B18	63	32	4.41	1589	422	129	107	B18	63	21	2.90	2611	740	184	363
A24	64	32	4.11	1342	357	138	107	A24	64	22	2.55	2409	592	169	276
B19	64	32	4.17	1361	427	105	114	B19	64	22	3.17	2951	701	159	256
A25	65	33	4.14	1092	397	122	89	A25	65	22	2.89	2472	701	166	266
P20	65	33	4.78	1189	375	116	90	P20	65	22	3.23	2592	562	143	241
B20	66	33	4.36	1458	376	93	85	B20	66	22	2.80	2427	625	156	251
B21	67	34	4.59	1425	298	90	71	B21	67	23	3.14	2217	563	153	240
B22	68	34	4.17	1303	383	81	77	B22	68	23	2.87	2257	552	121	232
A26	69	35	4.30	940	302	89	69	A26	69	23	2.94	2340	542	116	196
P21	70	35	4.79	946	299	82	64	P21	70	24	3.36	2062	478	111	206
P22	76	38	1.95	529	172	12	12	P22	76	26	1.33	659	210	25	30
P23	76	38	2.44	697	204	29	18	P23	76	26	1.67	966	237	26	39
B23	78	39	4.89	1056	183	45	37	B23	78	26	3.31	1798	358	79	132
A27	80	40	4.43	627	150	56	37	A27	80	27	3.04	1388	267	83	116
M1	101	51	4.73	197	38	7	6	M1	101	34	3.20	1071	81	11	16
P24	101	51	1.83	179	33	4	1	P24	101	34	1.25	318	50	6	12
M2	121	61	3.54	113	14	1	1	M2	121	41	2.44	226	52	11	12
M3	151	76	5.69	87	3	0	0	M3	151	51	3.71	119	19	3	3
M4	200	100	7.92	46	0	0	0	M4	200	67	5.29	63	0	0	0
G1	262	131	11.79	7	0	0	0	G1	262	88	8.13	9	0	0	0

structure, so the earlier neighborhood structures are searched more often. But still it can be observed that for most but the largest instances even $\mathcal{N}_4$ was useful as it sometimes led to an improvement where the other neighborhood structures did not. Note that the numbers for the smallest instances are very small compared to the larger instances. This is because by using the solution archive the optimal solution was found within seconds after which the algorithm terminated. In Section 5.7.3 this issue is further discussed.

Solution Archive

The next experiments are performed to investigate the impact of the solution archive to the overall results. Therefore, the algorithm with the parameters described before (*without SA*) is compared to the algorithm with the solution archive and no conversion within the VND (*GASA* SA_{conv}*=0*) and to the configuration with solution archive and conversion (*GASA* SA_{conv}*=1*). Furthermore, the relative number of identified duplicate solutions per instance (dups) is recorded.

Table 5.7: Results for the configurations without the solution archive (without SA), with the SA and no conversion in the VND (GASA SA_{conv}=0), and with the SA and with conversion (GASA SA_{conv}=1) for smaller instances with $\theta = 2$.

	without SA		GASA SA_{conv}=0			GASA SA_{conv}=1		
$\theta = 2$	$\overline{obj}$	*sd*	$\overline{obj}$	*sd*	dups[%]	$\overline{obj}$	*sd*	dups[%]
P1	245.34	0.00	245.34	0.00	39.89	245.34	0.00	33.48
P2	146.82	0.00	146.82	0.00	93.01	146.82	0.00	92.86
P3	149.02	0.00	149.02	0.00	92.64	149.02	0.00	92.92
P4	160.48	0.00	160.48	0.00	86.81	160.48	0.00	85.24
P5	162.95	0.00	**161.36**	0.00	86.75	**161.36**	0.00	87.48
P6	323.59	0.01	323.59	0.00	24.28	323.59	0.01	21.64
P7	312.51	0.00	312.51	0.01	17.76	312.51	0.01	15.81
B1	419.91	0.00	419.91	0.00	11.95	419.91	0.00	11.06
A1	538.49	3.49	**520.04**	0.00	41.98	**520.40**	0.94	24.93
A2	455.15	0.00	455.15	0.00	39.80	455.15	0.00	26.17
A3	468.10	0.19	**467.95**	0.00	45.02	**467.96**	0.07	23.23
A4	503.34	2.07	**498.15**	0.00	46.05	**498.35**	1.09	26.59
B2	466.80	0.00	466.80	0.00	13.40	466.80	0.00	12.53
B3	619.24	0.00	619.24	0.00	12.89	619.24	0.00	11.52
A5	506.95	0.00	**506.40**	0.79	58.80	506.68	0.63	40.58
A6	454.29	3.95	**447.86**	0.00	76.72	**447.86**	0.00	57.30
A7	609.17	3.78	**590.59**	3.49	15.18	**595.03**	8.16	11.05
A8	481.97	0.00	481.97	0.00	29.78	481.97	0.00	20.95
B4	479.92	0.00	479.92	0.00	10.18	479.87	0.26	9.50
A9	567.41	0.00	567.41	0.00	16.84	567.41	0.00	11.75
A10	561.25	0.00	**560.61**	0.40	15.91	**561.02**	0.39	11.52
B5	356.43	0.00	356.43	0.00	16.33	356.43	0.00	16.44
P8	296.44	0.00	**296.36**	0.05	17.50	**296.35**	0.04	18.46
B6	483.26	0.00	483.22	0.15	9.58	483.26	0.00	9.08
B7	485.46	0.00	485.46	0.00	9.22	485.46	0.00	8.61
A11	627.86	0.00	627.86	0.00	10.66	627.86	0.00	8.93
B8	563.95	0.00	563.95	0.00	8.19	563.95	0.00	7.81
A12	621.23	0.00	621.23	0.00	10.29	621.23	0.00	8.44
A13	692.89	0.00	692.89	0.00	8.24	693.09	1.09	7.47
B9	502.02	0.00	502.02	0.00	10.76	502.02	0.00	10.52
B10	482.91	0.00	482.91	0.00	7.95	482.91	0.00	7.44
P9	340.68	0.00	**340.50**	0.06	10.16	**340.53**	0.09	9.60
A14	624.05	0.00	**622.84**	1.32	8.03	**622.75**	1.32	7.54
A15	686.42	0.00	686.42	0.00	7.62	686.42	0.00	7.25
B11	454.09	0.00	454.09	0.00	7.92	454.09	0.00	7.68
B12	923.53	0.00	923.53	0.00	6.81	923.53	0.00	6.49
P10	423.34	0.57	**422.24**	1.50	7.06	**421.86**	1.45	6.58
P11	354.47	0.00	354.47	0.00	8.51	354.47	0.00	8.12
P12	377.66	0.00	377.62	0.21	7.39	377.62	0.21	6.97

Table 5.8: Results for the configurations without the solution archive (without SA), with the SA and no conversion in the VND (GASA SA_{conv}=0), and with the SA and with conversion (GASA SA_{conv}=1) for larger instances with $\theta = 2$.

	without SA		GASA SA_{conv}=0			GASA SA_{conv}=1		
$\theta = 2$	$\overline{obj}$	sd	$\overline{obj}$	sd	dups[%]	$\overline{obj}$	sd	dups[%]
B13	682.70	0.00	682.70	0.00	6.85	682.70	0.00	6.47
P13	451.79	0.00	451.79	0.00	6.86	451.79	0.00	6.51
B14	458.66	0.33	**458.39**	0.34	7.00	**458.34**	0.33	6.65
A16	633.47	0.00	632.78	2.79	6.55	**632.03**	4.48	6.21
A17	722.02	2.62	721.54	3.63	6.50	722.38	0.60	6.12
A18	718.26	0.51	718.11	0.06	6.33	718.45	1.80	6.03
P14	420.69	0.00	420.69	0.00	6.30	420.69	0.00	5.97
P15	560.92	0.01	560.86	0.30	6.21	**560.69**	0.60	5.93
P16	361.87	0.00	361.87	0.00	6.84	361.87	0.00	6.53
P17	362.07	0.03	**362.04**	0.02	7.03	**362.03**	0.00	6.70
B15	474.92	0.00	474.92	0.00	6.09	474.89	0.15	5.84
B16	779.30	0.41	**778.60**	0.91	5.94	**778.76**	0.92	5.69
B17	967.33	0.00	967.33	0.00	5.91	967.33	0.00	5.72
A19	815.86	0.00	815.86	0.00	5.79	815.86	0.00	5.59
P18	452.86	0.00	452.86	0.00	5.78	452.86	0.00	5.45
P19	572.08	0.00	572.08	0.00	5.85	572.06	0.37	5.50
A20	658.06	8.24	653.64	9.18	5.60	**653.30**	8.81	5.46
A21	755.75	0.00	755.75	0.00	5.81	755.75	0.00	5.51
A22	830.88	0.53	830.88	0.53	5.45	830.80	0.66	5.25
A23	946.39	0.00	946.39	0.00	5.44	946.39	0.00	5.22
B18	852.87	0.00	852.87	0.00	5.54	852.87	0.00	5.28
A24	837.31	0.00	837.31	0.00	5.41	837.31	0.00	5.12
B19	514.92	0.00	514.92	0.00	5.32	514.92	0.00	5.05
A25	712.14	0.00	712.14	0.00	5.42	712.14	0.00	5.11
P20	501.39	0.00	501.34	0.30	5.43	501.39	0.00	5.25
B20	818.42	0.00	818.42	0.00	5.15	818.42	0.00	4.92
B21	672.40	0.00	672.40	0.00	4.93	672.42	0.05	4.80
B22	738.48	0.00	738.48	0.00	4.97	738.13	1.91	4.76
A26	706.39	6.91	707.90	6.07	4.85	707.67	6.52	4.68
P21	504.96	0.00	504.96	0.00	4.90	504.96	0.00	4.69
P22	394.20	0.00	**392.81**	1.01	8.55	**392.89**	0.89	8.43
P23	409.93	0.00	409.93	0.00	6.08	409.93	0.00	5.96
B23	873.27	12.83	863.99	18.93	4.00	865.28	18.52	3.80
A27	1049.26	0.71	1049.26	0.71	3.79	1049.39	0.00	3.60
M1	569.06	17.30	569.15	19.41	3.47	572.23	18.08	3.44
P24	462.25	3.27	**458.43**	0.35	28.98	**458.37**	0.28	24.97
M2	896.42	83.21	860.22	112.30	3.09	886.63	148.42	2.88
M3	966.97	102.47	983.18	209.26	2.39	993.70	211.74	2.20
M4	1967.18	166.82	1680.24	605.77	2.02	1740.09	545.62	1.71
G1	8709.61	775.64	8669.10	741.93	1.52	8526.76	550.58	1.21

Table 5.9: Results for the configurations without the solution archive (without SA), with the SA and no conversion in the VND (GASA SA_{conv}=0), and with the SA and with conversion (GASA SA_{conv}=1) for smaller instances with $\theta = 3$.

	without SA		GASA SA_{conv}=0			GASA SA_{conv}=1		
$\theta = 3$	$\overline{obj}$	sd	$\overline{obj}$	sd	dups[%]	$\overline{obj}$	sd	dups[%]
P1	170.37	0.00	170.37	0.00	7.46	170.37	0.00	10.36
P2	112.10	0.00	112.10	0.00	70.94	112.10	0.00	72.02
P3	117.31	0.00	117.31	0.00	72.52	117.31	0.00	72.00
P4	117.07	0.00	117.07	0.00	71.52	117.07	0.00	70.90
P5	111.19	0.00	111.19	0.00	87.77	111.19	0.00	87.66
P6	245.83	0.00	245.83	0.00	31.92	245.83	0.00	29.28
P7	183.59	0.00	183.59	0.00	20.44	183.59	0.00	19.78
B1	355.73	0.00	355.73	0.00	28.09	355.73	0.00	24.59
A1	386.91	0.00	386.91	0.00	85.45	386.91	0.00	85.33
A2	318.03	0.00	318.03	0.00	89.28	318.03	0.00	88.76
A3	364.59	0.00	364.59	0.00	86.83	364.59	0.00	87.40
A4	419.12	0.00	419.12	0.00	71.34	419.12	0.00	70.76
B2	363.09	0.00	363.09	0.00	72.98	363.09	0.00	73.14
B3	501.47	0.00	**500.87**	0.00	45.85	**500.87**	0.00	47.17
A5	399.90	0.00	399.90	0.00	74.41	399.90	0.00	71.99
A6	359.13	0.00	359.13	0.00	83.34	359.13	0.00	83.65
A7	430.99	0.00	430.99	0.00	48.11	430.99	0.00	48.19
A8	371.80	0.00	371.80	0.00	80.60	371.80	0.00	78.99
B4	389.04	0.77	**386.25**	0.30	28.90	**386.25**	0.30	24.60
A9	371.41	0.00	371.41	0.00	63.80	371.41	0.00	64.09
A10	416.03	0.00	416.03	0.00	59.30	416.03	0.00	60.26
B5	281.48	0.00	281.48	0.00	88.47	281.48	0.00	88.64
P8	214.75	0.00	214.75	0.00	80.90	214.75	0.00	78.42
B6	404.26	0.00	404.26	0.00	39.34	404.26	0.00	39.92
B7	347.65	0.00	347.65	0.00	57.08	347.65	0.00	55.01
A11	508.85	0.70	**506.60**	1.09	26.19	**506.48**	1.03	24.82
B8	402.02	0.00	402.02	0.00	14.85	402.02	0.00	14.13
A12	478.22	0.00	478.22	0.00	26.14	478.22	0.00	24.67
A13	488.02	0.00	488.02	0.00	11.66	488.02	0.00	11.53
B9	417.03	0.00	417.03	0.00	66.42	417.03	0.00	67.98
B10	358.99	0.00	358.99	0.00	16.81	358.99	0.00	15.99
P9	239.36	0.00	239.36	0.00	59.32	239.36	0.00	58.08
A14	470.96	0.00	**466.82**	2.22	18.53	**466.34**	1.85	18.58
A15	462.55	0.00	462.55	0.00	15.58	462.55	0.00	14.26
B11	398.38	0.00	398.38	0.00	15.97	398.38	0.00	14.11
B12	600.66	0.00	600.64	0.05	7.87	600.65	0.03	7.63
P10	302.37	0.00	302.37	0.00	8.00	302.37	0.00	8.30
P11	261.31	0.00	261.31	0.00	21.17	261.31	0.00	17.35
P12	273.12	0.80	**268.91**	0.00	18.91	**268.91**	0.00	16.08

Table 5.10: Results for the configurations without the solution archive (without SA), with the SA and no conversion in the VND (GASA SA_{conv}=0), and with the SA and with conversion (GASA SA_{conv}=1) for larger instances with $\theta = 3$.

	without SA		GASA SA_{conv}=0			GASA SA_{conv}=1		
$\theta = 3$	$\overline{obj}$	sd	$\overline{obj}$	sd	dups[%]	$\overline{obj}$	sd	dups[%]
B13	513.02	0.00	513.02	0.00	12.84	513.02	0.00	12.34
P13	313.41	0.00	313.41	0.00	7.57	313.41	0.00	7.87
B14	360.50	0.00	360.50	0.00	22.16	360.50	0.00	18.56
A16	443.87	0.00	443.87	0.00	17.87	443.87	0.00	17.86
A17	490.54	0.00	490.54	0.00	10.00	490.54	0.00	9.86
A18	474.05	0.00	474.05	0.00	10.24	474.05	0.00	9.24
P14	316.65	0.00	**313.37**	2.18	6.99	**313.67**	2.25	6.72
P15	396.20	0.00	396.20	0.00	6.38	396.12	0.44	6.10
P16	274.22	0.00	274.22	0.00	12.14	274.22	0.00	12.97
P17	276.33	0.00	276.33	0.00	10.92	276.33	0.00	11.55
B15	358.81	0.26	**357.91**	0.26	10.65	**357.84**	0.00	10.44
B16	565.26	1.06	**564.53**	0.55	9.42	**564.37**	0.12	9.09
B17	689.78	8.96	**681.73**	11.45	6.35	682.01	11.30	6.14
A19	620.10	3.03	616.92	5.72	6.67	616.68	8.13	6.28
P18	328.89	0.00	**328.83**	0.05	6.41	**328.83**	0.05	5.94
P19	372.63	0.00	372.63	0.00	5.83	372.63	0.00	5.66
A20	482.51	0.00	482.51	0.00	6.11	482.51	0.00	5.96
A21	617.56	0.00	617.56	0.00	6.31	617.56	0.00	6.02
A22	611.54	0.00	611.54	0.00	5.86	611.66	0.68	5.69
A23	665.59	1.46	664.95	1.64	5.84	664.66	1.61	5.67
B18	604.67	0.07	604.66	0.09	6.80	604.62	0.10	6.53
A24	564.46	0.00	564.46	0.00	6.44	564.33	0.73	6.25
B19	457.24	0.00	457.24	0.00	8.13	457.24	0.00	7.62
A25	525.03	0.00	525.03	0.00	6.06	525.03	0.00	5.78
P20	378.48	0.00	378.48	0.00	5.68	378.48	0.00	5.56
B20	627.36	0.00	627.22	0.35	7.29	627.29	0.26	7.14
B21	561.71	0.00	561.71	0.00	5.74	561.71	0.00	5.58
B22	538.59	1.91	539.25	1.46	6.01	538.70	1.89	5.74
A26	523.77	0.00	523.77	0.00	7.94	523.77	0.00	7.84
P21	386.15	0.00	**385.82**	0.19	5.28	**385.88**	0.21	4.93
P22	310.40	0.00	310.40	0.00	34.38	310.40	0.00	32.96
P23	310.40	0.00	310.40	0.00	23.82	310.40	0.00	27.64
B23	620.11	0.00	620.11	0.00	7.73	620.11	0.00	7.48
A27	748.99	6.03	751.46	6.18	5.25	752.06	6.15	5.09
M1	468.10	14.60	463.48	8.51	5.32	463.06	5.95	5.33
P24	378.68	0.00	**371.93**	0.00	18.22	**372.74**	2.15	18.26
M2	578.24	18.02	**561.00**	18.40	4.87	566.74	18.97	4.57
M3	538.05	30.39	532.71	44.21	4.13	523.75	33.84	4.04
M4	936.83	113.82	**834.52**	158.87	2.89	**829.13**	163.78	2.59
G1	3769.61	317.90	3843.03	324.70	2.31	3898.56	302.18	2.09

The results of these tests are shown in Tables 5.7 and 5.8 for instances with $\theta = 2$ and in Tables 5.9 and 5.10 for instances with $\theta = 3$. The average objective values ($\overline{obj}$) and corresponding standard deviations (sd) over 30 runs are given along with the number of duplicate solutions relative to all generated solutions. The bold numbers in the column for the configuration without the SA mean that on these instances the algorithm without the SA achieved statistically better results (using a pairwise Wilcoxon rank sum test as described before) than either GASA SA_{conv}=0 or GASA SA_{conv}=1. The bold numbers in the other columns indicate statistically better results for the respective configuration compared only to the GA without the SA. These tables show that through all instance sizes the SA is able to improve the algorithm as it produces in 20 out of 79 instances with $\theta = 2$ and in 14 out of 79 instances with $\theta = 3$ significantly better results whereas the GA without the archive was never significantly better. When considering the number of duplicate solutions, it is observed that generally the larger the instances the fewer duplicates are produced and the average number of duplicates is 16.30% for instances with $\theta = 2$ and 28.29% for instances with $\theta = 3$ (for SA_{conv}=0). A summary of the results is given in Table 5.11 where it becomes more obvious that both configurations using the SA achieve significantly better results than the GA without the SA at a significance level of 1%. However, when comparing SA_{conv}=0 to SA_{conv}=1 the results do not show a clear indication which one performed better, but SA_{conv}=0 is used for the remaining tests since the total average gap to the BKS and the geometric mean is lower for this configuration.

Table 5.11: Performance of the GA with different variants of the SA.

	Instances with $\theta = 2$			Instances with $\theta = 3$		
	no SA	SA_{conv}=0	SA_{conv}=1	no SA	SA_{conv}=0	SA_{conv}=1
$\overline{obj}$	679.91	674.57	674.12	460.92	459.77	460.36
$\overline{obj_g}$	545.15	542.90	543.41	398.58	397.42	397.42
$\overline{gap}$ to BKS	3.22%	2.58%	2.72%	1.37%	1.02%	1.02%
# Best results	43	60	55	56	62	65
p-Value (< no SA)	-	<0.000001	<0.000001	-	<0.000001	<0.000001
p-Value (< SA_{conv}=0)	>0.999999	-	0.949769	>0.999999	-	0.641107
p-Value (< SA_{conv}=1)	>0.999999	0.050273	-	>0.999999	0.359077	-

Bounding Extension

For the evaluation of the bounding extension the probability of a bound computation on the visit of a trie node is set to 5% and as already stated in Section 5.6.4 the coarsest approximation of the DP is used for computing lb_1.

To investigate the impact of the bounding extension on the algorithm first it is determined how successful the bound computations are. In this context *successful* means that after the bound computation the subtrie could actually be cut off, i.e., the computed lower bound on this partial solution is already higher than the global upper bound given by the best solution found so far. To get an overview of the results the relative number of bound cuts is grouped by the trie levels which are divided into four quarters in which the bounds are computed.

Table 5.12: Successful bound cuts grouped by instance and part of the trie where they were computed for smaller instances.

$\theta = 2$	0-25[%]	26-50[%]	51-75[%]	76-100[%]	$\theta = 3$	0-25[%]	26-50[%]	51-75[%]	76-100[%]
P1	0.00	0.00	0.00	0.00	P1	0.00	0.00	0.00	0.00
P2	24.36	68.98	94.08	98.62	P2	23.10	85.17	97.84	100.00
P3	16.35	58.85	91.73	98.70	P3	7.04	55.04	96.43	100.00
P4	8.64	56.54	91.24	98.87	P4	2.11	56.65	98.20	99.77
P5	8.64	61.21	93.34	99.26	P5	26.20	69.65	95.93	98.44
P6	0.00	5.97	36.61	58.74	P6	0.66	10.95	38.90	52.22
P7	0.00	0.00	0.60	1.48	P7	0.00	1.51	12.33	22.65
B1	2.04	5.70	19.93	52.47	B1	0.55	5.82	42.14	75.16
A1	18.51	29.46	55.61	81.62	A1	16.36	61.68	92.43	99.06
A2	13.59	26.76	57.28	82.25	A2	10.27	63.27	94.08	99.53
A3	11.83	28.28	57.13	82.00	A3	6.73	65.43	92.35	99.12
A4	20.76	35.39	60.46	84.44	A4	29.18	52.41	80.32	95.01
B2	3.34	13.65	49.10	77.29	B2	27.58	61.50	88.67	96.21
B3	0.46	9.85	42.46	76.86	B3	22.92	44.71	75.78	89.48
A5	19.48	35.85	64.10	88.70	A5	10.22	52.40	84.36	96.48
A6	23.82	48.40	68.69	92.24	A6	16.09	55.16	85.11	96.95
A7	1.58	9.65	39.31	77.19	A7	14.90	36.45	72.13	87.82
A8	9.48	27.05	48.70	83.96	A8	33.88	62.06	85.18	96.72
B4	0.33	8.27	35.36	63.84	B4	21.58	34.48	68.48	84.00
A9	1.69	11.78	38.91	73.40	A9	11.22	41.05	77.66	93.15
A10	2.77	12.80	40.60	74.55	A10	17.42	44.38	77.05	91.96
B5	14.93	27.88	58.20	82.91	B5	59.44	82.71	97.39	99.78
P8	9.79	21.41	45.14	77.91	P8	23.69	53.42	74.92	94.77
B6	1.42	7.33	44.00	80.95	B6	29.05	45.74	74.50	91.72
B7	0.03	3.35	32.67	73.67	B7	31.74	57.65	77.59	94.08
A11	0.53	6.60	36.27	73.91	A11	2.17	24.03	54.29	84.30
B8	0.01	1.31	24.68	64.77	B8	1.60	15.38	52.98	80.71
A12	1.60	9.32	39.19	76.30	A12	3.32	21.33	47.93	81.74
A13	0.02	1.37	20.29	60.32	A13	0.32	6.15	36.58	73.74
B9	2.24	11.99	46.06	85.50	B9	53.73	63.08	81.45	96.77
B10	0.38	3.58	22.70	66.08	B10	3.68	15.53	47.88	81.54
P9	5.20	17.22	41.87	81.12	P9	14.92	42.35	59.49	90.87
A14	0.38	5.55	33.19	72.75	A14	6.72	19.84	46.65	76.94
A15	0.12	2.80	21.94	59.50	A15	3.73	14.90	44.38	75.62
B11	2.54	10.94	45.15	77.34	B11	10.57	19.04	57.26	84.89
B12	0.00	0.02	5.66	37.12	B12	0.00	1.24	29.63	60.06
P10	0.06	1.74	24.28	61.04	P10	0.32	3.76	33.82	65.34
P11	1.73	7.19	35.85	73.10	P11	7.13	19.16	46.49	76.01
P12	0.55	5.01	33.22	68.40	P12	5.18	16.18	45.38	74.69

Tables 5.12 and 5.13 show the number of successful bound cuts. Column 0-25 corresponds to the top quarter of the trie, column 26-50 to the second quarter and so on. As expected, this value increases on higher levels as more of the solution is already fixed. However, also for the lower levels this number is surprisingly high for some of the smaller instances. Even if the number of bound cuts on the first quarter of the trie is often less than 1% for the larger instances, one successful cut on a top level drastically reduces the search space, as a cut on level i removes $(m - i + 1)!$ solution candidates, which are not considered in later iterations anymore. On average the successful bound

Table 5.13: Successful bound cuts grouped by instance and part of the trie where they were computed for larger instances.

$\theta = 2$	0-25[%]	26-50[%]	51-75[%]	76-100[%]	$\theta = 3$	0-25[%]	26-50[%]	51-75[%]	76-100[%]
B13	0.20	4.94	32.80	66.08	B13	8.01	24.95	56.29	82.17
P13	0.03	1.56	21.64	60.19	P13	0.11	2.22	28.23	61.43
B14	0.68	7.17	40.21	74.88	B14	14.54	27.65	60.58	87.43
A16	0.42	4.92	30.92	71.31	A16	7.38	18.42	49.76	82.64
A17	0.03	1.57	20.36	64.63	A17	1.94	10.90	41.67	76.49
A18	0.25	3.45	25.48	64.52	A18	4.70	15.58	44.44	79.71
P14	0.15	2.83	21.75	59.34	P14	0.10	3.02	25.61	67.13
P15	0.00	0.16	7.35	39.64	P15	0.00	0.24	13.44	49.72
P16	1.45	8.56	32.95	69.56	P16	2.40	11.29	35.00	75.81
P17	1.75	8.84	33.94	70.90	P17	1.98	10.80	35.30	75.28
B15	0.61	5.57	31.63	66.08	B15	4.68	15.89	48.02	84.96
B16	0.02	0.91	19.40	60.83	B16	0.35	7.42	39.14	79.63
B17	0.00	0.09	7.49	36.96	B17	0.01	1.92	17.44	59.09
A19	0.01	0.92	16.79	58.12	A19	0.13	3.15	26.76	64.46
P18	0.10	2.77	24.13	62.89	P18	0.31	3.47	25.64	61.94
P19	0.00	0.34	10.13	47.70	P19	0.01	1.32	18.72	51.65
A20	0.17	2.92	23.89	64.39	A20	0.43	4.09	32.11	68.87
A21	0.13	2.75	23.80	68.27	A21	0.15	3.76	29.69	67.98
A22	0.02	1.33	16.55	54.39	A22	0.10	2.50	30.90	66.13
A23	0.00	0.62	10.36	45.26	A23	0.04	1.52	20.93	57.48
B18	0.03	1.99	16.62	53.05	B18	1.45	10.06	38.99	73.59
A24	0.02	1.13	15.13	53.13	A24	0.38	5.70	31.05	70.06
B19	0.86	6.54	24.01	60.17	B19	0.68	6.38	34.56	74.20
A25	0.34	4.48	28.94	63.92	A25	1.04	6.89	34.28	71.82
P20	0.10	1.81	22.24	61.10	P20	0.11	2.12	23.11	63.25
B20	0.00	0.71	13.62	51.56	B20	0.18	5.68	31.98	72.01
B21	0.07	2.27	19.61	58.78	B21	0.20	3.65	30.50	72.79
B22	0.01	0.80	14.97	54.15	B22	0.21	4.30	30.09	73.62
A26	0.15	2.65	23.24	65.45	A26	0.93	6.18	30.37	74.23
P21	0.07	1.38	17.69	60.77	P21	0.22	2.21	20.72	60.28
P22	4.94	11.84	38.76	79.72	P22	18.93	28.92	47.04	80.43
P23	1.86	8.18	34.45	77.95	P23	15.82	24.08	45.51	79.60
B23	0.01	0.37	9.88	51.07	B23	0.14	2.87	28.00	70.48
A27	0.01	0.55	9.56	45.63	A27	0.03	1.25	13.42	59.07
M1	0.41	2.51	14.68	45.68	M1	0.30	2.35	19.23	64.11
P24	9.74	14.03	36.32	80.61	P24	7.23	14.18	35.56	75.67
M2	0.12	0.34	12.33	46.12	M2	0.55	1.70	24.17	63.63
M3	0.20	0.61	8.73	33.08	M3	0.43	1.22	12.03	52.82
M4	0.00	0.00	0.00	17.81	M4	0.04	0.05	0.84	27.47
G1	0.00	0.00	0.00	8.43	G1	0.00	0.00	0.00	13.10

cuts for instances with $\theta = 2$ are, starting from the first quarter, 3.22%, 10.42%, 31.75%, and 64.47% and for the instances with $\theta = 3$, 8.25%, 22.55%, 47.79%, 75.31%.

Optimal Solutions

Using the solution archive and the bounding extension within the GA has the side effect to enhance the algorithm into an exact bounded enumeration method. This is basically a theoretic result but if the computed bounds are strong enough it may be sufficient to solve smaller instances to proven optimality. Therefore, a set of experiments is conducted in which the global run-time was not limited but instead a memory limit of 20 GB is used.

Table 5.14: List of the instances which could be solved optimally within the memory limit of 20 GB with and without the bounding extension of the solution archive.

		GASA	GASA +bounding			GASA	GASA +bounding
$\theta = 2$	obj*	t[s]	t[s]	$\theta = 3$	obj*	t[s]	t[s]
P1	245.34	1.16	0.86	P1	170.37	0.06	0.06
P2	146.82	63.56	22.30	P2	112.10	0.11	0.08
P3	149.02	63.34	31.48	P3	117.31	0.13	0.11
P4	160.48	756.62	282.02	P4	117.07	0.13	0.11
P5	161.36	756.54	300.48	P5	111.19	0.72	0.46
P6	323.59	1445.44	1335.61	P6	245.83	6.31	7.20
P7	312.48	13334.70	14126.00	P7	183.59	0.98	0.79
				B1	355.73	782.63	721.70
				A1	386.91	835.08	288.87
				A2	318.03	842.92	332.92
				A3	364.59	832.20	306.27
				A4	419.12	10697.70	2538.82
				B2	363.09	10595.70	1959.18
				B3	500.87	10740.90	3144.64
				A5	399.90	10702.10	3802.57
				A8	371.80	-	13415.50
				B5	281.48	-	6535.06

Table 5.14 shows the instances, along with their optimal objective values and the needed time, which could be solved within the memory limit with and without the bounding extension. We observe that all instances with up to 12 clusters could be solved optimally. The largest instance which could be solved was B5 with $\theta = 3$, 39 nodes, and 13 clusters, which is solved in less than two hours. For most of the solved instances the bounding extension is able to reduce the needed run-time by up to a factor of more than 5.0 (for instance B2) and for the largest two instances it is even able to find the optimal solution whereas the GA with the solution archive alone could not.

Final Results

Finally, the GASA and the GASA with the bounding extension are compared to each other and the proposed VNS. All algorithms are terminated after 300 seconds and in Tables 5.15 and 5.16 the average objective values over 30 runs and the corresponding standard deviations are shown. As in Section 5.7.3 bold values indicate that the corresponding algorithm performed significantly better than the other two methods on a 1% error level according to a one-sided paired Wilcoxon rank sum test.

Table 5.15: Results of the GASA with and without the bounding extension and the VNS for smaller instances.

	VNS		GASA		GASA + bounding			VNS		GASA		GASA + bounding	
$\theta = 2$	$\overline{obj}$	sd	$\overline{obj}$	sd	$\overline{obj}$	sd	$\theta = 3$	$\overline{obj}$	sd	$\overline{obj}$	sd	$\overline{obj}$	sd
P1	245.34	0.00	245.34	0.00	245.34	0.00	P1	170.37	0.00	170.37	0.00	170.37	0.00
P2	146.82	0.00	146.82	0.00	146.82	0.00	P2	112.10	0.00	112.10	0.00	112.10	0.00
P3	149.02	0.00	149.02	0.00	149.02	0.00	P3	117.31	0.00	117.31	0.00	117.31	0.00
P4	160.48	0.00	160.48	0.00	160.48	0.00	P4	117.07	0.00	117.07	0.00	117.07	0.00
P5	161.36	0.00	161.36	0.00	161.36	0.00	P5	111.19	0.00	111.19	0.00	111.19	0.00
P6	323.95	0.91	**323.59**	0.00	**323.59**	0.00	P6	245.83	0.00	245.83	0.00	245.83	0.00
P7	312.51	0.00	312.51	0.01	312.51	0.01	P7	183.59	0.00	183.59	0.00	183.59	0.00
B1	419.91	0.00	419.91	0.00	419.91	0.00	B1	355.73	0.00	355.73	0.00	355.73	0.00
A1	521.92	5.73	520.04	0.00	520.04	0.00	A1	386.91	0.00	386.91	0.00	386.91	0.00
A2	455.34	0.25	**455.15**	0.00	**455.15**	0.00	A2	318.03	0.00	318.03	0.00	318.03	0.00
A3	468.76	0.08	**467.95**	0.00	**467.95**	0.00	A3	364.59	0.00	364.59	0.00	364.59	0.00
A4	498.15	0.00	498.15	0.00	498.15	0.00	A4	419.12	0.00	419.12	0.00	419.12	0.00
B2	466.80	0.00	466.80	0.00	466.80	0.00	B2	363.09	0.00	363.09	0.00	363.09	0.00
B3	619.24	0.00	619.24	0.00	619.24	0.00	B3	501.39	0.21	**500.87**	0.00	**500.87**	0.00
A5	506.95	0.00	**506.40**	0.79	**506.46**	0.77	A5	399.90	0.00	399.90	0.00	399.90	0.00
A6	447.86	0.00	447.86	0.00	447.86	0.00	A6	359.13	0.00	359.13	0.00	359.13	0.00
A7	608.39	0.86	**590.59**	3.49	**589.70**	0.71	A7	430.99	0.00	430.99	0.00	430.99	0.00
A8	481.98	0.00	**481.97**	0.00	**481.97**	0.00	A8	371.80	0.00	371.80	0.00	371.80	0.00
B4	**479.44**	0.69	479.92	0.00	479.92	0.00	B4	388.84	1.05	**386.25**	0.30	**386.25**	0.30
A9	567.91	0.00	**567.41**	0.00	**567.41**	0.00	A9	371.41	0.00	371.41	0.00	371.41	0.00
A10	561.25	0.00	**560.61**	0.40	**560.73**	0.44	A10	417.78	0.33	**416.03**	0.00	**416.03**	0.00
B5	356.48	0.00	**356.43**	0.00	**356.43**	0.00	B5	281.48	0.00	281.48	0.00	281.48	0.00
P8	296.44	0.00	**296.36**	0.05	**296.33**	0.00	P8	214.75	0.00	214.75	0.00	214.75	0.00
B6	483.26	0.00	483.22	0.15	483.20	0.18	B6	404.26	0.00	404.26	0.00	404.26	0.00
B7	487.02	2.20	**485.46**	0.00	**485.46**	0.00	B7	347.65	0.00	347.65	0.00	347.65	0.00
A11	627.86	0.00	627.86	0.00	627.86	0.00	A11	508.98	0.00	**506.60**	1.09	**505.32**	0.59
B8	563.96	0.00	**563.95**	0.00	**563.95**	0.00	B8	402.02	0.00	402.02	0.00	402.02	0.00
A12	621.23	0.00	621.23	0.00	621.23	0.00	A12	478.22	0.00	478.22	0.00	478.22	0.00
A13	692.89	0.00	692.89	0.00	692.89	0.00	A13	488.02	0.00	488.02	0.00	488.02	0.00
B9	502.02	0.00	502.02	0.00	502.02	0.00	B9	419.35	0.79	**417.03**	0.00	**417.03**	0.00
B10	482.91	0.00	**482.91**	0.00	**482.91**	0.00	B10	358.99	0.00	358.99	0.00	358.99	0.00
P9	340.48	0.00	340.50	0.06	340.49	0.04	P9	239.36	0.00	239.36	0.00	239.36	0.00
A14	623.01	1.16	622.84	1.32	622.58	1.31	A14	471.34	0.50	**466.82**	2.22	**465.62**	0.00
A15	686.42	0.00	686.42	0.00	686.42	0.00	A15	462.55	0.00	462.55	0.00	462.55	0.00
B11	454.09	0.00	454.09	0.00	454.09	0.00	B11	398.38	0.00	398.38	0.00	398.38	0.00
B12	923.53	0.00	923.53	0.00	923.53	0.00	B12	604.66	1.64	**600.64**	0.05	**600.62**	0.06
P10	431.22	1.31	**422.24**	1.50	**421.36**	1.33	P10	302.37	0.00	302.37	0.00	302.37	0.00
P11	354.47	0.00	354.47	0.00	354.47	0.00	P11	261.31	0.00	**261.31**	0.00	**261.31**	0.00
P12	377.66	0.00	377.62	0.21	377.62	0.19	P12	273.27	0.00	**268.91**	0.00	**268.91**	0.00

The results show that both GASA and GASA + bounding outperform the VNS on most of the instances. Specifically, GASA found on 35 instances with $\theta = 2$ and on 28 instances with $\theta = 3$ significantly better results and GASA + bounding on 38 instances with $\theta = 2$ and on 30 instances with $\theta = 3$ better results than the VNS. The VNS, however, achieved only in five instances better results than any of the other two algorithms. When we compare GASA with GASA + bounding in Tables 5.15 and 5.16

Table 5.16: Results of the GASA with and without the bounding extension and the VNS for larger instances.

	VNS		GASA		GASA + bounding			VNS		GASA		GASA + bounding	
$\theta = 2$	$\overline{obj}$	sd	$\overline{obj}$	sd	$\overline{obj}$	sd	$\theta = 3$	$\overline{obj}$	sd	$\overline{obj}$	sd	$\overline{obj}$	sd
B13	682.27	1.32	682.70	0.00	682.70	0.00	B13	513.02	0.00	**513.02**	0.00	**513.02**	0.00
P13	451.79	0.00	451.79	0.00	451.64	0.52	P13	313.41	0.00	313.41	0.00	313.41	0.00
B14	458.87	0.14	**458.39**	0.34	**458.57**	0.19	B14	360.50	0.00	360.50	0.00	360.50	0.00
A16	636.61	1.89	**632.78**	2.79	**631.10**	5.99	A16	443.87	0.00	**443.87**	0.00	**443.87**	0.00
A17	721.48	2.64	721.54	3.63	720.96	4.12	A17	490.54	0.00	490.54	0.00	490.54	0.00
A18	730.53	4.91	**718.11**	0.06	**718.12**	0.00	A18	474.05	0.00	474.05	0.00	474.05	0.00
P14	424.54	0.19	**420.69**	0.00	**420.69**	0.00	P14	316.65	0.00	**313.37**	2.18	**312.14**	0.87
P15	560.92	0.00	**560.86**	0.30	**560.62**	0.67	P15	395.57	0.76	396.20	0.00	396.16	0.27
P16	370.43	5.70	**361.87**	0.00	**361.87**	0.00	P16	274.22	0.00	274.22	0.00	274.22	0.00
P17	362.21	0.00	**362.04**	0.02	**362.03**	0.00	P17	276.33	0.00	**276.33**	0.00	**276.33**	0.00
B15	478.10	0.00	**474.92**	0.00	**474.92**	0.00	B15	358.85	0.19	**357.91**	0.26	**357.84**	0.00
B16	779.43	0.19	**778.60**	0.91	**778.69**	1.13	B16	567.66	0.23	**564.53**	0.55	**564.35**	0.00
B17	967.33	0.00	967.33	0.00	967.33	0.00	B17	692.38	3.10	**681.73**	11.45	**674.93**	8.63
A19	816.39	0.00	**815.86**	0.00	**815.86**	0.00	A19	617.87	4.70	616.92	5.72	615.61	5.95
P18	455.26	0.00	**452.86**	0.00	**452.86**	0.00	P18	328.89	0.00	**328.83**	0.05	**328.79**	0.00
P19	572.08	0.00	572.08	0.00	572.07	0.09	P19	372.63	0.00	372.63	0.00	372.63	0.00
A20	662.94	0.00	**653.64**	9.18	**648.92**	7.85	A20	482.51	0.00	482.51	0.00	482.51	0.00
A21	755.77	0.00	**755.75**	0.00	**755.75**	0.00	A21	617.56	0.00	617.56	0.00	617.56	0.00
A22	**830.79**	0.00	830.88	0.53	830.88	0.53	A22	611.54	0.00	611.54	0.00	611.54	0.00
A23	946.39	0.00	946.39	0.00	946.39	0.00	A23	666.46	0.00	**664.95**	1.64	**663.65**	1.12
B18	852.87	0.00	852.87	0.00	852.87	0.00	B18	604.68	0.06	604.66	0.09	**604.59**	0.11
A24	837.31	0.00	837.31	0.00	837.31	0.00	A24	563.57	3.39	564.46	0.00	564.02	2.44
B19	514.92	0.00	514.92	0.00	514.92	0.00	B19	457.24	0.00	**457.24**	0.00	**457.24**	0.00
A25	712.74	0.00	**712.14**	0.00	**712.14**	0.00	A25	525.03	0.00	525.03	0.00	525.03	0.00
P20	501.39	0.00	501.34	0.30	501.39	0.00	P20	378.53	0.00	**378.48**	0.00	**378.48**	0.00
B20	818.42	0.00	818.42	0.00	818.42	0.00	B20	627.36	0.00	627.22	0.35	627.22	0.35
B21	674.95	0.00	**672.40**	0.00	**672.40**	0.00	B21	561.71	0.00	561.71	0.00	561.71	0.00
B22	738.48	0.00	738.48	0.00	738.48	0.00	B22	539.10	1.63	539.25	1.46	538.88	1.74
A26	711.19	0.00	707.90	6.07	708.78	5.48	A26	523.77	0.00	523.77	0.00	523.77	0.00
P21	504.96	0.00	504.96	0.00	504.96	0.00	P21	386.07	0.17	**385.82**	0.19	**385.69**	0.00
P22	394.16	0.20	**392.81**	1.01	**392.46**	1.12	P22	310.40	0.00	310.40	0.00	310.40	0.00
P23	409.93	0.00	409.93	0.00	409.93	0.00	P23	310.40	0.00	**310.40**	0.00	**310.40**	0.00
B23	**840.94**	0.52	863.99	18.93	**839.53**	0.00	B23	620.11	0.00	**620.11**	0.00	**620.11**	0.00
A27	1064.86	0.00	**1049.26**	0.71	**1049.13**	0.98	A27	757.24	1.70	**751.46**	6.18	**743.00**	0.00
M1	590.38	0.03	**569.15**	19.41	**544.82**	0.55	M1	467.14	0.03	**463.48**	8.51	**463.96**	0.00
P24	462.18	0.83	**458.43**	0.35	**458.31**	0.23	P24	371.93	0.00	371.93	0.00	371.93	0.00
M2	**769.86**	0.00	860.22	112.30	**745.93**	0.01	M2	565.77	0.00	**561.00**	18.40	**545.87**	1.25
M3	**732.85**	0.80	983.18	209.26	**692.56**	0.41	M3	530.05	3.73	532.71	44.21	**506.51**	20.49
M4	3372.85	97.94	**1680.24**	605.77	**1398.44**	69.30	M4	2338.33	45.86	**834.52**	158.87	**646.41**	17.22
G1	13817.9	252.8	**8669.10**	741.93	**10741.71**	338.27	G1	8544.93	161.56	**3843.03**	324.70	**4435.65**	345.10

it is not clear which should be preferred. Therefore, an overall summary of all three algorithms is given in Table 5.17, which is constructed like the previous summaries. Although GASA + bounding has a higher arithmetic mean than GASA, the geometric mean, the average gap to the BKS, and the number of best results are better which is also reflected in the statistical tests which showed that GASA + bounding performs significantly better on the given problem instances. Additionally it has the property that for smaller instances it can actually find proven optimal solutions which makes GASA + bounding the superior algorithm.

Table 5.17: Summary of the performance of the GASA with and without the bounding extension compared to the VNS.

	Instances with $\theta = 2$			Instances with $\theta = 3$		
	VNS	GASA	GASA + bounding	VNS	GASA + bounding	GASA
$\overline{obj}$	758.17	674.57	691.39	539.00	459.77	464.08
$\overline{obj_g}$	549.48	542.90	539.14	407.29	397.42	396.28
$\overline{gap}$ to BKS	5.66%	2.58%	1.70%	6.14%	1.02%	0.72%
# Best results	31	53	66	48	56	74
p-Value (< VNS)	-	<0.000001	<0.000001	-	<0.000001	<0.000001
p-Value (< GASA)	>0.999999	-	<0.000001	>0.999999	-	0.000948
p-Value (< GASA+bounding)	>0.999999	>0.999999	-	>0.999999	0.999053	-

5.8 Conclusions

In this chapter solution algorithms for the generalized vehicle routing problem with stochastic demands using the (optimal) preventive restocking strategy are presented. As neither the GVRPSD nor the VRPSD with preventive restocking have been considered by exact algorithms so far, an initial attempt is made. Results showed that this approach is effective for solving smaller instances up to about 40 nodes and 13 clusters with $E[nr] \leq 2$. Having obtained optimal results or at least bounds for the benchmark instances a step towards heuristically solving the GVRPSD is made by developing an efficient solution evaluation method by using a multi-level evaluation scheme which is used by a VNS. The VNS further used concepts from both the related VRPSD and the GTSP and the results showed that especially the ML-ES was able to significantly reduce the needed run-time for the solution evaluations. The third proposed solution algorithm for the GVRPSD uses a complete trie-based solution archive in combination with an evolutionary algorithm. Within this EA both the ML-ES and the neighborhood structures from the VNS are used. The basic concept of the solution archive was extended to permutation encodings and a bounding extension was introduced such that the considered solution space could be significantly pruned. All presented algorithms, components, and configurations were extensively analyzed to show their contribution to the resulting solution quality. The results show that the GASA method is superior to the VNS and even able to solve smaller instances optimally with the help of the solution archive.

The computational study for the GVRPSD showed that also for this problem, the solution archive and especially the bounding extension is beneficial to the final solution quality and improved the basic algorithms. The combination of the SA and the other employed method led to a new state-of-the-art heuristic solution method for the GVRPSD which showed find optimal or near-optimal results.

Ideas for future work include the application of the methods, especially the ML-ES and the SA, to similar problems, e.g., when a maximum route duration is given such that more than one tour has to be planned. Also a more in-depth analysis when the GASA is applied to the VRPSD could be interesting. For the solution archive another promising research direction is the utilization of the computed bounds for making a more intelligent branching decision.

CHAPTER 6

Conclusions and Future Work

Metaheuristics and, more generally, stochastic search methods for solving combinatorial optimization problems have the common weakness that they usually do not keep track of their search history. Therefore, already evaluated solution candidates are frequently revisited and unnecessarily reevaluated. Especially in evolutionary algorithms such duplicate solutions can cause problems by reducing the diversity and therefore the algorithm could potentially converge prematurely. To overcome this problem, in this thesis an efficient method for recognizing and converting duplicate solutions occurring in stochastic search procedures for combinatorial optimization problems was developed, extended, and applied to two relevant problem domains with real-world applications. Such a *complete trie-based solution archive* is especially useful in combination with an evolutionary algorithm to reduce or avoid the risk of premature convergence and to maintain diversity in the population. Also for other metaheuristic algorithms a solution archive can be beneficial to avoid possibly expensive, time-consuming, and unnecessary re-evaluations of already generated solution candidates.

In this thesis we extended the basic idea of complete trie-based solution archives to different, constrained solution representations. We extended the trie structure from simple unconstrained binary strings to constrained bit vectors and permutations. For the considered problems, a particular trie structure and conversion method was developed which can be applied also to other problems having the same or similar solution representations. Advanced concepts like trie randomization and bound computations were applied and their effectiveness empirically shown.

The first considered problem belongs to the family of competitive facility location problems and an evolutionary algorithm using such a solution archive was developed and applied to several different scenarios of the problem. We showed that some of these scenarios, especially the proportional customer behavior, were much more difficult to solve than the easier ones like the binary customer behavior. For all scenarios the solution archive was able to significantly increase the final solution quality of the algorithm and the developed evolutionary algorithm was able to outperform previous state-of-the-art algorithms. The evolutionary algorithm with solution archive found new best results for more than 45 Euclidean instances from the literature using binary customer behavior and more than 30 new best results for instances with proportional customer behavior. Therefore, a new state-of-the-art heuristic solution algorithm for the considered type of competitive facility location problems was found which profoundly relies on the complete trie-based solution archive.

The second considered problem is the generalized vehicle routing problem with stochastic demands and preventive restocking which in this form has not yet been considered in the literature. In this thesis a new exact branch-and-cut algorithm was developed which was able to solve 11 out of 27 smaller benchmark instances to optimality. Using a variable neighborhood search and an innovative multi-level evaluation scheme also larger instances could be tackled and for all instances with a known optimal solution value, it was able to find an optimal solution, usually in only a fraction of the run-time. The second developed metaheuristic for this problem is an evolutionary algorithm using a solution archive and a variable neighborhood descent algorithm as subprocedure. Also for this problem, by employing the solution archive the results significantly improved. With the bounding extension of the solution archive the algorithm was frequently able to cut off a large part of the search space such that smaller instances could even be solved to optimality which was not possible in reasonable time without the bounding extension. Specifically, 24 out of the whole set of 158 benchmark instances could be solved to optimality using the bounding extension. Also for this problem a new state-of-the-art algorithm was found which also relies on the solution archive and the developed extensions to it.

Although this thesis showed that complete trie-based solution archives are able to boost the performance of metaheuristics for some problems, they are not always suitable as standard extension to any metaheuristic for every problem. If there does not exist (or cannot be found) a reasonable compact solution representation of a problem, the memory consumption of the solution archive can be too high to be usable. Even if the solution representation is compact but the instance size is too big, memory problems can occur because the height of the trie, hence the size of the solution archive, depends on the length of the solution. For the used solution representations, i.e., a constrained binary string and permutations, the solution archive was beneficial and the memory consumption was reasonably low. However, especially for the permutation representation, we could already observe that for larger instances the memory consumption increased strongly, so we would run into problems when considering larger instances and / or longer run-times. Although the time overhead of the solution archive is negligible when the solution evaluation is time-consuming, as in our considered problems, for problems with a fast solution evaluation method, the time consumption of the archive can be more substantial and the overhead of the archive might not outweigh its advantages which could lead to an overall slower, and therefore worse, algorithm.

For many practical relevant combinatorial optimization problems, however, the use of complete trie-based solution archives is still a viable approach which can lead, as we have seen in this thesis, to huge improvements of the performance of the solution algorithm.

6.1 Future Work

There are several promising possibilities for future work on complete trie-based solution archives. First, we have seen the application of solution archives on various different solution representations. It would be interesting to find a more fine-grained distinction for which solution representations solution archives are fruitful and for which they are not and to generally investigate the application of solution archives to other kinds of problems.

With the current knowledge about solution archives it is still an unsolved challenge how to use a solution archive with a solution representation for strongly constrained problems with complex side constraints. In particular, given a solution representation of the form $(x_1, \ldots, x_m)$ and the partial solution $(x_1, \ldots, x_l)$ with $l < m$. For creating a trie node for variable x_{l+1} we need to know the feasible values for x_{l+1} during insertion, otherwise the conversion method could create infeasible solutions. If the computation of those values is time-consuming, e.g., needs more than constant time, the time consumption of the insertion method will in general be huge.

Another open challenge of solution archives are large domains of the decision variables. As the size of a trie node is determined by the size of the domain, the memory consumption would be huge. To overcome this memory problem one can use a *linked trie* instead of an indexed trie, in which the entries are created on the fly when needed and not beforehand. A drawback of this approach is, however, that the access time of the entries would not be constant anymore which would increase the overall time consumption of the insertion and conversion method. For some problems this could still be a viable approach but this has not been accessed in this thesis and could be investigated in future work.

As already mentioned in the last section even if the solution representation is compact the memory consumption can be too high if large instances are considered. An option to significantly reduce the memory usage in this case would be to split the solution into multiple parts and use one trie for each of them. By applying this technique, however, the solution archive is not complete anymore. Even if the solution archives classifies a solution as a duplicate in each of the tries, it does not need to be the case that it has already been considered before because each individual part can have appeared in different solutions. Investigating such a split solution archive nevertheless seems promising for future work when facing large instances.

Finally, it appears interesting to investigate the bounding extension of the solution archive further. So far only dual bounds on partial solutions were considered in the bounding extension. Frequently, it is possible to come up with a fast method for computing primal bounds. Especially in the second considered problem of this thesis such primal bounds can rather easily be computed. These bounds may indicate promising solution subspaces which might further guide the metaheuristic search.

Bibliography

[1] E. Aarts and J. K. Lenstra. *Local search in combinatorial optimization.* Princeton University Press, 2003.

[2] H. M. Afsar, C. Prins, and A. C. Santos. Exact and heuristic algorithms for solving the generalized vehicle routing problem with flexible fleet size. *International Transactions in Operational Research*, 21(1):153–175, 2014.

[3] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102, 2002.

[4] E. Alekseeva and Y. Kochetov. Matheuristics and exact methods for the discrete $(r|p)$-centroid problem. In El-Ghazali Talbi, editor, *Metaheuristics for Bi-level Optimization*, volume 482 of *Studies in Computational Intelligence*, pages 189–219. Springer Berlin Heidelberg, 2013.

[5] E. Alekseeva, N. Kochetova, Y. Kochetov, and A. Plyasunov. A hybrid memetic algorithm for the competitive p-median problem. In Natalia Bakhtadze and Alexandre Dolgui, editors, *Information Control Problems in Manufacturing*, volume 13, pages 1533–1537. International Federation of Automatic Control, 2009.

[6] E. Alekseeva, N. Kochetova, Y. Kochetov, and A. Plyasunov. Heuristic and exact methods for the discrete $(r|p)$-centroid problem. In Peter Cowling and Peter Merz, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6022 of *LNCS*, pages 11–22. Springer Berlin Heidelberg, 2010.

[7] B. A. Alireza and R. Z. Farahani. Facility location dynamics: An overview of classifications and applications. *Computers & Industrial Engineering*, 62(1):408–420, 2012.

[8] M. G. Ashtiani. Competitive location: a state-of-art review. *International Journal of Industrial Engineering Computations*, 7(1):1, 2016.

[9] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA journal on computing*, 6(2):126–140, 1994.

[10] T. Bektaş, G. Erdoğan, and S. Røpke. Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. *Transportation Science*, 45(3):299–316, 2011.

[11] D. J. Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585, 1992.

[12] D. J. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[13] J. Bhadury, H. A. Eiselt, and J. H. Jaramillo. An alternating heuristic for medianoid and centroid problems in the plane. *Computers & Operations Research*, 30(4):553–565, 2003.

[14] L. Bianchi, M. Birattari, M. Chiarandini, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms*, 5(1):91–110, 2006.

[15] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[16] B. Biesinger. A hybrid evolutionary algorithm for the discrete $(r|p)$-centroid problem. Austrian Workshop on Metaheuristics 9, Vienna, Austria, 2013.

[17] B. Biesinger, B. Hu, and G. R. Raidl. An evolutionary algorithm for the leader-follower facility location problem with proportional customer behavior. In *Conference Proceedings of Learning and Intelligent Optimization Conference (LION 8)*, volume 8426 of *LNCS*, pages 203–217. Springer, 2014.

[18] B. Biesinger, B. Hu, and G. R. Raidl. A hybrid genetic algorithm with solution archive for the discrete $(r|p)$-centroid problem. *Journal of Heuristics*, 21(3):391–431, 2015.

[19] B. Biesinger, B. Hu, and G. R. Raidl. An integer L-shaped method for the generalized vehicle routing problem with stochastic demands. In *7th International Network Optimization Conference, INOC*, 2015. To appear.

[20] B. Biesinger, B. Hu, and G. R. Raidl. Models and algorithms for competitive facility location problems with different customer behavior. *Annals of Mathematics and Artificial Intelligence*, 76(1):93–119, 2015.

[21] B. Biesinger, B. Hu, and G. R. Raidl. A variable neighborhood search for the generalized vehicle routing problem with stochastic demands. In Gabriela Ochoa and Francisco Chicano, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2015*, volume 9026 of *LNCS*, pages 48–60. Springer, 2015.

[22] B. Biesinger, B. Hu, and G. R. Raidl. A genetic algorithm in combination with a solution archive for solving the generalized vehicle routing problem with stochastic demands. 2016. submitted to a journal.

[23] B. Biesinger, B. Hu, and G. R. Raidl. A memetic algorithm for competitive facility location problems. In Natalie Jane de Vries and Pablo Moscato, editors, *Business and Consumer Analytics: New Directions (Vol1)*, pages 1–23. 2016. To appear.

[24] C. Blum. Beam-ACO—hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.

[25] C. Blum, P. Pinacho, M. López-Ibáñez, and J. A. Lozano. Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. *Computers & Operations Research*, 68(0):75–88, 2016.

[26] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[27] C. Blum and G. R. Raidl. *Hybrid Metaheuristics.* Springer International Publishing, 2016.

[28] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[29] M. Boschetti and V. Maniezzo. Benders decomposition, lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15(3):283–312, 2009.

[30] M. Boschetti, V. Maniezzo, and M. Roffilli. Decomposition techniques as metaheuristic frameworks. In *Matheuristics*, pages 135–158. Springer, 2009.

[31] S. Bouamama, C. Blum, and A. Boukerram. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. *Applied Soft Computing*, 12(6):1632–1639, 2012.

[32] C. Campos-Rodríguez, J. A. Moreno-Pérez, H. Noltemeier, and D. R. Santos-Peñate. Two-swarm pso for competitive location problems. In N. Krasnogor, M. B. Melián-Batista, J. A. Moreno-Pérez, J. M. Moreno-Vega, and D. A. Pelta, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2008)*, volume 236 of *Studies in Computational Intelligence*, pages 115–126. Springer Berlin Heidelberg, 2009.

[33] C. Campos-Rodríguez, J. A. Moreno-Pérez, and D. R. Santos-Peñate. Particle swarm optimization with two swarms for the discrete $(r|p)$-centroid problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory (EUROCAST 2011)*, volume 6927 of *LNCS*, pages 432–439. Springer Berlin Heidelberg, 2012.

[34] C. H. Christiansen and J. Lysgaard. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters*, 35(6):773–781, 2007.

[35] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.

[36] C. Darwin. *On the Origin of Species.* John Murray, 1859.

[37] I. A. Davydov, Y. Kochetov, and E. Carrizosa. VNS heuristic for the $(r|p)$-centroid problem on the plane. *Electronic Notes in Discrete Mathematics*, 39:5–12, December 2012.

[38] I. A. Davydov, Y. Kochetov, N. Mladenovic, and D. Urosevic. Fast metaheuristics for the discrete $(r|p)$-centroid problem. *Automation and Remote Control*, 75(4):677–687, 2014.

[39] I. A. Davydov, Y. Kochetov, and A. Plyasunov. On the complexity of the $(r|p)$-centroid problem in the plane. *Top*, 22(2):614–623, 2014.

[40] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[41] G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[42] M. Dorigo, M. Birattari, and T. Stützle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[43] T. Drezner. Optimal continuous location of a retail facility, facility attractiveness, and market share: An interactive model. *Journal of Retailing*, 70(1):49–64, 1994.

[44] T. Drezner. Location of multiple retail facilities with limited budget constraints — in continuous space. *Journal of Retailing and Consumer Services*, 5(3):173–184, 1998.

[45] T. Drezner, Z. Drezner, and S. Salhi. Solving the multiple competitive facilities location problem. *European Journal of Operational Research*, 142(1):138–151, 2002.

[46] Z. Drezner. On a modified one-center model. *Management Science*, 27(7):848–851, 1981.

[47] H. A. Eiselt. *Foundations of Location Analysis*, chapter Equilibria in Competitive Location Models, pages 139–162. Springer US, Boston, MA, 2011.

[48] H. A. Eiselt and G. Laporte. Sequential location problems. *European Journal of Operational Research*, 96(2):217–231, 1997.

[49] H. A. Eiselt, V. Marianov, and T. Drezner. *Location Science*, chapter Competitive Location Models, pages 365–398. Springer International Publishing, Cham, 2015.

[50] R. Z. Farahani, M. Hekmatfar, A. B. Arabani, and E. Nikbakhsh. Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering*, 64(4):1096–1109, 2013.

[51] J. Fernández, B. Pelegrı, F. Plastria, and B. Tóth. Solving a huff-like competitive location and design model for profit maximization in the plane. *European Journal of Operational Research*, 179(3):1274–1287, 2007.

[52] M. Fischetti and A. Lodi. Local branching. *Mathematical programming*, 98(1-3):23–47, 2003.

[53] M. Fischetti, J. Salazar-González, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–123, 1995.

[54] M. Fischetti, J. Salazar-González, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[55] J. J. H. Forrest, J. P. H. Hirst, and J. A. Tomlin. Practical solution of large mixed integer programming problems with umpire. *Management Science*, 20(5):736–773, 1974.

[56] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 22. W.H. Freeman and Company, New York, 2000.

[57] M. Gendreau, G. Laporte, and R. Séguin. An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transportation science*, 29(2):143–155, 1995.

[58] M. Gendreau, G. Laporte, and R. Séguin. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44(3):469–477, 1996.

[59] M. Gendreau and J. Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213, 2005.

[60] M. Gendreau and J. Potvin. *Handbook of metaheuristics*, volume 2. Springer, 2010.

[61] M. Gendreau and J. Potvin. Tabu search. In *Handbook of Metaheuristics*, pages 41–59. Springer, 2010.

[62] G. Ghiani and G. Improta. An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research*, 122(1):11–17, 2000.

[63] D. E. Goldberg. *Genetic algorithms in search optimization and machine learning*, volume 412. Addison-Wesley Reading Menlo Park, 1989.

[64] J. C. Goodson, J. W. Ohlmann, and B. W. Thomas. Cyclic-order neighborhoods with application to the vehicle routing problem with stochastic demand. *European Journal of Operational Research*, 217(2):312–323, 2012.

[65] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.

[66] M. H. Hà, N. Bostel, A. Langevin, and L. Rousseau. An exact algorithm and a metaheuristic for the generalized vehicle routing problem with flexible fleet size. *Computers & Operations Research*, 43(0):9–19, 2014.

[67] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, 1964.

[68] S. L. Hakimi. On locating new facilities in a competitive environment. *European Journal of Operational Research*, 12(1):29–35, 1983.

[69] P. Hansen, N. Mladenović, J. Brimberg, and J. A. Moreno-Pérez. Variable neighborhood search. In *Handbook of Metaheuristics*, pages 61–87. Springer, 2010.

[70] C. Hjorring and J. Holt. New optimality cuts for a single-vehicle stochastic routing problem. *Annals of Operations Research*, 86(0):569–584, 1999.

[71] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[72] H. Hotelling. Stability in competition. *The Economic Journal*, 39(153):41–57, 1929.

[73] B. Hu and G. R. Raidl. An evolutionary algorithm with solution archive for the generalized minimum spanning tree problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, volume 6927 of *LNCS*, pages 287–294. Springer, 2012.

[74] B. Hu and G. R. Raidl. An evolutionary algorithm with solution archives and bounding extension for the generalized minimum spanning tree problem. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 393–400, Philadelphia, PA, USA, 2012. ACM Press.

[75] D. L. Huff. Defining and estimating a trading area. *The Journal of Marketing*, 28(3):34–38, 1964.

[76] O. Jabali, W. Rei, M. Gendreau, and G. Laporte. Partial-route inequalities for the multi-vehicle routing problem with stochastic demands. *Discrete Applied Mathematics*, 177(0):121–136, 2014.

[77] I. Kara and T. Bektaş. Integer linear programming formulation of the generalized vehicle routing problem. In *Proc. of the 5-th EURO/INFORMS Joint International Meeting*, pages 6–10, 2003.

[78] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[79] R. M. Karp. Complexity of computer computations. chapter Reducibility among Combinatorial Problems, pages 85–103. Springer US, 1972.

[80] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.

[81] Y. Kochetov, N. Kochetova, and A. Plyasunov. A matheuristic for the leader-follower facility location and design problem. In H. Lau, P. Van Hentenryck, and G. R. Raidl, editors, *Proceedings of the 10th Metaheuristics International Conference (MIC 2013)*, pages 32/1–32/3, Singapore, 2013.

[82] J. Kratica. Improving performances of the genetic algorithm by caching. *Computers and artificial intelligence*, 18(3):271–283, 1999.

[83] D. Kress and E. Pesch. Sequential competitive location on networks. *European Journal of Operational Research*, 217(3):483–499, 2012.

[84] H. Küçükaydin, N. Aras, and I. K. Altınel. Competitive facility location problem with attractiveness adjustment of the follower: A bilevel programming model and its solution. *European Journal of Operational Research*, 208(3):206–220, 2011.

[85] G. Laporte and S. Benati. Tabu Search Algorithms for the $(r|X_p)$-medianoid and $(r|p)$-centroid Problems. *Location Science*, 2:193–204, 1994.

[86] G. Laporte and F. V. Louveaux. Solving stochastic routing problems with the integer L-shaped method. In T. G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 159–167. Springer, 1998.

[87] G. Laporte, F. V. Louveaux, and L. Van Hamme. An integer L-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research*, 50(3):415–423, 2002.

[88] G. Laporte, S. Nickel, and F. S. da Gama. *Location science*, volume 145. Springer, 2015.

[89] S. J. Louis and G. Li. Combining robot control strategies using genetic algorithms with memory. In *Evolutionary Programming VI*, LNCS, pages 431–441. Springer, 1997.

[90] M. Marinaki and Y. Marinakis. A glowworm swarm optimization algorithm for the vehicle routing problem with stochastic demands. *Expert Systems with Applications*, 46(0):145–163, 2016.

[91] Y. Marinakis, G. Iordanidou, and M. Marinaki. Particle swarm optimization for the vehicle routing problem with stochastic demands. *Applied Soft Computing*, 13(4):1693–1704, 2013.

[92] Y. Marinakis and M. Marinaki. Combinatorial expanding neighborhood topology particle swarm optimization for the vehicle routing problem with stochastic demands. In *Proceedings of the 15th annual Conference on Genetic and Evolutionary Computation*, pages 49–56. ACM, 2013.

[93] Y. Marinakis, M. Marinaki, and A. Migdalas. A hybrid clonal selection algorithm for the vehicle routing problem with stochastic demands. In *Learning and Intelligent Optimization*, pages 258–273. Springer, 2014.

[94] Y. Marinakis, M. Marinaki, and P. Spanou. A memetic differential evolution algorithm for the vehicle routing problem with stochastic demands. In I. Fister and I. F. Jr., editors, *Adaptation and Hybridization in Computational Intelligence*, volume 18 of *Adaptation, Learning, and Optimization*, pages 185–204. Springer, 2015.

[95] M. L. Mauldin. Maintaining diversity in genetic search. In *National Conference on Artificial Intelligence*, volume 19, pages 247–250. AAAI, William Kaufmann, 1984.

[96] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE, 1972.

[97] N. Mladenović, J. Brimberg, P. Hansen, and J. A. Moreno-Pérez. The *p*-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3):927–939, 2007.

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

[99] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1–68, 1989.

[100] H. Noltemeier, J. Spoerhase, and H. Wirth. Multiple voting location and single voting location on trees. *European Journal of Operational Research*, 181(2):654–667, 2007.

[101] I. M. Oliver, D. J. D. Smith, and J. R. C. Holland. Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms.* Hillsdale, NJ: L. Erlhaum Associates, 1987.

[102] P. S. Ow and T. E. Morton. Filtered beam search in scheduling†. *The International Journal Of Production Research*, 26(1):35–62, 1988.

[103] C. H. Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[104] P. H. Peeters and F. Plastria. Discretization results for the huff and pareto-huff competitive location models on networks. *Top*, 6(2):247–260, 1998.

[105] D. Pisinger and S. Ropke. Large neighborhood search. In *Handbook of Metaheuristics*, pages 399–421. Springer, 2010.

[106] F. Plastria. Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research*, 129(3):461–470, 2001.

[107] P. C. Pop, L. Fuksz, and A. H. Marc. A variable neighborhood search approach for solving the generalized vehicle routing problem. In M. Polycarpou, A. C. P. L. F. de Carvalho, J. Pan, M. Woźniak, H. Quintian, and E. Corchado, editors, *Hybrid Artificial Intelligence Systems*, volume 8480 of *LNCS*, pages 13–24. Springer, 2014.

[108] P. C. Pop, O. Matei, C. P. Sitar, and C. Chira. A genetic algorithm for solving the generalized vehicle routing problem. In E. Corchado, M. Graña Romay, and A. Manhaes Savio, editors, *Hybrid Artificial Intelligence Systems*, volume 6077 of *LNCS*, pages 119–126. Springer, 2010.

[109] J. Puchinger, G. R. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010.

[110] G. R. Raidl. A unified view on hybrid metaheuristics. In *Hybrid Metaheuristics*, pages 1–12. Springer, 2006.

[111] G. R. Raidl. Decomposition based hybrid metaheuristics. *European Journal of Operational Research*, 244(1):66–76, 2015.

[112] G. R. Raidl and B. Hu. Enhancing genetic algorithms by a trie-based complete solution archive. In Peter Cowling and Peter Merz, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6022 of *LNCS*, pages 239–251. Springer Berlin Heidelberg, 2010.

[113] C. R. Reeves. Genetic algorithms. In *Handbook of Metaheuristics*, pages 109–141. Springer, 2010.

[114] W. Rei, M. Gendreau, and P. Soriano. A hybrid monte carlo local branching algorithm for the single vehicle routing problem with stochastic demands. *Transportation Science*, 44(1):136–146, 2010.

[115] M. C. Roboredo and A. A. Pessoa. A branch-and-cut algorithm for the discrete $(r|p)$-centroid problem. *European Journal of Operational Research*, 224(1):101–109, 2013.

[116] S. Ronald. Duplicate genotypes in a genetic algorithm. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 793–798, 1998.

[117] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[118] M. Ruthmair and G. R. Raidl. A memetic algorithm and a solution archive for the rooted delay-constrained minimum spanning tree problem. In *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, pages 351–358, 2012.

[119] N. Saidani, F. Chu, and H. Chen. Competitive facility location and design with reactions of competitors already in the market. *European journal of operational research*, 219(1):9–17, 2012.

[120] M. E. Sáiz, E. M. T. Hendrix, and B. Pelegrín. On nash equilibria of a competitive location-design problem. *European Journal of Operational Research*, 210(3):588–593, 2011.

[121] D. Serra and C. Revelle. Competitive location in discrete space. Economics Working Papers 96, Department of Economics and Business, Universitat Pompeu Fabra, Nov 1994. Techn. Report.

[122] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer, 1998.

[123] N. Z. Shor. Utilization of the operation of space dilatation in the minimization of convex functions. *Cybernetics and Systems Analysis*, 6(1):7–15, 1972.

[124] T. Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519–1539, 2006.

[125] R. Suárez-Vega, D. Santos-Peñate, and D. Pablo. Competitive multifacility location on networks: the $(r|X_p)$-medianoid problem. *Journal of Regional Science*, 44(3):569–588, 2004.

[126] E. Talbi. *Hybrid Metaheuristics*, volume 22 of *Studies in Computational Intelligence*. Springer-Verlag Berlin Heidelberg, 2013.

[127] R. M. Van Slyke and R. Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663, 1969.

[128] L. A. Wolsey. *Integer programming.* Wiley, 1998.

[129] W. Yang, K. Mathur, and R. H. Ballou. Stochastic vehicle routing problem with restocking. *Transportation Science*, 34(1):99–112, 2000.

[130] D. B. Yudin and Nemirovskii A. Informational complexity and efficient methods for the solution of convex extremal problems. *Matekon*, 13(0):25–45, 1977.

[131] S. Y. Yuen and C. K. Chow. A non-revisiting genetic algorithm. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 4583–4590. IEEE, 2007.

APPENDIX A

Full Result Tables

Here, we present the full result tables of the presented algorithms for both the competitive facility location problem and the generalized vehicle routing problem with stochastic demands.

A.1 Competitive Facility Location Problem

In this section we show the results of the various algorithm configurations for all instances of the benchmark set.

A.1.1 Solution Evaluation

Table A.1 shows the results of the tests for the different types of solution evaluation methods as described in the paper.

Table A.1: Results of different solution evaluation methods using the standard configuration and a runtime of 600 seconds.

	greedy			LP			exact		
Instance	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code111w_rp10	4359,00	0,00	14,80	**4361,00**	0,00	130,30	**4361,00**	0,00	70,60
Code111w_rp15	4547,11	6,01	20,10	**4596,00**	0,00	64,10	**4596,00**	0,00	55,60
Code111w_rp20	**4508,50**	6,09	253,30	4505,47	11,22	343,30	4502,90	11,26	217,70
Code1__150w_rp10	7132,20	130,36	250,20	7138,37	112,88	88,60	**7167,43**	51,47	94,00
Code1__150w_rp15	7008,63	54,17	138,40	7077,97	35,79	341,20	**7088,83**	43,99	398,50
Code1__150w_rp20	7070,67	52,46	314,20	7198,27	19,01	380,20	**7198,53**	22,50	370,60
Code1__200w_rp10	9349,60	69,78	406,60	9476,17	107,30	200,60	**9478,50**	92,39	369,70
Code1__200w_rp15	9814,13	185,24	351,30	**10001,40**	92,78	394,40	10000,30	82,92	475,60
Code1__200w_rp20	9615,13	135,94	411,90	**9753,07**	77,54	572,80	9697,53	85,19	586,90
Code211w_rp10	5309,47	2,92	26,50	**5310,00**	0,00	43,50	**5310,00**	0,00	46,10
Code211w_rp15	**5373,00**	0,00	97,10	**5373,00**	0,00	111,80	**5373,00**	0,00	95,70
Code211w_rp20	**5431,57**	2,37	284,50	5404,43	29,69	291,60	5405,63	31,19	365,20
Code2__150w_rp10	7181,53	52,91	332,10	7247,47	53,43	292,50	**7253,30**	71,29	291,00
Code2__150w_rp15	7590,23	92,37	154,60	**7743,20**	4,70	281,40	7742,00	5,57	358,40

Code2_150w_rp20	7673,90	83,24	255,00	**7772,13**	40,91	349,50	7755,50	46,49	347,80
Code2_200w_rp10	9032,00	71,74	221,20	9231,63	75,55	249,80	**9254,53**	62187,00	448,00
Code2_200w_rp15	9274,23	153,66	312,40	**9539,27**	70,94	516,40	9505,43	109,72	438,40
Code2_200w_rp20	9381,90	138,87	475,30	**9579,83**	118,18	508,00	9570,30	110,98	548,80
Code311w_rp10	4392,47	42,88	17,80	**4483,00**	0,00	25,80	**4483,00**	0,00	24,50
Code311w_rp15	4782,10	18,23	221,60	**4800,00**	0,00	73,60	**4800,00**	0,00	63,20
Code311w_rp20	4853,47	8,76	100,50	**4892,80**	0,61	297,90	4892,67	0,76	250,80
Code3_150w_rp10	7240,20	75,69	362,80	7286,93	16,36	310,70	**7291,87**	13,30	369,20
Code3_150w_rp15	7499,30	44,12	161,60	7589,00	18,83	285,80	**7589,27**	18,01	200,50
Code3_150w_rp20	7520,40	63,06	303,20	**7624,43**	34,03	309,10	7624,37	34,20	411,20
Code3_200w_rp10	9224,03	52,98	202,70	**9300,23**	70,30	291,70	9287,13	74,85	362,90
Code3_200w_rp15	9145,17	210,97	378,30	9304,57	71,44	386,40	**9308,37**	70,27	459,10
Code3_200w_rp20	8902,30	210,90	468,70	**9197,97**	155,51	516,80	9145,73	107,33	574,10
Code411w_rp10	**4994,00**	0,00	10,10	**4994,00**	0,00	22,10	**4994,00**	0,00	22,40
Code411w_rp15	5045,43	4,26	108,80	**5064,00**	0,00	117,40	**5064,00**	0,00	125,40
Code411w_rp20	5184,70	5,87	34,40	5204,73	16,24	298,10	**5205,63**	13,61	189,80
Code4_150w_rp10	7232,53	62,64	156,00	**7246,33**	98,75	290,60	7201,73	112,80	151,40
Code4_150w_rp15	7324,93	79,35	120,20	**7409,00**	0,00	285,00	7404,07	14,62	329,30
Code4_150w_rp20	7867,20	7,92	213,90	7915,93	7,44	276,10	**7916,37**	8,83	260,00
Code4_200w_rp10	8594,47	77,24	231,80	8852,40	43,37	191,10	**8873,50**	30,67	289,40
Code4_200w_rp15	8912,20	189,16	387,10	**9106,43**	87,24	403,10	9092,47	99,50	410,80
Code4_200w_rp20	9069,10	194,06	373,00	**9335,67**	113,20	502,40	9300,20	88,94	521,90
Code511w_rp10	4862,93	6,96	7,20	**4906,00**	0,00	39,60	**4906,00**	0,00	38,50
Code511w_rp15	**5131,00**	0,00	162,80	5123,90	7,40	332,30	5128,07	8,37	276,80
Code511w_rp20	**5334,00**	0,00	70,70	5330,33	9,94	101,10	5329,27	9,65	223,70
Code5_150w_rp10	6891,63	16,27	169,60	6965,73	10,59	329,10	**6966,27**	6,38	276,90
Code5_150w_rp15	7039,60	24,26	374,80	7084,93	34,91	392,70	**7097,00**	33,52	323,90
Code5_150w_rp20	7212,87	55,66	291,60	**7320,77**	8,91	343,80	7313,73	24,91	408,30
Code5_200w_rp10	8992,13	96,25	398,60	9159,13	66,25	338,70	**9184,13**	95,30	308,30
Code5_200w_rp15	9094,07	87,52	395,80	9167,60	71,37	413,00	**9175,10**	78,45	374,40
Code5_200w_rp20	9163,33	143,66	398,70	**9442,13**	90,25	499,20	9417,00	101,19	502,20
Code611w_rp10	4589,13	3,60	112,70	4594,73	1,46	110,90	**4595,00**	0,00	95,30
Code611w_rp15	4868,97	25,55	74,10	**4881,00**	0,00	74,50	**4881,00**	0,00	83,00
Code611w_rp20	**4949,33**	8,91	173,10	4941,53	18,58	248,40	4946,13	11,68	266,50
Code6_150w_rp10	6990,83	63,95	129,30	7015,83	64,84	242,70	**7019,17**	65,40	321,50
Code6_150w_rp15	7050,13	70,94	198,60	**7162,53**	32,51	319,90	7160,63	50,74	202,60
Code6_150w_rp20	7325,03	48,11	197,20	**7356,23**	39,25	368,20	7355,60	37,80	353,60
Code6_200w_rp10	9542,23	143,32	233,00	**9750,17**	130,57	207,40	9710,83	107,38	247,90
Code6_200w_rp15	9747,10	242,75	361,80	**10025,27**	150,55	442,70	10007,77	137,66	439,00
Code6_200w_rp20	9864,70	190,84	406,00	**10192,63**	116,02	570,30	10118,63	131,96	580,60
Code711w_rp10	5555,27	10,51	25,30	**5586,00**	0,00	79,70	**5586,00**	0,00	85,00
Code711w_rp15	5826,43	3,10	162,40	**5827,00**	0,00	83,90	**5827,00**	0,00	94,70
Code711w_rp20	**5893,00**	0,00	38,50	**5893,00**	0,00	73,60	**5893,00**	0,00	55,70
Code7_150w_rp10	6042,90	71,68	177,90	6223,90	44,98	293,90	**6230,60**	41,73	220,00
Code7_150w_rp15	6779,17	70,04	292,60	**6840,00**	0,00	231,80	**6840,00**	0,00	218,40
Code7_150w_rp20	7253,33	23,61	215,70	7266,87	42,85	290,10	**7272,97**	41,70	273,00
Code7_200w_rp10	8835,33	166,84	260,20	**9142,37**	149,21	274,10	9122,20	144,57	375,00
Code7_200w_rp15	9413,57	127,38	385,90	**9527,73**	61,94	518,30	9527,67	73,61	424,10
Code7_200w_rp20	9719,33	83,12	330,40	9837,63	58,28	476,10	**9863,23**	53,11	482,90
Code811w_rp10	4576,33	41,45	280,10	4605,20	14,09	301,10	**4607,60**	1,52	356,80
Code811w_rp15	4647,70	19,03	45,70	**4670,77**	11,42	180,70	4658,87	24,49	223,80
Code811w_rp20	4847,00	0,00	212,40	**4858,00**	0,00	77,40	**4858,00**	0,00	75,10
Code8_150w_rp10	7692,70	57,34	236,00	7729,33	14,61	217,60	**7729,43**	14,06	246,90
Code8_150w_rp15	7628,13	100,02	171,20	**7659,73**	8,45	340,50	7659,13	7,04	340,30
Code8_150w_rp20	7784,87	24,04	215,30	**7838,13**	19,77	285,60	7833,87	29,02	294,40
Code8_200w_rp10	8896,83	125,44	382,10	8997,83	122,64	375,90	**9045,00**	89,40	413,80
Code8_200w_rp15	8869,87	111,97	429,20	8983,57	81,04	530,30	**9005,33**	59,87	501,80
Code8_200w_rp20	8882,37	192,45	480,60	**9230,93**	97,88	512,80	9203,87	105,27	565,80
Code911w_rp10	**5302,00**	0,00	8,00	**5302,00**	0,00	21,80	**5302,00**	0,00	21,00
Code911w_rp15	5083,93	6,90	48,90	**5155,07**	6,15	339,40	5150,10	13,94	332,30
Code911w_rp20	5450,87	4,97	20,50	**5457,00**	2,03	239,90	5456,63	1,97	195,10
Code9_150w_rp10	6801,07	137,08	323,00	6838,70	39,05	220,30	**6848,60**	24,36	231,40

Code9_150w_rp15	6651,40	48,96	362,20	**6877,93**	15,49	274,40	6865,97	31,32	327,80
Code9_150w_rp20	6897,60	97,04	321,10	7073,23	63,63	365,00	**7113,53**	52,19	432,00
Code9_200w_rp10	8468,00	253,61	192,40	8818,73	119,21	204,00	**8824,73**	162,21	209,30
Code9_200w_rp15	8880,57	84,92	226,40	9074,87	75,56	445,30	**9097,67**	53,24	547,00
Code9_200w_rp20	9025,07	172,46	445,70	**9409,13**	62,21	539,00	9391,50	78,89	515,80
Code1011w_rp10	**5005,00**	0,00	49,60	5000,53	11,83	110,70	4998,73	13,05	93,10
Code1011w_rp15	**5195,00**	0,00	98,30	**5195,00**	0,00	254,00	5194,60	2,19	189,00
Code1011w_rp20	5359,03	37,90	101,50	**5399,00**	0,00	222,80	**5399,00**	0,00	204,10
Code10_150w_rp10	6531,80	76,84	226,90	**6701,30**	43,97	114,40	6699,90	48,60	171,00
Code10_150w_rp15	6826,63	127,95	148,50	7004,00	34,14	228,80	**7009,47**	17,54	313,30
Code10_150w_rp20	7089,93	89,07	237,70	**7176,17**	40,36	336,30	7171,73	54,05	277,90
Code10_200w_rp10	9282,37	108,47	313,30	**9333,13**	118,39	437,50	9317,03	116,95	438,50
Code10_200w_rp15	8990,13	164,61	307,40	**9221,83**	63,44	465,20	9197,70	79,54	533,60
Code10_200w_rp20	9310,17	184,23	433,50	9585,33	100,70	512,40	**9588,63**	133,12	522,80
geometric mean	6907,43			**6995,12**			6993,29		
#best results	11			**53**			48		
#unique best res.	6			**35**			31		

A.1.2 Genetic Algorithm

In Table A.2 the computational results of different configurations of the genetic algorithm are given.

Table A.2: Results for the different configurations of the GA. The pure genetic algorithm (GA), the genetic algorithm with the local search (GA + LS), the genetic algorithm with the solution archive (GA + solA) and the genetic algorithm with the solution archive and the local search (GA + LS + solA) with a runtime of 600 seconds.

	GA			GA + LS			GA + solA			GA + LS + solA		
Instance	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code111w_rp10	4331,80	12,32	338,20	4348,97	25,00	113,00	4334,20	10,75	402,60	**4361,00**	0,00	14,70
Code111w_rp15	4572,37	21,18	461,10	4586,43	16,65	38,50	4582,67	6,63	412,80	**4596,00**	0,00	16,10
Code111w_rp20	4452,23	17,50	538,40	4474,97	23,52	162,50	4464,17	21,42	521,40	**4505,47**	11,22	209,50
Code1_150w_rp10	6503,63	123,24	538,20	**7163,57**	54,75	113,40	6606,03	120,76	530,10	7138,37	112,88	29,20
Code1_150w_rp15	6823,13	63,36	549,60	7021,47	75,64	149,30	6876,50	49,83	546,80	**7077,97**	35,79	133,00
Code1_150w_rp20	6900,63	109,91	550,20	7163,03	58,69	187,30	6980,63	105,85	560,30	**7198,27**	19,01	241,40
Code1_200w_rp10	8810,10	201,26	531,40	9443,60	105,48	246,70	8926,30	189,46	509,10	**9476,17**	107,30	243,10
Code1_200w_rp15	9079,43	265,11	572,00	9956,77	112,63	349,90	9212,53	227,11	553,60	**10001,40**	92,78	297,10
Code1_200w_rp20	8899,63	157,41	562,00	9683,20	119,85	479,60	8996,93	219,23	560,80	**9753,07**	77,54	460,50
Code211w_rp10	5289,47	25,60	521,20	**5310,00**	0,00	51,20	5291,13	28,09	473,40	**5310,00**	0,00	8,10
Code211w_rp15	5279,47	35,12	546,90	5362,33	19,73	42,40	5294,43	31,31	493,60	**5373,00**	0,00	23,10
Code211w_rp20	5250,03	52,47	556,90	5351,80	55,16	135,10	5268,93	47,01	546,80	**5404,43**	29,69	82,40
Code2_150w_rp10	6962,67	46,48	519,90	7229,90	89,61	233,40	6969,40	43,43	512,40	**7247,47**	53,43	242,70
Code2_150w_rp15	7423,00	94,91	537,40	7702,23	103,32	148,00	7496,67	66,09	516,30	**7743,20**	4,70	96,90
Code2_150w_rp20	7439,37	65,12	540,10	7713,43	70,96	208,40	7500,33	60,97	551,80	**7772,13**	40,91	211,50
Code2_200w_rp10	8609,17	151,04	524,40	9181,07	127,89	228,80	8717,20	131,21	529,70	**9231,63**	75,55	130,10
Code2_200w_rp15	8639,43	159,89	523,30	9482,17	119,41	342,10	8678,77	144,79	520,70	**9539,27**	70,94	392,00
Code2_200w_rp20	8626,47	119,66	557,10	9508,30	119,17	521,50	8726,30	112,43	547,80	**9579,83**	118,18	421,30
Code311w_rp10	4472,50	34,65	331,60	**4483,00**	0,00	21,20	**4483,00**	0,00	336,20	**4483,00**	0,00	8,10
Code311w_rp15	4775,93	19,79	473,20	4785,40	22,85	70,40	4781,13	21,89	534,60	**4800,00**	0,00	13,30
Code311w_rp20	4835,77	15,62	534,70	4879,17	14,61	116,60	4849,67	11,70	495,30	**4892,80**	0,61	65,20
Code3_150w_rp10	6975,17	75,29	522,30	7252,50	69,75	136,20	7004,47	71,96	493,00	**7286,93**	16,36	35,40
Code3_150w_rp15	7333,73	124,31	551,10	7554,00	47,95	131,20	7391,33	95,72	523,60	**7589,00**	18,83	142,50
Code3_150w_rp20	7358,33	70,60	543,60	7601,30	45,29	214,80	7406,17	50,70	559,40	**7624,43**	34,03	274,00
Code3_200w_rp10	8832,40	181,41	553,50	9229,40	86,17	239,00	8856,23	185,09	535,80	**9300,23**	70,30	227,00
Code3_200w_rp15	8232,10	269,27	563,40	9265,43	98,36	333,10	8497,40	203,87	562,00	**9304,57**	71,44	281,90
Code3_200w_rp20	8091,83	188,37	564,80	9150,37	174,73	509,90	8251,03	134,42	550,60	**9197,97**	155,51	426,90
Code411w_rp10	4984,67	9,57	466,20	**4994,00**	0,00	22,40	4988,00	9,03	506,10	**4994,00**	0,00	7,90
Code411w_rp15	5044,90	19,96	476,30	5046,63	16,87	113,20	5051,83	15,50	431,40	**5064,00**	0,00	48,80
Code411w_rp20	5081,93	27,01	514,60	5131,80	55,58	129,10	5101,10	26,28	468,60	**5204,73**	16,24	39,60
Code4_150w_rp10	6503,53	136,21	530,50	7174,67	90,48	110,00	6611,83	117,85	534,90	**7246,33**	98,75	38,30
Code4_150w_rp15	6934,27	174,97	551,90	7353,67	74,30	237,60	7047,40	139,91	536,30	**7409,00**	0,00	71,30
Code4_150w_rp20	7382,80	148,05	549,80	7900,00	25,49	201,90	7405,40	139,86	565,90	**7915,93**	7,44	251,30
Code4_200w_rp10	8385,63	93,41	496,60	8849,43	45,21	232,20	8380,77	105,28	493,90	**8852,40**	43,37	115,60

Code4_200w_rp15	8377,87	139,43	541,50	9055,10	101,98	339,30	8407,60	141,00	519,60	**9106,43**	87,24	241,30
Code4_200w_rp20	8633,17	137,69	570,10	9313,53	98,52	482,20	8770,83	178,30	550,00	**9335,67**	113,20	388,50
Code511w_rp10	**4906,00**	0,00	318,00	**4906,00**	0,00	49,20	**4906,00**	0,00	254,40	**4906,00**	0,00	8,90
Code511w_rp15	5030,73	37,47	534,10	5086,10	31,13	141,50	5047,93	30,46	466,90	**5123,90**	7,40	63,40
Code511w_rp20	5224,03	26,17	524,80	5305,67	37,00	208,70	5238,90	33,37	565,80	**5330,33**	9,94	76,00
Code5_150w_rp10	6777,10	78,29	509,00	6954,27	20,27	190,10	6788,97	75,57	514,60	**6965,73**	10,59	32,90
Code5_150w_rp15	6849,03	80,05	507,10	7065,03	60,02	161,20	6904,83	63,76	524,60	**7084,93**	34,91	214,60
Code5_150w_rp20	6768,50	86,03	560,90	7292,87	47,62	345,60	6849,43	99,17	555,50	**7320,77**	8,91	227,30
Code5_200w_rp10	8566,63	138,43	512,50	9137,57	74,39	220,10	8645,13	106,16	527,10	**9159,13**	66,25	268,20
Code5_200w_rp15	8494,37	151,08	554,50	9115,17	120,27	319,60	8660,67	216,45	547,10	**9167,60**	71,37	320,90
Code5_200w_rp20	9008,77	94,42	511,50	9398,47	103,93	480,20	9045,77	72,74	509,70	**9442,13**	90,25	345,90
Code611w_rp10	4582,17	26,63	435,10	4590,50	11,28	107,00	4573,27	36,36	452,90	**4594,73**	1,46	17,70
Code611w_rp15	4752,67	37,81	528,40	4858,90	41,88	56,80	4769,70	40,16	508,70	**4881,00**	0,00	15,70
Code611w_rp20	4811,00	29,47	554,50	4913,00	36,56	187,20	4827,60	12,76	536,70	**4941,53**	18,58	96,00
Code6_150w_rp10	6680,50	86,69	512,80	6988,53	76,38	100,20	6772,37	67,94	523,40	**7015,83**	64,84	36,90
Code6_150w_rp15	6776,40	97,73	556,50	7126,77	82,94	165,50	6821,50	91,48	564,20	**7162,53**	32,51	71,60
Code6_150w_rp20	6894,73	98,39	550,10	7267,33	94,63	194,40	6950,80	55,80	541,00	**7356,23**	39,25	133,90
Code6_200w_rp10	8783,10	181,04	464,50	9710,20	139,67	283,00	8986,30	195,73	520,10	**9750,17**	130,57	197,50
Code6_200w_rp15	8873,43	178,65	545,20	9957,50	164,71	396,90	9013,73	235,39	564,80	**10025,27**	150,55	326,70
Code6_200w_rp20	9222,03	170,11	561,20	10153,33	129,86	521,50	9404,47	158,70	567,50	**10192,63**	116,02	452,50
Code711w_rp10	5384,73	95,76	531,90	5585,10	4,93	26,40	5431,20	72,86	525,20	**5586,00**	0,00	8,70
Code711w_rp15	5763,40	18,20	486,30	5820,17	17,61	67,00	5767,93	18,34	500,40	**5827,00**	0,00	31,60
Code711w_rp20	5863,40	12,67	512,00	5889,30	11,29	51,60	5861,47	28,80	537,50	**5893,00**	0,00	29,80
Code7_150w_rp10	5895,37	81,25	531,90	6193,73	75,22	160,40	5886,43	75,72	544,20	**6223,90**	44,98	190,10
Code7_150w_rp15	6615,50	71,31	543,90	6823,97	39,32	142,60	6667,40	46,97	542,40	**6840,00**	0,00	82,90
Code7_150w_rp20	6920,20	56,37	573,10	7221,87	60,19	211,60	6961,73	46,05	530,70	**7266,87**	42,85	203,10
Code7_200w_rp10	8177,73	205,33	548,00	9040,83	192,16	250,90	8296,80	130,13	532,90	**9142,37**	149,21	222,80
Code7_200w_rp15	8743,93	138,58	550,40	9469,50	96,36	356,30	8891,03	112,83	543,60	**9527,73**	61,94	283,90
Code7_200w_rp20	9159,53	183,23	564,60	**9848,93**	58,16	482,80	9265,63	94,41	545,60	9837,63	58,28	361,90
Code811w_rp10	4468,77	31,34	447,50	4558,27	34,34	148,90	4482,23	33,97	483,10	**4605,20**	14,09	21,60
Code811w_rp15	4472,03	55,21	562,40	4615,97	47,53	210,30	4489,60	56,09	544,40	**4670,77**	11,42	41,60
Code811w_rp20	4799,47	23,40	553,70	4854,03	9,35	162,90	4818,70	9,89	524,30	**4858,00**	0,00	24,40
Code8_150w_rp10	7091,50	178,99	562,80	7664,23	123,59	113,20	7164,93	132,71	529,60	**7729,33**	14,61	28,70
Code8_150w_rp15	7133,27	128,62	542,30	7624,73	48,91	161,90	7242,30	148,03	551,20	**7659,73**	8,45	103,10
Code8_150w_rp20	7534,20	31,35	530,30	7785,50	86,49	199,50	7554,07	37,92	531,00	**7838,13**	19,77	188,20
Code8_200w_rp10	8499,73	167,54	496,30	**9031,87**	103,59	282,50	8564,03	194,36	476,60	8997,83	122,64	170,60
Code8_200w_rp15	8496,33	159,99	534,20	8960,00	107,96	349,80	8540,93	133,65	546,80	**8983,57**	81,04	357,90
Code8_200w_rp20	8479,23	120,61	563,20	9201,30	88,44	511,60	8574,77	72,10	559,80	**9230,93**	97,88	484,30
Code911w_rp10	5300,80	6,57	294,40	**5302,00**	0,00	23,30	**5302,00**	0,00	235,70	**5302,00**	0,00	7,50
Code911w_rp15	4989,77	55,79	527,80	5119,67	35,51	140,20	5000,17	60,71	551,40	**5155,07**	6,15	220,60
Code911w_rp20	5385,73	36,02	562,10	5448,63	12,93	58,80	5424,17	16,11	552,40	**5457,00**	2,03	92,50
Code9_150w_rp10	6548,50	67,94	499,90	6810,67	69,48	271,50	6574,13	59,01	521,50	**6838,70**	39,05	55,50
Code9_150w_rp15	6561,50	65,00	528,50	6811,50	91,06	164,70	6591,00	54,31	505,30	**6877,93**	15,49	148,40
Code9_150w_rp20	6661,43	59,53	563,10	7020,67	87,25	207,80	6716,47	48,97	552,70	**7073,23**	63,63	299,90
Code9_200w_rp10	8020,73	178,59	552,60	**8836,20**	134,15	254,10	8064,03	164,39	506,00	8818,73	119,21	182,90
Code9_200w_rp15	8220,90	135,00	558,40	9020,63	105,81	357,90	8289,27	138,26	550,00	**9074,87**	75,56	335,40
Code9_200w_rp20	8370,97	112,10	519,30	9377,47	83,44	486,70	8484,93	161,26	554,50	**9409,13**	62,21	416,80
Code1011w_rp10	4927,47	31,00	515,90	4982,97	22,94	95,80	4941,87	26,25	487,60	**5000,53**	11,83	18,20
Code1011w_rp15	5162,80	17,34	517,50	5172,60	22,95	126,40	5167,57	10,31	428,60	**5195,00**	0,00	29,20
Code1011w_rp20	5268,87	35,60	546,70	5342,00	31,38	80,10	5277,20	35,12	544,00	**5399,00**	0,00	60,80
Code10_150w_rp10	6191,23	111,42	514,90	6662,80	84,69	108,00	6239,10	96,49	522,10	**6701,30**	43,97	30,20
Code10_150w_rp15	6477,40	112,59	523,80	6998,43	31,17	131,60	6493,60	104,47	554,20	**7004,00**	34,14	104,30
Code10_150w_rp20	6776,33	88,64	512,50	7112,63	80,67	219,70	6820,37	66,04	548,30	**7176,17**	40,36	175,30
Code10_200w_rp10	8744,50	185,66	514,40	9312,07	125,38	297,10	8827,13	174,75	535,80	**9333,13**	118,39	151,70
Code10_200w_rp15	8825,93	89,75	488,50	9174,60	94,69	328,70	8859,43	64,82	470,40	**9221,83**	63,44	434,30
Code10_200w_rp20	8911,50	118,03	549,90	**9607,57**	97,67	485,20	8980,20	154,75	541,90	9585,33	100,70	460,40
geometric mean	6643,72			6964,10			6691,70			**6995,12**		
#best results	1			10			3			**85**		
#unique best res.	0			5			0			**80**		

A.1.3 Neighborhoods of the Local Search

Table A.3 shows the results of using the different strategies for utilizing the solution archive within the local / tabu search as described in Section 4.7.

Table A.3: Results of using different neighborhood structures for intermediate local search with a total runtime of 600 seconds.

	complete NB			reduced NB			conversion NB			TS with reduced NB		
Instance	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code1011w_rp10	4998,73	13,05	249,20	5000,53	11,83	110,70	5000,07	11,43	123,20	**5003,67**	7,30	93,80
Code111w_rp10	**4361,00**	0,00	133,80	**4361,00**	0,00	130,30	**4361,00**	0,00	72,20	**4361,00**	0,00	47,00
Code111w_rp15	**4596,00**	0,00	44,90	**4596,00**	0,00	64,10	**4596,00**	0,00	79,40	**4596,00**	0,00	55,10
Code111w_rp20	4488,20	22,42	299,50	4505,47	11,22	343,30	4497,93	14,69	343,60	**4506,23**	8,39	268,70
Code1_150w_rp10	7157,07	89,96	113,60	7138,37	112,88	88,60	7171,30	47,65	137,40	**7180,00**	0,00	118,00
Code1_150w_rp15	7055,70	39,71	240,40	7077,97	35,79	341,20	7070,77	45,79	339,40	**7143,30**	31,63	337,00
Code1_150w_rp20	7187,87	22,59	289,30	7198,27	19,01	380,20	7190,87	24,84	400,70	**7221,50**	27,75	455,20
Code1_200w_rp10	9471,80	85,42	286,70	9476,17	107,30	200,60	9508,00	67,29	400,10	**9532,50**	56,10	350,30
Code1_200w_rp15	9959,17	133,66	483,60	**10001,40**	92,78	394,40	9988,87	100,51	461,90	9986,13	99,43	500,60
Code1_200w_rp20	9706,37	70,27	530,90	9753,07	77,54	572,80	9720,67	74,29	521,80	**9760,10**	69,67	600,00
Code211w_rp10	**5310,00**	0,00	85,50	**5310,00**	0,00	43,50	**5310,00**	0,00	29,00	**5310,00**	0,00	33,10
Code211w_rp15	5367,60	8,39	62,40	**5373,00**	0,00	111,80	**5373,00**	0,00	100,40	**5373,00**	0,00	162,80
Code211w_rp20	5386,77	37,42	291,80	5404,43	29,69	291,60	5404,13	30,62	439,50	**5427,37**	9,90	200,00
Code2_150w_rp10	7234,77	55,54	303,70	7247,47	53,43	292,50	7265,53	58,72	216,00	**7290,17**	48,81	401,10
Code2_150w_rp15	7738,40	13,75	306,10	**7743,20**	4,70	281,40	7742,30	5,53	367,50	7741,60	6,29	342,30
Code2_150w_rp20	7737,80	51,87	360,50	7772,13	40,91	349,50	7771,40	39,29	390,40	**7774,13**	59,61	317,60
Code2_200w_rp10	9214,43	102,90	266,60	9231,63	75,55	249,80	9230,23	94,53	395,30	**9252,07**	89,10	376,50
Code2_200w_rp15	9525,37	92,86	419,00	9539,27	70,94	516,40	9518,10	93,23	439,20	**9561,30**	83,08	591,40
Code2_200w_rp20	9542,93	106,14	549,80	9579,83	118,18	508,00	9549,43	101,89	549,80	**9619,70**	67,33	600,00
Code311w_rp10	**4483,00**	0,00	25,40	**4483,00**	0,00	25,80	**4483,00**	0,00	23,80	**4483,00**	0,00	31,10
Code311w_rp15	**4800,00**	0,00	85,90	**4800,00**	0,00	73,60	**4800,00**	0,00	50,80	**4800,00**	0,00	49,20
Code311w_rp20	4890,63	3,95	183,30	**4892,80**	0,61	297,90	4892,67	0,76	264,90	4892,73	0,69	213,90
Code3_150w_rp10	7285,67	15,76	113,40	7286,93	16,36	310,70	7293,10	12,10	314,30	**7299,00**	0,00	130,90
Code3_150w_rp15	7567,03	34,68	193,00	7589,00	18,83	285,80	**7589,90**	18,86	286,60	7583,77	20,38	192,40
Code3_150w_rp20	7605,33	47,90	298,50	7624,43	34,03	309,10	**7627,93**	26,49	331,80	7620,67	46,21	429,70
Code3_200w_rp10	9265,97	102,41	247,40	9300,23	70,30	291,70	9275,87	85,79	320,20	**9339,97**	76,46	375,40
Code3_200w_rp15	9279,60	91,05	383,20	9304,57	71,44	386,40	9301,60	68,80	438,00	**9317,83**	66,40	496,10
Code3_200w_rp20	9149,43	118,80	518,40	9197,97	155,51	516,80	9147,50	115,48	517,80	**9220,97**	105,33	600,00
Code411w_rp10	**4994,00**	0,00	25,00	**4994,00**	0,00	22,10	**4994,00**	0,00	25,20	**4994,00**	0,00	32,50
Code411w_rp15	5059,97	10,25	140,10	**5064,00**	0,00	117,40	5062,27	6,67	191,10	5063,80	1,10	125,10
Code411w_rp20	5163,60	38,10	211,20	5204,73	16,24	298,10	5206,73	12,42	239,00	**5209,00**	0,00	163,80
Code4_150w_rp10	7207,10	93,32	225,00	7246,33	98,75	290,60	7242,53	108,65	230,40	**7281,60**	67,11	152,70
Code4_150w_rp15	7384,97	30,96	363,40	**7409,00**	0,00	285,00	7399,00	29,61	304,00	**7409,00**	0,00	225,20
Code4_150w_rp20	7911,43	10,11	229,20	7915,93	7,44	276,10	7914,17	10,51	232,30	**7919,00**	7,47	300,30
Code4_200w_rp10	8870,13	32,68	228,90	8852,40	43,37	191,10	8868,47	35,77	360,30	**8874,80**	28,02	312,40
Code4_200w_rp15	9071,50	119,82	352,40	**9106,43**	87,24	403,10	9070,27	131,37	417,30	9069,73	112,00	476,90
Code4_200w_rp20	9279,47	115,51	520,90	9335,67	113,20	502,40	9285,10	99,60	531,00	**9341,20**	82,14	600,00
Code511w_rp10	**4906,00**	0,00	39,60	**4906,00**	0,00	39,60	**4906,00**	0,00	53,90	**4906,00**	0,00	52,60
Code511w_rp15	5099,53	20,20	206,70	**5123,90**	7,40	332,30	5115,77	14,33	212,00	5122,30	5,54	259,30
Code511w_rp20	5318,53	15,85	233,60	5330,33	9,94	101,10	5328,40	14,62	165,80	**5332,57**	7,85	122,00
Code5_150w_rp10	6961,43	11,62	215,90	6965,73	10,59	329,10	6963,13	9,49	265,70	**6972,20**	5,70	144,80
Code5_150w_rp15	7088,87	42,55	275,30	7084,93	34,91	392,70	7090,03	41,84	352,00	**7096,03**	45,75	320,90
Code5_150w_rp20	7304,07	30,18	416,80	**7320,77**	8,91	343,80	7307,43	34,81	375,90	7316,13	23,14	454,70
Code5_200w_rp10	9187,27	86,31	278,00	9159,13	66,25	338,70	9153,13	99,28	242,90	**9202,97**	63,13	327,20
Code5_200w_rp15	9160,30	79,24	385,50	9167,60	71,37	413,00	9179,13	68,50	482,60	**9192,50**	81,90	480,60
Code5_200w_rp20	9413,07	75,87	490,10	9442,13	90,25	499,20	9420,60	108,68	514,30	**9450,87**	84,16	591,40
Code611w_rp10	**4595,00**	0,00	104,60	4594,73	1,46	110,90	**4595,00**	0,00	155,80	**4595,00**	0,00	81,50
Code611w_rp15	4879,33	9,13	86,40	**4881,00**	0,00	74,50	**4881,00**	0,00	75,20	**4881,00**	0,00	61,80
Code611w_rp20	4936,80	25,44	237,20	4941,53	18,58	248,40	**4946,23**	11,56	321,70	4945,17	18,24	146,10
Code6_150w_rp10	7016,53	50,58	356,20	7015,83	64,84	242,70	7031,67	36,71	269,40	**7050,00**	0,00	146,00
Code6_150w_rp15	7141,17	55,31	270,10	7162,53	32,51	319,90	7152,30	51,89	309,20	**7181,37**	25,38	288,80
Code6_150w_rp20	7360,67	25,43	320,30	7356,23	39,25	368,20	7352,20	42,01	394,70	**7386,00**	0,00	328,20
Code6_200w_rp10	9749,27	127,43	278,30	9750,17	130,57	207,40	9729,67	130,58	252,10	**9798,00**	65,27	348,40
Code6_200w_rp15	9991,10	146,05	410,70	10025,27	150,55	442,70	9999,57	156,43	430,20	**10071,83**	106,40	552,90
Code6_200w_rp20	10187,07	107,08	539,10	10192,63	116,02	570,30	10195,33	118,35	528,70	**10234,30**	77,10	600,00
Code711w_rp10	**5586,00**	0,00	96,10	**5586,00**	0,00	79,70	5585,10	4,93	100,90	**5586,00**	0,00	39,50
Code711w_rp15	**5827,00**	0,00	74,50	**5827,00**	0,00	83,90	**5827,00**	0,00	105,00	**5827,00**	0,00	111,40
Code711w_rp20	**5893,00**	0,00	79,40	**5893,00**	0,00	73,60	**5893,00**	0,00	52,50	**5893,00**	0,00	98,40
Code7_150w_rp10	6201,60	54,54	304,40	6223,90	44,98	293,90	6202,33	60,52	237,40	**6235,20**	49,09	262,70
Code7_150w_rp15	6833,93	6,60	141,90	**6840,00**	0,00	231,80	6839,57	2,37	313,40	**6840,00**	0,00	214,90
Code7_150w_rp20	7255,87	57,45	279,10	7266,87	42,85	290,10	7265,03	41,55	299,30	**7267,90**	38,74	444,80

Code7_200w_rp10	9137,07	146,47	242,40	9142,37	149,21	274,10	9168,00	90,67	241,30	**9194,97**	97,00	331,50
Code7_200w_rp15	9529,87	49,08	416,50	9527,73	61,94	518,30	**9538,90**	56,23	511,20	9524,90	75,55	503,80
Code7_200w_rp20	9826,40	62,92	524,20	9837,63	58,28	476,10	9839,83	79,96	492,80	**9886,83**	64,06	600,00
Code811w_rp10	4574,37	35,63	432,40	4605,20	14,09	301,10	4606,43	9,62	340,70	**4609,00**	0,00	134,10
Code811w_rp15	4647,07	25,37	247,10	**4670,77**	11,42	180,70	4670,20	14,13	187,70	4664,63	20,80	214,30
Code811w_rp20	**4858,00**	0,00	116,80	**4858,00**	0,00	77,40	**4858,00**	0,00	86,70	**4858,00**	0,00	79,30
Code8_150w_rp10	7729,33	14,61	176,00	7729,33	14,61	217,60	**7732,00**	0,00	294,20	7723,47	46,74	142,90
Code8_150w_rp15	7647,80	30,24	364,90	7659,73	8,45	340,50	7654,67	24,28	303,80	**7660,80**	3,66	261,90
Code8_150w_rp20	7816,23	55,06	301,90	7838,13	19,77	285,60	7818,63	59,88	302,50	**7840,13**	17,64	360,60
Code8_200w_rp10	9003,17	132,27	302,40	8997,83	122,64	375,90	9041,93	94,40	268,10	**9050,13**	95,65	406,70
Code8_200w_rp15	8994,00	79,88	460,00	8983,57	81,04	530,30	8984,13	83,64	500,50	**9007,90**	76,47	527,50
Code8_200w_rp20	9171,17	128,52	536,00	9230,93	97,88	512,80	**9245,07**	93,85	527,90	9242,60	86,11	600,00
Code911w_rp10	**5302,00**	0,00	21,80	**5302,00**	0,00	21,80	**5302,00**	0,00	24,80	**5302,00**	0,00	33,10
Code911w_rp15	5138,67	22,18	313,90	5155,07	6,15	339,40	5154,77	6,22	357,80	**5156,53**	2,47	307,50
Code911w_rp20	5455,70	2,98	200,20	5457,00	2,03	239,90	5456,60	1,99	167,30	**5457,70**	1,76	175,90
Code9_150w_rp10	6845,00	26,98	240,80	6838,70	39,05	220,30	**6851,80**	17,53	185,10	6849,97	19,90	278,10
Code9_150w_rp15	6860,70	42,20	281,60	**6877,93**	15,49	274,40	6857,10	46,96	248,90	6875,87	18,06	324,40
Code9_150w_rp20	7072,17	64,04	307,20	7073,23	63,63	365,00	7107,07	60,49	324,00	**7127,70**	55,41	508,00
Code9_200w_rp10	8834,53	105,05	260,70	8818,73	119,21	204,00	8808,93	143,98	259,60	**8869,87**	146,19	352,30
Code9_200w_rp15	9051,40	56,97	348,40	9074,87	75,56	445,30	9052,70	96,45	500,50	**9079,77**	53,21	489,00
Code9_200w_rp20	9369,40	104,99	496,80	9409,13	62,21	539,00	9383,87	81,97	512,10	**9427,07**	44,51	600,00
Code1011w_rp15	5192,10	7,92	297,20	**5195,00**	0,00	254,00	5194,60	2,19	209,00	**5195,00**	0,00	134,70
Code1011w_rp20	5368,00	28,86	169,50	**5399,00**	0,00	222,80	**5399,00**	0,00	246,30	**5399,00**	0,00	178,60
Code10_150w_rp10	6696,30	55,79	154,00	6701,30	43,97	114,40	6698,47	43,41	155,00	**6708,43**	35,97	122,20
Code10_150w_rp15	7006,17	17,42	220,20	7004,00	34,14	228,80	7000,53	46,09	290,10	**7008,00**	18,09	290,70
Code10_150w_rp20	7144,17	60,73	394,40	7176,17	40,36	336,30	7170,10	40,19	390,50	**7198,20**	19,17	435,10
Code10_200w_rp10	9322,00	95,62	337,10	9333,13	118,39	437,50	9308,50	94,90	454,70	**9356,07**	83,08	477,00
Code10_200w_rp15	9211,87	74,97	431,10	9221,83	63,44	465,20	9190,40	83,05	397,20	**9289,80**	60,00	468,90
Code10_200w_rp20	9581,93	101,29	512,60	9585,33	100,70	512,40	9584,77	126,69	516,30	**9587,97**	106,61	600,00
geometric mean	6983,653735			6995,12021			6992,376022			**7006,534831**		
#best results	13			27			22			**74**		
#unique best results	0			9			7			**55**		

A.1.4 Multi-Level Evaluation Scheme

The computational results for testing the multi-level evaluation scheme (ML-ES) are listed in Table A.4.

Table A.4: Results for the multi-level evaluation scheme and the local/tabu search improvement compared to the standard LP solution evaluation with a runtime of 600 seconds.

	GA + solA + LP + LS			GA + solA + ML-ES + LS			GA + solA + ML-ES + improved LS			GA + solA + ML-ES + improved TS		
Instance	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$	$\overline{obj}$	sd	$t^*[s]$
Code111w_rp10	**4361,00**	0,00	130,30	**4361,00**	0,00	13,00	**4361,00**	0,00	12,70	**4361,00**	0,00	14,70
Code111w_rp15	**4596,00**	0,00	64,10	**4596,00**	0,00	22,20	**4596,00**	0,00	12,20	**4596,00**	0,00	16,10
Code111w_rp20	4505,47	11,22	343,30	4507,77	4,15	181,80	4511,47	1,38	231,30	**4511,87**	0,73	209,50
Code1_150w_rp10	7138,37	112,88	88,60	**7180,00**	0,00	23,50	**7180,00**	0,00	33,70	**7180,00**	0,00	29,20
Code1_150w_rp15	7077,97	35,79	341,20	7080,23	45,92	110,30	7130,97	36,67	291,60	**7153,93**	0,25	133,00
Code1_150w_rp20	7198,27	19,01	380,20	7198,70	23,39	264,90	7208,13	18,97	282,50	**7247,27**	7,97	241,40
Code1_200w_rp10	9476,17	107,30	200,60	9515,23	54,65	201,80	9558,83	37,26	298,50	**9594,00**	10,37	243,10
Code1_200w_rp15	10001,40	92,78	394,40	10026,23	67,17	134,20	10077,10	42,96	289,20	**10095,00**	37,02	297,10
Code1_200w_rp20	9753,07	77,54	572,80	9742,10	93,20	225,20	9807,57	74,67	460,80	**9831,97**	56,35	460,50
Code211w_rp10	**5310,00**	0,00	43,50	**5310,00**	0,00	8,60	**5310,00**	0,00	8,30	**5310,00**	0,00	8,10
Code211w_rp15	**5373,00**	0,00	111,80	**5373,00**	0,00	37,00	**5373,00**	0,00	17,80	**5373,00**	0,00	23,10
Code211w_rp20	5404,43	29,69	291,60	5427,80	10,53	252,60	**5432,00**	0,00	165,70	5431,57	2,37	82,40
Code2_150w_rp10	7247,47	53,43	292,50	7276,70	61,74	98,60	7328,97	23,91	166,70	**7337,00**	0,00	242,70
Code2_150w_rp15	7743,20	4,70	281,40	7735,93	9,15	60,30	7744,00	3,81	102,90	**7745,00**	0,00	96,90
Code2_150w_rp20	7772,13	40,91	349,50	7759,60	40,59	181,30	7789,27	28,71	177,00	**7802,03**	15,79	211,50
Code2_200w_rp10	9231,63	75,55	249,80	9238,23	78,81	58,30	9307,13	53,11	236,50	**9321,13**	26,28	130,10
Code2_200w_rp15	9539,27	70,94	516,40	9471,07	78,21	119,40	9593,53	53,40	345,90	**9626,67**	17,34	392,00
Code2_200w_rp20	9579,83	118,18	508,00	9599,40	88,02	241,50	9643,80	80,47	402,70	**9666,37**	52,72	421,30
Code311w_rp10	**4483,00**	0,00	25,80	**4483,00**	0,00	5,90	**4483,00**	0,00	7,90	**4483,00**	0,00	8,10
Code311w_rp15	**4800,00**	0,00	73,60	**4800,00**	0,00	13,50	**4800,00**	0,00	19,80	**4800,00**	0,00	13,30
Code311w_rp20	4892,80	0,61	297,90	4889,33	6,19	100,10	**4893,00**	0,00	103,30	**4893,00**	0,00	65,20
Code3_150w_rp10	7286,93	16,36	310,70	7292,50	13,38	62,90	7296,83	8,33	64,10	**7299,00**	0,00	35,40
Code3_150w_rp15	7589,00	18,83	285,80	7580,97	23,60	45,60	7597,77	13,80	207,80	**7603,10**	2,75	142,50

Code3_150w_rp20	7624,43	34,03	309,10	7605,77	60,47	111,10	7636,47	15,28	289,20	**7646,87**	4,32	274,00
Code3_200w_rp10	9300,23	70,30	291,70	9320,53	62,72	164,60	9358,97	48,66	283,30	**9374,30**	28,15	227,00
Code3_200w_rp15	9304,57	71,44	386,40	9310,33	71,22	192,00	9353,33	39,29	388,40	**9365,97**	17,19	281,90
Code3_200w_rp20	9197,97	155,51	516,80	9265,93	107,10	252,20	9285,73	81,47	416,80	**9296,67**	70,96	426,90
Code411w_rp10	**4994,00**	0,00	22,10	**4994,00**	0,00	5,30	**4994,00**	0,00	5,30	**4994,00**	0,00	7,90
Code411w_rp15	**5064,00**	0,00	117,40	**5064,00**	0,00	49,70	**5064,00**	0,00	53,20	**5064,00**	0,00	48,80
Code411w_rp20	5204,73	16,24	298,10	**5209,00**	0,00	92,40	**5209,00**	0,00	89,80	**5209,00**	0,00	39,60
Code4_150w_rp10	7246,33	98,75	290,60	7275,20	66,09	56,30	7302,43	77,20	128,70	**7318,00**	0,00	38,30
Code4_150w_rp15	**7409,00**	0,00	285,00	7387,43	24,88	71,60	**7409,00**	0,00	124,70	**7409,00**	0,00	71,30
Code4_150w_rp20	7915,93	7,44	276,10	7918,43	8,52	79,50	7925,50	5,69	221,10	**7927,50**	2,74	251,30
Code4_200w_rp10	8852,40	43,37	191,10	8841,70	66,08	50,60	8884,87	15,15	271,30	**8888,47**	14,39	115,60
Code4_200w_rp15	9106,43	87,24	403,10	9094,30	108,36	120,90	9153,63	52,72	217,60	**9179,03**	32,68	241,30
Code4_200w_rp20	9335,67	113,20	502,40	9334,77	118,92	253,60	9355,37	97,49	362,10	**9404,70**	89,41	388,50
Code511w_rp10	**4906,00**	0,00	39,60	**4906,00**	0,00	16,90	**4906,00**	0,00	9,50	**4906,00**	0,00	8,90
Code511w_rp15	5123,90	7,40	332,30	5126,47	4,03	114,90	**5127,80**	3,99	141,20	5123,00	0,00	63,40
Code511w_rp20	5330,33	9,94	101,10	**5334,00**	0,00	95,20	**5334,00**	0,00	89,90	**5334,00**	0,00	76,00
Code5_150w_rp10	6965,73	10,59	329,10	6966,83	6,43	153,50	**6975,00**	0,00	162,90	**6975,00**	0,00	32,90
Code5_150w_rp15	7084,93	34,91	392,70	7102,20	39,38	205,00	7124,87	27,57	273,90	**7139,97**	26,56	214,60
Code5_150w_rp20	7320,77	8,91	343,80	7314,83	14,82	108,50	7324,90	3,19	344,40	**7326,50**	3,29	227,30
Code5_200w_rp10	9159,13	66,25	338,70	9187,77	60,06	70,60	9229,40	66,03	208,20	**9273,10**	27,45	268,20
Code5_200w_rp15	9167,60	71,37	413,00	9193,97	84,84	118,10	9203,77	86,14	262,50	**9252,03**	42,10	320,90
Code5_200w_rp20	9442,13	90,25	499,20	9429,90	96,86	201,60	9465,43	108,54	340,60	**9512,10**	42,91	345,90
Code611w_rp10	4594,73	1,46	110,90	**4595,00**	0,00	22,60	**4595,00**	0,00	17,80	**4595,00**	0,00	17,70
Code611w_rp15	**4881,00**	0,00	74,50	**4881,00**	0,00	17,50	**4881,00**	0,00	16,90	**4881,00**	0,00	15,70
Code611w_rp20	4941,53	18,58	248,40	4951,13	3,30	95,00	**4952,00**	0,00	120,30	**4952,00**	0,00	96,00
Code6_150w_rp10	7015,83	64,84	242,70	7049,13	4,75	51,10	**7050,00**	0,00	73,20	**7050,00**	0,00	36,90
Code6_150w_rp15	7162,53	32,51	319,90	7150,47	69,57	152,40	**7186,00**	0,00	112,10	**7186,00**	0,00	71,60
Code6_150w_rp20	7356,23	39,25	368,20	7356,77	52,27	262,10	7384,53	8,03	258,50	**7386,00**	0,00	133,90
Code6_200w_rp10	9750,17	130,57	207,40	9800,73	54,01	227,10	9831,83	46,47	340,90	**9850,53**	5,58	197,50
Code6_200w_rp15	10025,27	150,55	442,70	10033,50	115,47	131,70	10123,40	53,01	253,30	**10148,23**	27,71	326,70
Code6_200w_rp20	10192,63	116,02	570,30	10181,83	100,37	229,60	**10274,20**	82,03	366,10	10261,53	91,67	452,50
Code711w_rp10	**5586,00**	0,00	79,70	**5586,00**	0,00	34,90	**5586,00**	0,00	7,20	**5586,00**	0,00	8,70
Code711w_rp15	**5827,00**	0,00	83,90	**5827,00**	0,00	56,70	**5827,00**	0,00	21,10	**5827,00**	0,00	31,60
Code711w_rp20	**5893,00**	0,00	73,60	**5893,00**	0,00	20,10	**5893,00**	0,00	22,80	**5893,00**	0,00	29,80
Code7_150w_rp10	6223,90	44,98	293,90	6247,07	17,08	216,20	6238,17	38,36	316,70	**6248,10**	0,55	190,10
Code7_150w_rp15	**6840,00**	0,00	231,80	6827,97	21,41	44,80	**6840,00**	0,00	59,90	**6840,00**	0,00	82,90
Code7_150w_rp20	7266,87	42,85	290,10	7280,63	39,41	201,00	**7295,77**	6,76	143,40	7290,83	14,02	203,10
Code7_200w_rp10	9142,37	149,21	274,10	9185,57	80,24	264,50	9223,70	57,39	234,20	**9270,30**	20,44	222,80
Code7_200w_rp15	9527,73	61,94	518,30	9533,13	52,77	127,50	9573,93	23,12	367,30	**9580,30**	35,03	283,90
Code7_200w_rp20	9837,63	58,28	476,10	9877,93	61,18	251,20	9918,77	39,99	363,50	**9943,10**	33,88	361,90
Code811w_rp10	4605,20	14,09	301,10	**4609,00**	0,00	116,90	**4609,00**	0,00	91,60	**4609,00**	0,00	21,60
Code811w_rp15	4670,77	11,42	180,70	4662,20	22,13	102,90	**4675,00**	0,00	66,20	**4675,00**	0,00	41,60
Code811w_rp20	**4858,00**	0,00	77,40	**4858,00**	0,00	39,60	**4858,00**	0,00	25,80	**4858,00**	0,00	24,40
Code8_150w_rp10	7729,33	14,61	217,60	**7732,00**	0,00	41,10	**7732,00**	0,00	68,00	**7732,00**	0,00	28,70
Code8_150w_rp15	7659,73	8,45	340,50	7659,20	5,16	155,60	**7662,00**	0,00	139,50	**7662,00**	0,00	103,10
Code8_150w_rp20	7838,13	19,77	285,60	7834,67	30,79	83,30	**7847,33**	9,94	174,50	7846,73	11,06	188,20
Code8_200w_rp10	8997,83	122,64	375,90	9066,20	63,40	149,90	**9092,57**	2,37	149,60	**9092,57**	2,37	170,60
Code8_200w_rp15	8983,57	81,04	530,30	8978,17	119,21	165,70	9061,40	35,42	429,30	**9063,10**	41,76	357,90
Code8_200w_rp20	9230,93	97,88	512,80	9204,63	95,46	236,40	9298,53	66,63	445,50	**9342,90**	23,35	484,30
Code911w_rp10	**5302,00**	0,00	21,80	**5302,00**	0,00	5,70	**5302,00**	0,00	5,30	**5302,00**	0,00	7,50
Code911w_rp15	5155,07	6,15	339,40	5138,07	20,95	220,60	5157,43	1,48	325,60	**5157,93**	0,25	220,60
Code911w_rp20	5457,00	2,03	239,90	5458,07	1,72	109,10	**5459,00**	0,00	203,40	**5459,00**	0,00	92,50
Code9_150w_rp10	6838,70	39,05	220,30	**6855,00**	0,00	46,20	**6855,00**	0,00	41,30	**6855,00**	0,00	55,50
Code9_150w_rp15	6877,93	15,49	274,40	6852,30	49,77	179,80	6880,70	12,07	136,40	**6883,40**	0,93	148,40
Code9_150w_rp20	7073,23	63,63	365,00	7069,77	64,64	99,70	7130,27	44,37	382,80	**7160,40**	41,30	299,90
Code9_200w_rp10	8818,73	119,21	204,00	8897,87	143,78	78,00	8946,43	103,98	238,70	**9011,40**	8,76	182,90
Code9_200w_rp15	9074,87	75,56	445,30	9066,30	81,41	117,50	9144,33	44,68	312,80	**9168,20**	23,40	335,40
Code9_200w_rp20	9409,13	62,21	539,00	9418,93	74,78	231,10	9448,47	16,11	382,90	**9452,57**	16,55	416,80
Code1011w_rp10	5000,53	11,83	110,70	4998,70	11,61	74,70	**5005,00**	0,00	19,10	**5005,00**	0,00	18,20
Code1011w_rp15	**5195,00**	0,00	254,00	**5195,00**	0,00	78,80	**5195,00**	0,00	54,40	**5195,00**	0,00	29,20
Code1011w_rp20	**5399,00**	0,00	222,80	**5399,00**	0,00	91,20	**5399,00**	0,00	72,40	**5399,00**	0,00	60,80
Code10_150w_rp10	6701,30	43,97	114,40	6701,47	35,67	90,90	**6715,00**	0,00	35,30	**6715,00**	0,00	30,20
Code10_150w_rp15	7004,00	34,14	228,80	7010,60	13,56	95,80	**7014,00**	0,00	90,70	**7014,00**	0,00	104,30
Code10_150w_rp20	7176,17	40,36	336,30	7189,13	31,81	260,90	7200,13	17,52	216,30	**7203,40**	10,21	175,30
Code10_200w_rp10	9333,13	118,39	437,50	9400,20	15,97	361,30	9408,63	7,57	329,00	**9411,00**	0,00	151,70
Code10_200w_rp15	9221,83	63,44	465,20	9218,17	109,00	130,30	9287,37	56,94	332,60	**9312,40**	51,91	434,30
Code10_200w_rp20	9585,33	100,70	512,40	9611,67	131,72	224,70	9651,30	143,66	390,50	**9688,73**	74,95	460,40
geometric mean	6995,12			7000,54			7019,37			**7027,16**		
#best results	19			24			43			**85**		
#unique best results	0			0			5			**47**		

A.2 Generalized Vehicle Routing Problem with Stochastic Demands

In this section we show the full result tables of the tests performed with the variable neighborhood search for the GVRPSD with preventive restocking. Table A.5 shows the results of the tests for the different types of the VND as described in Section 5.5.3 for instances with $\theta = 2$ and Table A.6 lists the results for the instances with $\theta = 3$.

Table A.5: Results for the different configurations of the VND for instances with $\theta = 2$.

				FI + VND		GTSP + VND		GTSP + VND + ML-ES	
Instance	n	m	$E[nr]$	*obj*	$t[s]$	*obj*	$t[s]$	*obj*	$t[s]$
P-n16-k8-C8-V5	16	8	4,2	208,511	1	**208,158**	1	**208,158**	1
P-n19-k2-C10-V2	19	10	1,08	**146,825**	13	**146,825**	<1	**146,825**	<1
P-n20-k2-C10-V2	20	10	1,04	**149,024**	13	**149,024**	<1	**149,024**	<1
P-n21-k2-C11-V2	21	11	1,04	162,77	21	**160,481**	<1	**160,481**	<1
P-n22-k2-C11-V2	22	11	1,04	**161,365**	24	162,952	18	162,952	8
P-n22-k8-C11-V5	22	11	3,87	**326,225**	5425	**326,225**	4988	**326,225**	190
P-n23-k8-C12-V5	23	12	4,53	313,084	8	**312,512**	2	**312,512**	2
B-n31-k5-C16-V3	31	16	2,03	**419,912**	57	**419,912**	23	**419,912**	25
A-n32-k5-C16-V2	32	16	2	547,267	65	**539,161**	20	**539,161**	12
A-n33-k5-C17-V3	33	17	2,33	**455,146**	96	455,665	39	455,665	7
A-n33-k6-C17-V3	33	17	2,74	**468,337**	87	468,775	31	468,775	4
A-n34-k5-C17-V3	34	17	2,25	541,021	73	**504,136**	36	**504,136**	6
B-n34-k5-C17-V3	34	17	2,05	**466,803**	113	**466,803**	43	**466,803**	25
B-n35-k5-C18-V3	35	18	2,23	**619,24**	115	**619,24**	45	**619,24**	26
A-n36-k5-C18-V2	36	18	1,98	**506,953**	146	**506,953**	36	**506,953**	7
A-n37-k5-C19-V3	37	19	2,1	**447,859**	145	**447,859**	131	**447,859**	19
A-n37-k6-C19-V3	37	19	2,93	626,813	165	**610,072**	59	**610,072**	13
A-n38-k5-C19-V3	38	19	2,42	485,88	124	**481,977**	0	**481,977**	0
B-n38-k6-C19-V3	38	19	2,75	487,754	157	**479,918**	67	**479,918**	27
A-n39-k5-C20-V3	39	20	2,47	**567,414**	210	567,906	77	567,906	14
A-n39-k6-C20-V3	39	20	2,68	**560,574**	301	561,253	72	561,253	20
B-n39-k5-C20-V3	39	20	2,33	**356,43**	220	356,484	<1	356,484	<1
P-n40-k5-C20-V3	40	20	2,26	299,248	480	**296,443**	133	**296,443**	10
B-n41-k6-C21-V3	41	21	2,88	**483,257**	360	**483,257**	106	**483,257**	28
B-n43-k6-C22-V3	43	22	2,78	**490,23**	337	**490,23**	165	**490,23**	77
A-n44-k6-C22-V3	44	22	2,91	628,954	332	**627,856**	211	**627,856**	44
B-n44-k7-C22-V4	44	22	3,16	**559,111**	435	563,957	126	563,957	80
A-n45-k6-C23-V4	45	23	3,12	622,394	405	**621,23**	150	**621,23**	31
A-n45-k7-C23-V4	45	23	3,16	704,585	476	**692,887**	219	**692,887**	97
B-n45-k5-C23-V3	45	23	2,45	**502,021**	447	**502,021**	285	**502,021**	34
B-n45-k6-C23-V4	45	23	3,05	**480,861**	491	482,913	182	482,913	92
P-n45-k5-C23-V3	45	23	2,35	342,329	865	**340,48**	388	**340,48**	34
A-n46-k7-C23-V4	46	23	3,12	**624,051**	578	**624,051**	156	**624,051**	62
A-n48-k7-C24-V4	48	24	3,25	693,444	684	**686,417**	249	**686,417**	152
B-n50-k7-C25-V4	50	25	3,01	**454,088**	715	**454,088**	216	**454,088**	34
B-n50-k8-C25-V5	50	25	4,26	951,949	659	**923,532**	292	**923,532**	199
P-n50-k10-C25-V5	50	25	4,77	**428,477**	516	431,461	296	431,461	131
P-n50-k7-C25-V4	50	25	3,18	354,473	1133	**354,467**	599	**354,467**	58
P-n50-k8-C25-V4	50	25	3,98	400,619	666	**377,659**	255	**377,659**	57
B-n51-k7-C26-V4	51	26	3,56	699,957	650	**682,701**	157	**682,701**	61
P-n51-k10-C26-V6	51	26	5,04	480,028	691	**451,79**	138	**451,79**	58
B-n52-k7-C26-V4	52	26	3,12	461,436	888	**458,949**	427	**458,949**	91
A-n53-k7-C27-V4	53	27	3,38	643,524	801	**637,534**	442	**637,534**	110
A-n54-k7-C27-V4	54	27	3,43	**719,822**	1126	722,494	287	722,494	91

A-n55-k9-C28-V5	55	28	4,18	**733,084**	1013	733,496	405	733,496	145
P-n55-k10-C28-V5	55	28	4,72	424,796	1639	**424,598**	570	**424,598**	111
P-n55-k15-C28-V8	55	28	7,76	563,252	618	**560,924**	204	**560,924**	91
P-n55-k7-C28-V4	55	28	3,19	**361,871**	3080	374,094	615	374,094	78
P-n55-k8-C28-V4	55	28	3,39	**362,202**	2885	362,205	1269	362,205	108
B-n56-k7-C28-V4	56	28	3,19	**474,611**	1100	478,102	497	478,102	313
B-n57-k7-C29-V4	57	29	3,68	**766,237**	1377	779,483	803	779,483	446
B-n57-k9-C29-V5	57	29	4,19	1018,781	1569	**967,332**	481	**967,332**	395
A-n60-k9-C30-V5	60	30	4,11	897,263	1617	**816,393**	597	**816,393**	257
P-n60-k10-C30-V5	60	30	4,83	459,237	2078	**455,262**	749	**455,262**	225
P-n60-k15-C30-V8	60	30	7,23	580,988	1136	**572,084**	748	**572,084**	316
A-n61-k9-C31-V5	61	31	4,51	**644,702**	2465	662,945	746	662,945	181
A-n62-k8-C31-V4	62	31	3,66	794,707	1680	**755,773**	960	**755,773**	261
A-n63-k10-C32-V5	63	32	4,69	851,511	2033	**830,794**	904	**830,794**	438
A-n63-k9-C32-V5	63	32	4,52	980,597	3037	**946,39**	578	**946,39**	229
B-n63-k10-C32-V5	63	32	4,41	881,188	1933	**852,866**	642	**852,866**	475
A-n64-k9-C32-V5	64	32	4,11	**831,768**	3232	837,309	563	837,309	352
B-n64-k9-C32-V5	64	32	4,17	**514,915**	2574	**514,915**	984	**514,915**	234
A-n65-k9-C33-V5	65	33	4,14	726,861	2908	**712,743**	1050	**712,743**	509
P-n65-k10-C33-V5	65	33	4,78	**498,862**	4287	501,391	1668	501,391	336
B-n66-k9-C33-V5	66	33	4,36	830,581	2983	**818,424**	816	**818,424**	590
B-n67-k10-C34-V5	67	34	4,59	692,05	3376	**674,948**	946	**674,948**	647
B-n68-k9-C34-V5	68	34	4,17	**721,533**	3519	738,479	971	738,479	507
A-n69-k9-C35-V5	69	35	4,3	**695,626**	3547	711,188	1012	711,188	379
P-n70-k10-C35-V5	70	35	4,79	512,773	6176	**504,961**	2145	**504,961**	336
P-n76-k4-C38-V2	76	38	1,95	844,06	>10000	**394,195**	>10000	**394,195**	366
P-n76-k5-C38-V3	76	38	2,44	701,248	>10000	410,097	>10000	**409,933**	439
B-n78-k10-C39-V5	78	39	4,89	843,099	7736	**839,6**	3335	**839,6**	1375
A-n80-k10-C40-V5	80	40	4,43	1065,212	8307	**1049,388**	2035	**1049,388**	1745
M-n101-k10-C51-V5	101	51	4,73	1430,448	>10000	550,072	>10000	**545,681**	4000
P-n101-k4-C51-V2	101	51	1,83	1646,051	>10000	468,871	>10000	**461,77**	780
M-n121-k7-C61-V4	121	61	3,54	2881,864	>10000	765,489	>10000	**760,821**	>10000
M-n151-k12-C76-V6	151	76	5,69	2398,81	>10000	731,037	>10000	**714,182**	>10000
M-n200-k16-C100-V8	200	100	7,92	3314,08	>10000	**905,613**	>10000	906,447	>10000
G-n262-k25-C131-V12	262	131	11,79	13691,641	>10000	**3666,883**	>10000	3667,864	>10000

Table A.6: Results for the different configurations of the VND for instances with $\theta = 3$.

				FI + VND		GTSP + VND		GTSP + VND + ML-ES	
Instance	n	m	$E[nr]$	*obj*	$t[s]$	*obj*	$t[s]$	*obj*	$t[s]$
P-n16-k8-C6-V4	16	6	3	**162,17**	<1	**162,17**	<1	**162,17**	<1
P-n19-k2-C7-V1	19	7	0,71	**112,105**	5	**112,105**	3	**112,105**	<1
P-n20-k2-C7-V1	20	7	0,68	**117,306**	4	**117,306**	<1	**117,306**	<1
P-n21-k2-C7-V1	21	7	0,64	**117,071**	3	**117,071**	<1	**117,071**	<1
P-n22-k2-C8-V1	22	8	0,73	**111,194**	10	**111,194**	5	**111,194**	<1
P-n22-k8-C8-V4	22	8	2,82	**246,082**	2681	**246,082**	1927	**246,082**	97
P-n23-k8-C8-V3	23	8	2,55	194,018	2	**183,586**	1	**183,586**	1
B-n31-k5-C11-V2	31	11	1,38	**355,729**	25	**355,729**	16	**355,729**	5
A-n32-k5-C11-V2	32	11	1,39	**386,909**	20	388,597	10	388,597	2
A-n33-k5-C11-V2	33	11	1,52	**318,028**	17	**318,028**	15	**318,028**	3
A-n33-k6-C11-V2	33	11	1,91	**367,629**	23	**367,629**	16	**367,629**	4
A-n34-k5-C12-V2	34	12	1,66	**419,124**	29	**419,124**	25	**419,124**	4
B-n34-k5-C12-V2	34	12	1,34	**363,089**	33	**363,089**	13	**363,089**	5
B-n35-k5-C12-V2	35	12	1,54	**501,47**	32	**501,47**	14	**501,47**	6
A-n36-k5-C12-V2	36	12	1,34	404,579	30	**399,905**	23	**399,905**	7
A-n37-k5-C13-V2	37	13	1,43	**359,133**	45	**359,133**	20	**359,133**	3
A-n37-k6-C13-V2	37	13	1,95	467,266	31	**430,987**	32	**430,987**	7
A-n38-k5-C13-V2	38	13	1,71	**371,795**	57	**371,795**	20	**371,795**	2

B-n38-k6-C13-V2	38	13	1,93	**386,195**	55	389,241	27	389,241	7
A-n39-k5-C13-V2	39	13	1,48	390,4	47	**371,41**	20	**371,41**	8
A-n39-k6-C13-V2	39	13	1,83	**417,844**	43	**417,844**	40	**417,844**	8
B-n39-k5-C13-V2	39	13	1,45	**281,482**	50	**281,482**	<1	**281,482**	<1
P-n40-k5-C14-V2	40	14	1,51	214,775	175	**214,753**	49	**214,753**	4
B-n41-k6-C14-V2	41	14	1,82	**404,261**	93	**404,261**	34	**404,261**	11
B-n43-k6-C15-V2	43	15	1,81	394,529	74	**347,65**	33	**347,65**	6
A-n44-k6-C15-V2	44	15	2	**505,129**	105	508,981	51	508,981	15
B-n44-k7-C15-V3	44	15	2,24	**402,02**	119	**402,02**	57	**402,02**	25
A-n45-k6-C15-V3	45	15	2,09	**478,219**	105	**478,219**	56	**478,219**	12
A-n45-k7-C15-V3	45	15	2,06	516,508	94	**488,017**	99	**488,017**	34
B-n45-k5-C15-V2	45	15	1,51	**419,613**	116	**419,613**	59	**419,613**	8
B-n45-k6-C15-V2	45	15	1,96	**358,989**	72	**358,989**	83	**358,989**	31
P-n45-k5-C15-V2	45	15	1,61	239,568	172	**239,357**	94	**239,357**	6
A-n46-k7-C16-V3	46	16	2,08	**465,624**	209	471,98	82	471,98	16
A-n48-k7-C16-V3	48	16	2,13	474,21	150	**462,548**	95	**462,548**	35
B-n50-k7-C17-V3	50	17	2,05	**398,38**	160	**398,38**	103	**398,38**	33
B-n50-k8-C17-V3	50	17	2,72	**600,656**	169	605,714	123	605,714	23
P-n50-k10-C17-V4	50	17	3,32	**302,371**	181	**302,371**	106	**302,371**	23
P-n50-k7-C17-V3	50	17	2,21	**261,343**	402	**261,343**	214	**261,343**	19
P-n50-k8-C17-V3	50	17	2,77	**273,27**	276	**273,27**	134	**273,27**	17
B-n51-k7-C17-V3	51	17	2,44	**513,021**	184	**513,021**	64	**513,021**	11
P-n51-k10-C17-V4	51	17	3,31	313,594	162	**313,41**	94	**313,41**	27
B-n52-k7-C18-V3	52	18	2,18	**360,496**	269	**360,496**	143	**360,496**	21
A-n53-k7-C18-V3	53	18	2,09	450,973	268	**443,875**	97	**443,875**	16
A-n54-k7-C18-V3	54	18	2,19	507,805	201	**490,544**	134	**490,544**	41
A-n55-k9-C19-V3	55	19	2,75	475,919	292	**474,048**	114	**474,048**	15
P-n55-k10-C19-V4	55	19	3,21	**311,949**	300	316,648	254	316,648	54
P-n55-k15-C19-V6	55	19	5,27	402,374	290	**396,226**	204	**396,226**	66
P-n55-k7-C19-V3	55	19	2,17	275,083	781	**274,223**	496	**274,223**	31
P-n55-k8-C19-V3	55	19	2,31	**276,328**	796	**276,328**	393	**276,328**	27
B-n56-k7-C19-V3	56	19	2,2	**357,843**	264	358,882	109	358,882	25
B-n57-k7-C19-V3	57	19	2,39	569,003	317	**567,698**	287	**567,698**	128
B-n57-k9-C19-V3	57	19	2,55	**691,991**	377	693,717	285	693,717	124
A-n60-k9-C20-V3	60	20	2,8	**614,515**	517	620,897	361	620,897	117
P-n60-k10-C20-V4	60	20	3,19	340,535	627	**328,893**	324	**328,893**	36
P-n60-k15-C20-V5	60	20	4,78	**372,634**	428	**372,634**	156	**372,634**	57
A-n61-k9-C21-V4	61	21	3,09	495,72	528	**482,511**	219	**482,511**	61
A-n62-k8-C21-V3	62	21	2,59	629,651	382	**617,56**	260	**617,56**	93
A-n63-k10-C21-V4	63	21	3,14	627,932	398	**611,536**	283	**611,536**	80
A-n63-k9-C21-V3	63	21	2,96	678,592	398	**666,458**	277	**666,458**	109
B-n63-k10-C21-V3	63	21	2,9	**604,49**	460	604,701	181	604,701	50
A-n64-k9-C22-V3	64	22	2,55	**551,108**	786	564,462	417	564,462	145
B-n64-k9-C22-V4	64	22	3,17	**457,245**	640	**457,245**	337	**457,245**	85
A-n65-k9-C22-V3	65	22	2,89	550,806	683	**525,03**	259	**525,03**	90
P-n65-k10-C22-V4	65	22	3,23	**378,526**	1326	**378,526**	1082	**378,526**	123
B-n66-k9-C22-V3	66	22	2,8	649,179	568	**627,357**	279	**627,357**	121
B-n67-k10-C23-V4	67	23	3,14	561,816	934	**561,712**	385	**561,712**	109
B-n68-k9-C23-V3	68	23	2,87	546,556	715	**539,815**	441	**539,815**	131
A-n69-k9-C23-V3	69	23	2,94	**523,774**	1130	**523,774**	389	**523,774**	88
P-n70-k10-C24-V4	70	24	3,36	390,951	1649	**386,148**	1088	**386,148**	120
P-n76-k4-C26-V2	76	26	1,33	461,753	>10000	**310,397**	4312	**310,397**	58
P-n76-k5-C26-V2	76	26	1,67	373,937	>10000	**310,397**	3748	**310,397**	56
B-n78-k10-C26-V4	78	26	3,31	**620,112**	1941	**620,112**	774	**620,112**	342
A-n80-k10-C27-V4	80	27	3,04	760,272	1878	**757,547**	943	**757,547**	381
M-n101-k10-C34-V4	101	34	3,2	720,202	>10000	**465,866**	9949	**465,866**	958
P-n101-k4-C34-V2	101	34	1,25	992,679	>10000	**371,926**	9979	**371,926**	397
M-n121-k7-C41-V3	121	41	2,44	1480,015	>10000	559,549	>10000	**550,556**	5051
M-n151-k12-C51-V4	151	51	3,71	1542,827	>10000	526,302	>10000	**491,768**	9090
M-n200-k16-C67-V6	200	67	5,29	2315,641	>10000	661,227	>10000	**657,803**	8382
G-n262-k25-C88-V9	262	88	8,13	8582,402	>10000	**2756,278**	>10000	2757,088	>10000

Curriculum Vitae

Personal Details

Name:	Benjamin Biesinger
Address:	Stättermayergasse 8/21-22, 1150 Vienna, Austria
E-Mail:	bbiesinger@gmail.com
Date of Birth:	January 6th, 1987

Education

since 2012	**PhD studies in Computer Science**, Vienna University of Technology. Dissertation on the topic of "Complete Solution Archives for Evolutionary Combinatorial Optimization"
2009–2012	**Master studies in Computational Intelligence**, Vienna University of Technology. Master thesis on the topic of "Enhancing an Evolutionary Algorithm with a Solution Archive to Reconstruct Cross Cut Shredded Text Documents"
2006–2009	**Bachelor studies in Computer Engineering**, University of Applied Sciences Wr. Neustadt.
1997–2005	**Secondary School**, Bundesgymnasium Babenbergerring Wr. Neustadt.

Working Experience

since 2012	**Research assistant**, Algorithms and Complexity Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology.
2013–2015	**University assistant**, Algorithms and Complexity Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology.
2010–2012	**Student assistent**, Information Technology Services, Vienna University of Technology.
Feb–Apr 2009	**Internship**, Fotec GmbH, software engineer.
Aug–Sep 2008	**Internship**, Fotec GmbH, software engineer.
Jul 2008	**Internship**, SOREX Wireless Solutions GmbH, software engineer.
Jul 2005 / 2006	**Internship**, Bizerba Waagen, technical service department.

Publications

[1] Benjamin Biesinger, Bin Hu, and Günther R. Raidl. A genetic algorithm in combination with a solution archive for solving the generalized vehicle routing problem with stochastic demands, pages 1–29. 2016. submitted to a journal.

[2] Benjamin Biesinger, Bin Hu, and Günther R. Raidl. A memetic algorithm for competitive facility location problems. *In Business and Consumer Analytics: New Directions (Vol1)* (Natalie Jane de Vries and Pablo Moscato, eds.), pages 1–23. 2016. To appear.

Publications (continued)

[3] Benjamin Biesinger, Bin Hu, and Günther R. Raidl. Models and algorithms for competitive facility location problems with different customer behavior. *Annals of Mathematics and Artificial Intelligence*, 76(1):93–119, 2015.

[4] Benjamin Biesinger, Bin Hu, and Günther R. Raidl. An integer L-shaped method for the generalized vehicle routing problem with stochastic demands. In *7th International Network Optimization Conference (INOC)*, 2015. To appear.

[5] Matthias Lanzinger, Benjamin Biesinger, Bin Hu, Günther R. Raidl. A Genetic Algorithm for the Capacity and Distance Constrained Plant Location Problem. *Proceedings of the 11th Metaheuristics International Conference*, pages 89/1–89/9, 2015.

[6] Benjamin Biesinger, Bin Hu, Günther R. Raidl. A Variable Neighborhood Search for the Generalized Vehicle Routing Problem with Stochastic Demands. *Evolutionary Computation in Combinatorial Optimization (EvoCOP'15)* (Gabriela Ochoa, Francisco Chicano, eds.), volume 9026 of LNCS, pages 48-60, 2015, Springer International Publishing.

[7] Christoph Weiler, Benjamin Biesinger, Bin Hu, Günther R. Raidl. Heuristic Approaches for the Probabilistic Traveling Salesman Problem. In *Computer Aided Systems Theory – EUROCAST 2015* (Roberto Moreno-Díaz, Franz Pichler, Alexis Quesada-Arencibia, eds.), volume 9520 of LNCS, pages 342–349, 2015, Springer.

[8] Christoph Weiler, Benjamin Biesinger, Bin Hu, and Günther R. Raidl. Heuristic Approaches for the Probabilistic Traveling Salesman Problem. In *Extended Abstracts of the 15th International Conference on Computer Aided Systems Theory*, pages 99–100, 2015.

[9] Benjamin Biesinger, Bin Hu, Günther R. Raidl. An Evolutionary Algorithm for the Leader-Follower Facility Location Problem with Proportional Customer Behavior. *Conference Proceedings of Learning and Intelligent Optimization Conference (LION 8)*, volume 8426 of LNCS, pages 203-217, 2014, Springer.

[10] Benjamin Biesinger, Bin Hu, Günther R. Raidl. A Hybrid Genetic Algorithm with Solution Archive for the Discrete $(r|p)$-Centroid Problem. *Journal of Heuristics*, 21(3):391–431, 2015.

[11] Benjamin Biesinger, Christian Schauer, Bin Hu, Günther R. Raidl. Enhancing a Genetic Algorithm with a Solution Archive to Reconstruct Cross Cut Shredded Text Documents. In *Computer Aided Systems Theory – EUROCAST 2013* (Roberto Moreno-Díaz, Franz Pichler, Alexis Quesada-Arencibia, eds.), volume 8111 of LNCS, pages 380–387, 2013, Springer.

[12] Benjamin Biesinger, Christian Schauer, Bin Hu, and Günther R. Raidl. Reconstructing cross cut shredded documents with a genetic algorithm with solution archive. In *Extended Abstracts of the 14th International Conference on Computer Aided Systems Theory*, pages 226–228, 2013.