FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Hybrid Metaheuristics and Matheuristics for Problems in Bioinformatics and Transportation

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

DI Sandro Pirkwieser, Bakk.techn.
Registration Number 0116200

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: ao.Univ.-Prof. DI Dr. Günther R. Raidl

The dissertation has been reviewed by:

(ao.Univ.-Prof. DI Dr. Günther R. Raidl)

(Univ.-Prof. Mag. Dr. Karl F. Dörner)

Wien, 25.05.2012

(DI Sandro Pirkwieser, Bakk.techn.)

ERKLÄRUNG ZUR VERFASSUNG DER ARBEIT

DI Sandro Pirkwieser, Bakk.techn.
Ignazgasse 11/30, 1120 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.

(Ort, Datum) (Unterschrift Verfasser)

DANKSAGUNG

Als erstes möchte ich mich bei Prof. Günther Raidl für die wirklich beispiellose Betreuung in all den Jahren bedanken. Die vorliegende Arbeit sowie ich selbst haben sehr davon profitiert durch „Günthers Schule“ gegangen zu sein. Zudem war er auch ein äußerst angenehmer Chef und bleibt vor allem weiterhin ein sehr geschätzter Kollege. Ein Dankeswort möchte ich auch an Prof. Karl Dörner richten, der sich sehr gerne der Zweitbegutachtung angenommen hat.

Generell möchte ich allen Kolleginnen und Kollegen vom ADS danken die mich während des Doktorats begleitet haben. Durch euch werde ich die dortige Zeit in bester Erinnerung behalten. Egal ob es um Forschung, Lehre, private Anliegen, oder einfach nur um Zerstreuung (u.a. „Sinnvolles“) ging, fand ich stets offene Ohren sowie Rat und Tat. An dieser Stelle auch ein Dankeschön an Rubén Ruiz-Torrubiano für die gute Zusammenarbeit zum Thema Consensus Tree als auch für die seither bestehende Freundschaft.

Ich honoriere auch die finanzielle Unterstützung seitens des Bundesministeriums für Wissenschaft und Forschung, des Fonds zur Förderung der wissenschaftlichen Forschung, und des Österreichischen Austauschdienstes (nunmehr OeAD GmBH) im Laufe des Studiums.

Andreas Chwatal, dem Co-Gründer unserer Firma, möchte ich Dank dafür aussprechen, dass er mir des Öfteren den Rücken freigehalten hat um zügiger an der Dissertation weiterarbeiten zu können. Weiterhin auf eine – nun wieder vermehrte – erfolgreiche Zusammenarbeit!

Eine ganz besondere Dankbarkeit empfinde ich gegenüber meiner Familie, insbesondere meinen Eltern und Großeltern. Mittels eurer Unterstützung in vielerlei Hinsicht habt ihr mir einen sorgloseren Bildungsweg ermöglicht aber seid mir auch abseits davon immer in allen Belangen zur Seite gestanden. Ihr habt einen großen Anteil daran, dass ich soweit gekommen bin und nun mein Doktorat abschließen kann. Dabei bedaure ich, die Freude über den Abschluss nicht mehr mit meinen Großvätern teilen zu können.

Der größte Dank jedoch gebührt dir, Marlene. Seit wir zusammen sind bereicherst du ganz entscheidend mein Leben. Ich bin sehr glücklich mittlerweile auch dein Ehemann sein zu dürfen und mit dir eine Familie gegründet zu haben. Du hast mich wo immer möglich unterstützt und bist stets, seit einiger Zeit gemeinsam mit unserer kleinen Johanna, der so wichtige Gegenpol zu Arbeit und Studium. Ich liebe euch über alles und blicke voller Freude und Aufgeregtheit unserem nächsten gemeinsamen Lebensabschnitt, bald zu viert, entgegen!

ABSTRACT

The general aim of this doctoral thesis was to thoroughly investigate diverse *hybrid optimization strategies* for certain classes of $\mathcal{NP}$-hard combinatorial optimization problems. For this basic concepts should be refined and further developed. The ultimate goals were twofold: to come up with highly effective, new state-of-the-art methods for solving the selected benchmark problems, and to gain further experience and knowledge of the specific pros and cons in order to apply the methods more generally in meaningful ways also to other problems.
In general, such hybrids try to combine in various ways the strengths of two or more methods from possibly different streams. It was further intended to focus in particular on combining exact and (meta-)heuristic algorithms, especially exploiting the power of mathematical programming techniques, yielding so-called *matheuristics* (or *model-based metaheuristics*).
Although we did not decide on the problems which would be tackled right from the start—as I was more interested in the methodical aspect—it eventually turned out that we dealt with problems that are not only interesting from an academic perspective but highly relevant in practical application areas, too. The first is the *consensus tree problem* which primarily arises in phylogenetics and thus belongs to the domain of bioinformatics. Its objective is to build a single solution tree out of several phylogenetic trees given as input, somehow best representing the whole available information. All remaining problems arise in the field of transportation and are extensions of the *capacitated vehicle routing problem* (CVRP) motivated by important real-world aspects. These variants are in fact generalizations, as the CVRP can be considered a special case of each one. Following ones are considered: the *periodic vehicle routing problem* and the *periodic vehicle routing problem with time windows*, where customers usually need to be visited multiple times in a given planning horizon, also respecting (hard) customer time windows in case of the latter; the *location-routing problem* as well as the *periodic location-routing problem*, which add to the CVRP the task of simultaneously placing some facilities at given locations (i.e. corresponding to the $\mathcal{NP}$-hard *facility location problem*); and finally the *vehicle routing problem with compartments*, considering not a single loading area and product but several compartments and products, possibly involving certain incompatibilities.
Several forms of hybridization are investigated in this work: collaboratively exchanging solutions, tight integration of the concepts of a method in another one, multilevel refinement, the guidance of a method by information gathered by another one, heuristic column generation as well as heuristic cut separation, very large neighborhood search based on integer linear programming and a more sophisticated variant of it also realizing an optimal merging by exploiting the information of several solutions, and finally solving subproblems to optimality.

We show that for all considered problems a skillfull hybridization of the developed exact and heuristic methods, or of several heuristics, leads to a significant improvement in general. In fact, the exact add-ons for heuristics and vice versa, representing an integrative combination, give in our cases almost always a considerable performance boost to the main (or host) method. Thereby either heuristic components are able to notably reduce the required runtime or exact components can significantly increase the solution quality. Moreover, the collaborative combinations clearly benefit from the diverse algorithms in use.
In addition, the role of the individual methods or the single underlying method is not to be underestimated. In our case variants of variable neighborhood search (VNS) are the most prominent metaheuristics applied, and for all but one problem a solution approach based on VNS is presented for the first time. The simple elegance of VNS offers a great flexibility when it comes to extension as well as specialization, as neighborhood structures can be added like building blocks in order to eventually assemble a powerful solution method. Especially meaningful problem-tailored neighborhood structures, which vary on the level/part of the problem they operate, contribute a lot to the overall success. Combined with appropriate embedded local search components, and in some cases to also accept worse solutions with a certain probability as well as to allow infeasible solutions, we always achieve a good balance of exploration and intensification.
In thorough comparisons to previous solution approaches we almost always achieve at least competitive results. In many cases they are even clearly better, hence obtaining currently leading approaches. This is also documented by numerous new best known solutions obtained. However, the improvement is not only in solution quality, but often our methods also exhibit much better runtime behaviors and thus scalability to larger instances. As a consequence of this, already competitive results can often be obtained with considerably less runtime. Since the means to compare to other approaches are after all quite limited, we are all the more concerned with comparing our "baseline methods" to the subsequently enhanced hybrid methods whenever meaningful. Overall, it turns out that our hybrid methods almost always show statistically significant better results, in some cases for whole instance sets; with nearly none or at most a moderate increase in runtime.
Note that each of these hybrid variants has its strengths and weaknesses, which are addressed in this work. Not surprisingly, none clearly dominates all others and is the preferred variant for each possible problem – also for hybrid methods there is "no free lunch". Nevertheless, our work provides additional guidelines concerning under which conditions which hybridization schemes can be promising. For one thing, our devised matheuristics not only seem promising in particular for other, possibly even richer variants of routing problems, but their concept can fairly easily be applied to other classes of combinatorial optimization problems as well. Especially the applied combination of very large neighborhood search and optimal merging is recommendable for problems exhibiting a similar structure.
Despite all their potential benefits, hybrid methods generally also have some drawbacks which one should be aware of: they have a higher complexity, they require more effort for design and implementation, to combine algorithms/concepts from different streams an appropriate knowledge of each individual stream is a prerequisite, and they are likely to be harder to tune. However, if one copes with these issues such hybridizations might give rise to promising solution approaches for many problems.

KURZFASSUNG

Das allgemeine Ziel dieser Dissertation bestand in der gründlichen Untersuchung unterschiedlicher *hybrider Optimierungsstrategien* für bestimmte Klassen $\mathcal{NP}$-schwieriger kombinatorischer Optimierungsprobleme. Dazu sollten grundlegende Konzepte verfeinert und weiterentwickelt werden. Zweierlei Endziele wurden dabei verfolgt: mit sehr effektiven, neuen State-of-the-Art Methoden zur Lösung der gewählten Benchmarkprobleme aufwarten zu können, als auch weitere Erfahrungen und Wissen hinsichtlich der spezifischen Vor- und Nachteile der Methoden zu erlangen, um sie generell auch sinnvoll auf andere Probleme anzuwenden.
Grundsätzlich versuchen derartige Hybride die Stärken von zwei oder mehr Verfahren, aus möglicherweise verschiedenen Richtungen, auf unterschiedliche Art und Weise zu kombinieren. Weiters war es beabsichtigt den Fokus gezielt auf die Kombination von exakten und (meta-)heuristischen Algorithmen zu legen, um speziell die Stärke von Methoden der mathematischen Programmierung auszunutzen, was in sogenannten *Matheuristiken* (oder *modellbasierten Metaheuristiken*) resultiert. Obwohl wir die zu behandelnden Probleme nicht von Anfang an festgelegt haben – da ich auch eher am methodischen Aspekt interessiert war – stellte sich schließlich heraus, dass diese nicht nur von einem akademischen Standpunkt aus interessant sind, sondern auch hinsichtlich praktischer Anwendungsbereiche eine hohe Relevanz besitzen. Das erste ist das Konsensus-Baum Problem (*Consensus Tree Problem*), welches hauptsächlich in der Phylogenetik auftritt und daher dem Bereich der Bioinformatik angehört. Das Ziel ist einen einzelnen Lösungsbaum anhand von mehreren gegebenen phylogenetischen Bäumen derart zu erstellen, sodass dieser die gesamte Information bestmöglich repräsentiert. Die restlichen Probleme entstammen dem Transportbereich und stellen allesamt Erweiterungen des kapazitierten Tourenplanungsproblems (*Capacitated Vehicle Routing Problem* (CVRP)) dar, die durch wichtige reale Aspekte motiviert sind. Genaugenommen handelt es sich bei diesen Varianten um Generalisierungen des CVRP, da dieses jeweils als Spezialfall angesehen werden kann. Folgende werden betrachtet: das periodische Tourenplanungsproblem (*Periodic Vehicle Routing Problem*) und das periodische Tourenplanungsproblem mit Zeitfenster (*Periodic Vehicle Routing Problem with Time Windows*), bei denen Kunden üblicherweise mehrmals innerhalb eines Planungszeitraums besucht werden müssen, wobei bei zweiterem auch (strikte) kundenseitige Zeitfenster zu berücksichtigen sind; das Standort-Tourenplanungsproblem (*Location-Routing Problem*) als auch das periodische Standort-Tourenplanungsproblem (*Periodic Location-Routing Problem*), welche das CVRP um die Aufgabe erweitern gleichzeitig bestimmte Einrichtungen an gewissen Standorten zu platzieren, welches dem $\mathcal{NP}$-schweren Standortplanungsproblem (*Facility Location Pro-*

blem) entspricht; und schlussendlich das Tourenplanungsproblem mit Ladeabteilen (*Vehicle Routing Problem with Compartments*), bei dem nicht nur eine Ladefläche und ein Produkt sondern mehrere Ladeabteile sowie Produkte berücksichtigt werden, wobei bestimmte Inkompatibilitäten auftreten können.
Verschiedene Arten der Hybridisierung werden in dieser Arbeit betrachtet: kollaboratives Austauschen von Lösungen, enge Integration von Konzepten einer Methode in eine andere, Multilevel Refinement, das Lenken einer Methode anhand von Informationen gewonnen durch eine andere, heuristische Spaltengenerierung als auch heuristisches Separieren von Schnittebenen, sehr große Nachbarschaftssuche basierend auf ganzzahliger linearer Programmierung und eine komplexere Variante davon die zusätzlich ein optimales Kombinieren realisiert, welches die Information von mehreren Lösungen ausnutzt, und zuletzt das optimale Lösen von Subproblemen.
Wir zeigen, dass eine geschickte Hybridisierung der entwickelten exakten und heuristischen Verfahren, oder unterschiedlicher Heuristiken, für alle behandelten Probleme im Allgemeinen zu einer signifikanten Verbesserung führt. Tatsächlich verleihen die exakten Erweiterungen für Heuristiken und umgekehrt, welche eine integrative Kombination darstellen, in so gut wie all unseren Fällen der Haupt- bzw. Host-Methode einen beträchtlichen Performancegewinn. Dabei können entweder heuristische Komponenten erheblich die Laufzeit reduzieren oder exakte Komponenten signifikant die Lösungsgüte erhöhen. Zudem profitieren kollaborative Kombinationen deutlich von den unterschiedlichen Algorithmen in Verwendung.
Des Weiteren darf auch die Rolle der individuellen Verfahren oder des alleinigen zugrundeliegenden Verfahrens nicht unterschätzt werden. In unserem Fall stellen Varianten der *Variablen Nachbarschaftssuche* (VNS) die meist verwendeten Metaheuristiken dar, und für alle außer einem Problem wird zum ersten mal ein auf VNS basierender Lösungsansatz vorgestellt. Die schlichte Eleganz der VNS bietet ein hohes Maß an Flexibilität hinsichtlich Erweiterung und Spezialisierung, da Nachbarschaftsstrukturen wie Bausteine hinzugefügt werden können um letzten Endes ein leistungsfähiges Lösungsverfahren zusammenzustellen. Insbesondere sinnvolle, auf das Problem zugeschnittene Nachbarschaftsstrukturen, die auf unterschiedlichen Leveln/Teilen des Problems operieren, tragen maßgeblich zum Gesamterfolg bei. Kombiniert mit geeigneten eingebetteten Komponenten zur lokalen Suche, und zum Teil der Akzeptanz von schlechteren Lösungen mit einer gewissen Wahrscheinlichkeit sowie dem Erlauben von ungültigen Lösungen, erreichen wir immer eine gute Balance zwischen Exploration und Intensivierung.
In gründlichen Vergleichen zu vorherigen Lösungsansätzen erzielen wir fast immer zumindest gleichwertige Ergebnisse. In vielen Fällen sind diese sogar deutlich besser, womit wir mit derzeit führenden Ansätzen aufwarten können. Dies wird auch durch zahlreich gefundene neue beste Lösungen dokumentiert. Allerdings spiegelt sich die Verbesserung nicht nur in der Lösungsqualität wider, sondern unsere Methoden zeigen oftmals auch ein viel besseres Laufzeitverhalten und damit auch Skalierbarkeit gegenüber größeren Instanzen. Infolgedessen können oftmals gleichwertige Ergebnisse bereits mit wesentlich weniger Laufzeit erzielt werden. Da die Mittel sich mit anderen Ansätzen zu vergleichen doch recht begrenzt sind, sind wir umso mehr damit befasst, wann immer sinnvoll, einen Vergleich unserer "Basismethoden" mit den in weiterer Folge verbesserten hybriden Methoden anzustellen. Insgesamt

stellt sich heraus, dass unsere Hybridverfahren fast immer statistisch signifikant bessere Lösungen aufweisen, teilweise für gesamte Instanz-Sets; mit fast keinem oder allenfalls moderatem Anstieg der Laufzeit.
Es ist zu beachten, dass jede dieser Hybrid-Varianten ihre Stärken und Schwächen hat, welche in dieser Arbeit behandelt werden. Wenig überraschend dominiert keine alle anderen und ist die bevorzugte Variante für alle möglichen Probleme – auch für Hybridverfahren gibt es nichts umsonst („no free lunch“). Dennoch bietet unsere Arbeit zusätzliche Richtlinien unter welchen Bedingungen welche Hybridisierungsschemata vielversprechend sein können. Zum Beispiel erscheinen die erarbeiteten Matheuristiken nicht nur speziell für andere, möglicherweise umfangreichere Varianten von Tourenplanungsproblemen vielversprechend, sondern ihr Konzept lässt sich auch relativ leicht auf andere Klassen von kombinatorischen Optimierungsproblemen anwenden. Vor allem die verwendete Kombination von sehr großer Nachbarschaftssuche und dem optimalen Kombinieren empfiehlt sich für Probleme die eine ähnliche Struktur aufweisen.
Trotz aller möglichen Vorteile haben hybride Verfahren in der Regel auch einige Nachteile, derer man sich bewusst sein sollte: sie besitzen eine höhere Komplexität, sie erfordern mehr Design- als auch Implementierungsaufwand, um Algorithmen/Konzepte aus unterschiedlichen Richtungen zu kombinieren ist ein entsprechendes Wissen jeder einzelnen Richtung eine Voraussetzung, und sie sind sehr wahrscheinlich schwieriger einzustellen bzw. zu parametrisieren. Wenn man jedoch mit diesen Problemen zurechtkommt, dann können derartige Hybridisierungen zu vielversprechenden Lösungsansätzen für viele Probleme führen.

CONTENTS

CHAPTER 1

INTRODUCTION

Once upon a time . . . or to be more precise, in the winter term 2004, I was for the first time effectively introduced to *combinatorial optimization* in the course "Heuristic Optimization Techniques" held by Günther Raidl from the Algorithms and Data Structures group (in the following denoted as ADS) at the Vienna University of Technology. I was immediately fascinated by this field: it offers problems that are often easy to state yet very hard to solve, and there exist many diverse solution methods. In fact the problems are mostly $\mathcal{NP}$-hard, meaning that no polynomial-time and hence efficient algorithm is known for solving them so far, and it is unlikely that one will be found someday (unless $\mathcal{P} = \mathcal{NP}$). In the mentioned course we considered and applied, as the name suggests, primarily *heuristic* (mostly *metaheuristic*) solution approaches. With them one can frequently obtain very good or even (near-)optimal solutions in relatively short time, which, however, comes at the price of having no guarantee about their quality. My interest grew during a project (Praktikum) at the ADS in the course of my master studies in 2005. There I mostly applied an *exact* solution approach to a network design problem. Such approaches are characterized by yielding the optimal solution but taking in the worst case exponential runtime to do so. Despite a lot of research and engineering efforts they are generally only applicable to instances of limited size (though this size definitely increased over the years). In contrast, the performance of heuristics usually scales better with the instance size, making them especially attractive to be applied in practice. Exact and heuristic methods are therefore somewhat diametric and one cannot have optimality and, say, broad applicability at the same time. An adequate quote from [237] on this circumstance:

> An old engineering slogan says, "Fast. Cheap. Reliable. Choose two." Similarly, if $\mathcal{P} \neq \mathcal{NP}$, we can't simultaneously have algorithms that (1) find optimal solutions (2) in polynomial time (3) for any instance. At least one of these

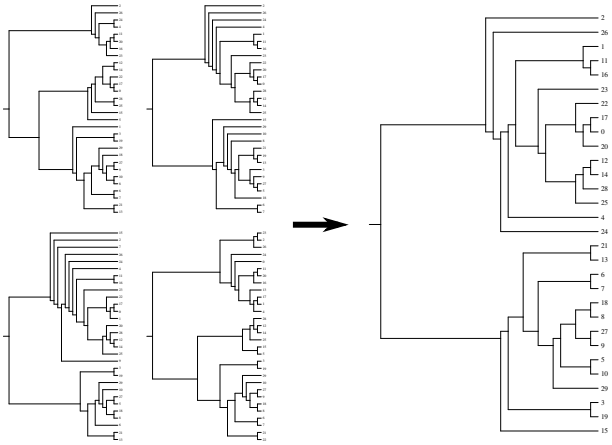


Figure 1.1: Four input trees and a resulting consensus tree.

requirements must be relaxed in any approach to dealing with an $\mathcal{NP}$-hard optimization problem.

Equipped with some knowledge of both "solving worlds" and with even more enthusiasm I decided to write my master's thesis at the ADS, too, under supervision by Günther Raidl and Jakob Puchinger. I tackled the same network design problem as in the project before, this time with different methods, and it turned out that an appropriate combination of them yielded the best results [156, 168]. In this case it was even possible to yield for most instances proven optimal solutions in short time and clearly improve upon previous solution methods. In general, such *hybrid methods* try to combine the strengths of two or more methods—possibly from different streams—in various ways such as to come up with solution approaches outperforming the individual algorithms and showing an overall favorable performance; for more details see Section 2.5. After finishing my master's thesis I was more than ever interested in combinatorial optimization in general, and especially in hybrid algorithms involving exact and (meta-)heuristic parts (which in case the exact method is based on mathematical programming are also referred to as *matheuristics*), such that it was "inevitable" to start my doctorate (PhD) studies at the ADS. The following chapters document my pursuit of more thoroughly investigating and devising such hybrid solution methods, with a focus on combining exact and heuristic algorithms where this appeared promising.

Although this thesis' work is basically rather method-driven it was naturally based on several problems which were chosen as "testbed" for different reasons (explained at the beginning of each corresponding chapter). The first problem we tackle is the *consensus tree problem* (CTP). It arises in *bioinformatics*, a domain where computer science and information technology is applied to the field of biology and medicine. The CTP is related to the inference of phylogenetic trees, which is one of the most important and challenging tasks in systematic biology. From molecular sequence data or another form of dissimilarity information, trees are sought that represent the evolutionary history of a collection of biological entities. The estimation of this evolutionary history is highly useful for many tasks such as multi-

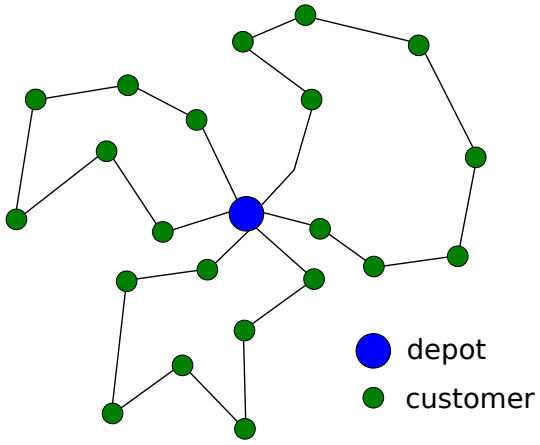


Figure 1.2: Exemplary solution (not likely to be optimal) to a small VRP.

ple sequence alignment, protein structure prediction, or molecular epidemiological studies of viruses. Unfortunately, this inference problem can be shown to be hard under many different formulations. Based on the available data and the used metric, the methods to build phylogenetic trees can roughly be divided into three classes, namely maximum likelihood, distance, and maximum parsimony methods. Different approaches to compute the desired tree exist for each of these classes. Most of them are heuristics due to the complexity of the problem. The different approaches to compute phylogenetic trees in general lead to a collection of different solutions for a specific instance with no information which of these trees is the really correct one from the biological point of view. Hence the objective of the CTP is to build a single solution tree out of several input trees somehow "best" representing the whole available information; see Figure 1.1.

All remaining problems considered in this thesis arise in the field of transportation. In general, transportation problems appear in many practically highly relevant areas of our daily life. They usually include the assignment of produced goods to customers and decisions on how and at which times the goods are picked up and delivered. Improvements in solutions often have a direct and substantial impact on costs and on other important factors like customer satisfaction. Because of the many facets and decisions to be made, such transportation problems are often complex combinations of assignment, scheduling, and routing problems.

The basis for all of them is the $\mathcal{NP}$-hard *vehicle routing problem* (VRP), which is arguably one of the most important, and well-studied, combinatorial optimization problems. It can be considered a generalization of the $\mathcal{NP}$-hard *traveling salesman problem* (TSP), which is perhaps *the* most prominent combinatorial optimization problem. The VRP variant where the vehicles have a certain capacity, the *capacitated VRP* (CVRP), was already introduced in 1959 by Dantzig and Ramsen [54] under the term "truck dispatching problem", where they deal with an optimum routing of a fleet of gasoline delivery trucks between a bulk terminal (depot) and a large number of service stations (customers) supplied by the terminal; see Figure 1.2 for a simple example. There exists a rich literature on the VRP and its many

variants, and a lot of heuristic and exact solution approaches as well as several hybrid variants were proposed. An overview on selected topics is given in a book by Toth and Vigo [220]. More recent works are e.g. a survey paper by Laporte [127] and a book on latest advances and new challenges by Golden et al. [98].

The transportation problems in this work are all extensions of the CVRP motivated by important real-world aspects. These variants are in fact generalizations, as the CVRP can be considered a special case of each one. Following ones are considered here: the *periodic vehicle routing problem* (PVRP) (and the *periodic TSP* (PTSP) as a special case of it) and the *periodic vehicle routing problem with time windows* (PVRPTW) where customers usually need to be visited multiple times in a given planning horizon, also respecting (hard) customer time windows in case of the PVRPTW, the *location-routing problem* (LRP) as well as the *periodic location-routing problem* (PLRP) which adds to the CVRP the task of simultaneously placing some facilities at given locations (i.e. corresponding to the $\mathcal{NP}$-hard *facility location problem*), and finally the *vehicle routing with compartments* (VRPC) where there is not a single loading area and product but several compartments and products, possibly involving certain incompatibilities. Of course more details are given in the corresponding chapters.

Note that two possibilities were suggested in [54] to actually solve VRPs: "The calculations may be readily performed by hand or by an automatic digital computing machine." Though to be fair, we should mention that they were faced with a problem involving four vehicles and twelve customers. Anyway, after a careful consideration we chose the latter option, otherwise we would surely have . . . *calculated happily ever after.*

1.1 Outline of the Thesis

The remainder of this thesis is organized as follows. In the next chapter we give a short introduction to the types of problems we are facing, i.e. mainly combinatorial optimization problems as well as to a lesser extent constraint satisfaction problems, and to computational complexity. Next we review prominent exact and (meta-)heuristic solution approaches, with a focus on those that are actually applied in this work. Finally we also describe the motivations and benefits of devising hybrid methods, concentrating on several use cases.

In general, each chapter on a specific problem gives a proper introduction, also stating our previously presented and published work, discusses previous and related work, details our contributions, introduces available as well as often also newly generated instances, reports on the results and findings of thorough computational tests (probably at more than one place), and finishes with conclusions and ideas for potential future work. In the following we will therefore only outline our contributions in short. Note that all newly generated test instances are publicly available at `https://www.ads.tuwien.ac.at/w/Research/Problem_Instances`.

The CTP is topic of Chapter 3. The major part deals with several metaheuristics and appropriate combinations of them to maximize a fine-grained non-linear similarity measure. For this we introduce several meaningful tree neighborhood structures with incremental update

schemes which are subsequently utilized to extend an existing evolutionary algorithm (EA) to yield a memetic algorithm (MA). We further propose a variable neighborhood descent (VND) and a variable neighborhood search (VNS) also based on them, probably embedding the VND inside the VNS, obtaining a so-called general VNS. Sequential and intertwined collaborative combinations of the EA/MA and the VND/VNS to yield hybrid metaheuristics are presented next. We also propose to examine the moves defined by a neighborhood structure in the order of their improvement potential according to a measure related to the objective function, hence realizing guided neighborhoods. In a second line of work we investigate two integer linear programming (ILP) formulations based on other, linear objective functions. Also the heuristic generation of variables (heuristic column generation), so-called lazy constraints to speed up the solving process, and the combination with the developed metaheuristics are considered. For testing we generate additional instances according to a developed scheme.

Chapter 4 is about the PVRPTW. Due to presenting results after most individual method sections and not only at the end, all test instances used, including newly generated ones, are described before the algorithms. We introduce a VNS for the problem, which is itself a hybrid variant as it integrates the concept of simulated annealing to better escape local optima. Next we realize a cooperative multistart search via multiple cooperating VNS instances performed in an intertwined way, which we denote as multiple VNS (mVNS). Right after describing an EA to be hybridized later, we introduce a set-covering ILP formulation for the PVRPTW which gives rise to a column generation (CG) approach. An exact labeling algorithm based on dynamic programming using different dominance rules (in a cascade) and several (meta-)heuristics to solve the $\mathcal{NP}$-hard pricing subproblem are presented. This CG approach is then extended by a branching scheme to yield a branch-and-price approach, which is subsequently extended to branch-and-cut-and-price. For the latter we adapt the 2-path cuts to the problem at hand as well as apply the subset-row cuts. The separation of the 2-path cuts involves a heuristic component similar to VND. Finally coming to the core of this chapter, the different matheuristics: VNS and mVNS are each hybridized with the set-covering ILP, realizing combined variants of very large neighborhood search and optimal merging, the CG approach is combined with the EA, where information of the former guides the latter, and the concepts of two other variants are described as well. A straightforward column generation based heuristic is further devised to compare to (beside the individual algorithms). Extensive computational results for a diverse set of instances are reported.

In Chapter 5 we tackle the PVRP and the PTSP. We merge a VNS, which is conceptually similar to those of the PVRPTW, with the idea of the multilevel refinement strategy. To arrive at what we call a *multilevel VNS* we introduce a suitable coarsening scheme based on segments, incorporating the periodic aspect, as well as a corresponding solution-based recoarsening scheme. Contrary to existing approaches the multilevel refinement is smoothly integrated into the VNS. Subsequently we also propose an according multilevel VND, which utilizes the multilevel refinement in a more standard way. For evaluation we created larger instances than previously available, as multilevel refinement is especially suitable to large instances in general.

Chapter 6 is dedicated to hybrid solution approaches for the LRP and the PLRP. Again we design a similar (hybrid) VNS than before using problem-specific neighborhood structures for shaking and, building upon previous experience, similar well-performing local improvement methods. It can more or less be readily applied to both problem variants. Later we devise two conceptually different ILP-based very large neighborhood searches. One is rather high-level and operates on the routes as well as on the depots (facilities), whereas the other is lower-level and operates on customers, i.e. on route sequences. For the first search two variants are proposed, a simpler one using information from a single incumbent solution only and a more sophisticated one which can utilize the information of a set of solutions. In case of the PLRP the periodic aspect needs to be dealt with accordingly. Finally the VNS is combined with several combinations of these searches, realizing integrative combinations.

The research conducted on the last problem considered in this thesis, the VRPC, is documented in Chapter 7. After introducing possible problem scenarios which are considered later on we directly deal with the core of the problem: the packing subproblem, which we denote as the compartment assignment problem (CAP). In contrast to previous work we devoted quite some effort for tackling it: The investigated solution methods range from simple construction heuristics over heuristic improvement methods, made possible via introducing a suitable density measure, to exact solution approaches based on ILP techniques and on constraint programming. Next also for the VRPC a solution method based on VNS is proposed, which is in contrast to the other VRP variants a "pure" VNS here, as it only accepts improved solutions. We more concentrate on devising meaningful problem-specific neighborhood structures for shaking. For re-insertion of customers we propose a greedy insertion as well as a regret-k insertion. Building upon the VNS components we also derive an adaptive large neighborhood search. To better test all our extensions we eventually generate additional instances exhibiting a harder packing problem, as the available ones turned out to be too "easy" with regard to this.

Overall conclusions are drawn in Chapter 8.

In Appendix A we provide supplementary material of the CTP and the VRPC, while Appendix B closes the thesis with my curriculum vitae.

CHAPTER 2

METHODOLOGIES

In this chapter we will present concepts and general solution approaches which essentially build the basis for the upcoming chapters. However, it is not our intention to present all methodologies in detail here, which is clearly not the purpose of this work and would also go beyond its scope. Moreover, there are a lot of good books and articles available which specifically provide an in-depth coverage. Hence we will give a short overview, cite classical as well as recent works, and rather highlight the main concepts. We further especially concentrate on methods which are applied in the remainder of this work. Some parts of this chapter have been published in similar form in our previous work [187].

Available techniques for solving hard combinatorial optimization problems can roughly be classified into two main categories: *exact* and *heuristic* algorithms. Exact algorithms are guaranteed to find an optimal solution and prove its optimality. Their run-time, however, often increases dramatically with a problem instance's size, and frequently only small or moderately-sized instances can be practically solved to proven optimality. For larger instances the only possibility usually is to turn to heuristic algorithms that trade optimality for run-time, i.e., they are designed to obtain good but not necessarily optimal solutions in reasonable time.

2.1 Problem Variants

Before coming to the different solution approaches, we characterize the actual types of problems that are considered. Note that a *problem* is a general class, and when we are given specific input values, we say this is an *instance* of the problem; i.e. a problem is a set of instances. A more formal definition, unifying those presented in [22, 23, 154], is the following:

Definition 1 *An instance $\mathcal{I}$ of a problem $\mathcal{P}$ is a quadruple (X, D, C, f) with*

- *a finite tuple of variables $X = (x_1, \ldots, x_n)$,*
- *corresponding variable domains $D_1, \ldots, D_n$, yielding the overall domain as the Cartesian product $D = D_1 \times \ldots \times D_n$,*
- *constraints C among variables, each defined on a subset of D,*
- *and an objective function f to be minimized or maximized (depending on the problem), where $f : D \to \mathbb{R}^+$.*

The set of all possible assignments $\mathcal{S}$, not necessarily respecting the constraints, is the *search (or solution) space* or the set of *candidate solutions*. Every $s \in \mathcal{S}$ is assigned an *objective value* $f(s)$. Naturally, we are more interested in the set of *feasible solutions*:

$$\mathcal{S}^f = \{s = \{(x_1, v_1), \ldots, (x_n, v_n)\} \mid v_i \in D_i, s \text{ satisfies all the constraints } C\} .$$

Solving an optimization problem amounts to finding a best solution, defined as a *globally optimal solution* or *global optimum*.

Definition 2 *A solution $s^* \in \mathcal{S}^f$ is said to be globally optimal if, assuming a minimization problem, $\forall s \in \mathcal{S}^f : f(s^*) \leq f(s)$.*

Following this there can be multiple globally optimal solutions. Note that in the remainder of this chapter we assume without loss of generality a minimization problem, since a maximization problem can be easily transformed into its corresponding minimization variant by taking $-f$. We can basically differentiate between problems having real-valued or discrete variable domains, being subject to *continuous* or *discrete* optimization, respectively. In this work we are mainly interested in solving discrete optimization problems, which are due to their nature also denoted as *combinatorial optimization problems* (COPs). An additional rather informal definition of COPs according to [154] is to look for an object from a given basic set (either finite or countably infinite), usually being an integer number, a subset, a permutation or a graph structure.
Somewhat related are *constraint satisfaction problems* (CSPs), where the focus entirely lies on the feasibility aspect and feasible solutions are not distinguished. CSPs could be regarded as COPs having a constant objective function. In fact, there is also the notion of *constraint optimization problems* which are CSPs together with an objective function, hence closing the circle.

2.2 Computational Complexity

In this section we will glimpse into the subject of computational complexity theory, which primarily deals with complexity classes in general as well as to identify the hardness of specific problems and their membership in one of these classes. For in-depth information we

refer to [154, 126], dealing among others with this topic, and to [89, 211, 212, 99] especially focusing on it. Note that in these investigations corresponding decision variants (yielding a "yes" or "no" answer) of optimization problems are considered, i.e. asking whether a solution having an objective value less than a requested value exists. Furthermore, the algorithms are assumed be executed on a common fictitious machine, the *Turing machine*, an abstraction of a real computer. This does, however, not impair the general findings, as the Church-Turing thesis suggests that this abstract device and all other (reasonable) computational models are equally powerful.

Definition 1 *The time complexity function of an algorithm expresses its time requirement by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size.*

Definition 2 *Big Oh. A function $f(n)$ is $\mathcal{O}(g(n))$ whenever there exist constants $c > 0, n_0 \in \mathbb{N}$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all values of $n \geq n_0$. Thus $c \cdot g(n)$ is an (asymptotic) upper bound for $f(n)$.*

Definition 3 *An algorithm runs in polynomial time (or is a polynomial time algorithm) if its time complexity function is $\mathcal{O}(p(n))$, where p is some polynomial function and n is the size of the instance (or its input length). If k is the largest exponent of such a polynomial in n, the corresponding problem is said to be solvable in $\mathcal{O}(n^k)$.*

Definition 4 *If an algorithm's time complexity function cannot be bounded by a polynomial in n, the algorithm is called an exponential time algorithm.*

Definition 5 *An optimization or constraint satisfaction problem is efficiently solvable if there exists a polynomial time algorithm for solving it. The problem is then considered "well-solved".*

Definition 6 *The class of problems that are efficiently solvable is denoted by $\mathcal{P}$ (standing for polynomial time).*

There are other algorithms which are in practice (i.e. for most practical instances) frequently considered efficient enough: the *pseudo-polynomial algorithms*. For them the time complexity function is polynomial in the size of the instances but also depends on actual instance input numbers. This refinement is made because when only considering the instance size n then parameters might be contained whose size is exponential in n, although for many problems the magnitudes are implicitly bounded by a polynomial in n. Hence a pseudo-polynomial algorithm runs in polynomial time when all input numbers are represented in *unary* (in base 1, i.e., as a sum of 1s), but in exponential time when all input numbers are represented in binary.

Unfortunately many (important) COPs do not seem to lie inside $\mathcal{P}$ since no polynomial time algorithms are known to solve them in general, but only solution approaches taking

exponential time in the worst case. This at most exponential effort originates from the fact that theoretically all possible solutions could be investigated, which is to some respect the "fallback strategy" which will be mentioned in the next section. However, for these problems it is at least possible to efficiently check the validity of a given solution. Otherwise we would even not be able to efficiently recognize a solution, yet preventing the heuristic solution approaches mentioned later. Such "intractable" problems belong to the class $\mathcal{NP}$, where it holds that $\mathcal{P} \subseteq \mathcal{NP}$. Yet despite tremendous research efforts it is up to date not known whether $\mathcal{NP} \subseteq \mathcal{P}$ and hence $\mathcal{P} = \mathcal{NP}$ (being one of the great unsolved problems of mathematics[1]), though it is conjectured that $\mathcal{P} \neq \mathcal{NP}$. Letting theory aside, also intuition suggests the latter, since solving would demand the same effort than checking/recognizing otherwise. A statement by Scott Aaronson also highlights this[2]:

> If $\mathcal{P} = \mathcal{NP}$, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps", no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett. ...

Inside $\mathcal{NP}$ exist so-called $\mathcal{NP}$-complete problems which are considered as the "hardest" among them. The ancestor of all such problems is the Boolean satisfiability problem (SAT), which Cook proved to be $\mathcal{NP}$-complete [36] (known as Cook's theorem or Cook-Levin theorem).

Definition 7 *A (decision) problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and every problem in $\mathcal{NP}$ is reducible to it in polynomial time.*

Showing that two problems are related is done by "reducing" one to the other.

Definition 8 *A **reduction** from problem A to B is a (polynomial time) constructive transformation that maps any instance of A into an equivalent instance of B. These are further called many-one reductions, and are denoted as $A \leq_m B$.*

Hence $A \leq_m B$ implies that any algorithm that solves B is also able to solve A, where a polynomial Since then many other problems where also shown to be $\mathcal{NP}$-complete, starting with the list of 21 problems by Karp [120]. This means that solving one of them efficiently would imply that all of them can be solved in an efficient way. Given the fact that for many problems specifically dedicated solution algorithms were devised, yet not "efficient" ones as denoted above, suggests even more that $\mathcal{P} \neq \mathcal{NP}$. There is further the notion of an $\mathcal{NP}$-hard problem to which every $\mathcal{NP}$-complete problem is reducible to, yet it does not necessarily lie in $\mathcal{NP}$ itself. They are therefore at least as hard as $\mathcal{NP}$-complete problems. Finally, if there exists a pseudo-polynomial algorithm for an $\mathcal{NP}$-complete or $\mathcal{NP}$-hard problem,

[1] see at `http://www.claymath.org/millennium/P_vs_NP/`

[2] see "The Philosophical Argument" at `http://www.scottaaronson.com/blog/?p=122`

then it is denoted as weakly $\mathcal{NP}$-complete and weakly $\mathcal{NP}$-hard, respectively. Contrary, if representing the input numbers of a problem in unary still does not permit to solve it in polynomial time it is denoted as strongly $\mathcal{NP}$-complete or strongly $\mathcal{NP}$-hard. In fact, the question is whether we deal with a *number problem*:

Definition 9 *A problem is a number problem when the magnitudes are not polynomially bounded by the instance size.*

Under the assumption $\mathcal{P} \neq \mathcal{NP}$ only $\mathcal{NP}$-complete problems that are number problems are potential candidates for being solved by pseudo-polynomial time algorithms.

Note that the theoretical definition of "efficiently solvable" not necessarily correlates with an efficient solvability in practice. Often instances of $\mathcal{NP}$-hard problems can be solved well, whereas solving instances of problems in $\mathcal{P}$ to optimality might require too much effort.

2.3 Exact Solution Approaches

Perhaps from a methodical point of view the simplest exact approach would be a complete enumeration of all possible assignments $\mathcal{S}$ (also referred to as exhaustive or brute-force search). Due to the inherent *combinatorial explosion* with respect to the size of the search space for hard COPs in general, this approach is only viable for very small instances. Therefore all practical exact solution approaches try to consider as much of the search space as possible only implicitly, hence ruling out regions where it is guaranteed that no better feasible solution can be found than a previously found one. Often these methods are based on a *tree search*, where the search space is recursively partitioned in a divide-and-conquer manner (see Section 2.3.2) into mutually disjoint subspaces by fixing certain variables or imposing additional constraints. Ruling out regions then amounts to (substantially) pruning the search tree. The scalability of a tree search thus depends essentially on the efficiency of this pruning mechanism. In *branch-and-bound* (B&B), upper and lower bounds are determined for the objective values of solutions, and subspaces for which the lower bounds exceed the upper bounds are discarded.

Next we will have a look at prominent exact solution approaches which will be used to differing extent in later chapters.

2.3.1 Integer Programming Techniques

This section introduces some basic notations and gives a short introduction into prominent integer programming techniques. For an in-depth coverage of the subject we refer to books on linear optimization [17, 55, 56, 223] as well as on combinatorial and integer optimization [149, 239, 18]. Further some important classical articles as well as works on current topics regarding IP are given in the book *50 Years of Integer Programming: 1958–2008* [115]. We also recommend a more informal paper about linear programming (also clarifying the "programming") by Dantzig [52].

A *linear program* (LP) is an optimization problem with a linear objective function subject to a set of constraints expressed as linear (in)equalities. A linear program where all the variables are required to be integers is an *integer (linear) program* (IP). We consider IP problems of the form

$$z_{\mathrm{IP}} = \min\{cx \mid Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}, \tag{2.1}$$

where x is an n-dimensional integer variable vector in column form and $c \in \mathbb{Q}^n$ an n-dimensional row vector. Their dot-product cx is the *objective function* that should be minimized. Matrix $A \in \mathbb{Q}^{m \times n}$ and the m-dimensional column vector $b \in \mathbb{Q}^m$ together define m inequality constraints. A *mixed integer program* (MIP) would involve a combination of integer and real-valued variables and can be written similarly as:

$$z_{\mathrm{MIP}} = \min\{cx + fy \mid Ax + By \geq d, x, y \geq 0, x \in \mathbb{Z}^n\}, \tag{2.2}$$

Maximization problems can be transformed into minimization problems by simply changing the sign of c. Less-than constraints are similarly brought into greater-than-or-equal form by changing the sign of the corresponding coefficients, and equalities can be translated to pairs of inequalities. Thus, we can handle all kinds of linear constraints by appropriate transformations. Without loss of generality, we may therefore restrict our following considerations to minimization problems of this standard form.

Relaxations and Duality

One of the most important concepts in integer programming are *relaxations*, where some or all constraints of a problem are loosened or omitted. Relaxations are mostly used to obtain related, simpler problems that can be solved efficiently yielding bounds and approximate (not necessarily feasible) solutions for the original problem. Embedded within a B&B framework, these techniques may lead to effective exact solution techniques.

The *linear programming* (LP) relaxation of the IP (2.1) is obtained by relaxing the integrality constraints, yielding

$$z_{\mathrm{LP}} = \min\{cx \mid Ax \geq b, x \geq 0, x \in \mathbb{R}^n\}. \tag{2.3}$$

Large instances of such LPs can be efficiently solved using simplex-based [55] or interior-point [119] algorithms. Although there exist scenarios where the simplex algorithm shows an exponential runtime (see the Klee-Minty cubes [124]), its average runtime is rather polynomial and it is known to be highly effective in practice. Therefore, it is today the most frequently used "workhorse" when it comes to solving LPs. Contrary, the interior-point algorithm has a guaranteed polynomial worst case runtime, and is usually also present in leading solver packages. The ellipsoid algorithm [121], despite also having a polynomial runtime, is more interesting from a theoretical perspective.

The solution to the LP relaxation provides a lower bound for the original minimization problem, i.e. $z_{\mathrm{IP}} \geq z_{\mathrm{LP}}$, since the search space of the IP is contained within the one of the LP and the objective function remains the same.

We can further associate a *dual problem* to an LP (2.3), which is defined by

$$w_{\mathrm{LP}} = \max\{ub \mid uA \leq c, u \geq 0, u \in \mathbb{R}^m\} \tag{2.4}$$

with u being the m-dimensional dual variable row vector. The dual of the dual LP is the original (*primal*) LP again. Important relations between the primal problem and its dual are known as weak and strong duality theorems, respectively:

Weak duality theorem: The value of every finite feasible solution to the dual problem is a lower bound for the primal problem, and each value of a finite feasible solution to the primal problem is an upper bound for the dual problem. As a consequence, if the dual is unbounded, the primal is infeasible and vice versa.

Strong duality theorem: If the primal has a finite optimal solution with value z^*_{LP}, than its dual has the same optimal solution value $w^*_{\mathrm{LP}} = z^*_{\mathrm{LP}}$ and vice versa.

The complementary slackness conditions follow from the strong duality theorem: Suppose x and u are feasible solutions for (2.3) and (2.4), respectively; then they are optimal if and only if the following conditions hold:

$$u(Ax - b) = 0 \quad \text{and} \tag{2.5}$$
$$x(c - uA) = 0. \tag{2.6}$$

In case of an IP we have to distinguish between weak and strong duals: A *weak dual* of an IP (2.1) is any maximization problem $w = \max\{w(u) \mid u \in S_{\mathrm{D}}\}$ such that $w(u) \leq cx$ for all $x \in \{Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}$. An obvious weak dual of (2.1) is the dual (2.4) of its LP relaxation (2.3). A *strong dual* is a weak dual that further has an optimal solution u^* such that $w(u^*) = cx^*$ for an optimal solution x^* of (2.1). For solving IPs, weak duals which are iteratively strengthened during the course of the optimization process are often utilized.

LP-Based Branch-and-Bound

By solving the LP relaxation of an IP we obtain a lower bound on the optimal IP solution value and the solution will in general contain fractional variable values. (If all variable values would be integer, we already would have solved the IP.) The standard way to continue towards an optimal integer solution is the already mentioned B&B. Branching usually takes place over some variable x_i with a fractional LP-value x^*_i, defining as first subproblem the IP with the additional inequality $x_i \leq \lfloor x^*_i \rfloor$ and as second subproblem the IP with inequality $x_i \geq \lceil x^*_i \rceil$. For these subproblems with the additional branching constraints, the LP relaxations are resolved leading to increased lower bounds and eventually solutions where all integer variables have integral values. As mentioned in the introduction, primal heuristics are usually also applied to each subproblem in order to find improved feasible solutions and corresponding global upper bounds, enabling a stronger pruning of the search tree.

Cutting Plane Algorithm and Branch-and-Cut

When modeling COPs as IPs an important goal is to find a *strong* formulation, for which the solution value of the LP relaxation in general provides a *tight* bound. For many COPs it is possible to strengthen an existing IP formulation significantly by including further inequalities, which would actually be redundant w.r.t. the integer optimum. In general it is even possible to strengthen a model such that the LP relaxation already yields an integer optimum; however, the number of required constraints often grows exponentially with the problem size. Naively solving such an LP by standard techniques might quickly become too costly in practice.

Dantzig et al. [53] proposed the *cutting plane algorithm* for this purpose, which usually only considers a fraction of all constraints explicitly but is nevertheless able to determine an optimal solution to the whole LP.

The cutting plane approach starts by solving a reduced LP consisting of a small subset of initial inequalities only. It then tries to find inequalities that are violated by the obtained solution but are valid for the original problem (i.e. contained in the full LP). These valid inequalities are called *cuts* or *cutting planes*, and they are added to the current reduced LP, which is then resolved. The whole process is iterated until no further cutting planes can be determined. If the algorithm computing the cuts provides a proof that no further violated inequality exists, the final solution is optimal for the original full LP. The subproblem of identifying cuts is called *separation problem*. In practice it is crucial to have an efficient method for separating cuts as usually a significant number of valid inequalities must be derived until the cutting plane algorithm terminates.

From a theoretical point of view it is possible to solve any IP using a pure cutting plane approach with appropriate classes of cuts. There exist generic types of cuts, such as the Chvatal-Gomory cuts [239], which guarantee such a result. In practice, however, it may take far too long for such a cutting plane approach to converge to the optimum, partly because it is often a hard subproblem to separate effective cuts and partly because of the large number of needed cuts.

The combination of B&B with cutting plane methods yields the highly effective class of *branch-and-cut* algorithms which are widely used. Specialized branch-and-cut approaches have been described for many applications and are known for their effectiveness. Cut separation is usually applied at each node of the B&B tree to tighten the bounds of the LP relaxation and to exclude infeasible solutions as far as possible.

For cutting plane separation effective heuristic methods come into play once again: For strengthening the LP relaxations it is often sufficient to generate cuts heuristically since the correctness of the final solution does not depend on the generated cuts as long as they are valid. Almost all modern mixed integer programming (MIP) solvers include sophisticated generic cut separation heuristics, and they play a major role in the success of these solvers.

Column Generation and Branch-and-Price

Often it is possible to model COPs via strong formulations involving a huge number of variables. Dantzig-Wolfe decomposition [57] is a technique for obtaining such models from compact formulations in a systematic way. It replaces the original problem variables by linear combinations of the extreme points and extreme rays of the original search space, yielding a potentially exponential number of new variables. The obtained models can result in much stronger relaxations than their compact counterparts.

Despite the many variables, the LP relaxations of such formulations can often be efficiently calculated. The *column generation* approach starts with only a small subset of all variables (corresponding to columns in the matrix notation of the IP) and solves the respective restricted LP relaxation. It is then tried to identify one or more so far ignored variables whose inclusion may lead to an improved solution. This subproblem is called *pricing problem*. For a minimization problem a variable can eventually improve the current LP solution if it has negative reduced costs. After adding such a new variable to the restricted LP, it is resolved and the process iterated until no further variables with negative reduced costs exist. The final solution is an optimal solution for the complete LP.

Column generation can be seen as dual to the cutting plane approach, since inequalities correspond to variables in the dual LP. For a recent review on column generation see [131] as well as the book [61]. The cutting stock problem is an early example for the successful application of column generation based methods [94]. The task is to cut some one-dimensional blanks of fixed size into several pieces to satisfy customer demands. Instead of directly deciding on which blank a particular piece is to be cut from, one might consider the cutting of blanks according to some pattern. Hence every possible cutting pattern is represented by a variable and the pricing problem corresponds to the classical 0–1 knapsack problem, which can be solved efficiently in pseudo-polynomial time.

As the column generation algorithm only solves the LP relaxation, it must in general also be embedded in a B&B in order to obtain optimal integer solutions. When column generation is performed for each node of the B&B tree, the approach is called *branch-and-price*. One of the main difficulties in the implementation of such methods frequently lies in the development of appropriate branching rules as branching on the set of dynamically generated variables would typically split the search space in a very skewed and ineffective way. Furthermore, the individual LPs may sometimes be degenerated, or newly added columns may only improve the solutions marginally leading to many iterations until convergence. In the latter cases, stabilization techniques as discussed in [67] often improve the situation. A more recent treatment of stabilization for constrained tree problems is provided by Leitner et al.[129].

Similarly as cutting plane separation may be performed by effective heuristics, one can also heuristically solve the pricing problem in column generation. Care must be taken that in the final iteration it is necessary to prove that no further columns with negative reduced costs exist so that the obtained solution value is guaranteed to be a lower bound for the original IP.

Finally, it occasionally makes sense to combine a cutting plane approach with column generation and embed both in B&B. Such methods, called *branch-and-cut-and-price*, are sometimes extremely successful but are typically also rather complex and highly specialized.

2.3.2 Dynamic Programming

Dynamic programming (DP), proposed by Richard Bellman [14], basically incorporates the divide-and-conquer principle and is a powerful paradigm for algorithm design in general. The name was chosen such that "*it was something not even a Congressman could object to*" [66]. Also see Chapter 15 of [44] for an introduction.

For "pure" divide-and-conquer the recursively solved subproblems are disjoint and there is a unique way to combine them to eventually yield an overall optimal solution to the problem. DP is typically applied to optimization problems and following conditions must hold to successfully apply it: (parts of) the subproblems are overlapping, and recursively solving the overall problem in a bottom-up fashion amounts to choosing the right subproblem solutions (i.e. the problem exhibits an *optimal substructure*). Perhaps the most crucial part is that the subproblems are not disjoint or independent anymore. This fact is exploited via storing their solution's values in some sort of table (or another systematic way) to efficiently retrieve them at the re-occurrence of the subproblems. Hence memory is traded for computational effort. Often the actual solution needs to be reconstructed afterwards, albeit it is usually possible to already derive the required information during the solution process.

A well-known algorithm utilizing these ideas is the Floyd-Warshall algorithm to compute all-pairs shortest paths in a graph with possibly negative edge weights [79, 233]. Another prime example is the 0–1 knapsack problem (with integer numbers), where a pseudo-polynomial algorithm based on DP can be devised [134]. Note that in [89] the authors report that all pseudo-polynomial algorithms for $\mathcal{NP}$-hard problems known to them make use of DP or similar techniques.

2.3.3 Constraint Programming

Whereas integer linear programming is mainly applied to optimization problems, *constraint programming* (CP) is primarily utilized to solve constraint satisfaction problems. In general both concepts are somewhat complementary to each other. Also in CP the term "programming" is actually related to "computer programming", too, as besides being a (declarative) programming paradigm, the user often needs to program the strategy to solve the problem as well. Nevertheless, the idea still is that the user states a problem (via constraints) which is then solved by a general purpose constraint solver. For an introduction we refer to the book by Apt [10], a broad overview also including some more advanced topics is given in the *Handbook of CP* by Rossi et al.[200], a comparison between ILP and CP is presented by Lustig and Puget [132].

The two main basically orthogonal concepts for solving CSPs are *search* and *inference*. For CSPs also a tree-search method is used: *backtracking*. It differs from exhaustive search by checking the constraints after each branching decision and in—the simplest case—going back to the previous node in case a violation is encountered and continue the search from there. Hence also here subtrees are pruned. Though the search component alone would be sufficient to solve the problem, it is usually too inefficient and would seldom be a viable approach on its own (even though many improvements were proposed in the literature).

For instance during simple backtracking it frequently happens that very similar subtrees are unnecessarily investigated again and again, only to fail and being pruned in the end (this behavior is denoted as *thrashing*). Here comes inference into play, "*the real power engine behind CP*" [13], which essentially tightens the constraints, i.e. reduces the domains of the variables that are involved, eventually eliminating parts of the search space. In practice one might be familiar with this concept when solving Sudokus from newspapers. More precisely, inference removes (*filters*) inconsistent domain values, hence is said to achieve consistency, and since the information contained in one constraint is propagated to the neighboring constraints this process is sometimes also called *constraint propagation*. Note that several notions (levels) of consistency with corresponding consistency techniques exist, demanding an increasing computational effort to achieve/apply. We mention three common ones: The simplest being node consistency (for unary constraints), followed by arc consistency (already achieving a high degree of consistency for binary constraints), and path consistency (to remove more but not provably all inconsistencies which were not covered by arc consistency; it was shown that achieving consistency on paths of length two is sufficient to achieve path consistency in general). Since every CSP can be transformed to an equivalent CSP using only binary constraints, much research effort was put into efficient algorithms to obtain arc consistency. Practical CSP solution approaches usually rely on incomplete consistency techniques combined with a non-deterministic search to yield a complete method. COPs can be solved too, in that case one applies a tree-search based on branch-and-bound.

There also exist *global constraints* (e.g. `sum`, `alldifferent`, `cumulative` etc.) that are on the one hand shorthands for frequently recurring patterns, which makes programming easier, and on the other hand facilitate the work of the constraint solver by providing it with a better view of the structure of the problem. For so-called *over-constrained* problems where it is unlikely or impossible that all constraints can be fulfilled, one can use *soft constraints* (as opposed to the usual *hard constraints*). They are suited to formalize desired properties rather than requirements that may not be violated.

Naturally a careful basic modeling is crucial for solving CSPs or COPs, but the performance might be further improved via: considering redundant constraints which might allow an enhanced filtering, the usage of readily available global constraints (depending on the solver), also consider the inclusion of the dual model (swapping the constraints for the variables and vice versa) or an alternative view on the initial model, and avoiding symmetries via posting corresponding constraints.

In this thesis, a subproblem in Chapter 7 will be tackled with CP.

2.4 (Meta-)Heuristic Solution Approaches

Heuristic problem solving techniques range from simple constructive techniques such as ad-hoc greedy algorithms over local search methods to various *metaheuristics* [218, 91]. Especially the latter category is well-developed and has proven to be highly useful in practice. As their name suggests, metaheuristics are defined on a higher and basically problem-independent level; note that the term "meta-heuristic" was first introduced by Fred Glover

in the context of tabu search [95]. These "solving concepts" describe how to efficiently explore the search space by guiding lower-level or subordinate heuristics to find (near-)optimal solutions. It is important that a metaheuristic appropriately balances *diversification* and *intensification* of the search. Several taxonomies were proposed in the context of metaheuristics [22, 218], which do make sense since meanwhile a lot of such solution approaches exist, where the differences are sometimes quite fuzzy. In fact, in recent years one can observe the appearance of an increasing number of rather "exotic" variants which are frequently inspired by metaphors from nature, physics and life [235]. It is sometimes hard to spot new and original ideas as often only slight variations, if at all, of established concepts are proposed. A critical view on this is given by Weyland [235]. Coming to the taxonomies again, they are also helpful in synthesizing (original) successful concepts and ideas among the different variants, fostering a better understanding of vital components yielding an improved search balance as mentioned above.

A meaningful criterion is the division into single-solution based methods (i.e. following a single search trajectory), which are often sophisticated variants of local search either using a single neighborhood or several ones, and population based methods (i.e. multiple search trajectories, usually running in an intertwined way). Prominent examples of the former class are variable neighborhood search (VNS), tabu search (TS) and simulated annealing (SA), while the broad class of evolutionary algorithms (EAs), swarm intelligence methods such as ant colony optimization (ACO) algorithms and particle swarm optimization (PSO), as well as scatter search (SS) belong to the latter class. Another criterion is whether the solutions are primarily constructed (e.g. greedy randomized adaptive search procedures or ACO) or improved (e.g. VNS, TS, SA, EAs, PSO, SS).

Mostly in the early years of metaheuristics – but sometimes to a lesser extent even today – certain communities strongly promoted *their* method of choice and seemingly rather tried to find problems where it could be applied to with success, and hence gather more evidence why it is the "only true" method. Yet according to the *"no free lunch" theorem* by Wolpert and Macready [238] no single algorithm can dominate all others on all problems, since any elevated performance over one class of problems is offset by performance over another class. Hence it only makes sense to favor one algorithm over another with respect to specific problems or classes of problems.

We define some required basic notions before dealing in more detail with some heuristic methods.

Definition 10 *A **neighborhood structure** is a function $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s. Often, neighborhood structures are implicitly defined by specifying the changes that must be applied to a solution s in order to generate all its neighbors. The application of such an operator that produces a neighbor $s \in \mathcal{N}(s)$ of a solution s is commonly called a **move**.*

In addition to the globally optimal solution defined in Section 2.1 we can also define a locally optimal solution (again for a minimization problem):

Definition 11 *A **locally optimal solution** (or **local optimum**) with respect to a neighborhood structure $\mathcal{N}$ is a solution s such that $\forall s' \in \mathcal{N}(s) : f(s) \leq f(s')$.*

2.4.1 Approximation Algorithms

The class of *approximation algorithms* [224, 237] are special heuristics, as in contrast to basically all others, they also guarantee a certain quality of the approximate solution. In the following the solution value of an (arbitrary) instance I of an optimization problem P by algorithm A will be denoted by $A(I)$, whereas the optimal solution value is denoted as $Opt(I)$.

Definition 12 *An approximation algorithm A has an **absolute performance guarantee** (or **absolute performance ratio**) k, ($k > 0$), if for every instance I holds that $|Opt(I) - A(I)| \leq k$.*

Only for a few problems such algorithms exist, more common are the next ones, giving a relative performance guarantee:

Definition 13 *An approximation algorithm A for a minimization problem has a **relative performance guarantee** (or **relative performance ratio**) k, ($k > 1$), if for every instance I holds that $A(I) \leq k \cdot Opt(I)$.*

Algorithm A is then also called a k-approximation algorithm. We might also consider the relative deviation:

$$\frac{A(I) - Opt(I)}{Opt(I)} \leq \epsilon \Leftrightarrow A(I) \leq (1 + \epsilon)Opt(I) \ .$$

An approximation algorithm A for a minimization problem with relative deviation ϵ is a $(1 + \epsilon)$-approximation algorithm. The case for a maximization problem is analogue, and we end up with a $(1 - \epsilon)$-approximation algorithm. There is also the notion of an asymptotic performance guarantee k, by adding a constant term d: $A(I) \leq k \cdot Opt(I) + d$, which is for dealing with small integer valued solution values.

A richer class of algorithms, in fact sets of algorithms, is when we consider the deviation itself as an input:

Definition 14 *An **approximation scheme** A is a family of algorithms $\{A_\epsilon\}$, where there is an algorithm for each $\epsilon > 0$, such that A_ϵ is a $(1 + \epsilon)$-approximation algorithm (for minimization problems) or a $(1 - \epsilon)$-approximation algorithm (for maximization problems).*

We can distinguish the approximation schemes according to their runtime:

Definition 15 *An approximation scheme A is a **polynomial-time approximation scheme** (PTAS) when the runtime of A is polynomial in the size of the instance.*

Definition 16 *An approximation scheme A is a **fully polynomial-time approximation scheme** (FPTAS) when the runtime of A is polynomial in the size of the instance and in $1/\epsilon$.*

Assuming that $Opt(I)$ is polynomially bounded by the instance size and the magnitudes of the numbers then (i) if there exists a PTAS for problem P, then there exists also a pseudo-polynomial algorithm for it, and (ii) if P is strongly $\mathcal{NP}$-complete it cannot have an FPTAS unless $\mathcal{P} = \mathcal{NP}$. If for a minimization problem P already the decision variant "Is there a solution with value $\leq k$?" is $\mathcal{NP}$-complete, then there exists no polynomial approximation algorithm with a performance guarantee smaller than $1 + 1/k$, hence there exists no PTAS.

Note that unfortunately for most non-trivial metaheuristics no practically useful performance guarantee can be derived.

2.4.2 Construction Heuristics

Construction heuristics iteratively build a solution from scratch. Although we are not aware of a taxonomy of such methods, probably because they are mostly designed ad-hoc, we mention some concepts which often occur. During construction they either add still missing solution elements one after another in some way "unconnected" elements are merged. Usually they rely on some greedy criterion to select the next element or move, e.g. nearest neighbor or cheapest insertion costs. Though also randomization might be involved to be able to generate more than one solution, which, if selecting the best solution out of the generated, might lead to a more robust performance. Also one could always select the next element at random and insert it in a greedy way again, or probably create a solution purely at random. The solutions produced are not necessarily feasible, which might sometimes be even intended, or simply required due to facing complicating constraints. As the obtained solution quality is usually quite limited (in case a feasible solution can be derived at all) it is common to apply a subsequent improvement method; in each of the following chapters we will encounter construction heuristics for this use case. The GRASP metaheuristic extends a randomized solution construction via guiding the construction process, see Section 2.4.6.

2.4.3 Local Search

Algorithm 1: Basic Local Search

Input: starting solution s
Result: possibly improved solution s after local search

```
repeat
    select s' ∈ N(s)
    if f(s') < f(s) then
        s = s'
until termination condition
```

Local search, depicted in Algorithm 1, belongs to the class of improvement heuristics which are applied on a solution at hand (feasible or not). It usually performs rather small local changes (moves), and further only accepts improving solutions. Therefore local search depends on the neighborhood structure $\mathcal{N}$, how it is processed (select $s' \in \mathcal{N}(s)$) and a given starting solution s. Regarding the processing of $\mathcal{N}(s)$ and accepting a new solution, the most common variants are: (1) *best improvement* scans the whole neighborhood and selects the local optimum, (2) *first improvement* scans the neighborhood in a predefined order and takes the first solution s' with $f(s') < f(s)$ and (3) *random improvement* selects a solution $s' \in \mathcal{N}(s)$ at random and accepts it if $f(s') < f(s)$.

The *termination condition(s)* might be based on a duration limit (either time or overall iterations), on how many iterations have passed without an improvement, or whether a local optimum has been reached. Such a basic local search is usually easy to implement and quite fast, but a major drawback is the inability to overcome local optima. Nevertheless, for almost all more sophisticated improvement methods local search is usually the backbone.

2.4.4 Variable Neighborhood Search

Variable neighborhood search (VNS) [103], introduced in [141] by Mladenović and Hansen, is built around the concept to utilize several neighborhood structures for a given problem and switch between them in a systematic way. It further relies on the following observations:

- A local optimum with respect to one neighborhood structure is not necessarily a local optimum to another one.
- A global optimum is a local optimum with respect to all possible neighborhood structures.
- For many problems, local optima with respect to different neighborhood structures are relatively close to each other.

These facts can be exploited in a deterministic and/or stochastic way. An approach realizing a deterministic usage is called *variable neighborhood descent* (VND), shown in Algorithm 2. VND basically extends simple local search via using multiple neighborhood structures, typically ordered according to increasing size or evaluation cost, but also some sort of adaptive ordering can be used [110]. Usually the best neighbor solution is selected for a given neighborhood, i.e. applying best improvement. If no improving solution is found within one neighborhood, the next neighborhood is considered; else the search is re-centered at the new incumbent and restarts with the first neighborhood structure. Naturally, VND terminates (at the latest) when all neighborhood structures have been considered.

While VND utilizes multiple neighborhoods for an improved intensification, VNS is doing so in an analogue way for diversification; depicted in Algorithm 3. This is accomplished by applying *shaking*, i.e. random moves ("jumps") in systematically increasing neighborhoods, yet even for the largest neighborhoods VNS is not meant to turn to a multi-start heuristic. Depending on the local search component (see line 5 of Algorithm 3) several flavors of VNS

Algorithm 2: Variable Neighborhood Descent (s)

Input: initial solution s; neighborhoods $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{l_{\max}}$
Output: (probably) improved solution s

```
l ← 1
repeat
    s' ← arg min_{s''∈N_l(s)} f(s'') // find best neighbor in N_l(s)
    if f(s') better than f(s) then
        s ← s'
        l ← 1
    else
        l ← l + 1
until l = l_max
return s
```

Algorithm 3: Variable Neighborhood Search (s)

Input: initial solution s; neighborhoods $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{k_{\max}}$
Output: (probably) improved solution s

```
repeat
    k ← 1
    repeat
        randomly select s' ∈ N_k(s) // shaking
        probably apply some local search on s' ∈ N_l(s)
        if f(s') better than f(s) then
            s ← s'
            k ← 1
        else
            k ← k + 1
    until k = k_max
until termination criterion
return s
```

exist: if no local improvement is applied at all, hence only relying on shaking and thus stochasticity, it is denoted as reduced VNS, if some best or first improvement based (simple) local search is applied, it is denoted as basic VNS, and finally, if VND is applied inside VNS (probably using different neighborhood structures), it is denoted as general VNS.

Despite reduced VNS, one has to ensure that the solution obtained via shaking is not immediately reverted to the initial solution due to the local search component. The most common criterion to choose the order of the neighborhood structures is the size of the corresponding neighborhoods, but as for VND there might also be applied a non-static order as well.

Algorithm 4: Simulated Annealing

Input: initial solution s; cooling scheme
Output: (probably) improved solution s

```
1  k ← 0
2  T_k ← initial temperature value
3  repeat
4      randomly select s' ∈ N(s)
5      if f(s') better than f(s) then
6          s ← s'
7      else if accept (s',s,T_k) then
8          s ← s'
9      k ← k + 1
10     cooling (T_k,k)
11 until some termination criterion is met
12 return s
```

For information on other variants, such as the skewed VNS or the variable neighborhood decomposition search, we refer again to [103]. We can conclude that VNS requires few to none parameters in general, making it easy to deploy with respect to this. The challenge, of course, is to derive adequate neighborhood structures, and decide on their order. Yet this concept allows on the one hand to assess the performance of the crucial parts, the neighborhood structures, in a straightforward way and on the other hand a fairly simple extension just by adding additional neighborhood structures.

As it turns out, VNS is the most prominent metaheuristic throughout this work, applying it to each problem tackled in the subsequent chapters. We will see that often well-designed, problem specific neighborhood structures give rise to an excellent performance.

2.4.5 Simulated Annealing

Simulated annealing (SA) is commonly referred to as the first heuristic that conceptually qualified as a metaheuristic, since it was the first with an explicit strategy to escape from local optima. The key feature is to accept with a certain probability also worse solutions than the current one, i.e. allowing *uphill* or *hill-climbing* moves. This idea is inspired by the annealing process of metal and glass, which assume a low energy configuration when first heated up and then cooled down sufficiently slowly, making SA a nature-inspired metaheuristic. An up-to-date survey is given in [151], initial works are presented by Kirkpatrick et al. [123], and, usually not mentioned, also by Černý [225]. We discuss relevant parts of SA on the basis of Algorithm 4, where we assume an already given initial solution, as the construction process is not influenced by SA. First an initial temperature T_0 is set, choosing a value such that the probability for an uphill move is rather high. Then in each iteration a neighboring solution s' of s is usually selected at random. Note that SA does not impose

any restrictions on this neighborhood at all, hence also multiple neighborhoods are possible. If s' improves upon s it simply replaces it. In contrast, if s' is an inferior solution then most commonly the Metropolis criterion [138] is applied: s' is accepted and replaces s with probability $e^{-|f(s')-f(s)|/T_k}$ based on the difference of objective values and the current temperature. To implement SA also a *cooling (or annealing) schedule* has to be defined in order to decrease the temperature in a systematic way; they can be static or adaptive. Common static schedules are the geometric schedule, i.e. $T_{new} = T_{old} \cdot \alpha$, $\alpha < 1$, or a linear schedule, i.e. $T_{new} = T_{old} - \beta$, $\beta > 0$. Usually several iterations are performed for a temperature level, hence in Algorithm 4 we additionally used k as a parameter to the cooling method, e.g. always applying cooling after a fixed amount of iterations. Beside the usual termination criteria as time or iterations, SA might be halted when a certain temperature level is reached. A possibility to continue/reapply the search without performing a complete restart is to apply a so-called *reheating*, where the temperature is raised to a higher level again. SA is rather simple to implement in general, and therefore one of the most widely used metaheuristics, but the tuning of its parameters, especially the cooling schedule, might be difficult. To alleviate this to some extent also adaptive schemes were proposed (for obtaining the initial temperature as well), leading rather to a "meta-tuning".

Although in our work we never apply SA on its own, it is hybridized with VNS to exploit its benefits in Chapters 4, 5, and 6.

2.4.6 Greedy Randomized Adaptive Search Procedure

The *greedy randomized adaptive search procedure* (GRASP) [76, 192] is a multi-start metaheuristic where each iteration consists basically of two phases: A construction phase building a feasible solution, which is then refined in a subsequent local search phase, e.g. by applying VND mentioned in the previous section. In case the solution built in the first phase is not feasible a repair procedure might be applied, though this will not be considered in the following. The best feasible solution obtained during the execution is returned. Algorithm 5 gives an overview on GRASP.

The key principle of GRASP is to systematically randomize a greedy solution construction procedure (lines 4 to 9) in order to generate good candidates for the subsequent local improvement. This is achieved by determining admissible solution elements and thus deriving a *candidate list* (CL). A promising subset of these viable candidates is selected for potential inclusion in the next step, yielding the *restricted candidate list* (RCL). Finally an element is chosen from the RCL at random and the partial solution is extended in an appropriate way. Obviously the behavior of the method can be adjusted via the process of building RCL from CL. In general, the criterion to decide on are the incremental costs $c(e)$ of inserting an element e to the solution under construction. A first option is then to simply consider the k best (cheapest) elements of CL, realizing a cardinality based selection. Alternatively a value based selection is possible by setting a threshold parameter $\alpha \in [0, 1]$: let $c_{\min}$ and $c_{\max}$ be the minimal and maximal incremental costs within CL, respectively, then all elements e satisfying $c(e) \leq c_{\min} + \alpha \cdot (c_{\max} - c_{\min})$ are added to RCL. The two extremes are to always select the best element in a pure greedy way ($\alpha = 0$), and selecting any viable element in

Algorithm 5: Greedy Randomized Adaptive Search Procedure

Input: instance data necessary to construct solutions
Output: best found feasible solution s^*

```
1  s* ← ∅
2  repeat
3      s ← ∅
4      while s is not a complete solution do
5          build CL // candidate list
6          derive RCL from CL // restricted candidate list
7          select an element e from RCL at random
8          add e to solution s
9      apply some local search on s
10     if f(s) better than f(s*) then
11         s* ← s
12 until some termination criterion is met
13 return s*
```

a complete random way ($\alpha = 1$). Although the average solution quality decreases with an increasing α, the variance of the generated solutions gets larger and usually the best solution obtained after local improvement is likely to improve, too. A good balance between quality and diversity of the constructed solutions is therefore crucial to yield an overall good performance, requiring to (fine-)tune the parameter α. However, relying on a single value might not be the best choice anyway, which is why schemes have been proposed to alternate α, either via choosing it at random according to some probability distribution or realizing a self-tuning via applying a *reactive GRASP*. Several other extensions and improvements were proposed, but since we do not make use of them in our work the reader is again referred to [192].

A GRASP will be applied to tackle a subproblem in Chapter 4.

2.4.7 Evolutionary Algorithms

Evolutionary Algorithms (EAs) [11, 70] are a class of population-based metaheuristics, where in contrast to the previously described metaheuristics a whole set of solutions (the *population*) is dealt with at the same time. The idea is to gain such an adaptive behavior like evolutionary processes occurring in nature by modeling them on the computer in a strongly simplified way, effectively realizing some sort of *in silico* Darwinian principles of natural selection. Thus the goal of the artificial evolution is to adapt to the "environment", represented by a specific problem, and thereby evolve good solutions. Note that in the context of EAs single solutions are often denoted as *individuals* or *chromosomes*, variables as *genes*, and variable values as *alleles*. The objective function value is further replaced by the *fitness* of an individual. Although this fitness often directly corresponds to the objective function

value of the individual, also a cost scaling or the diversity of the individual might be incorporated. What also sets EAs apart from most other metaheuristics is that one distinguishes between the representation, i.e. the genetic encoding (called the *genotype*), and the actual solution (called the *phenotype*). Hence in case of an indirect representation (e.g. a string of bits, integers, or real-values) the genotype is decoded to yield the phenotype. Alternatively one can apply a direct representation, where no encoding takes place. For efficient EAs, it is necessary either to design problem-specific representations and to use standard operators for recombination and mutation (described later), or to develop problem-specific operators and to use direct representations; see the book by Rothlauf [202] for more about this issue. One further aims at obtaining a high *locality*, i.e. small changes in the genotype (tend to) correspond to small changes in the phenotype and vice versa.

In the following we will specifically concentrate on *genetic algorithms* (GAs) [106] which are mainly applied to solve discrete problems. The main principles, however, are common to all EAs. The general outline of a GA is depicted in Algorithm 6. At the beginning an initial population P is generated, usually with one or more randomized constructive heuristics, to yield a diversified set of individuals, followed by evaluating P, i.e. determining the fitness of the individuals. The successive steps, comprising one *generation*, are repeated until some termination criterion is met (e.g. a limit on overall generations or on CPU-time).

Selection. From the current population P a subset Q_s of individuals (the *parents*) is selected as candidates for the subsequent recombination. Thereby individuals that are more desirable, i.e. those having a higher fitness, are given more opportunity to "breed". A scheme where the probability for selection is directly proportional to the fitness is the *roulette-wheel* method. As potentially several problems arise when applying such a direct scheme (e.g. negative fitness values, the scale on which the fitness is measured, or decreasing fitness values when doing minimization), often some sort of scaling is necessary, or a different scheme is used. Two common alternative schemes are *rank selection* and *tournament selection*. The latter will be used at some points in our work, where the selection for a tournament of size k works as follows: select k individuals out of P at random and take the fittest of them. In general, the ratio of the probability of selecting the fittest individual to the probability of selecting an average individual is termed the *selection pressure*. The implicit intensification of a GA's search can thus be adjusted by this pressure, where finding the right setting is crucial.

Recombination or Crossover. During recombination, which is considered the primary operator, the parent individuals Q_s are *recombined* to produce the new individuals (the *offspring*) Q_r. The offspring should be primarily made up of attributes (genes) of its parents, thus realizing a high degree of heritability. Although the case of having two parent individuals is most common also more are possible. The recombination operator highly depends on the considered problem and especially the representation needs to be taken into account. If it is inevitable or perhaps desired to generate infeasible offspring, a suitable repair operator might be applied.

Mutation. Next, the offspring Q_r is subject to a mutate operator, which is considered the secondary operator. It corresponds to some extent to natural mutation, and is in practice similar to a random move of a neighborhood based search. Typically with a rather small probability each individual is randomly changed to a small extent. Mutation introduces

Algorithm 6: Evolutionary Algorithm

```
1 P ← generate initial population
2 Evaluate(P)
3 repeat
4 |   Q_s ← Select(P)
5 |   Q_r ← Recombine(Q_s)
6 |   Q_m ← Mutate(Q_r)
7 |   Evaluate(Q_m)
8 |   P ← Replace(P, Q_m)
9 until some termination criterion is met
```

either new or already lost information into the population and is therefore responsible to keep a certain amount of diversity, as GAs tend to prematurely converge otherwise. This step results in the final offspring Q_m.

Replacement. After evaluating Q_m parts or even all of the old population gets replaced by the new offspring. If the new population consists only of individuals drawn from the offspring it is called *generational replacement*. When using *overlapping populations* then individuals can "survive" a generation, i.e. being transferred from the old to the new population. This can be taken even further to so-called *steady-state* or *incremental* strategies, in which usually only one offspring is created and integrated in the population, thereby replacing another ((e.g. the worst) chromosome. An *elitism* strategy is applied if the best solution always survives. The replacement process generally resembles the concept of the "survival of the fittest".

Assuming a "well-diversified" population GAs have the advantage of conducting a rather broad search and are thus generally less vulnerable to local optima. Note that in the standard GA no operator or the like is responsible for systematically optimizing single individuals. A remedy to this are *hybrid GAs* or *memetic algorithms* [145]. They are a class of GAs extended by means to (mostly) locally improve single individuals, either at the time of creation, before and/or after mutation.

GAs and/or MAs will be utilized in Chapter 3 and 4.

2.5 Hybrid Solution Approaches

Looking at the assets and drawbacks of the various exact and heuristic techniques, the approaches can to some degree be seen as complementary, and therefore it appears natural to either combine concepts from the different streams or to combine different methods within a common stream. In case the resulting method is of heuristic nature the potential benefit of such a synergy is an increase in solution quality and/or obtaining the same quality in less time (usually corresponding to less computational effort, letting parallel approaches aside). Here hybridization offers means for the "eternal struggle" of diversification (exploration) versus intensification (exploitation). Contrary, if the resulting method is exact might allow

to "push forward" the limit in the size of instances that can be solved to proven optimality within a reasonable time.

Although this idea is not new, such *hybrid optimization techniques* have become especially popular over the last years. Hundreds of publications reporting on such approaches and dedicated scientific events such as the *Workshops on Hybrid Metaheuristics* since 2004, the *Workshops on Matheuristics* since 2006, and the conferences on *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (CPAIOR) since 1999, as well as dedicated tracks and sessions on wider scope conferences, document the popularity, success, and importance of this specific line of research.

For the majority of more prominent problems some kind of hybrid approach belongs to the currently best known (leading) methods, entirely "pure" methods are seldom as successful. This is also because there occurred a shift from algorithm oriented to problem oriented research and it is (in a positive sense) inevitable to use an appropriate "mix" of solution concepts. In fact, devising such hybrids can be considered a quite creative task in general.

Although such hybrid approaches might basically consist of any possible combination, most common methods deal with the *"integration of a metaheuristic related concept with some other techniques (possibly another metaheuristic)"* [23] and are termed *hybrid metaheuristics* (or *metaheuristic hybrids*). For recent surveys see e.g. the book by Blum et al. [21] and the article by Raidl et al. [190], taxonomies are proposed in [71, 188]. In the latter work the main criteria to classify hybridizations are: (i) which kind of algorithms are involved, (ii) the level of hybridization (high or low level), (iii) the order of execution (sequential, intertwined, or parallel), and (iv) the control strategy (integrative or collaborative). While criteria (i) and (iii) should need no further explanation, we will describe (ii), the *level* (or strength) in more detail: High-level combinations generally retain the individual identities of the original algorithms and cooperate over a relatively well defined interface. There is no direct, strong relationship of the internal workings of the algorithms, hence obtaining a weak coupling. In contrast, algorithms in low-level combinations strongly depend on each other as individual components or functions are exchanged, resulting in a strong coupling. Finally, the meaning of (iv) will be clarified in the following.

A special yet prominent class are hybrid approaches where metaheuristic and exact algorithms are combined. Puchinger and Raidl [183] give a useful classification scheme and many examples for such hybrids, distinguishing between the following two main categories (defining the *control strategy*): *collaborative (cooperative) combination*, where the two methods exchange information, but are not part of each other, thereby running in sequential order or being executed in a parallel or intertwined way, and *integrative (coercive) combination*, with a distinguished master and at least one integrated slave algorithm, where exact algorithms are incorporated in (i.e. are subordinates of) metaheuristics and vice versa. In a subsequent work [189] the same authors especially focus on hybrids between metaheuristics and (integer) linear programming techniques. More recently such hybrids, exploiting in a suitable way the mathematical model of the problem, are often called *matheuristics* [133], or due to their nature *model-based metaheuristics*.

For a comprehensive collection of articles mainly concerned with hybridizations involving constraint programming we refer to the book "*Hybrid Optimization: The Ten Years of*

CPAIOR" edited by Milano and van Hentenryck [140].

Also worth mentioning is the book by Hooker [108], which deals with an integrated solving approach combining prominent concepts, primarily focusing on the unification of mathematical programming and constraint programming.

In the remainder of this section we list several use cases of hybridizations as in [190], also giving successful examples for each. Although a grouping according to some classification scheme would have been possible too, we deem the chosen approach more appropriate to highlight the possibilities and benefits.

2.5.1 Finding Initial or Improved Solutions

Perhaps the most often occurring hybridization among metaheuristics is to use some improvement method inside another, different method. A scenario could be a method that constructs good initial solutions which are subsequently improved by another one, e.g. applying GRASP with an embedded VND (although already pure GRASP is an example for this, as is the general VNS). Another fruitful combination is to combine a population-based approach with an improvement method, such that both, the rather explorative nature of the former and the strong intensification capabilities of the latter, are exploited. We already mentioned the class of *memetic algorithms* [144, 145] before, but e.g. also *ant colony optimization* [64] is usually enhanced by an improvement method to yield better solutions. We apply such kind of hybridization in Chapter 3 via realizing several combinations of an EA/MA with VND/VNS. Note that here the methods involved retain their characteristics and are thus a collaborative combination.

Remember that branch-and-bound relies on tight primal bounds that are most commonly obtained from feasible solutions. Obviously, heuristics and metaheuristics can be applied to the original problem before starting the B&B process, providing initial solutions, as well as be repeatedly applied throughout the whole tree search, providing possibly improved solutions. This results in providing feasible solutions at an early stage of the optimization process and the possibility to essentially speed up the overall optimization.

Note that beside such "manual" approaches current (commercial) generic MIP solvers also include very strong heuristics for finding initial feasible solutions, see e.g. the *feasibility pump* [77].

2.5.2 Multi-Stage Approaches

In multi-stage approaches the optimization process is divided into several stages or levels, which are usually iteratively tackled, possibly applying different methods. For some problems such an approach might be natural to apply, due to practical reasons (e.g. size of the problem), however, it might simply be inevitable. An example is a *preprocessing* step, where the problem instance is reduced, usually removing parts of it that are guaranteed not to be included in the best solution or any feasible solution. Another strategy is *multilevel refinement* [231], applying a (meta-)heuristic on several levels of the same problem. For this the

problem is initially *coarsened* and then iteratively solved and *refined* again. For more details as well as an application of it we refer to Chapter 5. Another possibility is to apply *variable fixing*, where e.g. some heuristic criterion decides which variables to fix to a certain value. The *core concept* applied to knapsack problems is an example for this, see the work by Puchinger and Raidl [186].

2.5.3 Decoder-Based Approaches

For the previously presented GAs it is often the case that actual solutions are represented in an indirect or incomplete way, applying a decoder to derive the phenotype from the genotype. If this transformation is not straightforward but involves a non-trivial optimization problem itself, then the overall performance is obviously strongly dependent on the quality and the speed of the decoder. Hu and Raidl [111] tackle the generalized traveling salesman problem with a VNS and use two different representations for their candidate solutions. Since both do not encode a complete solution they apply dynamic programming as well as the Lin-Kernighan heuristic as *intelligent decoder*. In [20] Blum and Blesa use dynamic programming to derive the best k-cardinality trees out of l-cardinality trees, $l > k$, built by an ant colony optimization algorithm.

2.5.4 Solution Merging

In *solution merging* new, possibly better solutions are created from attributes appearing in two or more promising heuristic solutions. Such an approach is based on the assumption that high-quality solutions often share many attributes. The recombination operator used by genetic algorithms can be seen as a classical solution merging approach. However, the offspring is usually derived by simple random inheritance of parental attributes and it is not tried to optimize this offspring, which therefore often is worse than its parents. The improvement is due to repeating these computationally cheap operations many times.

Alternatively, one can put more effort into the derivation of such offspring. A sometimes effective technique is *path relinking* (PR) [96] which is often applied together with scatter search [193]. PR traces a path in the search space from one parent to a second by repeatedly exchanging a single attribute only (or more generally by performing a series of moves in a simple neighborhood structure). The overall best solution lying on this path is finally taken as offspring.

This idea can further be extended by considering not just solutions on a single path between two parents, but the whole subspace of solutions induced by the joined attributes appearing in a set of two or more input solutions. An *optimal merging* operation returns a best solution from this subspace, i.e. it identifies a best possible combination of the parents' attributes. Depending on the underlying problem, identifying such an optimal offspring is often a hard optimization problem on its own, but due to the usually quite limited number of different attributes appearing in the parents, it can often be solved in reasonable time in practice.

For mixed integer programming, Rothberg [201] suggests a tight integration of an EA including optimal merging in a branch-and-cut based MIP solver. Experimental results indicate that

this hybrid often is able to find significantly better solutions than other heuristic methods for several very difficult MIPs. Similarly, the benefits of embedding branch-and-bound in EAs is shown by Cotta and Troya [49]. This concept is by far not limited to EAs only, e.g. in Section 4.9.2 we apply it to solutions of several VNS instances which are merged by solving an ILP.

2.5.5 Strategic Guidance of a Method by Another

A successful and hence often reported scheme for hybridization is to apply some means to guide the search process of an optimization method. This can be realized either via having an (independent) algorithm providing the information used for guidance, being collaborative, or the functionality of a method is directly enhanced with algorithmic components/concepts originating from other techniques, possibly from other streams, being integrative in nature.

This definition is rather broad, as in principle, any metaheuristic that provides incumbent solutions to a B&B-based approach might already be considered to fall into this class of approaches, as often some sort of indirect guidance occurs. Often the method used to gather the information solves a relaxation of the problem. This might be the LP relaxation, which is solved in [185] for several neighborhoods of a VND embedded in a VNS to determine a favorable ordering, this concept is denoted as *relaxation guided VNS*. In Section 4.9.3 we derive a combination of a column generation approach and an EA where the latter exploits information gathered by the former. Alternatively a Lagrangian relaxation or decomposition (LD) might be solved and subsequently exploit the information. In [168] Pirkwieser et al. apply such a concept to better solve the knapsack constrained maximum spanning tree problem. It is possible to shrink the graph by only considering edges also appearing in heuristic solutions of LD, Lagrangian dual variables are exploited by using final reduced costs for biasing the selection of edges in the EA's operators, and the best solution obtained from LD is provided to the EA as seed in the initial population.

Intertwined and parallel combinations allow for *mutual* guidance, i.e., all participating methods may exploit information from each other. Talukdar et al. [219] describe a very general agent-based model for such systems, called *asynchronous teams* (A-Teams). This problem solving architecture consists of a collection of agents and memories connected in a strongly cyclic directed way, and each optimization agent works on the target problem, a relaxation, or a subclass of the original problem. Denzinger and Offerman [59] describe a similar framework called TECHS (TEams for Cooperative Heterogeneous Search). It consists of teams of one or more agents using the same search paradigm. Communication between the agents is controlled by so-called send- and receive-referees. Gallardo, Cotta, and Fernández [86] present a EA/B&B hybrid evaluated on the multidimensional knapsack problem. The algorithms are executed in an intertwined way and are cooperating by exchanging information. The EA provides bounds for B&B, while B&B provides best and partial solutions to the EA. In [87], the same authors described a refined variant of their approach, which uses beam search as truncated B&B.

Next, we will give some examples realizing an integrated approach. Blum [19] hybridizes the solution construction mechanism of an ACO with a heuristic derivative of breadth-first B&B,

beam search, which restricts the search to a certain number of nodes based on bounding information instead of implicitly considering all nodes. Some loose and tight couplings of an ACO and CP are investigated by Meyer [139]. Especially a tight coupling where CP helps the ACO to handle (hard) constraints and results in a strengthened construction phase in the ACO appeared promising. Finally, in [214] Solnon also proposes to combine the expressive power of CP languages and the solving power of ACO.

In [105] the concepts of VNS and SA are "merged" into a single method, i.e. basically building upon the systematic neighborhood change, but with a certain possibility also accepting worse solutions like SA. Hence the ability of VNS to escape local optima is improved. This promising hybrid is also applied by us in Chapters 4–6.

Several extensions of generic branch-and-cut based MIP solvers exist that, conform to the concept of integration, do not directly make use of some heuristic, but are derived following the spirit of them in order to produce good heuristic solutions early during the exact tree search. An early representative is *local branching* by Fischetti and Lodi [78] introduces the spirit of classical k-opt local search in B&B by modifying the branching rule and the strategy for choosing the next tree node to process: based on a given incumbent solution a neighborhood is defined where at most k of some variables are allowed to change relative to this solution (adding a second node with the inverse constraint), this sub-MIP is then solved next. In case an improved solution is found the process is repeated using this new solution, k is increased up to some fixed value and this larger neighborhood based on the initial solution is searched otherwise. Danna et al. [51] suggest a different approach called *relaxation induced neighborhood search* (RINS) for exploring the neighborhoods of incumbent solutions more intensively. The central idea is to occasionally devise a sub-MIP at a node of the B&B tree that corresponds to a special neighborhood of an incumbent solution: Variables having the same values in the incumbent and in the current solution of the LP relaxation are fixed, and an objective cutoff is set based on the objective value of the incumbent. A sub-MIP is solved on the remaining variables with a given time limit. If a better solution can be found it is passed to the global MIP-search, which is resumed after the sub-MIP's termination.

2.5.6 Solving Large Neighborhoods or Subproblems

The general idea of *very large(-scale) neighborhood search* (VLNS) [5] is to apply a more sophisticated procedure than naive enumeration to search for a best (or better) solution within a larger but restricted part of the whole search space induced by an incumbent solution. Various techniques especially including (mixed) integer programming methods, dynamic programming (e.g. *Dynasearch* [34]), and constraint programming have been successfully used in VLNS as embedded optimization procedures. Hu et al. [109] fully explore some neighborhoods of a VND via dynamic programming and ILP techniques for the generalized minimum spanning tree problem. In this vein in [174] Prandtstetter and Raidl several different MIP-based neighborhoods are searched within a VNS framework for a car sequencing problem. Similar ideas, also utilizing ILP techniques, are applied in Section 4.9 and 6.4, though several of our approaches are somewhat a mixture of large neighborhood search and optimal solution merging.

As already pointed out in Section 2.3.1, in cut and column generation based integer programming methods the dynamic separation of cutting planes and the pricing of columns can be done by means of (meta-)heuristics in order to speed up the optimization process, especially when facing difficult subproblems. An example is the branch-and-cut algorithm by Gruber and Raidl [101] for the bounded diameter minimum spanning tree problem. The diameter bound is ensured via an exponentially large number of so-called jump inequalities. A sequence of methods is used for their separation, starting from a greedy construction technique over a local search procedure to a tabu search algorithm. In Section 4.8 we apply a heuristic multi-start method, basically similar to a VND, to find violated 2-path cuts for a routing problem in a B&C&P approach.

Puchinger and Raidl [184] describe an exact branch-and-price algorithm for the three-stage two-dimensional bin packing problem. The pricing problem occurring in this application is a three-stage two-dimensional knapsack packing problem. Fast column generation is performed by applying the following sequence: a greedy heuristic, an evolutionary algorithm, solving a restricted, simpler IP-model of the pricing problem using a MIP solver within a certain time-limit, and finally solving a complete IP-model by the MIP solver. The algorithms coming later in this sequence are only executed if the previous ones did not find columns with negative reduced costs. In Section 4.7.2 we also apply several methods of increasing computational effort to solve an $\mathcal{NP}$-hard pricing problem. Sometimes also CP is applied to generate colums [102].

Speaking of which, in CP a subproblem is to filter the inconsistent values of variable domains. Richter et al. [195] propose the `SomeDifferent` constraint and heuristic filtering algorithms, whereas in [85] Galinier et al. present a tabu search to handle this filtering.

Finally we note that sometimes a naturally occurring subproblem might be solved by means of an exact technique to improve the performance of the main heuristic approach. For example Fuellerer et al. [84] solve a two-dimensional loading problem via B&B for a combined vehicle routing and loading problem. Similar to generating columns as before, the exact method is only applied in case some heuristics failed to find a solution. We apply a similar scheme for a routing problem involving several compartments to be filled in Chapter 7, whereas we finally rely on CP.

CHAPTER 3

CONSENSUS TREE PROBLEM

3.1 Introduction

The *consensus tree problem* (CTP) has been first motivated and described in [3] alongside with a solution method. Although it most frequently appears in the domain of phylogenetics [107], it has potential applications in other clustering domains as well. But to start with, the reason for dealing with this interesting problem was the joint project entitled *Hybridizing Branch-and-Bound with Metaheuristics for Solving Tree-Structured Combinatorial Optimization Problems* supported by the Austrian exchange service (ÖAD) within the WTZ program (scientific and technological cooperation), where cooperations with partners from Spain were facilitated in the call "Acciones Integradas 2006–2007". Our partners were Carlos Cotta, Antonio José Fernández and Jose Enrique Gallardo from the University of Malaga. Among others one of the main aspects of the project was to develop new methods to especially deal with the inference of phylogenetic trees. For this we agreed on a two phase approach: our Spanish colleagues aimed at deriving high quality trees, and our task was to consolidate these trees into one final output tree, hence solving the CTP.

In Section 3.2 we report on previous as well as related work. The tree similarity measures utilized in our work are defined in Section 3.3. Meaningful tree neighborhood structures for local search based approaches are presented in Section 3.4. They are applied in a variable neighborhood search (VNS) with an embedded variable neighborhood descent (VND) as described in Section 3.7. The neighborhoods are further utilized in Section 3.6 to extend an existing evolutionary algorithm (EA) by local search to a memetic algorithm (MA). Finally, in Section 3.8 we consider sequential and intertwined combinations of the EA (MA) and VNS (VND). Experimental results on real and artificially generated CTP instances are given in Section 3.11, followed by concluding remarks in Section 3.12.

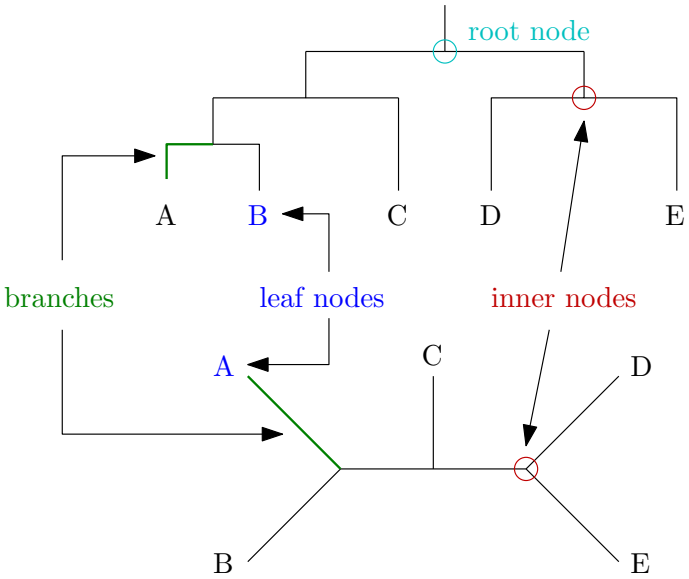


Figure 3.1: Exemplary rooted and unrooted evolutionary trees.

Parts of this work were presented at the *Austrian Workshop on Metaheuristics 5* in 2007 (AWM 5 '07), at the *2nd International Conference on Bioinformatics Research and Development* in 2008 (BIRD'08) [169] (also see our poster in the appendix in Section A.1), and at the *10th Annual Conference on Genetic and Evolutionary Computation* in 2008 (GECCO'08) [157].

3.1.1 Phylogenetics

To motivate the CTP we give a short and by no means complete survey on phylogenetic tree reconstruction. A *phylogenetic tree* is composed of nodes and branches (arcs) and models the evolutionary relationship between a set $\mathcal{L}$ of related objects called *taxa*.

These taxa are the labeled leaf nodes or operational taxonomic units of the tree, whereas unlabeled inner nodes represent probably extinct ancestors, also denoted as hypothetical taxonomic units. Exemplary evolutionary trees are depicted in Figure 3.1.

In the course of the project we decided to restrict our work on rooted unweighted binary trees, i.e. there exists a single distinguished root node denoting the common ancestor of all taxa, the relations represented by the tree are not weighted by any means (hence not representing a timeline), and each inner node always has exactly two direct descendants.

Next, we will introduce some definitions which will be of use in this chapter; some are taken from [30].

Definition 3 *A rooted (phylogenetic) tree is a rooted tree which has every leaf identified with a unique taxon and every node that is not a leaf (i.e. inner node) has at least two children.*

Definition 4 *A rooted tree is binary if every inner node has exactly two children.*

Definition 5 *A group is a subset of the set of taxa.*

Definition 6 *A cluster of a tree T is a group which contains all the descendants of its most recent common ancestor.*

Definition 7 *Two clusters A and B are compatible iff A is contained in B, or B is contained in A, or A and B are disjoint.*

Definition 8 *A cluster is compatible with a tree T if it is compatible with every cluster of T.*

Definition 9 *A rooted tree T refines another rooted tree T' on the same set of taxa if every cluster of T' is a cluster of T.*

Definition 10 *A rooted triple $ab|c$ denotes a grouping of a and b relative to c. $ab|c$ is said to be a rooted triple of tree T if the least common ancestor of a and b is a descendant of the least common ancestor of a, b and c. The set of all rooted triples of T is denoted as $r(T)$.*

Definition 11 *A set of rooted triples R is compatible if there exists a rooted tree T such that $R \subseteq r(T)$.*

The *phylogeny problem* is to infer the intermediate ancestors and branches, thus to derive the evolutionary relationships, from given species data. The latter is commonly given as biomolecular sequences, i.e. DNA, RNA or amino acid sequences. Often morphological features are used in addition, too. For inferring the tree a multitude of conceptually different inference methods exists, each having individual advantages and drawbacks [122]. On the one hand are approaches directly dealing with the sequences, e.g. maximum parsimony, where the "cheapest" tree is sought via minimizing the Hamming distance between connected sequences, and maximum likelihood, which uses a stochastic model of evolution, whereas on the other hand the sequences are used to obtain certain "distances" among the taxa and these approaches subsequently work on the derived distance matrix, seeking for a tree best resembling these distances. The distance based approaches can be regarded an intermediate strategy between maximum parsimony and maximum likelihood.

Stated as optimization problems, the maximum parsimony approach resembles a Steiner tree problem [81], and is thus $\mathcal{NP}$-complete. Since maximum likelihood approaches were recently shown to be connected to the former ones [196], the corresponding problem is $\mathcal{NP}$-complete, too. Unfortunately also distance based approaches, e.g. the least-squares-fit and the f statistic belong to the class of $\mathcal{NP}$-complete problems. Due to this it is quite common to apply heuristics, often relying on rather simple hill-climbing algorithms. Nevertheless, heuristically tackling distance based approaches is regarded to rather quickly yield reasonable phylogenies.

The difficulty of finding the (near-)optimal tree becomes even more evident, when looking at the possible number of trees for a given amount of species n. The number of possible unrooted trees is given by

$$(2n-5)!! = \frac{(2n-5)!}{2^{n-3}(n-3)!} \qquad \text{for } n \geq 3\,,$$

Table 3.1: The rapid growth of the amount of possible rooted trees.

$\lvert\mathcal{L}\rvert$	#rooted trees
2	1
4	15
6	945
8	135135
10	$3.44 \cdot 10^7$
12	$1.37 \cdot 10^{10}$
14	$7.90 \cdot 10^{12}$
16	$6.19 \cdot 10^{15}$
18	$6.33 \cdot 10^{18}$
20	$8.20 \cdot 10^{21}$

whereas the rooted case allows even more freedom – because of placing the root node – and therefore amounts to

$$(2n-3)!! = \frac{(2n-3)!}{2^{n-2}(n-2)!} \qquad \text{for } n \geq 2\,.$$

The extreme growth of the number of rooted trees, in which we are subsequently interested in, is demonstrated in Table 3.1.

The mentioned inference methods are utilized to derive phylogenetic trees. Unfortunately, it is likely that biologists end up with several different trees for one and the same taxa set $\mathcal{L}$ due to

- having multiple data sets available,
- inferring with different methods, or
- repeated runs of the same non-deterministic method.

This is where the consensus tree comes into play, which will be detailed in the next section.

A long-term goal in phylogenetics is to successively build up a vast evolutionary tree (or collections thereof), and eventually come as close as possible to the so-called *tree of life*. Two examples where many people collaborate on this task is the Tree Of Life Web Project [221] and Wikispecies [236].

Our journey into the domain of phylogenetics is finished with a quote of Charles Darwin of 1859 and his first drawing of an evolutionary tree shown in Figure 3.2.

> "The affinities of all the beings of the same class have sometimes been represented by a great tree... As buds give rise by growth to fresh buds, and these if vigorous, branch out and overtop on all sides many a feebler branch, so by generation I believe it has been with the great Tree of Life, which fills with its dead and broken branches the crust of the earth, and covers the surface with its ever branching and beautiful ramifications."

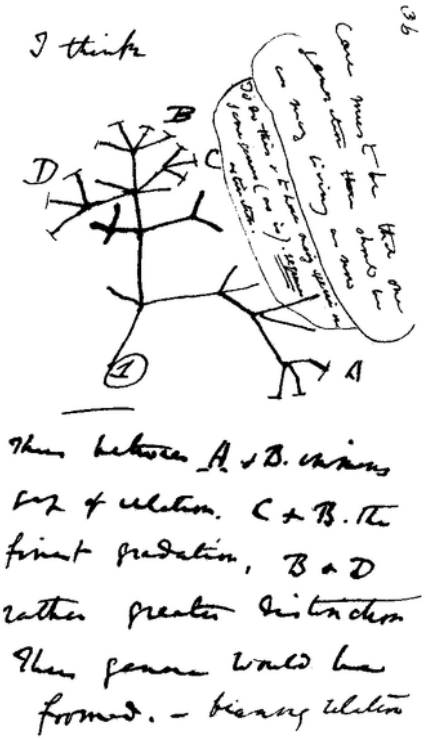


Figure 3.2: Darwin's first evolutionary tree in his "B" notebook on Transmutation of Species, 1837.

3.1.2 Consensus Tree Problem

The starting point is the question on how to take advantage of several probably high-quality but different and partly contradictory trees beside manually comparing and merging them. A possible solution is to look for a single tree over $\mathcal{L}$ "best" representing the given collection $\mathcal{T}$. This non-trivial task is known as the consensus tree problem. Notably, one assumes that independently derived trees are unlikely to have wrong or unreliable tree structures in common.

On the one hand the meaning of "best" depends on the desired information to retain in the consensus tree and on the other hand the possible consensus tree is literally restricted by the degree of strictness of the applied method as well as the granularity of the tree metric used; see Section 3.2 for more about this. Figure 3.3 shows a schematic representation of this circumstance. Generally, a strict method and a coarse-grained metric rather lead to poorly resolved trees with few inner nodes having high degrees, and a substantial portion of the information contained in the input trees is lost. In contrast, we aim at deriving fully resolved (thus, binary) high-quality consensus trees inheriting as much information as possible. Our approaches are therefore based on maximizing rather specific fine-grained measures, the so-called *TreeRank* score (and the contained *UpDown distance*) as well as the *Weighted Triple*

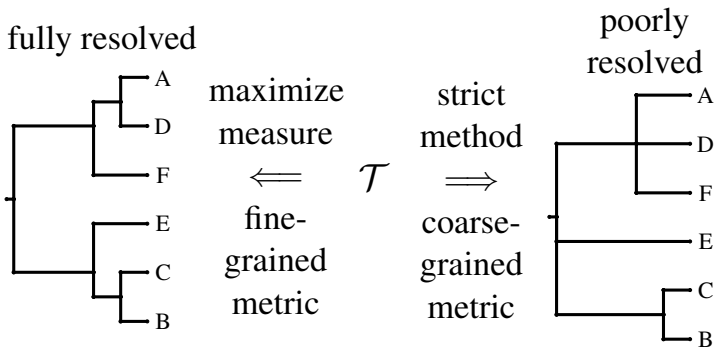


Figure 3.3: Consensus tree depending on method and metric.

score. Note that the actual complexity of the CTP when applying these measures as optimization criterion is not yet known.

Two related problems are the maximum subtree and consensus supertree problems, where the final tree does not contain all taxa and the input trees have different taxa sets, respectively.

3.2 Previous and Related Work

Several consensus tree methods have already been proposed in the literature. The work of Bryant [30] presents a good overview and comparison among the methods. Unfortunately, most methods have the drawback of being relatively strict, e.g. restricting the consensus tree to common substructures, and that the used tree metric is often coarse-grained, leading to quite poorly resolved or less intuitive solution trees. However, it is to be noted that using strict(er) methods can also be an advantage, since they are more cautious and thus implicitly reduce the chance of providing ungrounded information. In the following we will again mainly focus on methods for rooted trees. Prominent examples of classical consensus methods are the strict, majority, and loose consensus methods operating on clusters. The *strict consensus method* only retains clusters common to all input trees and the *majority consensus method* those appearing in more than half of them. The latter method can be regarded as a *median* method minimizing the number of non-common clusters, i.e. minimizing with respect to the *symmetric distance metric*:

$$d(T, \mathcal{T}) = \min \sum_{T' \in \mathcal{T}} d(T, T') \,, \tag{3.1}$$

with the symmetric difference distance $d(T_1, T_2)$ denoting the number of clusters appearing in one tree but not the other. Finally, the *loose consensus tree* contains exactly those clusters that are compatible with every tree in the input collection. It thus refines the strict consensus tree. Such state-of-the-art methods producing rather unresolved output trees can be efficiently implemented in polynomial time. Further to mention is that the classical methods do not make use of any sophisticated search procedures and rely, if at all, on simple greedy approaches: e.g. the *greedy consensus tree method* available in PHYLIP [75], which

basically adds additional clusters to the majority consensus tree, considers these clusters in decreasing order of frequency.

Phillips and Warnow [155] proposed the usage of the *asymmetric median tree* (AMT), which is an extension of the median tree. They presented a polynomial time algorithm in case of having two initial trees as well as showing the $\mathcal{NP}$-hardness when dealing with more than two trees. An approximation algorithm for the latter case yields trees which are at least as informative (refined) as median trees or strict consensus trees.

So far all mentioned consensus methods operate on clusters. An obvious drawback of this coarse metric is the case of having two trees sharing not a single cluster. A possible remedy is to base the metric on a more fine-grained tree structure: rooted triples. Hence the set of common rooted triples is often far more informative than those of common clusters. A classical method utilizing the triples is the *local consensus tree algorithm* [30]. It is based on the set R of triples appearing in each input tree, i.e. $R = \cap_{T \in \mathcal{T}} r(T)$, hence R being compatible. The local consensus tree is then built with the supertree algorithm of Aho et al. [4]. For the derived tree holds that $R \subseteq r(T)$. Another method is the R^* (R *star*) consensus tree, which complements the latter method. One determines the set R_{maj} of triples $ab|c$ that appear more frequently in the input collection than their conflicting triples $ac|b$ or $bc|a$. A maximum refined rooted tree T can be obtained via using the strong cluster algorithm of Berry and Bryant [15], such that $r(T) \subseteq R_{\text{maj}}$.

The *maximum consensus tree from rooted triples* (MCTT) problem is subject of the work by Wu [240]. Having a set of triples the existence of an exact consensus tree as well as the tree itself can efficiently be determined (again) by Aho et al.'s algorithm [4], where an exact consensus tree satisfies all given triples. The author tackles the case when no such exact consensus tree can be derived, hence instead resorting to satisfy as many given triples as possible. This amounts to solving the MCTT, which is shown to be $\mathcal{NP}$-hard by reduction from the Feedback Arc Set problem. An exact algorithm based on dynamic programming for smaller instances and a heuristic especially suitable for larger instances are presented.

A somewhat related problem appears when trying to obtain an unrooted tree from a set of quartets, which is the analogue construct to rooted triples, but being defined on four taxa. In [234] the authors for the first time present an integer linear programming model for the problem, maximizing the overall confidence of a tree's quartets.

Coming to the majority or median consensus tree again, its objective was to minimize its difference to the set of input trees utilizing the symmetric distance or partition metric (3.1). To alleviate the drawbacks of its coarseness one could use other distance metrics as objectives, too. Possible metrics would be the nearest neighbor interchange, subtree prune and regraft, as well as tree bisection and reconnection, which are described in [8]. Unfortunately, their applicability is hampered due to their either proven or assumed $\mathcal{NP}$-hardness.

Finally, the first metaheuristic approaches, based on evolutionary algorithms, applied to the consensus tree problem have been described by Cotta [45]. Therein the fine-grained TreeRank measure was used for the first time as an optimization criterion, which runs in polynomial time. More on this work is especially reported in Section 3.5.

While we are not aware of other metaheuristics to identify consensus trees, there already

exists a lot of such approaches for phylogenetic inference: Several EAs similar to the aforementioned are described in [47, 142]. In [48] an EA has been extended to a MA utilizing a local search based on subtree rotations. A more general survey of evolutionary computation in phylogenetics is given in [80]. Of further interest are applied metaheuristics apart from EAs like a greedy randomized adaptive search procedure (GRASP) and a VNS with embedded VND utilizing NNI, Step, and SPR neighborhood structures [9]; see Section 3.4. A GRASP/VND hybrid using a multiple SPR neighborhood (i.e. a composition of successive SPR moves) was introduced in [194]. In this work we adopt some of these well working strategies originally proposed for phylogenetic inference to also solve the consensus tree problem in better ways.

3.3 Applied Tree Similarity Measures

For our subsequently presented methods we basically apply two different similarity measures. The first one is the already mentioned TreeRank measure [232], which is the subject of the next section, followed by the so-called Weighted Triple score in Section 3.3.2.

3.3.1 TreeRank Score

A recently proposed tree (dis-)similarity measure, the *TreeRank measure* [232], originally introduced to handle database queries for similar trees in TreeBASE [222], allows for more sophisticated consensus tree procedures due to its fine granularity. This measure utilizes the quadratic Up matrix U which states for each pair of taxa (a, b) the number $U[a, b]$ of necessary up-traversals to reach from taxon a the least common ancestor of both taxa; see Figure 3.4 for an example. It can be derived in $O(|\mathcal{L}|^2)$ [232]. The authors also defined the Down matrix D in an analogous way, but since $U = D^{\mathrm{T}}$ it is redundant and the Up matrix is also called UpDown matrix. Having this matrix for two trees T_1 and T_2 and assuming equal taxa sets, one can calculate the *UpDown distance* between them by

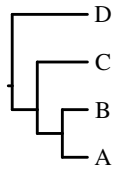
$$UpDownDist(T_1, T_2) = \sum_{u,v \in \mathcal{L}} |U_{T_1}[u, v] - U_{T_2}[u, v]| \; . \tag{3.2}$$

This distance is finally the basis for the *TreeRank* score between two trees:

$$TreeRank(T_1, T_2) = \left(1 - \frac{UpDownDist(T_1, T_2)}{\sum_{u,v \in \mathcal{L}} U_{T_1}[u, v]}\right) \cdot 100\% \; . \tag{3.3}$$

For the more general case of having different taxa sets in T_1 and T_2 see [232]; we will only need the stated definitions. The TreeRank score is thus a measure of the topological relationships in T_1 that are found to be the same or similar in T_2. It is bounded above by 100% but has no lower bound, which can be shown by comparing perfectly balanced and maximal unbalanced trees.

An alternative measure (to be minimized) is to solely use the previously introduced UpDown distance. Probably not as sophisticated as the TreeRank measure, it has the advantage of



	A	B	C	D
A	0	1	2	3
B	1	0	2	3
C	1	1	0	2
D	1	1	1	0

Figure 3.4: Exemplary tree and its Up matrix.

being a linear function on the values of the UpDown matrix, and it is symmetric. We will use the UpDown distance in our ILP-based exact methods later on.

3.3.2 Weighted Triple Score

In this section we propose to define a measure which is based on the rooted triples which are realized in a given tree, which can, as already mentioned, be regarded a fine-grained measure. For binary (fully resolved) trees with n leaves this set of triples always has the same size, since for every triple of taxa $\{a, b, c\} \subset \mathcal{L}$ exactly one of the three possible rooted triples $ab|c$, $ac|b$, or $bc|a$ is realized:

$$|R(T)| = \frac{\binom{n}{3}}{3} = n \cdot \binom{n-1}{2}. \tag{3.4}$$

Following this, we seek a consensus tree T *maximizing* the number of common rooted triples in the whole input tree collection $\mathcal{T}$, hence the *Weighted Triple* (WT) measure is defined by:

$$WT(T, \mathcal{T}) = \sum_{T' \in \mathcal{T}} |R(T) \cap R(T')| = \sum_{t_{ab|c}:\, ab|c \in R(T)} w^{\mathcal{T}}_{ab|c} t_{ab|c}\,, \tag{3.5}$$

where the coefficients $w^{\mathcal{T}}_{ab|c}$ are defined as the number of input trees in which triple $ab|c$ is present, and variables $t_{ab|c}$ denote whether triple $ab|c$ is present (i.e. realized) in tree T. In the style of the *quartet puzzling problem* [234] we might also term an optimization based on maximizing this score as the *triple puzzling problem*. However, in the former work they were already given a set of specific quartets, whereas we assume to have a collection of input trees from which we extract the information for optimizing the resulting tree.

The WT measure is linear and easy to calculate. Moreover, additional weights can be easily assigned to the input tree collection representing the confidence on each phylogenetic inference method used, which is especially interesting from a practical point of view.

3.3.3 Comparing the TreeRank Measure to a Triple-based Score

To investigate the relatedness of the TreeRank measure to the score which is based on the triples, we additionally define following slightly extended version of the previously defined

WT score:

$$WT_{\%}(T_1, T_2) = \frac{|R(T_1) \cap R(T_2)|}{|R(T_1)|} \cdot 100\% \,. \tag{3.6}$$

The number of common triples is simply divided by the (constant) number of realized triples (see previous section) and multiplied by 100 to also yield a measure in percent. Since the TreeRank measure is not symmetric we use following average measure:

$$\frac{\mathit{TreeRank}(T_1, T_2) + \mathit{TreeRank}(T_2, T_1)}{2} \,. \tag{3.7}$$

The correlation is shown in Figure 3.5, where for increasing sets of taxa either all (for $n = 4$ and $n = 5$) or a random sample of all possible trees is used. As can be seen the correlation coefficient decreases for increasing size of taxa sets. Also considering that we only investigated trees of up to 10 taxa, it is to be expected that optimizing one of the measures does not automatically yield high quality trees for the other measure as well. This result is of importance w.r.t. the exact methods to be developed.

3.4 Neighborhood Structures

In this section the applied neighborhood structures *Step*, *Swap*, *Rotate*, and *SPRr* are described, which are subsequently applied to improve the performance of the EA as well as to derive a VNS with an embedded VND. For the unrooted case, a good overview of Step, Rotate, and the general SPR is given in [9]. At first we consider the neighborhoods dealing with single taxa.

A **Step** move consists of removing a taxon with its predecessor node and reinserting them at some other branch in the tree or as new root, see Figure 3.6 for an example. In a tree containing n taxa, there are always $n - 2$ *inner branches* (i.e., arcs not directly leading to a taxon) and n *leaf branches* plus the position as new root. Hence after removing a single taxon with its leaf branch there are $((n - 2 - 1) + (n - 1) + 1) - 1 = 2n - 4$ possible new insertion positions, yielding $n(2n - 4)$ neighbors for a given tree reachable via one Step move. A Step move constitutes the smallest possible topology change of a tree. We search this neighborhood as well as the others following a deterministic first improvement strategy by utilizing the given pre-order traversal of the tree. In more detail, we go through the pre-order representation, consider each encountered taxon for removal and perform for it a nested tree traversal for enumerating all possible reinsertion points.

Our second neighborhood structure is defined by two related Step moves that exchange two taxa but keep the tree structure otherwise unchanged; thus, a **Swap** move is performed. Each pair of taxa can be swapped, resulting in $n(n - 1)/2$ neighbors, whereof at most $n(n - 1)/2 - 1$ are distinct because there exists at least one sibling pair. This neighborhood variant has not been used before; an example is given in Figure 3.7. When searching this neighborhood the possible Swap moves are deterministically enumerated similarly as the Step moves by two nested pre-order tree traversals.

The next two neighborhoods more generally operate on whole subtrees. The nearest neighbor interchange (NNI) move for the unrooted case from [9] can be interpreted as a rotation within

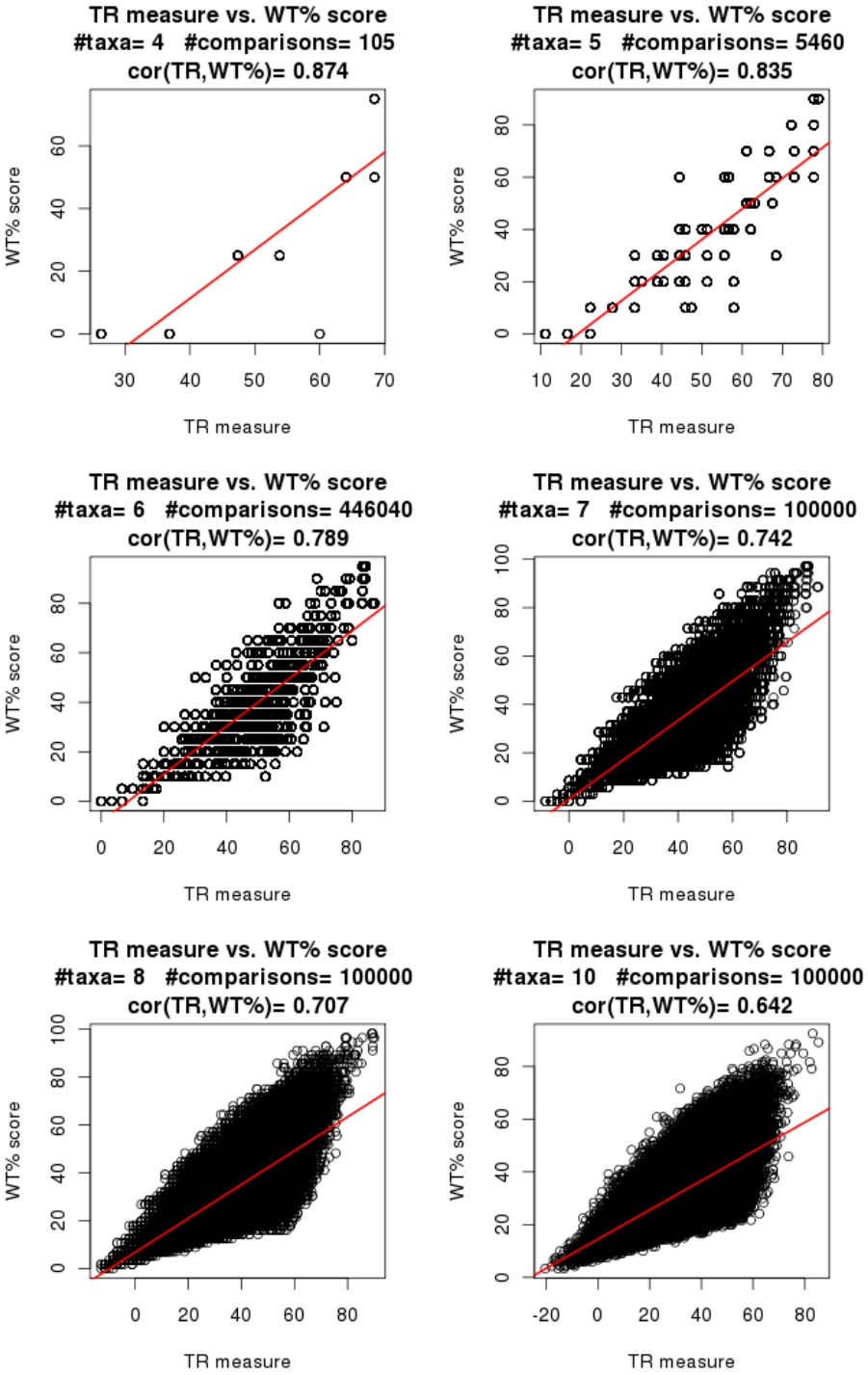


Figure 3.5: Plots showing the correlation of averaged TreeRank and Weighted Triple score in percent with a linear regression line for increasing taxa.

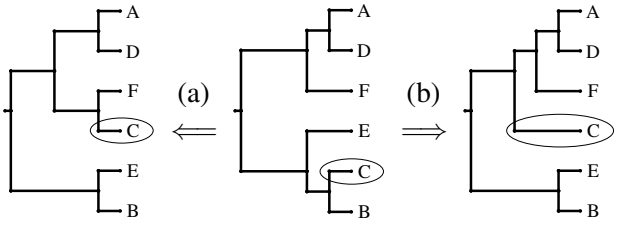


Figure 3.6: Exemplary Step moves of taxon C (a) to the leaf branch of F and (b) to an inner branch.

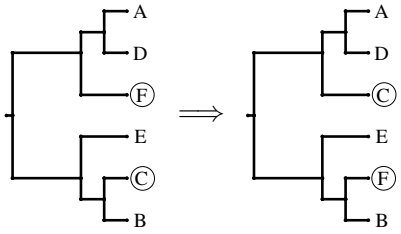


Figure 3.7: Exemplary Swap move of taxa C and F.

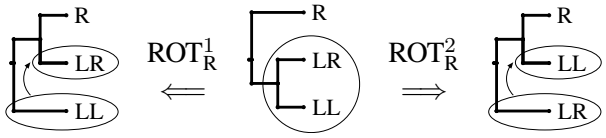


Figure 3.8: Schematic right rotations.

the tree for the rooted case; thus, we call it a **Rotate** move. There are four possible rotations distinguishable, two right rotations (see Figure 3.8) and two left rotations in an analogous way. The two variants per direction result from the possibilities of connecting either the left or the right inner subtree directly to the (sub-)tree's root. For every inner branch there are two rotations possible, thus there are $2(n-2) = 2n-4$ Rotate neighbors. Hence it is a relatively small neighborhood. We search it by performing a pre-order traversal to enumerate all inner branches, trying for each one all valid rotations, and taking a best one in case several rotations lead to an improvement.

The last neighborhood we use is a restricted form of the subtree prune and re-graft (SPR) neighborhood, excluding the movement of single taxa, thus denoted by **SPRr**. A SPRr move selects a non-trivial subtree, prunes it from the tree, and re-grafts it at some other branch. There exist $O(n^2)$ neighbors; the actual number depends on the current tree topology and is investigated for SPR in more detail in [216]. Two exemplary moves are presented in Figure 3.9. The SPRr neighborhood is searched similarly as Step, i.e. we traverse the tree in pre-order to enumerate all non-trivial subtrees for pruning and perform for each a nested tree traversal to determine all branches for re-grafting.

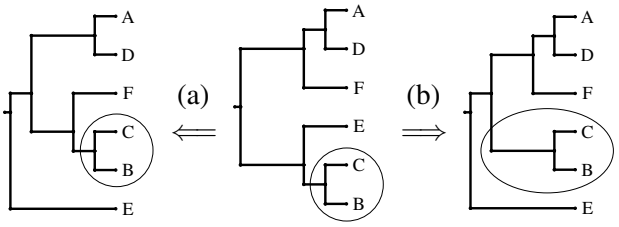


Figure 3.9: Exemplary SPRr moves of subtree (-1, B, C) (a) to the leaf branch of F and (b) to an inner branch.

3.4.1 Improvements

In order not to waste time on calculating the UpDown matrix always from scratch when evaluating a neighbor solution, we derived incremental update schemes for all the neighborhood structures. Only possibly affected parts of the UpDown matrix are efficiently updated, basically reducing the effort for obtaining the actual UpDown matrix from $O(\mathcal{L}^2)$ to $O(\mathcal{L})$. E.g. when swapping two taxa the corresponding columns of the UpDown matrix only need to be swapped, too. This strategy greatly improves the overall run-time. Unfortunately, it does not seem to be possible to follow this idea further by also calculating the UpDown distance and the TreeRank score itself in a significantly more efficient incremental way. Thus, these calculations are always completely performed.

A second improvement we consider is to avoid unnecessary moves that would result in the original tree again: swapping two adjacent taxa or moving a taxon or subtree to the same relative position. In this way some calculations of the TreeRank score can be saved.

3.5 Evolutionary Algorithm

In the work of Cotta [45] several evolutionary algorithms are presented differing in the way of whether applying mutation or crossover and the adopted evolution model. Tests on real-world instances indicate that solely applying the well-known *prune-delete-graft* (PDG) recombination operator [142] in combination with a steady-state model performs best. This recombination operator selects a subtree of the first parent at random, removes its leaves in the second parent and grafts the subtree therein at a random position. Considering subtrees as building blocks, they are well preserved, inherited, and mixed by this operator. Although this operator was shown to be more destructive to one parent than to the other [210], it is the de facto standard when dealing with phylogenetic trees in evolutionary algorithms in general. The variants utilizing only mutation by scrambling subtrees or swapping taxa turned out to be inferior. It is further important to include the given input tree collection $\mathcal{T}$ in the initial population, otherwise the results are significantly worse. As fitness function the average TreeRank score of a candidate solution to the set of input trees is used:

$$TreeRank(T, \mathcal{T}) = \frac{\sum_{T' \in \mathcal{T}} TreeRank(T, T')}{|\mathcal{T}|} \,. \tag{3.8}$$

Algorithm 7: Memetic Algorithm for the CTP

```
1 Initialize population
2 while stopping condition is not met do
3     Select parents x_a and x_b via binary tournaments
4     x_c ← Apply PDG recombination on x_a and x_b
5     if new best solution x_c found then
6         Apply local search on x_c
7     Insert x_c in population when there is no other solution with the same TreeRank
      score, replacing the worst solution
```

A solution tree is encoded in a pre-order traversal always stating the middle node followed by the nodes of the left and the right subtrees in a recursive way, yielding for the tree in Figure 3.4 (-1, -1, -1, A, B, C, D), with inner nodes being represented by -1. It is to be noted that although Cotta also proposed greedy heuristics, we do not consider them here due to their unsatisfying performance.

We re-implemented the EA variant that has been found to perform best in [45] as a basis, i.e. always applying PDG recombination and no mutation. The EA selects parents via binary tournaments with replacement and works in a steady-state fashion, i.e. in each iteration one offspring is derived, and it always replaces the worst solution in the population with one exception: To avoid duplicates and enforce a minimum diversity, new solutions having the same TreeRank score as solutions in the population are immediately discarded. The initial population consists of a copy of the input tree collection $\mathcal{T}$ and otherwise randomly created trees on $\mathcal{L}$ without any bias. In addition to [45] we also tried to use recombination and mutation together, though according to preliminary tests the performance was usually worse, hence solely recombination was finally used, too. The EA terminates when a given time or iteration limit is reached.

This EA also forms the basis for our extension to a memetic algorithm and for the combination with the VND/VNS, reported in Section 3.6 and 3.8, respectively.

3.6 Memetic Algorithm

In the EA described in the previous section we embedded different variants of local search utilizing the neighborhoods described in Section 3.4. SPRr was omitted as it yielded poorer results in preliminary tests, presumably because of the similarity to PDG recombination. Each of the variants uses a single neighborhood structure and applies a random neighbor step function; i.e. a random move is performed and the new solution is accepted if it is better than the original one. Improved trees are re-encoded in the chromosome in a Lamarckian manner. A local search phase terminates after a certain number of consecutive non-improving moves.

We further developed a *simple progressive search* (SPS) roughly following the idea in [97]. In this original work, the algorithm starts with the large SPR neighborhood and progressively reduces it by limiting the distance between the removal and insertion positions of a subtree

until the neighborhood converges to NNI having a distance limit of one. In contrast, we use several different neighborhoods without this smooth transition though maintaining the idea to reduce the size of the applied neighborhood: In the first third of the MA—either w.r.t. an iteration or time limit— we apply the Step local search, followed by Swap in the second third, and finally Rotate in the last third, whereas the local search variants are the ones described before.

Although a single application of one of the local search variants is relatively fast, trying to improve every offspring would dramatically increase the overall run-time without a substantial quality gain in the final solution. Preliminary tests further indicated that it is also not wise to apply local search on a certain portion of randomly chosen offsprings. Similarly as in [48], we perform it only on new incumbent solutions, which is restrained but turned out to be most effective. A pseudo-code of the MA is shown in Algorithm 7.

3.7 VNS with embedded VND

For the variable neighborhood descent (VND) we use the neighborhoods described in the previous section and apply the already mentioned first-improvement strategy, thus always immediately accepting the first solution yielding a better TreeRank score than the current. Due to the size and related evaluation effort of the neighborhoods and their impact on a tree's structure the following order is used for the VND: Rotate, Swap, Step, and finally SPRr.

In contrast to VND, variable neighborhood search (VNS) focuses more on diversification by applying shaking, i.e. random moves in larger neighborhoods. For intensification, VNS includes a local search component, in this case VND. Algorithm 8 shows the pseudo-code of the VNS with embedded VND. If the VNS is applied as a stand-alone algorithm, then we utilize an input tree from $\mathcal{T}$ with the highest TreeRank score as initial solution. For shaking, we perform a series of Step moves: A certain percentage of the taxa is randomly selected and moved to some other, also randomly chosen positions. We start with 5% of the taxa and gradually increase this portion by 5% up to 100% and hence the number of different VNS neighborhoods $k_{\max}$ is 20. Both VND and VNS terminate if an iteration or time limit is reached. VND also stops when the last neighborhood contains no better solution and thus, the current solution is locally optimal w.r.t. all VND neighborhood structures.

3.8 Hybrid Metaheuristic Variants

Considering our developed algorithms it is an obvious idea to combine the EA (or MA) with the VND or VNS. In our first approach, we do this as in the MA of the previous section: The VND is always only applied to new best solutions as a strong local improvement. This EA/VND hybrid will be denoted by Hyb_{B}.

Another combination possibility is to run the EA and the VNS in a pure sequential way: The EA's finally best solution is used as the starting point for the VNS. We term this algorithmic setting as Hyb_{S}.

Algorithm 8: VNS with embedded VND (x)

```
1  Set neighborhood structures for VND: N_1=Rotate, N_2=Swap, N_3=Step and N_4=SPRr
2  k ← 0; // shaking strength
3  while VNS stopping condition is not met do
4      while k ≠ k_max do
           // Shaking:
5          x' ← Apply random Step moves on k · 5%|L| taxa of x selected at random
           // Local search by VND:
6          l ← 1
7          while l ≠ 4 and VND stopping condition is not met do
               // Exploration of neighborhood l:
8              Search first improving neighbor x'' ∈ N_l(x')
               // Eventually move to x'' and
               // change neighborhood within VND:
9              if better solution x'' found then
10                 x' ← x''
11                 l ← 1
12             else l ← l + 1
           // Eventually move to x' and
           // change shaking strength:
13         if x' is better than x then
14             x ← x'; // set new incumbent
15             k ← 1; // reset shaking
16         else k ← k + 1; // increase shaking
       // finished VNS iteration
17     k ← 1; // reset shaking
```

A refinement of this approach is the intertwined execution of the EA and the VNS, as it is shown in Algorithm 9: A prespecified total execution time T is divided into 2τ slots and both algorithms are alternately applied. The EA starts and retains its population during its pauses. When the VNS takes over, it always begins with the EA's so far best solution and with its first neighborhood. Of course, each final solution of the VNS at the end of its slots is also inserted into the EA's population, replacing the worst solution and rejecting solutions having the same TreeRank scores as already existing ones. We denote such an intertwined setting with τ EA/VNS phases by $\text{Hyb}_{\text{I}\tau}$.

One might think it is even better to use the MA instead of the EA in the hybrid variants, but this is not true in general. Preliminary tests have clearly shown a worse performance of a sequential MA/VNS combination. This is probably due to the MA already focusing too strongly on local optima and thus seeding the VNS with a solution less suitable for further improvement. In the following, we therefore only consider in more detail an intertwined

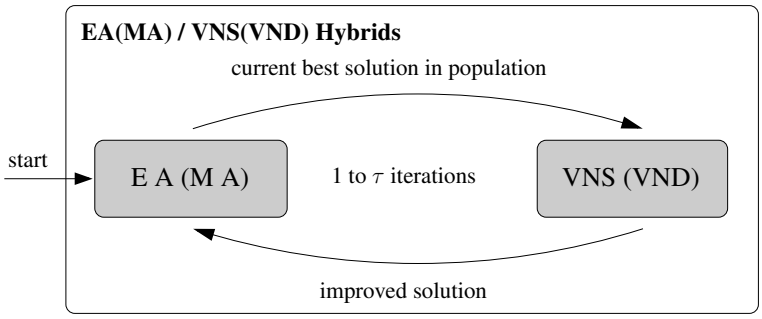


Figure 3.10: Information exchange between EA (MA) and VNS (VND).

Algorithm 9: Intertwined EA/VNS Hybrid (T, τ)

```
1  Initialize population of EA
2  t_slot = T/(2·τ); // time limit per slot
3  iter ← 0
4  while iter < τ do
5      Run the EA for duration t_slot
6      x ← best solution in EA's population
7      Run the VNS on x for duration t_slot
8      x' ← best solution found by VNS
9      if x' is better than x then
10         Insert x' in population of EA
11     iter ← iter + 1
```

MA/VNS hybrid which we denote by $\text{Hyb}^*_{\text{I}\tau}$.

A schematic overview of the actual information exchange of the hybrid variants is given in Figure 3.10.

3.9 Guided Neighborhood Variants

In Section 3.4 and subsequently for the VND in Section 3.7 we adhered to a fixed order for examining the possible moves of the presented neighborhoods, the pre-order traversal of the tree. Since we apply a first improvement scheme a potential advantage might be gained via using a dynamic, instance specific examination order: finding improved solutions more quickly and/or of better quality in the end. As information for the guided exploration we utilize the UpDown distance which directly indicates whether a taxa is well-placed or not. Due to implementation details we omit the guided variant of the Rotate neighborhood. For the Swap and the Step neighborhood we thus determine the UpDown distance to the input trees for every single taxa, whereas for the SPRr neighborhood a taxa always represents a whole subtree for which the corresponding single taxa's UpDown distances are summed up.

Afterwards the taxa are sorted in decreasing order of their assigned UpDown distances and subsequently considered in this order. In this way more emphasis is placed on changing "expensive" (i.e., dissimilar) tree structures first. Of course a certain overhead is to be expected due to determining the UpDown distances per taxa. This will be investigated in the results section.

3.10 ILP-based Exact Methods

The TreeRank measure is not (directly) applicable as objective function for an ILP model, since it is highly non-linear. Hence we will either apply the UpDown distance or the Weighted Triple score. Unfortunately, as we will see later on, especially an ILP model with the UpDown distance as objective function is in general only applicable to smaller instances as it soon requires too much computational effort.

3.10.1 Triple Model

The first proposed ILP model, denoted as *Triple model*, uses $t_{ab|c}$ variables representing the presence or absence of each possible triple, and further utilizes the Weighted Triple score (3.5) as objective function to be maximized (3.9); it is similar to the model for quartets presented in [234].

$$\max \sum_{t_{ab|c}:\, ab|c \in R(T)} w^{\mathcal{T}}_{ab|c} t_{ab|c} \tag{3.9}$$

$$\text{s.t.} \quad t_{ab|c} + t_{bc|a} + t_{ac|b} = 1 \qquad \forall a < b < c \in \mathcal{L} \tag{3.10}$$

$$t_{ab|c} + t_{ad|c} - t_{bd|c} \leq 1 \qquad \forall \{a,b,c,d\} \subset \mathcal{L} \tag{3.11}$$

$$t_{ab|c} + t_{ac|d} - t_{ab|d} \leq 1 \qquad \forall \{a,b,c,d\} \subset \mathcal{L} \tag{3.12}$$

$$t_{ab|c} = t_{ba|c} \qquad \forall a < b, c \in \mathcal{L} \tag{3.13}$$

$$t_{ab|c} \in \{0,1\} \qquad \forall \{a,b,c\} \subset \mathcal{L} \tag{3.14}$$

Constraints (3.10) ensure that exactly one possible triple $ab|c$, $bc|a$ or $ac|b$ is realized in the resulting tree. The inequalities (3.11) and (3.12) express the triple transitivity and so-called telescopic conditions, which are used in order to derive new necessary triples from other ones to ensure consistency. Equality (3.13) states that triples $ab|c$ and $ba|c$ are equivalent. Finally, constraints (3.14) enforce binary triple variables. This model implies $\Theta(n^3)$ variables and $\Theta(n^4)$ constraints and has the advantage of being relatively efficient, since also the objective function is easy to calculate. As we will see, moderately-sized instances can be solved well in practice.

3.10.2 Combined Triple and UpDown Distance Model

Our second ILP model, denoted as *Triple+UDD model*, is an extension of the previous one and includes both $t_{ab|c}$ variables as well as u_{ab} variables representing the values in the Up-

Down matrix of the final solution tree. The purpose of this is to allow the use of an objective function based on the values of the UpDown matrix—which is thus finer grained and assumed to be more realistic—and at the same time obtain the tree defined by the triple variables, in order to avoid the conversion from the UpDown matrix to the triple variables (since this algorithm is not yet known, and might be a topic of future investigation). Also for ensuring consistency of the UpDown matrix the $t_{ab|c}$ variables are used.

$$\min \quad \sum_{T\in\mathcal{T}} \sum_{a,b\in\mathcal{L}} \delta_{Tab} \tag{3.15}$$

$$\text{s.t.} \quad U_T[a,b] - u_{ab} \leq \delta_{Tab} \qquad \forall T \in \mathcal{T}; \forall a,b \in \mathcal{L} \tag{3.16}$$

$$u_{ab} - U_T[a,b] \leq \delta_{Tab} \qquad \forall T \in \mathcal{T}; \forall a,b \in \mathcal{L} \tag{3.17}$$

$$u_{aa} = 0 \qquad \forall a \in \mathcal{L} \tag{3.18}$$

$$u_{ab} \geq 1 \qquad \forall a,b \in \mathcal{L} \tag{3.19}$$

$$u_{ab} \leq n-1 \qquad \forall a,b \in \mathcal{L} \tag{3.20}$$

$$u_{ab} - u_{ac} < M(1 - t_{ab|c}) \qquad \forall \{a,b,c\} \subset \mathcal{L} \tag{3.21}$$

$$u_{ba} - u_{bc} < M(1 - t_{ab|c}) \qquad \forall \{a,b,c\} \subset \mathcal{L} \tag{3.22}$$

$$u_{ca} - u_{cb} \leq M(1 - t_{ab|c}) \qquad \forall \{a,b,c\} \subset \mathcal{L} \tag{3.23}$$

$$u_{cb} - u_{ca} \leq M(1 - t_{ab|c}) \qquad \forall \{a,b,c\} \subset \mathcal{L} \tag{3.24}$$

$$u_{ac} - u_{ab} - (u_{bc} - u_{ba}) \leq M(1 - t_{ab|c}) \qquad \forall \{a,b,c\} \subset \mathcal{L} \tag{3.25}$$

$$u_{bc} - u_{ba} - (u_{ac} - u_{ab}) \leq M(1 - t_{ab|c}) \qquad \forall \{a,b,c\} \subset \mathcal{L} \tag{3.26}$$

$$\min\{u_{ab} \mid b \in \mathcal{L} \setminus \{a\}\} = 1 \qquad \forall a \in \mathcal{L} \tag{3.27}$$

$$u_{ab} \in \mathbb{N}_0 \qquad \forall a,b \in \mathcal{L} \tag{3.28}$$

$$\delta_{Tab} \in \mathbb{N}_0 \qquad \forall T \in \mathcal{T}; \forall a,b \in \mathcal{L} \tag{3.29}$$

(3.10)–(3.14)

Here the UpDown distance (3.2) is used as an objective function: we introduce additional variables δ_{Tab} (3.29) which hold the absolute values linearized via (3.16) and (3.17), finally allowing to minimize the sum of them (3.15). The distance from a taxa to itself is zero (3.18) and the distance between two different taxa is at least one (3.19) and at most the number of taxa minus one (3.20), since the latter is the largest depth of a binary tree having n leaves. Constraints (3.21)-(3.26) ensure that the UpDown matrix is consistent. In these constraints M is a sufficiently large constant (hence often also referred to as *big-M* constraints) which can be set to the number of taxa n. This ensures that the constraints are activated only when the triple $ab|c$ is present (i.e. when $t_{ab|c} = 1$). Constraints (3.25)-(3.26) are called *path constraints*. They ensure that the values themselves of the UpDown matrix, and not only the relative distances, are consistent. The *row-min constraints* (3.27) ensure that no "artificial" inner nodes may be added to lower specific taxa, which might otherwise happen (also depending on the objective function in use). This model still implies $\Theta(n^3)$ variables and $\Theta(n^4)$ constraints.

3.10.3 Reduce Computational Effort with Lazy Constraints

For some ILP models quite a lot constraints are introduced to guarantee feasibility. They can be regarded consistency constraints. In such cases it might be beneficial to initially omit some of these constraints, but of course they would have to be respected during solving in an appropriate way. One rather naïve possibility is to check the feasibility of the solution obtained after solving the reduced model to optimality. After adding all violated constraints one has to do a complete re-solving, probably in an iterated manner, causing (most likely) too much overhead. Hence in order to gain more speed-up compared to solving the initial complete model, these constraints are at best added in an incremental way during the solving process. Ilog CPLEX offers an elegant way to do this via a pool of *lazy constraints*. For this one adds the desired consistency constraints in a pool (set), which in our case are the transitivity (3.11) and telescopic constraints (3.12), and they are henceforth treated differently. Whenever an integer solution is found it is checked whether one of these constraints is violated and adds them to the model if required; consult the user manual for more information. This scheme is in some sense analogue to the cutting plane approach when solving a linear program. Depending on the problem, this can dramatically improve the performance of the solver.

3.10.4 Heuristic Generation of Variables

Instead of reducing the number of (initial) constraints the optimization algorithm has to deal with, the number of variables considered for solving the problem can also be restricted at the beginning. Triple variables $t_{ab|c}$ corresponding to triples $ab|c$ for which it is assumed that they have value zero in the final solution are eliminated from the problem. This step is referred to as pruning the variable set. The important question is: How is the variable set to be pruned, so as not to discard the optimal solution? Since there is no theoretical background to answer this question, the variable set is pruned in a heuristic way: triples that are conjectured to most likely not appear in the optimal solution are pruned. These are exactly the triples not appearing in any input tree. Since the tree which best summarizes the information contained in the input trees is sought, it is expected that triples not appearing in any input tree will not be present in the final solution.

The idea of beginning with a reduced set of variables is closely related to column generation approaches, see Section 2.3.1 for a general outline and Section 4.7 for a concrete application of it. However, since for the CTP the pricing problem is most likely only solvable via complete enumeration of all variables (i.e. showing no special structure for which an effective algorithm is known), we settled here to propose a heuristic method in which variables are added in case they appear in an incumbent solution. For this purpose, an iterative rounding and repair procedure is applied to each non-integer candidate solution in order to find a feasible tree. In a first step this procedure investigates the triples in lexicographic order and sets those corresponding triple variable having the highest (fractional) value in the LP solution to one (randomly selecting a variable in case of equal values) and the other two variables to zero. Then for each subset of four taxa we check all according transitivity and telescopic constraints and in case of a violation repair them individually by setting the corresponding

triple variable to one and the other two to zero. This repair step is consecutively applied until no more violations occur.

We denote this whole method as heuristic column generation. In this approach, the initial set of variables is chosen by the previously described pruning method.

3.10.5 Hybridization of Heuristic and Exact Methods

Initially we planned to combine the strengths of the heuristic and exact approaches in a suitable way. We especially had in mind to select a subtree of a meaningful size (i.e. small enough to be solved in reasonable time yet large enough to allow for an improvement) and solve it to (near) optimality or to derive some sort of exact recombination operator (i.e. basically a solution merging). Unfortunately, for several reasons this did not work out. The major problem was the non-linearity of the favored TreeRank similarity measure and the fact, that optimizing using alternative linear measures does not imply maximizing the TreeRank measure. This makes it virtually impossible that the proposed exact methods are of a benefit for the metaheuristics. Considering the contrary hybridization, i.e. in some way boosting the performance of the exact approaches via utilizing the heuristics, also yielded no meaningful possibilities here. Examples of the latter would be to offer better initial solutions than the best input trees as well as to provide the exact methods with improved feasible solutions during its execution. On the one hand these approaches would not really represent something new from a methodical point of view and on the other hand the applicability of the exact approaches strongly depends on the problem size (also memory consumption becomes an issue due to the large number of constraints) and it is unlikely that good bounds from feasible solutions will allow to tackle larger instances.

3.11 Experimental Results

This section gives details of the algorithm settings, introduces the test instances, and presents the computational results as well as a comparison between them. All approaches have been implemented in C++ and were compiled with GCC 4.1.2. The experiments with the meta-heuristic algorithms were performed on a 2.2 GHz Dual-Core AMD Opteron 2214 PC with 4 GB RAM, those of the ILP-based approaches on a 2.8 GHz Intel Core i7 PC with 8 GB RAM. The best performing (pure) EA from [45] has been reimplemented as proposed in this original work and described in Section 3.5.

We adhere to the following abbreviations for the similarity measures: TR for TreeRank score, UDD for UpDown distance, WT for Weighted Triple score, and TR-WT for TreeRank based Weighted Triple score.

3.11.1 Algorithms Settings

The population size of the EA and MA was set to 100. In the MA local search is applied to all new incumbent solutions until 100 consecutive non-improving moves have been tried.

Due to space limitations, we present here only results for the MA with simple progressive search, which proved to generally outperform the variants where only a single neighborhood structure is considered. As mentioned earlier, the initial solution for the standalone VNS is the tree with highest TreeRank score of the input collection. Hyb_{B} is performing the VND on all new incumbents limiting run-time per call to at most 5% of the overall time limit. In case of Hyb_{S} the EA and the VNS are given 50% of the computation time each, thus it basically corresponds to Hyb_{I1}. The intertwined EA/VNS hybrid $\text{Hyb}_{\text{I}\tau}$ was applied with $\tau = 4$ alterations, thus in the following denoted by Hyb_{I4}. For $\text{Hyb}^*_{\text{I}\tau}$ we basically combine the settings of the best performing MA with SPS and of Hyb_{I4} and call it Hyb^*_{I4}. The variants using the guided neighborhood explorations use the same settings.

Following settings of the ILP-based methods proposed in Section 3.10 are used:

1. Triple model (TM)
2. Triple model plus lazy constraints (TM_{l})
3. Triple model using TreeRank based triple weighting (TM_{TR})
4. Triple model using TreeRank based triple weighting plus lazy constraints ($\text{TM}_{\text{TR+l}}$)
5. Triple+UDD model (TUDDM)
6. Triple+UDD model plus lazy constraints (TUDDM_{l})
7. Each setting with additionally pruning of the variable set, adding subscript '$_{\text{p}}$'

The corresponding best input tree (either w.r.t. to UpDown distance or Weighted Triple score) is used as an initial solution to (potentially) speed up the solution process. As a time limit for applying the ILP methods we set one hour, hence we also state the resulting gap. The general purpose MIP solver IBM ILOG CPLEX 12.2 is used for solving the models.

We further implemented the heuristic column generation approach (see Section 3.10.4) utilizing the open-source framework COIN-OR BCP[1]. However, even though we also used Ilog CPLEX as LP solver, it had an exceptionally high runtime. Looking closer at the performance suggests that the internal mechanisms of creating/initializing the model and accessing information from it were responsible for this circumstance. Though one has to keep in mind that we are faced with a large number of constraints here. Due to this, meaningful tests were not possible and we do not consider this approach in the following. Nevertheless, limited preliminary results did indicate the expected benefit for a few instances.

3.11.2 Test Instances

For experimental evaluation we use three types of instances: trees resulting from three simple agglomerative clusterings (single-link and complete-link [113] as well as average-link, also known as unweighted pair group method with arithmetic mean [213]), trees resulting from

[1] see at `http://www.coin-or.org/projects/Bcp.xml` [accessed August 16, 2010]

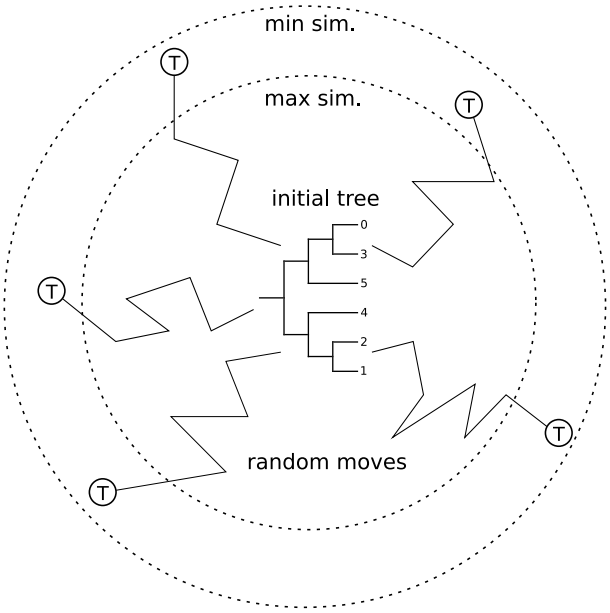


Figure 3.11: Artificial instances created by randomly perturbing an initial tree.

several runs of the scatter search approach in [46] (which were kindly provided by Carlos Cotta), and new artificial trees. The latter are created by generating one initial random tree and deriving the actual input trees by copying it and applying a series of perturbations in the neighborhoods described in Section 3.4: Random Step, Swap, Rotate, and SPRr moves are equally likely performed. In order to be able to control the similarity of the resulting input trees in a good way, we defined minimum and maximum pairwise TreeRank scores and performed the perturbations until a derived tree achieves a pairwise score w.r.t. the initial tree within these limits. If the actual score is less than the lower bound, the previous move is undone and the process continues. Note that the initial tree is finally discarded and not included in the input tree collection. A schematic presentation of this process is shown in Figure 3.11. An advantage of these artificially generated instances is that the known initial tree, although not necessarily the best possible consensus tree, lends itself as a reference solution. Having this procedure at hand also allows to generate small enough instances which can be tackled by the ILP-based exact methods. The instances are differing in the amount of taxa (10 to 178), the number of input trees (3 to 10) and their average similarity to each other (below 50% up to 90%).

3.11.3 Comparison of Algorithms

First, our aim was to make a comparison of the different heuristic algorithms. Hence per instance we ran the pure EA for $5 \cdot 10^5$ iterations and used the consumed CPU time as limit for all others. Following this we did 30 runs per algorithm setting and instance and state following TreeRank scores in Tables 3.2–3.5: the best result (best), the mean value

Table 3.2: Results on agglomerative clustering tree instances.

	M877 (3x134)				M971 (3x158)				M808agglom (3x178)			
	best	mean	sdv	med	best	mean	sdv	med	best	mean	sdv	med
EA	51.85	51.68	0.10	51.70	63.38	63.28	0.04	63.28	48.47	48.44	0.01	48.44
MA	51.90	51.73	0.11	51.75	63.36	63.28	0.05	63.28	48.51	48.46	0.02	48.46
VNS	51.97	51.88	0.05	51.88	63.34	63.32	0.01	63.32	48.53	48.50	0.01	48.49
Hyb_B	51.94	51.82	0.09	51.84	63.41	63.34	0.04	63.35	48.51	48.49	0.01	48.48
Hyb_S	51.97	**51.89**	0.04	51.89	63.41	**63.36**	0.03	63.36	48.52	48.50	0.02	48.50
Hyb_{I4}	52.00	**51.89**	0.06	51.88	63.40	63.35	0.03	63.36	48.53	48.51	0.01	48.51
Hyb^*_{I4}	51.99	51.88	0.07	51.89	63.41	63.33	0.05	63.34	48.53	**48.52**	0.01	48.52
using guided neighborhood explorations in VND												
VNS	51.95	51.84	0.07	51.84	63.37	63.29	0.02	63.28	48.49	48.49	0	48.49
Hyb_B	51.98	51.78	0.1	51.77	63.41	63.33	0.09	63.36	48.52	48.49	0.01	48.49
Hyb_S	51.99	**51.89**	0.05	51.88	*63.43*	63.35	0.07	63.36	48.52	48.50	0.01	48.50
Hyb_{I4}	51.99	51.88	0.06	51.87	*63.43*	63.35	0.05	63.36	48.53	48.51	0.01	48.52
Hyb^*_{I4}	*52.01*	51.86	0.08	51.87	63.41	**63.36**	0.04	63.37	*48.54*	**48.52**	0.01	48.52
best input	49.99				62.41				48.14			
CPU-time [s]	228				283				480			

(mean) with the corresponding standard deviation (sdv) as well as the median value (med). Overall best obtained mean values are printed bold, the best found solution per instance is printed in italics. For the agglomerative clustering and scatter search instances we also give the number of trees and taxa in parentheses, e.g. "(3x134)" for 3 trees with 134 taxa each. In case of the artificial tree instances the name itself holds this information and also the TreeRank score upper bound used during creation, e.g. "5x150_70" for 5 trees with 150 taxa each and a maximal TreeRank score of 70% w.r.t the initial tree. As lower limit the given upper bound minus 10% was used, i.e. 60% in the previous example. For these instances the TreeRank score of the initial input tree is given in line "init tree", too. We further list for each instance the TreeRank score of the best tree of the input collection $\mathcal{T}$ ("best input") and the aforementioned time limit ("CPU-time") in seconds. In the following discussion, all performance differences (i.e. differences in TreeRank score mean values) pointed out have been statistically verified by Wilcoxon rank sum tests and error levels are less than 5%.

For seven out of the 12 instances the MA yields on average significantly better solutions than the pure EA. In the remaining five cases the observed differences are not significant (although for four the MAs mean values are higher and only for instance M808scatter the EA achieved a slightly better mean). VNS is always significantly better than the pure EA except for M808scatter, 5x175_80, and 5x175_90, where the latter achieved higher scores. Compared to the MA, the VNS exhibits clearly better results on the real-world instances, but was less effective on the artificially generated ones, for which it achieved a higher mean score in only a single case (5x150_80). We believe the reason for this lies in the initial solution of

Table 3.3: Results on scatter search tree instances.

	Onco9 (9x148)				Onco10 (10x148)				M808scatter (10x178)			
	best	mean	sdv	med	best	mean	sdv	med	best	mean	sdv	med
EA	91.21	90.98	0.10	90.97	91.01	90.98	0.02	90.97	91.05	90.98	0.05	90.98
MA	91.21	90.99	0.11	90.98	91.07	90.98	0.03	90.98	91.08	90.96	0.07	90.97
VNS	91.16	91.16	0	91.16	91.09	**91.09**	0	91.09	90.96	90.96	0	90.96
Hyb_B	91.27	91.12	0.07	91.12	91.12	91.05	0.05	91.07	91.05	90.92	0.10	90.94
Hyb_S	*91.29*	**91.21**	0.04	91.22	*91.13*	**91.09**	0.02	91.09	91.09	**91.00**	0.04	91.00
Hyb_{I4}	91.26	91.20	0.04	91.22	*91.13*	**91.09**	0.02	91.09	91.10	90.99	0.06	91.00
Hyb^*_{I4}	91.28	**91.21**	0.03	91.22	91.12	**91.09**	0.01	91.09	*91.11*	90.99	0.06	90.99
using guided neighborhood explorations in VND												
VNS	91.24	**91.24**	0	91.24	91.09	**91.09**	0	91.09	90.96	90.96	0	90.96
Hyb_B	91.26	91.10	0.09	91.09	91.10	91.03	0.05	91.03	91.10	90.97	0.08	90.97
Hyb_S	91.28	91.23	0.03	91.23	91.12	91.08	0.03	91.08	91.08	**91.02**	0.04	91.02
Hyb_{I4}	91.25	91.19	0.05	91.20	*91.13*	**91.09**	0.02	91.08	91.08	90.99	0.10	91.02
Hyb^*_{I4}	91.28	91.20	0.04	91.21	91.11	**91.09**	0.01	91.09	91.06	90.98	0.09	91.00
best input	89.96				90.87				89.85			
CPU-time [s]	391				420				573			

the VNS, which is the input tree having the highest TreeRank score. While this best input tree turned out to lie relatively close to high quality consensus trees in case of our real-world instances, there are generally larger differences in the artificial instances. As the VNS is dominated by its strong local search of the embedded VND, which consumes the major part of the CPU-time on larger instances, its diversification abilities are less pronounced than those of the population-based MA.

Now we concentrate on the hybrid approaches. The results of Hyb_B, in which VND is used to locally optimize new incumbent solutions within the EA, was disappointing, as its final solutions are generally inferior to those of the other hybrids and VNS performed in most of the real-world instances better. The more systematic EA/VNS combinations Hyb_S, Hyb_{I4}, and Hyb^*_{I4} are more successful. They consistently achieve the overall best results; at least one of them performed on all instances significantly better than all other algorithms. The only exception is Onco10, where the VNS yields equally good results. Although there are only few statistically significant differences between the sequential and the intertwined hybrids, the former tends to yield better results for the real-world instances whereas the latter seems to be better suited for the artificial instances. Finally, the intertwined MA/VNS hybrid Hyb^*_{I4} shows for many instances a better performance—in fact sometimes even the best—when compared to the same variant utilizing the pure EA only (Hyb_{I4}). Altogether the intertwined variants can be clearly considered the most successful.

Table 3.4: Results on artificial tree instances with 150 taxa.

	5x150_70				5x150_80				5x150_90			
	best	mean	sdv	med	best	mean	sdv	med	best	mean	sdv	med
EA	68.86	67.81	0.40	67.70	78.56	77.95	0.28	77.98	88.65	87.99	0.31	88.01
MA	69.10	68.55	0.26	68.61	78.55	78.15	0.24	78.20	88.87	88.46	0.30	88.54
VNS	68.47	68.47	0	68.47	78.53	78.53	0	78.53	88.35	88.35	0	88.35
Hyb_B	69.40	68.90	0.21	68.83	78.83	78.46	0.26	78.53	88.91	88.50	0.34	88.59
Hyb_S	69.48	68.97	0.32	68.97	78.85	78.64	0.20	78.70	*88.96*	88.78	0.18	88.82
Hyb_{I4}	69.52	69.06	0.26	69.05	78.84	**78.67**	0.12	78.69	*88.96*	**88.84**	0.12	88.88
Hyb^*_{I4}	69.53	**69.16**	0.23	69.23	78.82	**78.67**	0.17	78.73	*88.96*	88.82	0.14	88.87
using guided neighborhood explorations in VND												
VNS	68.43	68.43	0	68.43	78.41	78.41	0	78.41	88.88	**88.88**	0	88.88
Hyb_B	69.53	68.93	0.25	68.86	*78.87*	78.49	0.25	78.57	*88.96*	88.48	0.36	88.50
Hyb_S	69.49	69.07	0.25	69.11	78.86	78.71	0.09	78.74	*88.96*	88.81	0.15	88.83
Hyb_{I4}	*69.64*	**69.17**	0.20	69.15	78.86	**78.67**	0.12	78.71	*88.96*	88.85	0.11	88.87
Hyb^*_{I4}	69.47	69.09	0.19	69.10	78.85	78.61	0.13	78.58	*88.96*	88.79	0.15	88.82
best input	64.54				72.65				84.34			
init tree	67.24				78.27				88.96			
CPU-time [s]	297				310				314			

Coming to the guided neighborhood variants, these were exploited in all algorithms incorporating the VND, more concretely in the VNS as well as in all hybrid variants. The results in bold again denote the best performing variant of this set of algorithms. Further, the same runtime limit was used for the tests. In general, the outcome of the tests is similar than when using a fixed neighborhood exploration: usually one of the hybrid variants performs best. An exception occurs for instance 5x150_90 where especially the VNS benefits from the dynamic exploration and performs best. When comparing the results of these variants with the previously proposed ones, it is notable that slightly more of the best known solutions as well as higher best average solution values are obtained. Although there is not a drastic performance boost, the inclusion of problem knowledge in the neighborhood exploration has a significant beneficial impact.

When comparing the scores of the best input trees with those of the overall best solutions found, it is apparent that our real instances offer less room for improvement, mostly below 2%, whereas the artificially generated ones allow for about 5% or even more. Another interesting fact is the relation between the TreeRank scores of the artificial instances' initial trees and the corresponding best solutions found. As can be observed in Tables 3.4 and 3.5, the initial tree is more likely the (nearly) optimal consensus tree (regarding the TreeRank score) when the derived input trees are close to the initial tree, hence when the radius of the inner circle in Figure 3.11 is small. The results of the best solutions also show that only the hybrid algorithms are consistently able to find consensus trees being better than or equal to

Table 3.5: Results on artificial tree instances with 175 taxa.

	5x175_70				5x175_80				5x175_90			
	best	mean	sdv	med	best	mean	sdv	med	best	mean	sdv	med
EA	69.57	68.66	0.38	68.65	76.94	75.98	0.48	75.96	*86.44*	85.71	0.47	85.62
MA	70.91	70.57	0.17	70.60	77.31	76.92	0.19	76.94	*86.44*	85.77	0.36	85.79
VNS	69.53	69.53	0	69.53	73.78	73.73	0.03	73.71	85.31	85.31	0.01	85.31
Hyb_B	70.95	70.47	0.24	70.51	77.51	76.95	0.33	76.95	86.33	85.95	0.28	86.00
Hyb_S	70.99	70.37	0.24	70.38	*77.56*	77.18	0.24	77.21	*86.44*	**86.37**	0.10	86.42
Hyb_{I4}	71.20	70.70	0.19	70.67	77.50	77.21	0.18	77.22	*86.44*	86.33	0.12	86.39
Hyb^*_{I4}	*71.27*	**70.94**	0.16	70.99	77.53	**77.27**	0.12	77.26	*86.44*	86.25	0.18	86.27
using guided neighborhood explorations in VND												
VNS	70.70	70.70	0	70.70	73.81	73.80	0.01	73.80	86.09	86.07	0.06	86.09
Hyb_B	71.12	70.77	0.16	70.79	77.51	77.01	0.26	77.02	*86.44*	85.99	0.27	85.94
Hyb_S	71.03	70.61	0.2	70.63	77.54	**77.25**	0.16	77.24	*86.44*	**86.36**	0.12	86.42
Hyb_{I4}	71.19	70.85	0.17	70.88	77.48	77.21	0.18	77.20	*86.44*	86.34	0.12	86.40
Hyb^*_{I4}	71.13	**70.95**	0.12	70.97	77.42	77.17	0.13	77.18	*86.44*	86.23	0.23	86.29
best input	64.75				72.10				81.34			
init tree	69.23				77.35				86.44			
CPU-time [s]	382				384				380			

the initial trees, where the latter happens for instances 5x150_90 and 5x175_90.

In the following we will have a look at the performance of the different VND neighborhoods for all algorithms using either the normal or the guided VND variant. In Table 3.6 we state how often the neighborhoods have been applied (used), how many times improvements were achieved (impr.), also giving the corresponding fraction of all applications and improvements in percent (%-total), and the average runtimes in seconds (t[s]). Since their order of application is Rotate, Swap, Step, and SPRr, the former neighborhoods are naturally applied more often in general. Moreover note that the average runtimes justify this order. Rotate and Swap account for most of the improvements, Step also contributes a little bit, while SPRr is hardly applied at all. For all hybrid variants using guided VND except of Hyb_B the Rotate neighborhood is roughly applied half the times as when using the standard VND. A reason might be that in these cases the solutions tackled by the VNS are already of a high quality and due to the better moves selected in guided Swap and Step there is less room for improvement for the rather locally operating Rotate neighborhood. The latter does not occur for Hyb_B because of the very strict time limit of the applied VND. Comparing the runtimes for Swap we can observe a longer average runtime (overhead due to determining the UpDown distances per taxon) but also a higher percentage of improvements in total for the guided variant. The latter also holds for guided Step, but here the average runtimes are less, hence earlier finding an improving move.

Table 3.6: Quantitative performance of different VND neighborhoods averaged per algorithm.

Algorithm	Rotate					Swap					Step					SPRr				
	used	%-total	impr.	%-total	t[s]	used	%-total	impr.	%-total	t[s]	used	%-total	impr.	%-total	t[s]	used	%-total	impr.	%-total	t[s]
VNS	279	71.33	186	67.05	0.07	92	23.66	74	26.63	2.25	18	4.67	16	6.08	7.19	1	0.34	0	0.24	10.90
Hyb_B	114	71.30	72	64.05	0.08	41	26.18	36	32.81	2.31	3	2.47	3	3.12	6.47	0	0.05	0	0.02	4.66
Hyb_S	155	71.60	101	65.87	0.07	53	24.69	46	30.05	1.94	7	3.24	5	3.86	8.21	1	0.47	0	0.22	15.00
Hyb_{I4}	178	76.05	130	74.39	0.06	47	20.17	39	22.40	1.97	7	3.12	5	3.04	10.59	1	0.67	0	0.16	15.01
Hyb^*_{I4}	167	75.03	120	72.81	0.06	47	21.11	39	23.82	2.06	7	3.20	5	3.17	10.50	1	0.67	0	0.20	13.89
using guided neighborhood explorations in VND																				
VNS	273	68.36	167	61.49	0.08	105	26.51	86	31.63	2.55	19	4.88	18	6.74	3.59	1	0.26	0	0.14	12.21
Hyb_B	111	71.03	68	63.50	0.10	42	27.01	36	34.03	2.63	3	1.94	2	2.47	4.99	0	0.02	0	0.01	5.29
Hyb_S	74	63.11	39	53.70	0.10	35	29.79	27	37.63	3.39	7	6.15	6	8.19	6.46	1	0.94	0	0.48	14.77
Hyb_{I4}	81	64.57	44	57.17	0.10	36	29.01	27	35.86	3.61	6	5.53	5	6.69	8.11	1	0.89	0	0.27	13.42
Hyb^*_{I4}	75	63.47	39	54.92	0.10	35	29.91	27	37.60	3.77	6	5.75	5	7.18	8.13	1	0.88	0	0.30	12.83

Table 3.7: Characteristics of the instances used for testing the exact methods.

instance	best input tree				initial tree			
	TR	UDD	WT	TR-WT	TR	UDD	WT	TR-WT
5x10_70	68.05	393	417	265.16	67.62	395	420	265.23
5x10_80	75.71	283	469	340.47	74.79	300	473	342.37
5x10_90	86.42	165	565	467.75	86.61	166	567	469.10
5x15_70	63.87	1232	1394	841.65	65.59	1156	1488	899.50
5x15_80	73.22	924	1683	1179.21	75.99	814	1871	1316.11
5x15_90	81.52	658	1938	1559.19	86.48	480	1982	1594.20
5x20_70	60.29	2829	3807	2244.75	65.14	2583	3806	2238.63
5x20_80	67.55	2208	4350	2961.32	73.92	2166	4441	3018.22
5x20_90	84.18	1227	4841	3930.53	84.86	1200	5006	4061.17
5x25_70	64.19	4540	7527	4684.92	66.79	4168	7849	4859.63
5x25_80	69.06	4266	8470	5570.38	73.98	3588	9005	5919.94
5x25_90	84.11	2094	10062	8272.29	87.41	1628	10242	8416.85
5x30_70	63.78	7729	14687	9156.35	68.46	7080	14990	9339.27
5x30_80	70.08	6395	15894	10767.20	76.21	5150	16500	11159.10
5x30_90	82.08	3875	18053	14603.30	86.62	2900	18508	14975.60
5x35_70	65.24	9930	22652	14325.10	68.54	9112	23427	14804.70
5x35_80	70.85	8602	22244	15584.60	76.28	7116	24527	17157.80
5x35_90	83.63	5240	28883	23365.20	86.06	4318	29514	23850.00
5x37_70	66.28	11574	29186	18730.20	68.03	10460	29477	18877.30
5x37_80	71.92	9482	29007	20300.40	75.33	8658	30216	21054.60
5x37_90	82.97	5756	34253	27873.20	85.82	4620	35191	28622.10
5x39_70	59.54	16394	32877	19195.30	67.17	13240	35915	20926.10
5x39_80	74.67	9810	35765	26245.30	78.18	8482	37226	27303.10
5x39_90	82.60	7686	38695	31431.90	87.37	5544	40248	32675.70

Finally, we will investigate the performance of the ILP-based exact solution methods. For this we created additional artificial instances which have a fewer number of taxa, ranging from 10 up to 39. The different scores of the best input tree and the initial tree used during creation of these instances are given in Table 3.7. The larger instances are used for testing the model utilizing the Weighted Triple score as well as the TreeRank based Weighted Triple score. These results are shown in Table 3.8 and 3.9, respectively. Therein the final solution value, the resulting percentage gap, and the CPU time in seconds is given. The last line states corresponding average values. In both cases using the model with lazy constraints yielded the best results, the standard model performs worst, and using heuristic pruning of variables leads to missing some improved solutions which also use triples not occurring in the input trees. As expected, pruning also leads to a significant decrease of runtime when applied to the

standard model, but this is even more true for using lazy constraints. Contrary, when pruning is applied in combination with lazy constraints the runtime increases again. Hence, reducing the search space is not necessarily beneficial, even with regard to runtime. The results of the more sophisticated ILP model with the UpDown distance as objective function are given in Table 3.10. Here we had to use rather small instances, since applying the standard model, even with lazy constraints, is quite soon very hard to solve. For this setting the pruning of variables seems mandatory. However, also when doing so 5 of 15 instances could not be solved to optimality within one hour.

3.12 Conclusions

In this chapter we tackled the consensus tree problem (CTP), mainly arising in the domain of phylogenetics and representing an important problem in bioinformatics. We described several tree similarity measures, as well as their advantages and drawbacks. Yet, the real practical applicability of some of them is an open question which cannot be dealt with in this work. On the heuristic side we introduced four meaningful neighborhood structures for the CTP, whereof Step and Rotate are adopted from approaches for deriving phylogenetic trees, SPRr is a restricted variant of SPR and Swap is a specific form of two Step moves and has not been described before. A previously presented evolutionary algorithm using the fine-grained TreeRank similarity measure has been extended to several memetic algorithm variants by embedding a randomized local search utilizing these neighborhood structures. Thereof the variant using the simple progressive search, which changes the neighborhoods over time, performs best. Furthermore, a variable neighborhood descent procedure that also uses these neighborhood structures has been developed and embedded in a variable neighborhood search. The latter performs shaking by applying an increasing number of random Step moves. The evaluation of neighborhoods could be substantially sped up by implementing a (partly) incremental scheme through updates of the UpDown matrix. Next, we hybridized the EA and MA with the VNS or the VND by running them either in sequential or intertwined order exchanging improved solutions. Finally, we also derived variants of the neighborhoods to be used in the VND where the exploration is done in an adaptive way via utilizing the UpDown instance to change "expensive" structures of the tree first.

In another line of research we investigated the application of ILP models for solving the problem. Since the TreeRank measure is non-linear we used two alternative linear measures: the Weighted Triple score and the UpDown distance. We proposed to consider some of the constraints as lazy constraints as well as to heuristically prune the set of variables (triples) in order to speed up the computation with no or only a small loss in solution quality.

The algorithm's performance has been evaluated with extensive tests on agglomerative clustering, scatter search, and carefully generated artificial instances providing more room for improvement. The MA utilizing SPS, the VNS, and the hybrid approaches are usually able to outperform the EA. On our real-world instances, the VNS turned out to be better than the MA, but on the artificial instances the situation was vice-versa. We explained this behavior with the VNS' stronger focus on intensification and weaker diversification abilities.

Table 3.8: Results of different ILP model settings on artificial instances using Weighted Triple score.

	TM			TM_p			TM_l			TM_{l+p}		
	WT	%-gap	t[s]	WT	%-gap	t[s]	WT	%-gap	t[s]	WT	%-gap	t[s]
5x25_70	8193	0.00	7.8	8193	0.00	3.6	8193	0.00	1.1	8193	0.00	1.1
5x25_80	9005	0.00	3.6	9005	0.00	2.1	9005	0.00	1.0	9005	0.00	0.8
5x25_90	10242	0.00	3.6	10242	0.00	1.0	10242	0.00	0.9	10242	0.00	0.4
5x30_70	15004	0.00	2149.9	15004	0.00	10.1	15004	0.00	35.6	15004	0.00	51.9
5x30_80	16624	0.00	8.7	16624	0.00	4.7	16624	0.00	2.1	16624	0.00	1.6
5x30_90	18508	0.00	8.1	18508	0.00	1.6	18508	0.00	2.1	18508	0.00	0.7
5x35_70	23532	0.00	58.9	23484	0.00	36.5	23532	0.00	9.5	23484	0.00	9.5
5x35_80	24551	0.00	22.2	24551	0.00	18.2	24551	0.00	4.6	24551	0.00	5.1
5x35_90	29645	0.00	16.2	29645	0.00	3.9	29645	0.00	4.0	29645	0.00	1.9
5x37_70	29235	2.33	3600.0	29836	0.00	1801.6	29897	0.00	252.9	29836	0.00	371.0
5x37_80	30246	0.00	86.7	29964	0.00	1579.8	30246	0.00	10.9	29964	0.00	230.6
5x37_90	35191	0.00	20.8	35191	0.00	5.0	35191	0.00	5.0	35191	0.00	2.5
5x39_70	35918	0.00	67.7	35918	0.00	39.3	35918	0.00	6.7	35918	0.00	11.1
5x39_80	37398	0.00	27.6	37398	0.00	35.1	37398	0.00	6.4	37398	0.00	7.7
5x39_90	40248	0.00	26.6	40248	0.00	16.3	40248	0.00	6.2	40248	0.00	5.3
avg	24236	0.15	407.8	24254	0.00	237.3	**24280**	0.00	23.3	24254	0.00	46.8

Table 3.9: Results of different ILP model settings on artificial instances using TreeRank based Weighted Triple score.

	TM_{TR}			TM_{TR+p}			TM_{TR+l}			TM_{TR+l+p}		
	TR-WT	%-gap	t[s]	TR-WT	%-gap	t[s]	TR-WT	%-gap	t[s]	TR-WT	%-gap	t[s]
5x25_70	5090.82	0.00	6.7	5090.82	0.00	3.1	5090.82	0.00	1.1	5090.82	0.00	0.9
5x25_80	5919.94	0.00	4.0	5919.94	0.00	2.1	5919.94	0.00	1.2	5919.94	0.00	1.0
5x25_90	8416.85	0.00	3.5	8416.85	0.00	1.0	8416.85	0.00	0.9	8416.85	0.00	0.4
5x30_70	9321.85	0.28	3600.0	9347.95	0.00	11.9	9347.95	0.00	77.4	9347.95	0.00	15.2
5x30_80	11240.00	0.00	11.6	11240.00	0.00	4.7	11240.00	0.00	2.1	11240.00	0.00	1.9
5x30_90	14975.60	0.00	8.1	14975.60	0.00	1.6	14975.60	0.00	2.1	14975.60	0.00	0.7
5x35_70	14870.90	0.00	165.5	14828.20	0.00	422.9	14870.90	0.00	12.9	14828.20	0.00	12.1
5x35_80	17177.00	0.00	76.9	17177.00	0.00	18.2	17177.00	0.00	4.8	17177.00	0.00	5.4
5x35_90	23962.70	0.00	16.5	23962.70	0.00	3.9	23962.70	0.00	3.9	23962.70	0.00	1.8
5x37_70	19127.00	0.18	3600.0	19120.20	0.00	2137.8	19151.90	0.00	101.1	19120.2	0.00	466.3
5x37_80	21082.40	0.00	63.6	20925.30	0.00	240.5	21082.40	0.00	7.7	20925.30	0.00	120.7
5x37_90	28622.10	0.00	20.8	28622.10	0.00	5.1	28622.10	0.00	4.9	28622.10	0.00	3.0
5x39_70	20928.40	0.00	85.2	20928.40	0.00	39.6	20928.40	0.00	7.3	20928.40	0.00	10.9
5x39_80	27432.60	0.00	31.2	27432.60	0.00	35.2	27432.60	0.00	6.3	27432.60	0.00	7.7
5x39_90	32675.70	0.00	26.3	32675.70	0.00	16.4	32675.70	0.00	6.2	32675.70	0.00	5.3
avg	17389.60	0.03	516.4	17377.60	0.00	196.3	**17393.00**	0.00	16.0	17377.60	0.00	43.6

Table 3.10: Results of different ILP model settings on artificial instances using UpDown distance.

	TUDDM			TUDDM$_p$			TUDDM$_l$			TUDDM$_{l+p}$		
	UDD	%-gap	t[s]	UDD	%-gap	t[s]	UDD	%-gap	t[s]	UDD	%-gap	t[s]
5x10_70	370	0.00	1910.6	370	0.00	3.7	370	0.00	162.9	370	0.00	5.4
5x10_80	275	0.00	1229.5	275	0.00	2.3	275	0.00	94.2	275	0.00	3.7
5x10_90	165	0.00	24.9	165	0.00	0.1	165	0.00	3.1	165	0.00	0.1
5x15_70	1232	18.85	3600.0	1232	14.24	3600.0	1232	17.68	3600.0	1118	0.00	3483.9
5x15_80	924	18.29	3600.0	796	0.00	22.8	924	16.99	3600.0	796	0.00	22.8
5x15_90	480	0.00	1644.8	480	0.00	0.4	480	0.00	1.1	480	0.00	0.3
5x20_70	2829	23.05	3600.0	2829	19.60	3600.0	2829	22.05	3600.0	2829	19.30	3600.0
5x20_80	2208	17.16	3600.0	2071	0.00	555.2	2208	15.60	3600.0	2071	0.00	780.6
5x20_90	1227	17.20	3600.0	1127	0.00	31.8	1227	15.92	3600.0	1127	0.00	56.3
5x25_70	4540	24.07	3600.0	4540	20.36	3600.0	4540	23.47	3600.0	4540	20.48	3600.0
5x25_80	4266	23.14	3600.0	4266	19.67	3600.0	4266	22.73	3600.0	4266	19.61	3600.0
5x25_90	2094	23.93	3600.0	1628	0.00	7.5	2094	23.47	3600.0	1628	0.00	4.6
5x30_70	7729	20.91	3600.0	7729	18.81	3600.0	7729	20.70	3600.0	7729	18.32	3600.0
5x30_80	6395	25.54	3600.0	6395	22.98	3600.0	6395	25.25	3600.0	6395	22.69	3600.0
5x30_90	3875	27.05	3600.0	2900	0.00	4.9	3875	26.78	3600.0	2900	0.00	13.0
avg	2574	15.95	2963.7	2454	7.71	1482.4	2574	15.38	2658.3	**2446**	6.69	1491.6

On the heuristic side the hybrid approaches generally perform best, clearly exploiting the benefits of the individual algorithms they combine. Especially the intertwined hybrids yield consistently excellent and in most cases the overall best solutions. This is still true when using the guided neighborhood explorations. Moreover, a slight improvement with regard to solution quality can be recognized.

When applying the exact ILP-based solution approaches it is consistently beneficial to use the proposed feature of lazy constraints, thereby often drastically reducing the required time to solve the model. In case of using the extended model with the UpDown distance as objective function to be minimized also pruning the considered variables, i.e. reducing the search space, is of benefit. Only with this setting it is possible to find an improved solution at all given the time limit of one hour.

Potential Future Work

Future work could include experiments on more problem instances with even more different properties. Promising also seems to be the investigation of intertwined hybrids with more sophisticated alteration schemes, in particular since we observed that in some cases the EA or MA is not able to improve the solution in later time-slots anymore. A further idea is to consider other neighborhood orders for the VND or even a (self-)adaptive strategy, which would probably nicely combine with the already proposed dynamic guided neighborhood exploration. Finally, a different, potentially suitable approach to follow is to incorporate the multilevel refinement strategy; a general overview is given in [230, 231] as well as an application in Section 5.4. It seems quite natural to either merge several taxa into so-called supertaxa or to fix certain subtrees. This would definitely speed up the neighborhood exploration and thus not only allow more (or even all) moves to be evaluated but also implicitly consider more powerful moves. Another interesting aspect is that multilevel refinement has to our knowledge not been applied to an optimization problem dealing with such tree structures before.

CHAPTER 4

PERIODIC VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

4.1 Introduction

The research efforts reported in the following were carried out while I was employed from January 2008 until September 2010 in a larger project entitled *Matheuristics: Hybrid Algorithms for Transportation Problems* funded by the Austrian Science Fund (FWF) under contract number P20342. Three research groups of two different universities were involved: Karl Dörner was the overall coordinator and local project leader at the Department of Business Administration, University of Vienna, my supervisor Günther Raidl was the local project leader on our side, and Walter Gutjahr was the local project leader at the Department of Statistics and Decision Support Systems, University of Vienna.

In this and the subsequent chapter we deal with a generalized variant of the classical *Vehicle Routing Problem* (VRP) where some customers must be served several times in a given planning period instead of only once on a single day. A large portion of our work is thus devoted to the *Periodic Vehicle Routing Problem with Time Windows* (PVRPTW), reported in this chapter, and to a lesser extent we tackle the *Periodic Vehicle Routing Problem* (PVRP), i.e. the former variant without time windows, in the next chapter. Hence our main focus lay on the PVRPTW as this was our task in the project. Such settings occur in many real-world applications as in courier services, grocery distribution, waste collection, or for various sorts of suppliers. However, although its practical applicability is evident, only few specific solution techniques have been described in the literature, which is especially true for the PVRPTW.

The basis forms the classical capacitated VRP which we informally introduced in Chapter 1. Here we already assume a rather general VRP variant to start with. It is defined on a complete graph $G = (V, A)$, where $V = \{v_0, v_1, \dots v_n\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in$

$V, i \neq j\}$ is the arc set. Vertex v_0 represents the depot at which are based m vehicles having capacities $Q_1, \ldots, Q_m$ and maximal daily working times $D_1, \ldots, D_m$. Each vertex of $V \setminus \{v_0\}$ corresponds to a customer and has an associated demand $q_i \geq 0$ as well as a service duration $d_i \geq 0$. For each arc $(v_i, v_j) \in A$ there are further given travel times or costs $c_{ij} \geq 0$. The first generalization is due to additionally specifying a time window $[e_i, l_i]$ per customer and the depot, where e_i and l_i are nonnegative integers and denote the earliest as well as latest beginning of the service, respectively. This yields the *VRP with time windows* (VRPTW) [38]. The second generalization concerns the extension to a planning horizon of t days: Each customer has defined a service frequency f_i and a set $C_i \subseteq \{T' \mid T' \subseteq T,\ |T'| = f_i\}$ of allowable combinations of visit days, thus a customer has to be visited periodically. The whole PVRPTW then consists of selecting a single visit combination per customer and designing (at most) m vehicle routes on each of the t days on G such that

(1) each route starts and ends at the depot,

(2) each customer i belongs to exactly f_i routes over the planning horizon,

(3) for each vehicle $k = 1, \ldots, m$, the total demand of each route does not exceed capacity limit Q_k, and its daily duration does not exceed the maximal daily working time D_k,

(4) the service at customer i begins in the interval $[e_i, l_i]$ and
every vehicle leaves the depot and returns to it in the interval $[e_0, l_0]$, and

(5) the total travel cost of all vehicles is minimized.

Arriving before e_i at a customer i implies a waiting time until this start of the time window (without further cost). Arriving later than l_i is not allowed, i.e. we assume hard time window constraints. In this work, we further assume a homogeneous vehicle fleet with $Q_1, \ldots, Q_m = Q$ and $D_1, \ldots, D_m = D$. This also holds for the test data we will consider later on. An exemplary solution of a PVRPTW instance involving 96 customers, six vehicles and a planning horizon of four days is displayed in Figure 4.1. At the first day (upper left) we highlight three customers having different visit frequencies, demanding one, two, and four visits. For the remaining days we also highlight the appearance of these customers.

In the following we will report on our work of solving the PVRPTW with metaheuristics, exact approaches as well as several hybrid variants composed out of these methods. Though the number of solution approaches might seem high, we, in fact, did not indent to develop a sheer amount of them, but instead they originated rather naturally in the course of our main project goal to investigate hybrid solution approaches. In Section 4.2 we refer to related work. A variable neighborhood search is reported in Section 4.4, a multiple variable neighborhood search in Section 4.5, and an evolutionary algorithm in Section 4.6. On the side of the exact methods we report on a column generation approach in Section 4.7 as well as on a branch-and-cut-and-price approach in Section 4.8. The hybrid variants are topic of Section 4.9. In general, previous experimental results are given after the corresponding method sections, though usually only in a summarized way alongside with a reference for further details. We wanted to omit presenting too many and often even outdated results. Final and

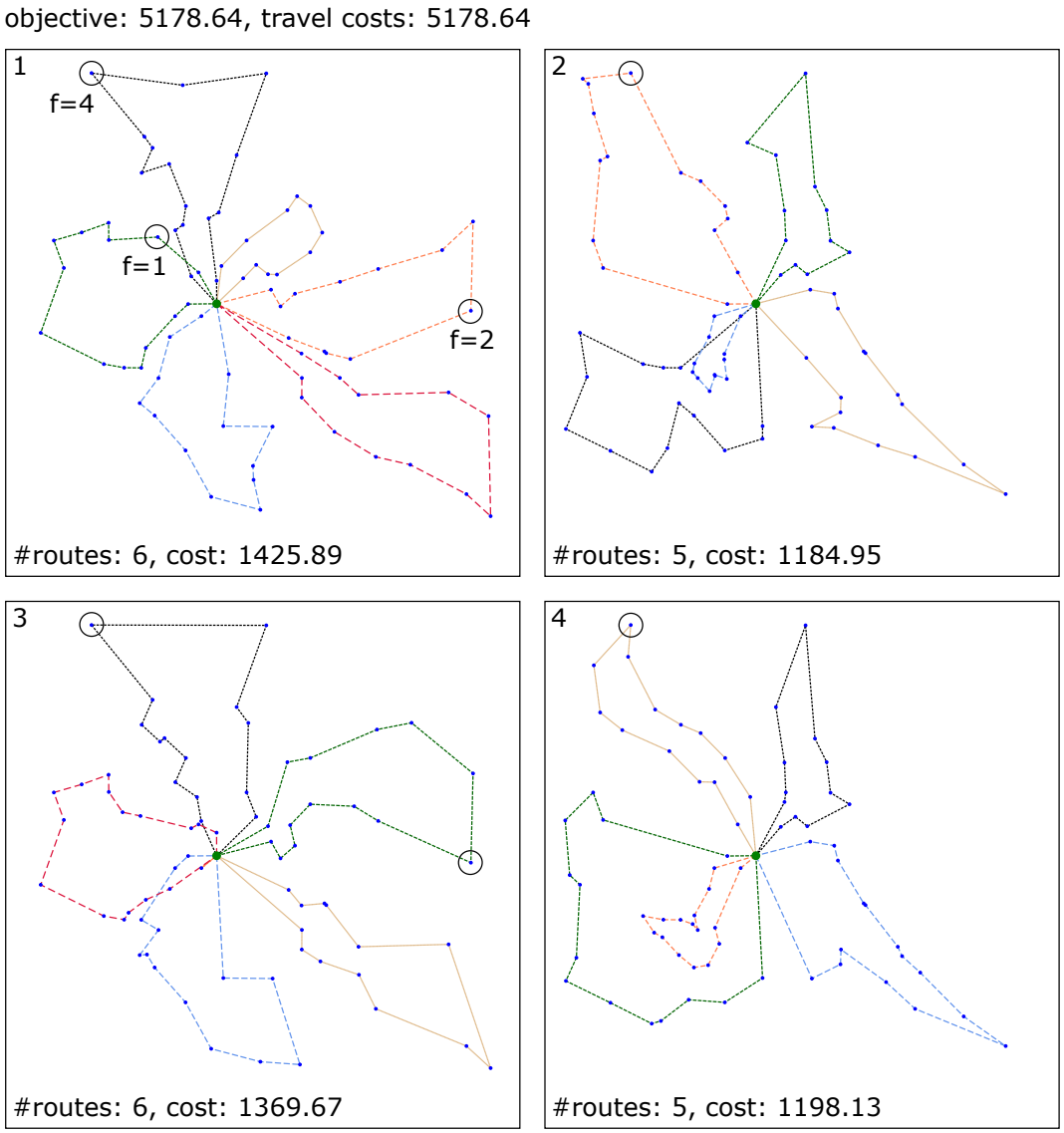


Figure 4.1: Exemplary solution of a PVRPTW instance with $n = 96$, $m = 6$, $t = 4$.

more detailed results of newly conducted test runs are given in Section 4.10, finishing with concluding remarks in Section 4.11.

Following parts of this work were presented before: the VNS at the *9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing* in 2008 [169], the column generation approach at the *International Network Optimization Conference 2009* [160], the VNS-ILP hybrid at the *8th Metaheuristic International Conference* in 2009 (MIC 2009) [159], the mVNS and the mVNS-ILP hybrid at the *6th International Workshop on Hybrid Metaheuristics* in 2009 (HM 2009) [161], as well as among other the evolutionary algorithm with the corre-

sponding hybrid variant at the *3rd International Workshop on Model-based Metaheuristic* in 2010 (Matheuristics 2010) [162].

4.2 Related Work

Although a large number of (meta-)heuristics have been described for the classical version of the VRP [39], and also the VRP with Time Windows (VRPTW) is covered to quite an extent [29], when we started the project the PVRPTW has to our knowledge previously been only dealt with in the works by Cordeau et al. [41, 42]. They developed a Tabu Search (TS) heuristic and applied it to the VRPTW and two generalizations, among those also the PVRPTW. For the PVRPTW the possible moves are to relocate a single customer to another route or to change its visit combination. Recently Cordeau and Maischberger [43] proposed a parallel iterated TS heuristic applicable to several routing problems, again also to the PVRPTW.

Related VNS metaheuristics were described by Polacek et al. [172] for the Multi-Depot VRPTW and by Hemmelmayr et al. [105] for the Periodic VRP. Our VNS for the PVRPTW combines, adapts, and extends concepts of these works for considering time windows and periodicity at the same time.

Evolutionary algorithms (EAs) for the VRPTW are subject of [28], applications to problem variants similar to the PVRPTW (e.g. the multi-depot case) can be found in [92]. Only very recently also the PVRPTW has been tackled with EAs. The first work is by Nguyen et al. [150] where a new, problem specific crossover is proposed and two metaheuristics, the TS of Cordeau et al. and our VNS, are applied to further improve ("educate") offspring solutions. Promising results are shown, though devoting quite excessive runtimes. A different EA based heuristic was proposed by Vidal et al. [227]. Their hybrid genetic search combines the explorativeness of genetic algorithms and the intensification via local search-based improvement procedures, effectively realizing a memetic algorithm. However, according to the results a large portion of the success of this method is due to the adaptive diversity management, which is seamlessly integrated into evaluating candidate solutions.

We are not aware of other exact or hybrid methods for the PVRPTW, yet similar PVRPs are dealt with in [82] and [146]. Also, very recently – in fact after we already completed our work – Baldacci et al. [12] proposed an exact solution approach for the PVRP based on a set-partitioning-like formulation which is basically similar to ours (in fact a stricter variant), introducing several relaxations of this formulation used to derive different bounding procedures. A more general survey of different PVRP variants and solution methods is given in [83]. An overview of exact approaches for the related VRPTW is given in [116]. A similar idea as one followed in this work was recently applied to a ready-mixed concrete delivery problem [206]. Our work also extends this by highlighting further aspects of this kind of hybridization. In a somewhat related work Danna and Le Pape [50] apply an ILP solver for deriving improved integer solutions during a branch-and-price procedure. In a recent survey Dörner and Schmid review matheuristic approaches applied to rich VRPs [65].

Although we do not explicitly consider parallelization in this work, one of our intertwined

approaches shares features with the replicated parallel VNS variant introduced among other approaches in [88] and also applied in [143]. Finally, also related to our work is [100], which evolved independently to ours, and presents a heuristic search based parallel VRP solver additionally making use of an ILP solver to obtain solutions to set covering problems. Though in their work Groër et al. tackled the VRP without any special constraints, they highlight the general applicability to richer VRPs as well.

4.3 Test Instances

Although it is usually common to give details about the test data rather at the end just before the computational results, we decided to put them here since we will present previous results after some methods.

Note that since each algorithm is implicitly designed to tackle instances of specific maximal size, we will not necessarily apply each algorithm on each instance set later on.

4.3.1 PVRPTW Instances from Cordeau et al.

The first PVRPTW instances were introduced in [41], originally proposed for the PVRP in [40] and extended with time windows. They are of type Euclidean, range from 48 to 288 customers, 3 to 20 homogeneous vehicles, and have a planning horizon of 4 or 6 days; more details are given in Table 4.1. Instances p01a–p10a and p01b–p10b have narrow and larger time windows, respectively. The customers are located at random around also randomly located clusters. For these type of instances no truncation/rounding is applied to the distances.

Results for comparison are reported in [41] and [150] without forward time slack, as well as in [42, 43, 227] with forward time slack.

4.3.2 Additional PVRPTW Instances Based on VRPTW Instances of Solomon

In the course of our work on the PVRPTW we further derived new instance sets from the Solomon VRPTW benchmark instances [215]. We proceeded by evenly assigning the possible visit combinations to all or only a subset of the customers at random.

We did so for the first five instances of type random (R), clustered (C), and mixed random and clustered (RC) for a planning horizon of four, six, and eight days, denoted by p4, p6, and p8, respectively. For p4 the customers need to be visited either one, two, or four times, for p6 either one, two, three, or six times, and for p8 either one, two, three, four, or eight times.

Following instance sets were derived: p4 with 36 customers, p4 with 50 customers, as well as p4, p6, and p8 with all 100 customers. In case less than 100 customers are used the actual number is given as a subscript; these smaller instances were specifically created for the exact approaches. During creation the number of vehicles m was altered (reduced) in such a way that few or none empty routes occur in feasible solutions, yet it is not too hard to find feasible

Table 4.1: Characteristics of Cordeau et al.'s test instances with best known solutions without and with forward time slack.

Id	n	m	t	D	Q	BKS	BKS^{FTS}
p01a	48	3	4	500	200	2989.58	2909.02
p02a	96	6	4	480	195	5107.51	5026.57
p03a	144	9	4	460	190	7158.77	7023.90
p04a	192	12	4	440	185	7981.85	7755.77
p05a	240	15	4	420	180	8584.35	8311.17
p06a	288	18	4	400	175	10935.60	10473.24
p07a	72	5	6	500	200	6892.71	6782.68
p08a	144	10	6	475	190	9751.66	9574.80
p09a	216	15	6	450	180	13707.30	13201.06
p10a	288	20	6	425	170	17754.20	16920.96
p01b	48	3	4	500	200	2284.83	2277.44
p02b	96	6	4	480	195	4141.15	4121.50
p03b	144	9	4	460	190	5567.15	5489.33
p04b	192	12	4	440	185	6471.74	6347.77
p05b	240	15	4	420	180	6963.11	6777.54
p06b	288	18	4	400	175	8855.97	8582.72
p07b	72	4	6	500	200	5509.08	5481.61
p08b	144	8	6	475	190	7677.68	7599.01
p09b	216	12	6	450	180	10874.80	10532.51
p10b	288	16	6	425	170	13851.40	13406.89

solutions quite early in the solution process. The capacity constraint ($Q = 200$) was mostly left untouched, except for p8 instances. Note that for these type of instances we truncate the distances to the first digit, e.g. as was done in [125]. The characteristics of the 100 customer instances along with the current best known solutions from [150] and mainly [227] are given in Table 4.2. Staying in line with [150, 227] we denote this whole 100 customer instance set as "Pirkwieser and Raidl".

4.3.3 Large-Scale PVRPTW Instances from Vidal et al. Based on Instances of Cordeau et al.

Recently Vidal et al. [227] also derived new, large-scale instances. They did so using the same creation procedure as Cordeau et al. [40, 41]. These instances range up to 960 customers, 58 vehicles, and a planning horizon of 12 days, thus being notably larger than previous instances; see Table 4.3 for more details. So far the only results are presented in [227], naturally without any comparison, which was an additional motivation to also consider them for testing in our work.

Table 4.2: Characteristics of new Solomon-based "Pirkwieser and Raidl" PVRPTW instances having 100 customers and a planning horizon of four, six, and eight days, plus the best known solutions used for calculating the gap.

Id	m	Q	BKS
p4r101	14	200	4082.0
p4r102	13	200	3724.3
p4r103	10	200	3153.1
p4r104	7	200	2566.0
p4r105	11	200	3638.9
p4c101	10	200	2907.4
p4c102	8	200	2882.9
p4c103	7	200	2734.5
p4c104	7	200	2419.0
p4c105	8	200	2884.1
p4rc101	10	200	3955.9
p4rc102	10	200	3755.7
p4rc103	8	200	3449.9
p4rc104	7	200	2991.5
p4rc105	11	200	3932.6
p6r101	14	200	5376.1
p6r102	12	200	5201.6
p6r103	9	200	3940.5
p6r104	8	200	3335.8
p6r105	9	200	4272.9
p6c101	7	200	3981.2
p6c102	7	200	3841.7
p6c103	6	200	3523.6
p6c104	6	200	3206.3
p6c105	7	200	4052.1
p6rc101	10	200	5781.5
p6rc102	9	200	5333.3
p6rc103	7	200	4273.1
p6rc104	7	200	4062.0
p6rc105	9	200	5227.1
p8r101	11	150	6471.3
p8r102	10	180	6097.9
p8r103	8	200	4687.0
p8r104	7	170	4355.8
p8r105	9	150	5473.2
p8c101	7	180	4679.1
p8c102	6	190	4933.3
p8c103	6	200	4664.0
p8c104	8	120	4591.6
p8c105	7	160	5134.2
p8rc101	9	150	6847.2
p8rc102	8	140	5763.3
p8rc103	7	145	5424.9
p8rc104	7	160	4929.5
p8rc105	9	135	6203.4

Table 4.3: Characteristics of Vidal et al.'s large-scale instances, also stating the (previously) best known solutions.

Id	n	m	t	BKS
p11a	360	24	4	20937.29
p12a	480	30	4	26483.68
p13a	600	38	4	31808.00
p14a	720	44	4	36954.39
p15a	840	50	4	41699.07
p16a	960	58	4	48375.16
p17a	360	22	6	28818.04
p18a	520	30	6	37385.82
p19a	700	38	6	48993.72
p20a	880	48	6	60144.66
p21a	420	22	12	54257.26
p22a	600	30	12	72978.33
p23a	780	38	12	90951.34
p24a	960	48	12	114712.30
p11b	360	18	4	15992.20
p12b	480	24	4	20753.17
p13b	600	30	4	24972.94
p14b	720	36	4	29790.14
p15b	840	48	4	41609.04
p16b	960	56	4	49470.50
p17b	360	18	6	22989.05
p18b	520	24	6	32093.04
p19b	700	32	6	42332.28
p20b	880	42	6	52863.23
p21b	420	18	12	43098.26
p22b	600	24	12	58814.76
p23b	780	30	12	74357.84
p24b	960	40	12	94395.56

4.4 Variable Neighborhood Search for the PVRPTW

The first metaheuristic we derive for the PVRPTW is a variable neighborhood search that incorporates, unites, and partly extends some ideas from other VNS methods reported for similar problems. The motivation for doing so were twofold: on the one hand, as reported in the previous section, VNS was successfully applied to other routing problems as well and we could benefit from these experiences, and on the other hand we needed a suitable candidate for our envisioned hybrid variants.

Below we report on the parts composing our VNS, the outline of the method is further given in Algorithm 10.

4.4.1 Penalized Cost Function

To help the VNS finding a good feasible solution we explicitly allow infeasible solutions during the search process, thus smoothing the search space, by relaxing conditions (3) and (4) of the problem definition. For a solution s, we denote the total travel cost by $c(s)$, the total violations of the load, duration, and time window constraints by $q(s)$, $d(s)$, and $w(s)$, respectively. While $q(s)$ and $d(s)$ are calculated on a route basis considering the values of Q_k and D_k, $w(s)$ is determined by $\sum_{i=1}^{n} \max\{0, a_i - l_i\}$, where a_i is the arrival time at customer i. The cost function is defined as

$$f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s)\,,$$

where α, β, and γ are positive weights which can be adapted in a dynamic way during the search. However, according to preliminary tests we settled on a common fixed value. If not stated otherwise we use a value of 100, which was also the case in [172]. Only for the results presented in Section 4.10 we use a different setting.

4.4.2 Initial Solution

Our initialization method is basically based on those introduced in [41], but with a few modifications. First we randomly choose a visit combination per customer, thus assigning customers to the days of the planning horizon. In case of Euclidean instances all customers are ordered according to the angle they make with the depot; ties are broken choosing the customer with the earlier center of its time window $(e_i + l_i)/2$ first. An arbitrary ordering can be used otherwise. Next a customer $j \in \{1, \ldots, n\}$ is chosen at random. At most m routes per day are then constructed by running the following procedure for each day of the planning horizon:

1. Set $k \leftarrow 1$.
2. For each customer $i = j, j+1, \ldots, n, 1, \ldots, j-1$ do:
   a) If the insertion of customer i into route k would result in the violation of load or duration constraints, set $k \leftarrow \min\{k+1, m\}$.
   b) Insert customer i into route k so as to minimize the increase of the cost function.

Using this procedure only the last route of each day might violate load or duration constraints, whereas all routes might violate time window constraints.

4.4.3 Shaking

In our VNS we make use of three different neighborhood structures utilized in the shaking phase. For each of these structures we define six moves with increasing maximal perturbation

Algorithm 10: VNS for the PVRPTW with initial solution x_{init}

$x_{best} \leftarrow x_{init}$; `// incumbent solution (though most likely infeasible at the beginning)`
$x_{vns} \leftarrow x_{init}$; `// solution utilized for shaking`
$k \leftarrow 1$; `// shaking strength`
initialize temperature to (average travel costs)/5
while *VNS stopping condition is not met* **do**
    while $k \neq k_{\max}$ **do**
        `// shaking:`
        Select $x' \in \mathcal{N}_k(x_{vns})$
        `// local search:`
        $x'' \leftarrow$ repeated first improving 2-opt neighbor of x'
        `// probably apply additional local search:`
        if *x'' lies in-between 2% of x and with small probability* **then**
            apply 2-opt* improvement on x''
        `// possibly update incumbent:`
        if *x'' is feasible and better than x_{best}* **then**
            `// additional local search:`
            apply 2-opt* improvement on x'' if not already done before
            if *x'' made infeasible* **then**
                revert previous local search
            $x_{best} \leftarrow x''$; `// set new incumbent`
            $k \leftarrow 1$; `// reset shaking`
        `// possibly update solution utilized by VNS:`
        if *(x'' is better than x_{vns}) or (x'' is worse than x_{vns} but accepted due to Metropolis criterion)* **then**
            $x_{vns} \leftarrow x''$
            $k \leftarrow 1$; `// reset shaking`
        else $k \leftarrow k + 1$; `// increase shaking`
        every 100^{th} iteration apply linear cooling
    `// finished VNS iteration`
    $k \leftarrow 1$; `// reset shaking`

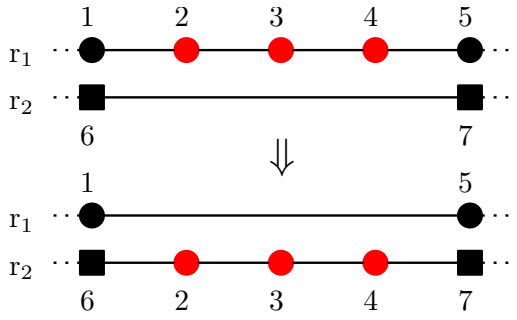


Figure 4.2: Exemplary segment move.

size δ, hence resulting in a total of 18 shaking neighborhoods. A move chooses the amount of change randomly in the interval $[1, \delta]$.

Change Visit Combination:

Change up to δ visit combinations randomly: Remove a customer from previous visit combination days and insert it in corresponding days of the new visit combination so as to minimize the increase of the cost function. It turned out to be beneficial to also allow "changing" the visit combination of customers offering only one combination, thus removing and inserting the customer on the same days. Due to the greedy insertion the latter operation can be regarded as a local search.

Move Segment:

Move a random segment of maximal length δ, if $\delta \in [1, 5]$, or bounded by the route size in case $\delta = 6$ from a route to another one, i.e. perform a customer relocation. Thereby reverse the segment with a small probability p_{rev} which is set to 0.1 according to preliminary tests. Such a move is shown in Figure 4.2.

Exchange Segments:

Exchange two random segments of varying maximal length (as in the previous move operator) between two routes, performing a CROSS exchange move. Reverse the segments with a small probability p_{rev}, occasionally performing an iCROSS exchange move [27]. Again, p_{rev} is set to 0.1. See Figure 4.3 for an example.

4.4.4 Shaking Neighborhood Order

According to findings in [105] we also use for our VNS a fixed neighborhood order of change combination, move segment, and finally exchange segments. The six moves of a specific

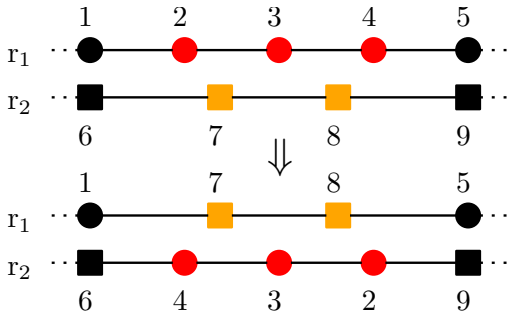


Figure 4.3: Exemplary segments exchange.

Table 4.4: Fixed shaking neighborhood order.

k	$\mathcal{N}_k$
1–6	*Change Visit Combination* up to $\delta = k$ times
7–11	*Move Segment* of maximal length $\delta = k - 6$
12	*Move Segment* of maximal length bounded by the route size
13–17	*Exchange Segments* of maximal length $\delta = k - 12$
18	*Exchange Segments* of maximal lengths bounded by corresponding route size

neighborhood structure are arranged in increasing order according to the perturbation size δ. This order is detailed in Table 4.4.

4.4.5 Local Search Procedures

We apply local search on each tour changed during the shaking phase. Thereby we make use of the well-known 2-opt neighborhood, where a single move corresponds to exchanging two edges within a tour, inverting a segment. The neighborhood is searched in lexicographic order and the search is repeatedly applied until no more improvement is possible. Extensive tests revealed that the best improvement variant clearly outperforms a first improvement strategy, both in terms of solution quality and runtime.

Additionally each new incumbent solution is subject to a 2-opt* inter-route exchange heuristic [173]. Hereby for each pair of routes of the same day all possible exchanges of the routes' end segments are tried; an exemplary application is depicted in Figure 4.4. Again, this local search iterates until no further improvement is possible, considering all days of the planning horizon. Contrary to before, for 2-opt* the first improvement version yielded slightly better results, hence only applying this setting. In later variants of the VNS, and for completeness also included in Algorithm 10, we further apply 2-opt* with a small probability to each

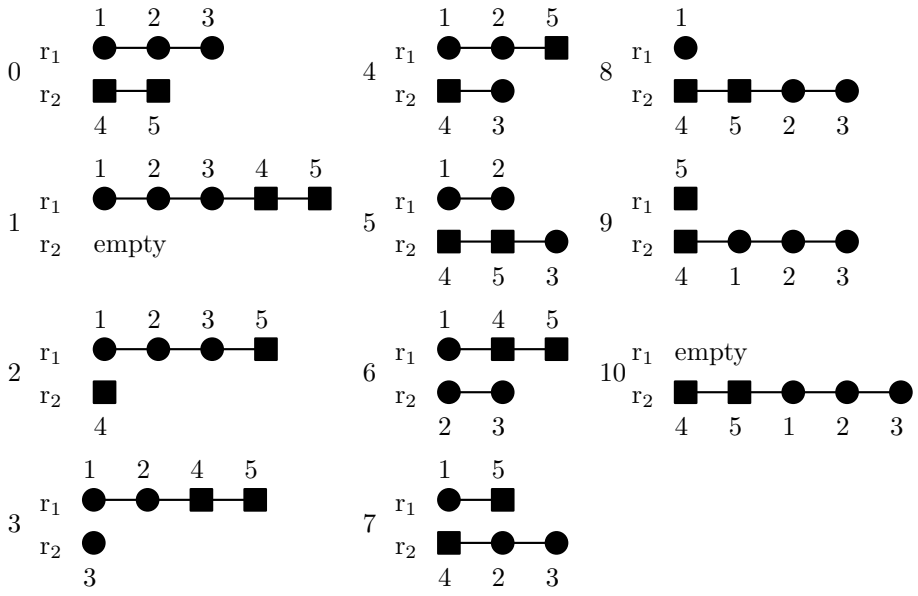


Figure 4.4: Exemplary application of the 2-opt* improvement procedure on routes r_1 and r_2.

newly derived solution lying within 2% to the current incumbent. The accurate values of this probability will be given in Section 4.10 since it depends on the instance set.

4.4.6 Acceptance Decision

To avoid that the VNS becomes too easily trapped in local optima, due to the cost function guiding towards feasible solutions and most likely complicating the escape of basins surrounded by infeasible solutions, we also allow to accept worse solutions under certain conditions. This is accomplished by utilizing a Metropolis criterion like in simulated annealing [123] for inferior solutions s'. They are accepted with a probability of $e^{-(f(s')-f(s))/T}$, depending on the cost difference to the actual solution s of the VNS process and the temperature T. We apply a linear cooling scheme and decrease T every τ_T iterations by an amount of $(T \cdot \tau_T)/\tau_{\max}$, where $\tau_{\max}$ denotes the maximal VNS iterations. Preliminary tests showed a satisfying performance over a wide range of instances when setting $\tau_T = 100$ and using an initial temperature value of $T_0 = 10$. Similar to the penalty weights only in Section 4.10 a different setting is applied.

4.4.7 Improved Route Evaluation

So far we implicitly assumed that each vehicle arrives as early as possible at the first customer j served by simply setting the waiting time at the depot to $\max\{e_0, e_j - c_{0j}\}$. An improved method involves the concept of forward time slack introduced in [205] for the VRPTW. This

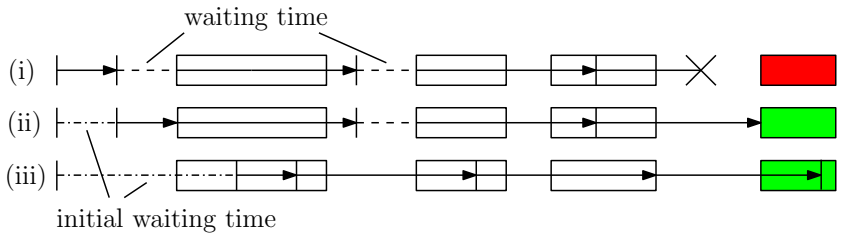


Figure 4.5: Different initial waiting times: (i) none at all, the route duration limit is exceeded before reaching the last customer (this setting is usually not used), (ii) set waiting time such that first customer is visited as early as possible (default setting), (iii) by considering *forward time slack* the initial waiting time is maximally increased to keep the route duration at a minimum.

was already applied to the PVRPTW by the TS heuristic [42], and later also used in [227]. The idea is to postpone leaving the depot as late as possible without increasing the time window violation, resulting in minimization of the route duration and probably rendering previously infeasible routes and thus solutions feasible. An example of this circumstance is shown in Figure 4.5. Even though this improved evaluation is only beneficial in case of a restriction on the route duration (of course coupled with the time windows). Further, its handling requires more computational effort. The concept of forward time slack will also be utilized for the exact approaches later on.

4.4.8 Previous Computational Results

First computational results were presented in [158]. There it was shown that 2-opt$*$ yields an improvement at nearly no computational cost, but note that at this time its application was restricted to new incumbent solutions only. Though a moderate iteration limit of 10^6 was used the VNS outperformed the TS of [41] as well as those in [42] utilizing the forward time slack. A larger improvement with regard to the best known solutions could be obtained without the forward time slack (-2.41% as opposed to -0.97%), i.e. compared to the results in [41]. Nevertheless, for both settings for all 20 instances considered (total of 40) the VNS yielded a better solution, i.e. a new best known solution.

More results as well as a comparison to the latest methods will be given in Section 4.10. Some more previous results will also appear in the following sections, since this “pure” VNS also represents a baseline for most of the other developed methods.

4.5 Multiple VNS

We extend the traditional VNS, which only has a single search trajectory, by considering multiple cooperating VNS instances performed in an intertwined way. Thus, our concern here is to investigate the possible benefits of a *sequential cooperative multistart search*. This new VNS variant is denoted as *multiple VNS* (mVNS).

Algorithm 11: Multiple VNS: $\#VNS$ refers to the number of VNS instances, $\#sec$ to the number of sections per VNS instance, and $iter_{max}$ is the total number of allowed iterations.

```
for i = 1 to #VNS do
    initialize VNS[i]
iter_sec ← ⌈iter_max/(#VNS · #sec)⌉
for sec = 1 to #sec do
    for i = 1 to #VNS do
        execute VNS[i] for iter_sec iterations
    x ← best solution of all VNS instances
    Replace solution of worst VNS instance by x
Return best solution of all VNS instances
```

Although it would be straight-forward to parallelize this approach, parallelization is not the issue we want to focus on here. A somewhat related approach is *replicated parallel VNS* [88, 143], in which multiple VNS instances are performed independently in parallel; the overall best solution is finally returned. In this case, the gain in performance is (almost) entirely due to the parallelization. In contrast, we aim at achieving better results within the same total CPU-time as required by a simple VNS run.

The multiple VNS algorithm is shown in Algorithm 11. We initialize each VNS instance independently by performing the method for creating a random solution 100 times and taking the best solution. This way each VNS instance most likely starts with a different initial solution. In the following the VNS instances are executed section-wise by setting an appropriate iteration limit given the total iteration limit and the number of sections. After each block of section-wise executions the actual best solution is determined and replaces the solution of the worst VNS instance. The latter is the cooperative part, where, considered locally, a worse performing VNS is supported by the best one, and seen from a global perspective, the search is intensified in the neighborhood of the so far best solution.

In some sense VNS instances can be said to be adaptively allocated to promising areas of the search space: If a solution is best after one iteration of the outer loop, one additional search trajectory is started from it. If the solution remains the incumbent over further iterations, more VNS instances are restarted from this point and a corresponding stronger intensification takes place. If, however, a new incumbent is found, no further VNS instances will be restarted from the previous one. Of course, in the unlucky event of a very captious local optimum this behavior could lead to a situation where all VNS instances are restarted from the same solution and no further progress is achieved, though this would be no worse than in the single VNS instance case.

4.5.1 Previous Computational Results

In [161] we compared the performance of the multiple VNS, as well as a matheuristic utilizing it (see Section 4.9.2), to those of the standard VNS. Back then we tested on our newly derived instances having a planning horizon of four and six days. In the following we present the comparison of the mVNS and the VNS in a concise way—averaging over the instances of a specific type—, extended by corresponding tests on the additional instances with an eight day planning horizon. We always did 30 runs per instance and VNS variant and determined how often the performance difference was statistically significant, using a Wilcoxon rank sum test with an error level of 5%. The runtimes are omitted since mVNS takes virtually the same time than VNS when executed with an equal iteration limit, in this case 10^6. Nevertheless, we also state the performance when compared to a VNS run with $2{\cdot}10^6$ iterations (which we mainly performed to compare to the matheuristics). In Table 4.5 we present results for mVNS running with 5, 8, 10, and 15 VNS instances, these are denoted by $\text{mVNS}_{\#VNS,\#sec}$, where $\#sec$ was here consistently set to 10 (as determined by preliminary tests). As can be seen multiple VNS is very often able to significantly outperform the standard VNS, ranging from 14 out of 15 times (93%) for p4 instances to 7 out of 15 times (46%) for p8 instances. Even when compared to the VNS allotted twice as many iterations the performance gain is notable for p4 and p6 instances. In this setting eight VNS instances ($\#VNS = 8$) seems to yield the best overall results. Though it is clear that choosing a suitable number depends on the instances at hand, especially on their size, but also on the number of sections. With an increasing length of the planning horizon $\#VNS$ should be rather small. This is not surprising since as the instance size gets larger in general (e.g. also more customers), so does the search space, and executing many VNS instances for rather few iterations is not beneficial anymore as the individual search attempts are not intensive enough. For further results we refer to Section 4.10 where mVNS will also be applied on other instances.

4.6 Evolutionary Algorithm

Although the following EA was designed already having the specific hybridization in mind, see Section 4.9.3, it achieves relatively good solutions on its own, but is clearly neither competitive to the VNS nor to the recently appeared EA-based algorithms. Further, it was in fact the first of its kind for periodic routing problems. Our intention was that the EA should mainly operate with whole (feasible) routes, in the sense that we assume to already have good routes at hand and more or less concentrate on solving the set covering aspect of the problem, of course also taking visit combinations properly into account. However it turned out very quickly that an appropriately combined repair/local search component is vital to also adequately adjust the routes.

The chromosomes are essentially represented as solutions of the VNS, i.e. in a direct way and not making use of any decoder. An array holds the currently selected visit combination per customer, and each day of the planning horizon holds an array of routes, where the latter directly represent the customer sequence. In accordance to the given fleet size we have m routes on each day, some of them might be empty in case less vehicles are in use.

Table 4.5: Concise results of mVNS on Pirkwieser and Raidl instances with a planning horizon of four, six and eight days, stating how often mVNS performs significantly better/worse than VNS.

Instances	$\text{mVNS}_{5,10}$	$\text{mVNS}_{8,10}$	$\text{mVNS}_{10,10}$	$\text{mVNS}_{15,10}$
significantly better/worse than VNS (10^6)				
p4r	4×/0×	5×/0×	4×/0×	4×/0×
p4c	5×/0×	5×/0×	5×/0×	5×/0×
p4rc	3×/0×	4×/0×	3×/0×	3×/0×
p6r	5×/0×	4×/0×	4×/0×	3×/0×
p6c	3×/0×	5×/0×	4×/0×	4×/0×
p6rc	3×/0×	2×/0×	2×/0×	0×/0×
p8r	2×/0×	2×/0×	3×/1×	0×/2×
p8c	2×/0×	1×/0×	3×/0×	2×/1×
p8rc	2×/0×	1×/0×	1×/1×	1×/2×
	29×/0×	29×/0×	29×/2×	22×/5×
significantly better/worse than VNS ($2 \cdot 10^6$)				
p4r	3×/0×	4×/0×	4×/0×	3×/0×
p4c	2×/0×	5×/0×	3×/0×	4×/0×
p4rc	2×/1×	3×/0×	2×/0×	0×/0×
p6r	3×/0×	1×/0×	2×/0×	2×/0×
p6c	1×/0×	4×/0×	2×/0×	2×/0×
p6rc	1×/0×	1×/1×	0×/3×	0×/3×
p8r	1×/1×	0×/2×	0×/2×	0×/3×
p8c	0×/3×	0×/3×	0×/3×	0×/2×
p8rc	1×/2×	1×/4×	1×/4×	1×/4×
	14×/7×	19×/10×	14×/12×	12×/12×

Algorithm 12: Evolutionary Algorithm for the PVRPTW

```
1  initialize population pop
2  repeat
3      p_1 ← binary tournament on pop
4      p_2 ← binary tournament on pop
5      o ← recombine(p_1, p_2)
6      o ← mutate(o)
7      o ← adjustVisitCombinations(o)
8      o ← removeRedundantCustomers(o)
9      o ← 2-opt(o)
10     o ← addMissingCustomers(o)
11     o ← 2-opt(o)
12     if o is new incumbent solution then
13         o ← 2-opt*(o)
14 until stopping criteria not met
```

The initial population of the stand-alone EA is created by repeatedly applying the initialization procedure also used for the VNS as described in Section 4.4. Also the same penalized objective function is used, however, this time with penalty weights of 1000 to almost enforce the selection of feasible solutions only.

For recombination we have three different operators affecting different aspects:

- apply recombination solely on the visit combinations,
- apply recombination solely on routes considering whole days as one unit (*building block*),
- or a more fine-grained variant of the latter applying recombination solely on routes considering single days' routes.

In preliminary tests we evaluated the performance of different standard crossover strategies, involving one-point, two-point, and uniform crossover. The latter yielded the best results, so in each case an uniform crossover is used. Note that during crossover the routes itself are not changed, so the resulting offspring solution might visit some customers more than once at a day and/or might visit them not at all.

Another vital component is the mutation operator, which might cause several possible changes:

- removal of a route selected at random,
- swapping of two routes of differing days,
- greedy insertion of a random customer on a random day,
- removal of one occurrence of a random customer,

- or changing one visit combination.

Each newly derived chromosome is subject to a procedure which eventually adjusts the visit combinations according to the routes, i.e. the visit combinations are independently chosen s.t. the least under-covering occurs, breaking ties w.r.t. the least over-covering. Though this is not applied if recombining the visit combinations, also visit combinations changed during mutation are left out. This adjustment step is to facilitate the set covering point of view, i.e. to first change the visit combinations in accordance to the routes and then vice versa. Hence afterwards, over-covering is tackled via removing redundant customers in a random sequence, followed by a 2-opt intra-route improvement on the altered routes. Finally, missing customers are added in a greedy way and subsequently 2-opt improvement is applied again. Adding missing customers is the most critical part potentially rendering a solution infeasible, most likely because of a time window violation. Due to this we initially adjust the visit combinations to reduce the amount of necessary insertions. Similar to the VNS each new incumbent solution is subject to the mentioned 2-opt* procedure. An outline of the EA is shown in Algorithm 12.

The EA applies a steady-state reproduction with a population of 100 individuals, using binary tournament selection with replacement, and accepting no duplicates (based on the objective function). All different recombination as well as mutation operators are applied with equal probability.

Since the EA is basically only compared to a matheuristic utilizing it, the combined results are given in Section 4.9.3.

4.7 Column Generation Approach for the PVRPTW

Among the most successful solution approaches for VRPs are algorithms based on column generation [63], where the initial basis is a restricted master problem gradually enriched by new columns by iteratively solving pricing subproblems. Therefore we focus on an ILP formulation suitable for such an approach. The master problem and the pricing subproblem are presented in Section 4.7.1 and 4.7.2, respectively. A schematic diagram of this process is shown in Figure 4.6.

4.7.1 Set-Covering Master Problem

We formulate the *integer master problem* (IMP) for the PVRPTW as a set-covering model, since such an approach also led to strong bounds in case of the VRPTW [26]:

$$\min \sum_{\tau \in T} \sum_{\omega \in \Omega} \gamma_\omega \, \chi_{\omega\tau} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{r \in C_i} y_{ir} \geq 1 \qquad \forall i \in V_C \tag{4.2}$$

$$\sum_{\omega \in \Omega} \chi_{\omega\tau} \leq m \qquad \forall \tau \in T \tag{4.3}$$

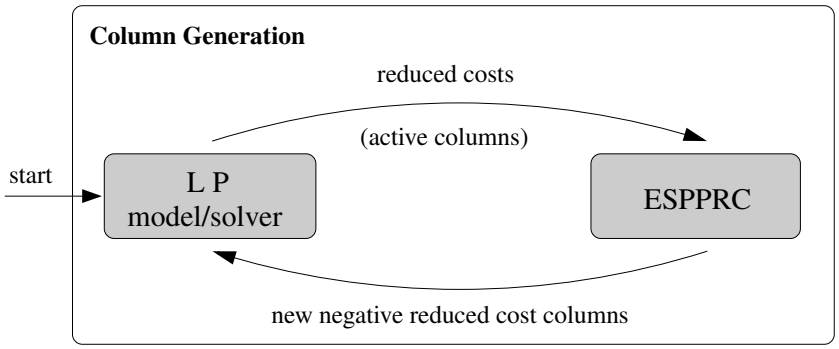


Figure 4.6: Information exchange between LP master problem and ESPPRC pricing subproblem.

$$\sum_{\omega \in \Omega} \alpha_{i\omega}\, \chi_{\omega\tau} - \sum_{r \in C_i} \beta_{ir\tau}\, y_{ir} \geq 0 \qquad \forall i \in V_C;\, \forall \tau \in T \tag{4.4}$$

$$y_{ir} \in \{0,1\} \qquad \forall i \in V_C;\, \forall r \in C_i \tag{4.5}$$

$$\chi_{\omega\tau} \in \{0,1\} \qquad \forall \omega \in \Omega;\, \forall \tau \in T \tag{4.6}$$

The set of all feasible routes, which grows exponentially with the number of customers, is denoted by Ω, and for each route $\omega \in \Omega$ its cost is γ_ω and $\chi_{\omega\tau}$ is the number of times route ω is selected on day τ. For each customer $i \in V_C$, variables y_{ir} indicate whether or not visit combination $r \in C_i$ is chosen. Following constraints are used: Cover constraints (4.2) guarantee that at least one visit day combination is selected per customer, fleet constraints (4.3) restrict the number of daily routes to not exceed the available vehicles m, and finally visit constraints (4.4) link the routes and the visit combinations, whereas $\alpha_{i\omega}$ and $\beta_{ir\tau}$ are binary constants and indicate whether route ω visits customer i and if day τ belongs to visit combination $r \in C_i$ of customer i, respectively. In the following we want to derive a good lower bound for the IMP by solving its LP relaxation. Therefore conditions (4.5) and (4.6) are replaced by $y_{ir} \geq 0$ and $\chi_{\omega\tau} \geq 0$, yielding the (linear) *master problem* (MP). Due to the exponential number of variables (columns) corresponding to routes, this LP cannot be solved directly. Instead, we restrict ourself to a small number of initial columns $\Omega' \subset \Omega$. The corresponding LP is referred to as *restricted master problem* (RMP). Additional columns (routes) that are able to improve the current LP solution are then generated by iteratively solving the so-called pricing subproblem.

Initial Column Set

An obvious way to start the column generation process is to populate the set of initial columns Ω' by the routes of at least one feasible primal solution, e.g. provided by a short run of the VNS or even only the construction heuristic. Of course this extra computation is only necessary in case of a rather limited fleet size (which is the case for all considered instances). As a side-note, suppose the fleet size equals or exceeds the number of customers.

Then we could simply insert a depot-customer-depot route for each customer and guarantee a feasible RMP.

However, if deriving initial columns is either not possible or not desired one can utilize slack variables—one per customer to be visited—and can thus start with an empty set of columns ($\Omega' = \emptyset$). These slack variables need to be added to constraints (4.4)

$$\sum_{\omega \in \Omega} \alpha_{i\omega}\, \chi_{\omega\tau} - \sum_{r \in C_i} \beta_{ir\tau}\, y_{ir} + s_i \geq 0 \qquad \forall i \in V_C;\, \forall \tau \in T$$

as well as with a penalty to the objective function (4.1)

$$\min \sum_{\tau \in T} \sum_{\omega \in \Omega} \gamma_\omega\, \chi_{\omega\tau} + \sum_{i \in V_C} s_i M\ .$$

The penalty M should be high enough such that it is still profitable to actually visit the customers.

4.7.2 Pricing Subproblem

In the pricing subproblem we search for a column (route) which potentially improves the LP relaxation of the RMP. Since it is a "new" column it implies that it was not considered before, i.e. belonging to the set $\Omega \setminus \Omega'$. Solving the problem for day $\tau \in T$ basically amounts to finding a route minimizing the *reduced costs*:

$$\min_{\omega \in \Omega} \gamma_\omega - \rho_\tau - \sum_{i \in V_C} \alpha_{i\omega} \pi_{i\tau}$$

and checking if this cost value is below zero, with ρ_τ and $\pi_{i\tau}$ being the corresponding dual variable values of constraints (4.3) and (4.4), respectively. In case these costs are greater or equal than zero then no potentially improving column could be found for day τ.

In order to formally state the detailed pricing subproblem we need to project the so far high-level representation of a route to the arc level. Hence following pricing subproblem holds for each day $\tau \in T$ and is solved on the auxiliary graph $G' = (V', A')$, with $V' = V \cup \{v_{n+1}\}$ and $A' = \{(v_0, i), (i, v_{n+1}) : i \in V_C\} \cup \{(i, j) : i, j \in V_C,\, i \neq j\}$, where v_{n+1} is a copy of the (starting) depot v_0 and acts as target node, we further assume a homogeneous fleet of vehicles:

$$\min \sum_{i \in V'} \sum_{j \in V'} \hat{c}_{ij\tau}\, x_{ij} \tag{4.7}$$

$$\text{s.t.} \quad \sum_{j \in V_C} x_{0j} = 1 \tag{4.8}$$

$$\sum_{i \in V'} x_{ik} - \sum_{j \in V'} x_{kj} = 0 \qquad \forall k \in V_C \tag{4.9}$$

$$\sum_{i \in V_C} x_{i,n+1} = 1 \tag{4.10}$$

$$\sum_{i \in V_C} \sum_{j \in V'} q_i \, x_{ij} \leq Q \tag{4.11}$$

$$a_{n+1} - w_0 \leq D \tag{4.12}$$

$$a_i + w_i + d_i + c_{ij} - M_{ij}(1 - x_{ij}) \leq a_j \qquad \forall (i,j) \in A' \tag{4.13}$$

$$e_i \leq (a_i + w_i) \leq l_i \qquad \forall i \in V' \tag{4.14}$$

$$w_i \geq 0 \qquad \forall i \in V' \tag{4.15}$$

$$a_i \geq 0 \qquad \forall i \in V' \setminus \{v_0\} \tag{4.16}$$

$$a_0 = 0 \tag{4.17}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A' \tag{4.18}$$

Variables x_{ij}, $\forall (i,j) \in A'$ denote which arcs from A' are used, and $\hat{c}_{ij\tau}$ is the *reduced cost* of using arc (i,j) on day τ:

$$\hat{c}_{ij\tau} = \begin{cases} c_{ij} - \rho_\tau & \text{if } i = v_0,\ j \in V_C \\ c_{ij} - \pi_{i\tau} & \text{if } i \in V_C,\ j \in V' \end{cases}.$$

Constraints (4.8)–(4.10) are the flow constraints, (4.11) and (4.12) guarantee feasibility regarding capacity and duration constraints, respectively. Finally, (4.13) and (4.14) are time constraints, with variable a_i denoting the arrival time at customer i and w_i being the waiting time occurring after this visit.

This pricing subproblem resembles a *shortest path problem with resource constraints* (SPPRC) [112]. Regarding the quality of the theoretically obtainable lower bound it is beneficial to restrict the search to elementary paths, hence only considering the *elementary SPPRC* (ESPPRC). A drawback, especially from a computational point of view, is that this condition renders the problem $\mathcal{NP}$ hard, whereas the SPPRC is solvable in pseudo-polynomial time. However, because of the better bounds we decided to go with the ESPPRC.

Exact Label Correcting Algorithm

The ESPPRC subproblem is solved by a dynamic programming approach based on [74, 31]. We use a label correcting algorithm and expand the partial paths from the depot v_0 to the target node v_{n+1}, thereby retaining only non-dominated labels. We will give a short outline of this method next, according to [112]. In the course of the algorithm two sets of paths, the unprocessed and useful paths, are dynamically changed. One usually starts with a single unprocessed path containing only the start depot. During the path extension step the next unprocessed path is selected and all feasible one-node extensions are created and added to the set of unprocessed paths, whereas the extended path itself is removed from it and added to the set of useful paths. Having a lot and/or tight constraints already rules out a lot of the extensions here, e.g. for the ESPPRC visiting a node twice is also not allowed. In a second step the algorithm applies the dominance rule(s) to reduce both sets to limit the number of necessary extension steps. It only retains the best (in fact the Pareto-optimal) paths and guarantees finding an optimal solution to the ESPPRC. Though one can trade optimality for less computational effort as we will see a bit later.

In the following we describe the label and the dominance criteria suitable for our problem setting in more detail, which are an important part of the whole algorithm. We are faced both with the existence of time windows and restrictions on route duration. To our knowledge this combination was not part of any work in the context of this algorithm before, though it is highly relevant in practice. Due to these constraints it is a non-trivial task to find non-dominated paths, since on the one hand partial paths arriving earlier might be beneficial regarding following time windows, whereas on the other hand partial paths arriving later might be able to reach more customers afterwards.

To minimize route duration we adhere to the concept of *forward time slack*, see Section 4.4.7, and maximize the initial waiting time w_0 at the (start) depot without introducing a time window violation. When building a path we need to determine the minimum of this time slack, also used to calculate the minimal route duration.

A label associated with a partial path p at node v_i holds the following resource information: accumulated cost $\mathcal{C}_i$, load $\mathcal{L}_i$, overall waiting time $\mathcal{W}_i$, as well as the arrival time $\mathcal{A}_i$, the actual minimal forward time slack $\mathcal{F}_i$, and a set $\overline{V_i}(p)$ containing already visited nodes and those unreachable due to the numerous restrictions. Having this information, we define two more resources which are calculated on-the-fly: the start of service time $\mathcal{S}_i = \max\{\mathcal{A}_i, e_i\}$ and the current minimal route duration $\mathcal{D}_i = \mathcal{S}_i - \min\{\mathcal{F}_i, \mathcal{W}_i\}$.

When moving from node v_i to node v_j the label resources are updated as follows:

- costs: $\mathcal{C}_j = \mathcal{C}_i + c_{ij}$
- load: $\mathcal{L}_j = \mathcal{L}_i + q_i$
- arrival time: $\mathcal{A}_j = \mathcal{A}_i + w_i + c_{ij} + d_i$
- waiting time: $\mathcal{W}_j = \mathcal{W}_i + \max\{0, e_j - \mathcal{A}_j\}$
- forward slack time: $\mathcal{F}_j = \min\{\mathcal{F}_i, \mathcal{W}_j + (l_j - \mathcal{S}_j)\}$

When a path reaches the target node v_{n+1} the initial waiting time at the depot is set to $w_0 = \mathcal{F}_{n+1}$ in order to yield the overall minimal route duration $\mathcal{D}_{n+1}$.

Finally, based on these resources we define three different dominance rules stating whenever partial path p^1 dominates partial path p^2, both ending at the same node v_i:

R1: $\mathcal{C}_i^1 \leq \mathcal{C}_i^2 \ \wedge \ \mathcal{D}_i^1 \leq \mathcal{D}_i^2 \ \wedge \ \mathcal{L}_i^1 \leq \mathcal{L}_i^2 \ \wedge \ \mathcal{A}_i^1 \leq \mathcal{A}_i^2$

R2: $\mathcal{C}_i^1 \leq \mathcal{C}_i^2 \ \wedge \ \mathcal{D}_i^1 \leq \mathcal{D}_i^2 \ \wedge \ \mathcal{L}_i^1 \leq \mathcal{L}_i^2 \ \wedge \ \mathcal{A}_i^1 \leq \mathcal{A}_i^2 \ \wedge \ \overline{V_i}(p^1) \subseteq \overline{V_i}(p^2)$

R3: $\mathcal{C}_i^1 \leq \mathcal{C}_i^2 \ \wedge \ \mathcal{D}_i^1 \leq \mathcal{D}_i^2 \ \wedge \ \mathcal{L}_i^1 \leq \mathcal{L}_i^2 \ \wedge \ \mathcal{A}_i^1 \leq \mathcal{A}_i^2 \ \wedge \ \overline{V_i}(p^1) \subseteq \overline{V_i}(p^2) \ \wedge \ \mathcal{F}_i^1 \geq \mathcal{F}_i^2$

A dominated partial path is discarded from further consideration. Dominance rule R1 is quite simple and fast, a similar one was used in [31]. However, it sometimes also filters out (near) optimal least cost paths. Contrary to rule R1, rules R2 and R3 are more refined and also consider sets $\overline{V_i}(p^1)$ and $\overline{V_i}(p^2)$. Especially, rule R3 additionally takes the actual forward time slack into account, which is the critical resource connecting the arrival time and the

duration. Due to rules R1 and R2 being too relaxed and therefore of heuristic nature, rule R3 must be applied at last to guarantee finding all least cost paths, since it is the only rule (implicitly) incorporating all resources. In order to decrease computation time we apply the following sequential dominance rule scheme:

(a) use rule R1 as long as more than 100 new columns could be found, else switch permanently to (b)

(b) use rule R2 until no new columns could be found, if time windows and limited route duration then switch to (c); terminate the column generation otherwise,

(c) finally use rule R3, returning to (b) in the next run whenever new columns could be found; terminate the column generation otherwise.

Optionally the dynamic programming algorithm can be stopped after a certain number of negative cost paths have been found, i.e. applying a *forced early stop* [128]. For this, one simply keeps track of the number of paths which are extended to the end depot, and prematurely halts the execution if it exceeds the desired number. Although the optimal path will most likely be missed (especially at the first applications when a lot paths are found), some good-enough paths are obtained after a short time.

Heuristic Pricing Algorithms

We further propose two heuristics to generate new columns having negative reduced costs. Both rely on a local search framework offering following neighborhood moves operating on a single path each:

- insert a new customer in the path,
- delete a visited customer from the path,
- move a visited customer to another position within the path,
- replace a visited customer with another not yet visited one, and
- exchange/swap two visited customers.

The first method is actually a metaheuristic which can be regarded a greedy randomized adaptive search procedure (GRASP) [191]: In each iteration we start with a virtually empty path (v_0, v_{n+1}) with zero costs and successively try to append arcs with negative cost such that the feasibility of the path is maintained, always appending to the currently last node before v_{n+1}. In case several arcs are available to append we always select one at random. Afterwards we apply up to ten random moves out of the set of neighborhood moves described above. Finally, we perform a local search also based on these moves, applying them in a random fashion and always accepting the first improving change. This setting turned out to yield better results on a large set of instances on average than using a strict order and/or a

best improvement strategy. Whenever an iteration results in a negative cost path it is stored and returned at the end of the heuristic, thereby avoiding to accept duplicate paths.

The second heuristic is quite similar, but instead of generating paths from scratch it exploits currently active routes in the master problem as initial routes. These paths also initially have a reduced cost of zero. As for the GRASP-based metaheuristic the method subsequently applies random perturbation and first improvement local search. This variant is denoted as REUSE heuristic. A similar idea was also successfully applied in combination with a tabu search in [62].

Hybrid Pricing Algorithms

It is a natural extension to consider some form of hybridization of the exact and the heuristic pricing algorithms which were presented before. An obvious approach is a pure sequential hybrid: start with one of the heuristic methods and let it run as long as it yields satisfying results (which is to be defined), then switch to the exact dynamic programming based algorithm, which is itself also applied in various phases.

Speeding Up the Column Generation

As soon as new columns are generated for one of the daily subproblems they can be inserted for all days and the RMP is re-solved. This strategy leads to a substantial speed-up compared to only inserting the columns in the corresponding day. In the following iteration the same daily subproblem is solved again. This process continues until a full iteration over all days yields no new columns.

4.7.3 Computational Results

The first set of test instances for the column generation approach was taken from Cordeau et al., described in Section 4.3.1. We also reduced some of them by selecting only a random subset of the customers and appropriately adapting the number of vehicles; in this case we give a subscript denoting the index of the reduced instance. For this tests the initial set of columns is provided by taking the routes of feasible solutions of the VNS. The algorithms have been implemented in C++, compiled with GCC 4.1 and executed on a single core of a 2.2 GHz Dual-Core AMD Opteron 2214 PC with 4 GB RAM. Ilog CPLEX 11.2 was used as LP solver. The ESPPRC subproblem is solved in four ways:

(i) dynamic programming (DP),

(ii) dynamic programming with forced early stop after generating more than 1000 columns (DP^S), and two hybrid methods composed of

(iii) the GRASP-based metaheuristic applied for at most 10000 iterations—in case it did not find new columns in the first 1000 iterations—and switching to DP^S if either less than 100 new columns could be found or the number of new columns decreased in the last five iterations (GRASP+DP^S),

(iv) as well as similar to (iii) but applying the REUSE heuristic instead (REUSE+DP^S).

The newly generated routes are always inserted in all days. For the hybrid non-deterministic variants we performed 10 runs on each instance.

In Table 4.6 we state the instances, the initially provided upper bounds by the VNS (UB_{VNS}), the derived lower bounds (LB), the percentage gaps between them, i.e. %-gap $= (UB - LB)/LB \cdot 100\%$, the CPU times of settings DP and DP^S, as well as the minimal and median times of setting GRASP+DP and REUSE+DP over 10 runs. It can be observed that applying a forced early stop (DP^S) is in general faster than using none, especially for instances with narrow time windows where the runtime is often halved. Using the hybrid variants is beneficial for all instances but $2b_{r1}$, showing only a small differene though. Thus it seems that the faster heuristically generated columns outweigh the probably higher quality columns of the DP algorithm. This difference is more obvious for instances having narrow time windows (1a–8a). Of both hybrid variants the one using the REUSE heuristic is almost always faster and therefore should be the method of choice. The resulting gaps are clearly smaller for instances with a period of four days, though this is to some part also affected by the method providing the initial solutions.

As second and third set of test instances we took those with a planning horizon of four days (p4) either having 50 or all 100 customers of the PVRPTW instances we created out of the Solomon VRPTW instances, see Section 4.3.2. For these tests we start with an empty set of initial columns, hence introducing the slack variables into the model. Note that due to this we cannot state a resulting percentage gap. Here we further always apply the forced early stop, yet already after generating more than 100 new columns, which is of course a speedup in the short term, but also turned out to be beneficial as a whole. Additionally a different setting than before is to insert the generated columns for the corresponding day only, denoted as DP^S_{1d}. The results with 50 customers are reported in Table 4.7, those with 100 customers in Table 4.8. Whenever a "–" is reported instead of a runtime the run could not be finished within 15 hours and was stopped. As was expected it is always clearly better to insert the generated routes for all days, otherwise often consuming drastically more runtime or even reaching the time limit. So apart from DP^S_{1d} all settings yield a solution in time. Concerning the performance when additionally using heuristic column generation, it is observable that for 50 customers this sometimes leads to an overhead and therefore solely applying the label correcting algorithm (with the different dominance rules though) is often still better, in fact the ratio is 7:7 and one draw. Looking at the performance on the 100 customer instances though, the hybrid variants clearly yield better results, this time for each instance. For both instance sets the hybrid variant using the REUSE heuristic outperforms the one with the GRASP based method, again performing best of all settings. The improvements are further not dependent on the instance type. Finally, we observed that when using either of the pricing heuristics the actual number of slack variables in use decreases much faster as when starting with the labeling algorithm, and only after a few iterations no slack variables are active anymore. Moreover, the good performance of the heuristics is not influenced by the reduced costs, whereas the labeling algorithm is quite sensitive to them and often takes notably longer.

Table 4.6: Experimental results of different subproblem algorithm settings on (reduced) instances of Cordeau et al.

Instance				UB_{VNS}	LB	%-gap	DP	DP^S	GRASP+DP^S		REUSE+DP^S	
No.	n	m	t				t[s]	t[s]	min t[s]	med t[s]	min t[s]	med t[s]
1a	48	3	4	2909.02	2882.01	0.94	11.2	**6.0**	4.7	6.7	5.7	**6.0**
2a	96	6	4	5032.06	4993.48	0.77	129.2	61.8	46.5	50.8	34.8	**41.3**
3a	144	9	4	7138.65	6841.44	4.34	799.6	432.3	353.8	374.1	269.5	**325.4**
$4a_{r1}$	160	10	4	6929.84	6641.67	4.34	2086.2	1517.6	748	908.2	683.1	**866.0**
7a	72	5	6	6784.71	6641.39	2.16	29.9	20.7	12.0	**14.2**	11.1	14.8
$9a_{r1}$	96	7	6	8545.80	8035.09	6.36	259.7	119.5	94.2	105.0	87.7	**102.7**
$9a_{r2}$	120	8	6	8598.40	8140.15	5.63	2205.2	1005.5	624.4	784.7	450.2	**695.2**
8a	144	10	6	9721.25	9153.79	6.20	1607.5	724.3	382.1	445.6	321.3	**415.1**
$2b_{r1}$	32	2	4	2709.15	2682.52	1.00	53.5	**49.5**	38.8	69.8	40.9	52.8
1b	48	3	4	2277.44	2258.85	0.82	194.3	159.9	84.8	112.0	63.6	**93.1**
$2b_{r2}$	64	4	4	2771.68	2733.55	1.40	209.5	240.2	131.8	174.0	126.2	**163.2**
$3b_{r1}$	72	4	4	3306.86	3241.90	2.00	549.9	628.6	441.9	540.8	332.8	**520.7**
$7b_{r1}$	24	2	6	3776.25	3677.21	2.70	**0.8**	**0.8**	0.7	1.0	0.7	**0.8**
$8b_{r1}$	36	2	6	3640.79	3476.43	4.73	13.4	9.1	6.4	8.8	6.4	**8.6**
$7b_{r2}$	48	3	6	3723.18	3599.72	3.43	45.9	35.8	19.0	**26.1**	19.2	26.2
$8b_{r2}$	60	3	6	4606.17	4324.87	6.50	1964.9	1553.8	964.6	1609.9	1025.6	**1327.1**

Table 4.7: Experimental results of different subproblem algorithm settings on new PVRPTW instances with 50 customers based on Solomon's instances.

Id	LB	DP^S_{1d}	DP^S	GRASP+DP^S		REUSE+DP^S	
		t[s]	t[s]	min t[s]	med t[s]	min t[s]	med t[s]
p4r101$_{50}$	2736.50	0.6	**0.2**	0.8	0.9	0.6	0.7
p4r102$_{50}$	2151.73	2.3	**0.7**	0.9	1.1	0.8	1.1
p4r103$_{50}$	1868.40	6.8	**1.5**	1.3	1.7	1.3	1.7
p4r104$_{50}$	1679.42	12321.4	7.0	2.8	4.0	2.6	**3.4**
p4r105$_{50}$	2252.23	3.0	**0.9**	0.7	1.0	0.6	**0.9**
p4c101$_{50}$	1966.00	3.4	**1.1**	1.4	1.6	1.3	1.5
p4c102$_{50}$	1970.10	25.1	3.5	1.5	**2.2**	1.8	2.3
p4c103$_{50}$	1825.77	16.2	4.4	2.7	3.2	2.3	**3.1**
p4c104$_{50}$	1784.73	–	801.6	116.7	241.9	132.8	**190.4**
p4c105$_{50}$	1978.80	9.0	2.7	1.9	2.1	1.6	**2.0**
p4rc101$_{50}$	2771.97	0.8	**0.4**	0.6	0.8	0.6	0.7
p4rc102$_{50}$	2420.57	3.0	**0.9**	0.7	1.0	0.8	1.2
p4rc103$_{50}$	2145.00	64.5	4.3	1.3	2.2	1.4	**2.1**
p4rc104$_{50}$	1787.56	43847.5	20.1	5.7	7.9	5.9	**6.7**
p4rc105$_{50}$	2593.00	3.4	**0.8**	0.8	1.0	0.8	1.0

4.8 Branch-and-Cut-and-Price for the PVRPTW

Although the previously presented column generation approach for the PVRPTW yields tight lower bounds, it does not provide upper bounds, i.e. feasible primal solutions, in general. In order to certainly produce both, and eventually even prove the optimality of the solution found, one needs to extend the column generation process with a branching scheme, finally yielding a *Branch-and-Price* (B&P); this method is the topic of Section 4.8.1. Additionally adding inequalities to strengthen the model yields a *Branch-and-Cut-and-Price* (B&C&P) approach. The latter is achieved for our problem via adding 2-path cuts as well as subset-row cuts, which is detailed in Section 4.8.2.

4.8.1 Branching Scheme

We propose three types of branching in order to obtain integer solutions. They are applied in the order as presented. The first one often appears in the literature and concerns the number of vehicles, also referred to as fleet size. Thereby we select the most fractional fleet over all

Table 4.8: Experimental results of different subproblem algorithm settings on new PVRPTW instances with 100 customers based on Solomon's instances.

Id	LB	DP^S_{1d}	DP^S	GRASP+DP^S		REUSE+DP^S	
		t[s]	t[s]	min t[s]	med t[s]	min t[s]	med t[s]
p4r101	4076.85	16.4	6.5	3.0	3.5	2.7	**3.2**
p4r102	3714.80	99.4	15.5	5.5	**6.0**	5.7	6.8
p4r103	3142.10	–	54.1	8.4	13.5	9.3	**11.4**
p4r104	2516.07	–	1127.5	98.4	145.4	78.5	**130.4**
p4r105	3590.48	104.6	14.4	5.2	5.9	4.3	**4.9**
p4c101	2904.50	123.8	19.9	5.7	6.7	4.9	**6.3**
p4c102	2869.30	7038.5	73.8	21.8	27.4	16.2	**20.1**
p4c103	2676.67	–	197.8	25.8	30.7	21.3	**25.8**
p4c104	2400.23	–	17488.7	6198.3	8360.9	3140.3	**6773.7**
p4c105	2876.30	174.2	51.7	8.8	10.6	8.6	**10.1**
p4rc101	3907.70	43.4	10.2	4.2	5.2	4.0	**4.9**
p4rc102	3716.95	292.0	22.0	6.7	**7.6**	6.7	**7.6**
p4rc103	3401.62	–	83.6	14.6	18.7	13.1	**15.2**
p4rc104	2941.71	–	4768.1	1003.2	1499.5	721.2	**1447.7**
p4rc105	3882.68	108.9	13.7	5.2	6.2	5.0	**5.5**

days:

$$\tau' = \underset{\tau \in T}{\operatorname{argmax}} \ \min \left(\sum_{\omega \in \Omega} \chi_{\omega\tau} - \left\lfloor \sum_{\omega \in \Omega} \chi_{\omega\tau} \right\rfloor , \left\lceil \sum_{\omega \in \Omega} \chi_{\omega\tau} \right\rceil - \sum_{\omega \in \Omega} \chi_{\omega\tau} \right) ,$$

afterwards we add two child nodes with the restriction:

- left child node with fleet size $\leq \left\lfloor \sum_{\omega \in \Omega} \chi_{\omega\tau'} \right\rfloor$, and
- right child node with fleet size $\geq \left\lceil \sum_{\omega \in \Omega} \chi_{\omega\tau'} \right\rceil$.

The next branching rule concerns the visit combinations y_{ir}. For this the most fractional visit combination is selected:

$$(i', r') = \underset{i \in V_C, r \in C_i}{\operatorname{argmax}} \ \min(y_{ir}, 1 - y_{ir})$$

and utilized to generate the following child nodes:

- left child node with $y_{i'r'} = 1$ and $y_{i'r} = 0, \forall r \in C_{i'}, r \neq r'$, and

- right child node with setting $y_{i'r'} = 0$.

Finally, we have to use a branching rule taking into account the flow on single arcs which is implicitly given by the route variables. Again, we select those arc $(v_{i'}, v_{j'})$ having the most fractional flow over all days and add these two child nodes:

- left child node where the arc $(v_{i'}, v_{j'})$ is "forced": whenever a route visits customer i' it must subsequently also visit j', and
- right child node where the arc $(v_{i'}, v_{j'})$ is forbidden.

Having a complete branching scheme defined, the difficulty in practice, however, is often the enforcement of the branchings in the actual master problem as well as in subsequent pricing rounds. Due to the second and third rule additional corresponding arcs have to be forbidden in order to enforce the chosen branching step.

4.8.2 Strengthening Inequalities

Although the previously defined branching scheme allows to generate primal solutions, it sometimes takes very long to close the gap between lower and upper bounds and prove optimality, usually because of the slowly increasing lower bound. Hence we propose two additional inequalities for strengthening the formulation. Finding cuts to strengthen the model—either known ones which can be applied or even new ones—is not trivial in general, and especially difficult in our case where already a lot of the "usual" cuts appearing in the VRP literature are already implicitly contained in the model. Therefore we resort to adapting suitable cuts to our problem which were reported recently.

2-Path Cuts

The 2-path cuts are a special form (an instance) of the k-path cuts which were introduced by Kohl et al. in 1999 for the VRPTW [125]. Assuming a directed graph, the incoming flow of a set of customers $S \subseteq V_C$ is defined as

$$X(S) = \sum_{i \in V_C \setminus S} \sum_{j \in S} x_{ij} \; .$$

For a 2-path cut we seek a set S such that $X(S) < 2$ but the minimal number of vehicles necessary to service all customers in S, in the following denoted by $k(S)$, is greater than one, i.e. $k(S) > 1$. Since the number of vehicles is integral it must hold that $k(S) \geq 2$.

In case of non-periodic VRPs each customer needs to be serviced exactly once, whereas for the PVRP(TW) occur "fractional daily visits" because of

$$0 \leq \sum_{r \in C_i} \beta_{ir\tau} \, y_{ir} \leq 1 \quad \forall i \in V \; .$$

The flow of a set S further depends on the day, thus is denoted by $X_\tau(S), \tau \in T$.

Hence we propose the 2-path cut for the PVRP(TW) in the following way:

$$X_\tau(S) + \sum_{i \in S}(1 - \sum_{r \in C_i} \beta_{ir\tau}\, y_{ir}) \geq k(S) \qquad \forall S \subseteq V_C;\ \forall \tau \in T$$

or equivalently

$$X_\tau(S) - \sum_{i \in S} \sum_{r \in C_i} \beta_{ir\tau}\, y_{ir} \geq k(S) - |S| \qquad \forall S \subseteq V_C;\ \forall \tau \in T\ .$$

For separating 2-path cuts one first needs to find sets $X_\tau(S) < 2$. Kohl et al. propose a greedy algorithm in [125], though it only guarantees to find all sets when the graph is acyclic. They further state an algorithm which enumerates all minimal sets, but its expected long runtime prevented anyone from applying it so far. Cook and Rich [37] used the random contraction algorithm of Karger [118, 117], which is a randomized min-cut algorithm. They use a special form of the algorithm finding all cuts with a weight lying within a multiplicative factor α of the minimum cut; they set $\alpha = 2$. Ropke et al. [198] applied a randomized greedy construction heuristic for finding the necessary sets.

In our work we implemented a heuristic multi-start method to obtain suitable sets of customers which bears some resemblance with a variable neighborhood descent; it is depicted in Algorithm 13. It always considers the customers in a random order and applies a first improvement fashion, at which only customers with their in-flow $X(\{v_i\})$ above a specified threshold are taken into account. The latter guarantees to focus on actually relevant customers at the time of separation, the inequality is weakened quite drastically otherwise. Further, a customer i is added to S' if it holds that $1 < X(S' \cup \{v_i\}) < 2$, an exchange of customers i and j occurs when $1 < X((S' \cup \{v_j\}) \setminus \{v_i\}) < 2$ and $X((S' \cup \{v_j\}) \setminus \{v_i\}) < X(S')$, i.e. the exchange increases the remaining flow. Finally, a removal of a customer is applied whenever neither adding nor exchanging was successful, this is tried for $n/2$ times and yields sets that also satisfy $1 < X(S') < 2$.

Having obtained some viable customer sets, we have to check whether more than one vehicle is necessary to serve the customers. For this we can directly use our exact label correcting algorithm developed for the ESPPRC pricing subproblem. Therefore we have to set the costs of the arcs in a suitable way, s.t. only a path departing from the depot, visiting all customers in S and ending at the target node will have a negative cost:

$$\begin{aligned} c_{0i} &= |S| - 0.5 && \forall i \in S \\ c_{ij} &= -1 && \forall i \in S;\ \forall j \in S \cup \{v_{n+1}\} \end{aligned}$$

Those arcs which were not assigned a cost are deactivated. Similarly to solving the actual pricing subproblem we can apply the different dominance rules in a sequential way, too. Moreover, we can stop the procedure after finding a single negative cost path.

The 2-path cuts do not intrinsically alter the subproblem and hence allow for a robust B&C&P. However, they need to be considered when deriving the reduced costs: for each arc from a customer out of $V_C \setminus S$ to one belonging to S the dual variable value of the corresponding

Algorithm 13: Separating 2-path cuts: find sets $S \subseteq V_C$ with $X_\tau(S) < 2$

```
Sets ← ∅
for i = 1 to n do
    S' ← {v_i}
    count ← 0
    repeat
        repeat
            added ← addCustomer(S')
            if NOT added then
                exchanged ← exchangeCustomer(S')
        until NOT (added ∨ exchanged)
        if checkSetForInclusion(S') then
            Sets ← Sets ∪ {S'}
        count ← count + 1
        remove ← removeCustomer(S')
    until NOT removed ∧ count ≥ n/2
return Sets
```

2-path cut must be taken into account when determining the reduced costs of the arc (by subtracting it).

Note that due to the heuristic for finding the necessary customer sets the separation of 2-path cuts as a whole is of heuristic nature.

Subset-Row Cuts

The *subset-row* (SR) inequalities were introduced by Jepsen et al. [114] and were derived from the clique inequalities for the set-packing problem [149]. Using our notation they can be stated as:

$$\sum_{\omega \in \Omega} \left\lfloor \frac{1}{k} \sum_{i \in S} \alpha_{i\omega} \right\rfloor \chi_{\omega\tau} \leq \left\lfloor \frac{|S|}{k} \right\rfloor \qquad \forall \tau \in T;\, \forall S \subseteq V_C;\, 0 < k \leq |S| \,.$$

The general idea behind these cuts is to avoid that customers are visited by more than one vehicle (route), which is clearly satisfied by each feasible integral solution. In the original work [114] they set $|S| = 3$ and $k = 2$, which turned out to be satisfying for our setting, too. Further, we also do a complete enumeration to identify violated inequalities. The latter poses no problem with regard to runtime. Their separation is thus done in an exact way.

A downside of the SR inequalities is that they alter the pricing subproblem, i.e. we end up with a non-robust B&C&P. This is because we need to introduce an additional resource m per inequality to the labels, setting its value to $m = |S \cap V(L)|$, where $V(L)$ are the nodes visited by the partial path. Naturally these resources need to be considered when checking for dominance; for details we refer to [114].

4.8.3 Computational Results

For evaluating the performance of the Branch-and-Price as well as its extension to the Branch-and-Cut-and-Price approach we took the newly created Solomon-based PVRPTW instances having a planning horizon of four days and 36 as well as 50 customers (the column generation approach was already applied to the latter ones). All algorithms have been implemented in C++, compiled with GCC 4.1, and performed on a single core of an Intel Xenon E5540 with 2.53 GHz and 3 GB RAM dedicated per core. The whole approach was implemented with SCIP [2], using Ilog CPLEX 12.1 as LP solver. We performed tests with following different settings:

- BP_1: Branch-and-Price solely using the labeling algorithm
- BP_2: apply the REUSE subproblem heuristic until it either finds no new columns for five times in a row or the number of generated columns decreased in the last five iterations, switch to the labeling algorithm afterwards
- BCP_1: like BP_1 plus separating at most 500 2-path cuts
- BCP_2: like BP_1 plus separating at most 100 subset-row cuts when dealing with 36 customers and 200 cuts in case of 50 customers
- BCP_3: a mixture of BP_1, BCP_1 and BCP_2

We always prematurely stop the pricing problem after generating more than 100 new columns and set a time limit of one hour (stopping is not enforced during solving the pricing subproblem). To not rely on any external method we introduce the slack variables to start without initial columns. The results are shown in Table 4.9 for 36 customers and in Table 4.10 for 50 customers, stating the resulting percentage gap (%-gap) or "–" when no gap is available, the number of solved nodes, and the runtime in seconds. Note that instance p4c104_{50} is omitted since no setting yielded a feasible solution for it. Whenever non-determinism is involved, i.e. for settings BP_2, BCP_1 and BCP_3, we performed 10 runs and state average results as well as in some rare cases (for instance p4rc104_{50}) as superscript the number of runs with a gap other than infinity. Per instance we print in bold either the resulting gap or the runtime of the setting which performed best, i.e. those obtaining the smallest gap or the fastest one in case of same gaps. We also state averaged results over all instances, where for BP_1 on the 50 customer instances we averaged the gap only over the 13 solved instances.

As was also observed for the column generation approach, for such rather small instances the heuristic pricing method yields not much gain. This is again not surprising since the switch (fallback) to the labeling algorithm occurs quite soon. Taking a closer look: For instances having 36 customers the gap is a little bit smaller, but the runtimes are slightly higher, whereas for instances with 50 customers both characteristics are slightly improved. Nevertheless, we were not convinced enough of BP_2 and hence we solely apply the labeling algorithm for the other settings, having also the advantage to investigate the influence of the cutting planes without any bias from non-deterministic column generation. Looking at the

Table 4.9: Results of several Branch-(and-Cut-)and-Price settings on smaller derived periodic Solomon instances with a planning horizon of four days and 36 customers.

Instance		BP_1			BP_2			BCP_1			BCP_2			BCP_3		
Id	m	%-gap	nodes	t[s]	%-gap	nodes	t[s]	%-gap	nodes	t[s]	%-gap	nodes	t[s]	%-gap	nodes	t[s]
p4r101$_{36}$	6	0.00	652	9.3	0.00	605	9.6	0.00	652	12.9	0.00	352	**6.4**	0.00	352	8.8
p4r102$_{36}$	7	0.59	94190	3601.9	0.61	106041	3601.6	0.00	87	**5.3**	0.00	140	6.3	0.00	99	8.5
p4r103$_{36}$	6	0.00	24203	1213.0	0.04	47199	3165.4	0.00	20793	1282.9	0.00	750	**46.5**	0.00	750	55.4
p4r104$_{36}$	4	0.00	4067	1342.1	0.00	4298	**1297.1**	0.00	3867	1498.3	0.00	3917	1684.4	0.00	3326	1574.2
p4r105$_{36}$	5	0.00	10440	526.8	0.00	13286	734.4	0.00	7798	420.7	0.00	3699	**133.7**	0.00	2884	167.9
p4c101$_{36}$	5	0.00	7	1.5	0.00	6	1.4	0.00	7	1.0	0.00	7	**0.9**	0.00	7	1.1
p4c102$_{36}$	4	0.00	60	11.8	0.00	122	16.8	0.00	35	**9.7**	0.00	38	11.0	0.00	40	12.9
p4c103$_{36}$	4	0.00	223	43.7	0.00	368	62.4	0.00	223	48.7	0.00	22	**9.2**	0.00	22	9.8
p4c104$_{36}$	3	0.00	544	875.4	0.00	575	983.2	0.00	634	911.7	0.00	634	**831.1**	0.00	634	880.6
p4c105$_{36}$	4	0.00	107	9.3	0.00	196	8.9	0.00	107	7.3	0.00	21	**3.8**	0.00	21	4.3
p4rc101$_{36}$	6	0.00	1841	46.6	0.00	2003	31.8	0.00	827	25.1	0.00	173	**5.9**	0.00	218	9.7
p4rc102$_{36}$	4	0.61	52075	3600.3	0.23	53924	2374.6	0.06	58642	3512.0	0.00	17809	**1316.8**	0.00	17779	1700.7
p4rc103$_{36}$	5	0.00	497	38.9	0.00	330	18.1	0.00	497	29.4	0.00	8	**2.0**	0.00	8	2.3
p4rc104$_{36}$	3	0.00	3971	2105.7	0.00	3118	**1655.9**	0.00	4264	2388.2	0.00	2172	2581.7	0.00	2254	2727.5
p4rc105$_{36}$	6	0.00	39460	1824.7	0.00	39609	1949.5	0.00	10046	344.6	0.00	1325	**43.3**	0.00	1343	58.3
average		0.08	15489	1016.7	0.06	18112	1060.7	0.00	7232	699.9	0.00	2071	445.5	0.00	1982	481.5
# best			0			2			2			11			0	

Table 4.10: Results of several Branch-(and-Cut-)and-Price settings on smaller derived periodic Solomon instances with a planning horizon of four days and 50 customers.

Instance		BP_1			BP_2			BCP_1			BCP_2			BCP_3		
Id	m	%-gap	nodes	t[s]	%-gap	nodes	t[s]	%-gap	nodes	t[s]	%-gap	nodes	t[s]	%-gap	nodes	t[s]
p4r101$_{50}$	10	0.00	7	**0.8**	0.00	10	1.9	0.00	7	1.1	0.00	8	1.0	0.00	8	1.3
p4r102$_{50}$	8	0.00	7506	809.0	0.00	7211	775.3	0.00	7500	1079.0	0.00	3185	614.3	0.00	3177	**586.5**
p4r103$_{50}$	7	0.00	176	**72.4**	0.00	471	177.9	0.00	496	204.9	0.00	227	117.7	0.00	227	138.4
p4r104$_{50}$	5	1.34	2168	3603.0	1.88	2144	3603.2	1.49	2005	3603.2	**0.85**	938	3602.7	0.88	862	3601.6
p4r105$_{50}$	6	5.32	32949	3600.6	4.12	34942	3600.7	1.81	19099	3606.2	0.48	13476	3600.5	**0.22**	12373	3600.7
p4c101$_{50}$	6	0.00	4	7.8	0.00	5	**2.6**	0.00	4	5.5	0.00	1	5.4	0.00	1	5.5
p4c102$_{50}$	5	0.00	53	34.2	0.00	95	**32.2**	0.00	53	36.5	0.00	48	44.5	0.00	48	39.8
p4c103$_{50}$	6	0.00	149	**127.2**	0.00	230	191.4	0.00	149	137.9	0.00	139	181.1	0.00	139	192.1
p4c105$_{50}$	5	0.00	3631	484.9	0.00	2919	412.0	0.00	3	15.4	0.00	1	14.4	0.00	1	**14.1**
p4rc101$_{50}$	8	0.28	52329	3602.9	0.27	53978	3602.6	0.15	68863	3600.8	0.16	36913	3601.8	**0.11**	48484	3601.2
p4rc102$_{50}$	6	0.00	25926	3482.9	0.00	11378	1619.9	0.00	14158	2551.6	0.00	14874	2442.5	0.00	2559	**692.6**
p4rc103$_{50}$	5	0.00	5126	1712.7	0.00	4487	1610.4	0.00	4745	1932.2	0.00	5332	2162.1	0.00	2350	**1227.7**
p4rc104$_{50}$	4	–	35	3618.5	1.81[3]	445	3669.9	1.47[8]	1135	3618.5	**1.14**	364	3600.5	1.19	301	3602.8
p4rc105$_{50}$	6	0.00	1285	146.8	0.00	4076	459.9	0.00	3295	487.1	0.00	287	**51.2**	0.00	579	112.3
average		0.50	9381	1521.7	0.58	8742	1411.4	0.35	8679	1491.4	0.19	5413	1431.4	0.17	5079	1244.0
# best			3			2			0			3			6	

results of the BCP variants it is clear that separating the proposed inequalities is always beneficial, especially the subset-row cuts: either the runtime is more than halved (36 customers) or the resulting gap is perceivably smaller (50 customers). When using subset-row cuts the number of solved nodes often decreases substantially, which is one one hand due to their strength and on the other hand—and which should not be neglected—due to the altered and even more demanding pricing subproblem. Only utilizing the 2-path cuts is less successful, but interestingly for the 50 customer instances the variant where both cuts are separated performs best. It is also observable that the clustered instances seem easiest to solve (despite the mentioned unsolvable instance p4c104_{50}), a fact which was not apparent when only solving the LP relaxation.

4.9 Matheuristic Variants for the PVRPTW

In the previous sections we introduced heuristic as well as exact methods for solving the PVRPTW, either delivering primal solutions or valid lower bounds. Since the very beginning of the project we aimed at combining compatible methods in a suitable way and come up with powerful Matheuristics. Therefore we will especially focus on the hybridization of the VNS and also the multiple VNS with the proposed set covering ILP model, which will be done in Section 4.9.1 and 4.9.2, respectively, as well as further on the combination of the column generation approach and the EA detailed in Section 4.9.3, followed by some additional concepts reported in Section 4.9.4.

4.9.1 Hybridizing the VNS and the Set Covering ILP

In the following the ILP model of Section 4.7.1, i.e. basically the master problem of the column generation approach, is used to boost the performance of the VNS of Section 4.4. This is achieved by introducing feasible VNS solutions into the set covering model, i.e. by adding the single routes of these solutions as new columns. This way the feasibility of the model is guaranteed. The resulting ILP is solved by a (basically) branch-and-bound based generic ILP solver, gradually fixing the visit combination (4.5) and route variables (4.6). A similar approach was introduced in [206], where the authors highlight the "global view" property of such an exact model. For a set of solutions' routes the ILP solver might be able to derive a more favorable combination and provide a better (less costly) solution in this way. Obviously the potential of the ILP solver depends on the routes contained in the model since it is neither able to alter routes nor create ones on its own.
Hence it is crucial to provide:

(i) a suitable amount of routes,

(ii) cost-effective routes,

(iii) and diverse enough routes.

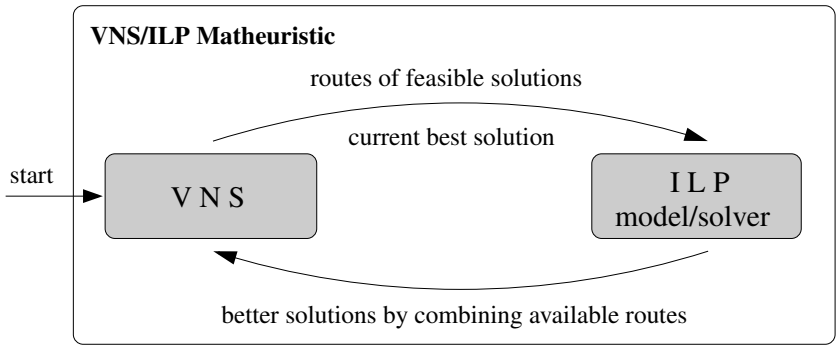


Figure 4.7: Information exchange between VNS and ILP model/solver.

Adding not enough or only weak routes might prevent finding a better solution at all, on the other hand a too large set naturally increases the runtime, which might also prevent finding better solutions quickly enough in case a time limit is given. Therefore solutions should not be added arbitrarily to the model. Only finding new combinations of routes constituting a feasible solution is not sufficient, the solution should also improve on the current incumbent in terms of travel costs. This is dealt with by primarily adding improved and feasible solutions found by the VNS (i.e. solutions that improved on the current best solution at the time they were derived), as it is done in [206]. However, we found it often not enough to solely add such improved solutions, since in case the VNS gets stuck for a while no improved solutions are available at all and no further solving of the ILP is meaningful. In order to be able to still exploit the power of the ILP solver we propose to further add some "intermediate" VNS solutions, i.e. feasible solutions derived in an iteration but not improving on the best solution. For ensuring a certain quality, we define a maximal deviation ε from the current best solution (at the time of checking), and avoid duplicates.

We apply the following hybrid scheme, which is a high-level integrative combination; see Figure 4.7. The hybrid algorithm divides the execution of a VNS run in several equally long sections $\mathcal{S}$, and after each section the current ILP model is solved. The latter is gradually enriched by columns extracted from selected feasible VNS solutions. For this we choose a number n_{sol} of overall solutions to consider and first try to insert only the most current improved VNS solutions. In case less than n_{sol} such solutions are provided by the previous VNS section we select the remaining ones from the set of intermediate solutions (also accumulated during the previous VNS section) at random, in order to obtain a set of diverse solutions. Further, the ILP solver is always initialized with the current best solution to speed up the process. If the ILP solver is able to improve on the current best solution this new solution is transformed and transferred to the VNS. During transformation over-covered solutions are repaired by choosing exactly one visit combination (the first active) and omitting customers from following routes if they are already covered or do not need to be covered on this day. This over-covering might happen since we use a set covering model. In contrast, a set partitioning model (derived by turning inequalities (4.2) and (4.4) into equalities)

Table 4.11: Concise results of VNS-ILP on Pirkwieser and Raidl instances with a planning horizon of four and six days reported in [159].

Instances	VNS-ILP$_{5,0\%}$	VNS-ILP$_{5,5\%}$	VNS-ILP$_{5,10\%}$	VNS-ILP$_{10,5\%}$	VNS-ILP$_{10,10\%}$
significantly better/worse than VNS (10^6) and not exceeding runtime by more than 20%					
p4r	2×/1×	4×/0×	4×/0×	4×/0×	3×/0×
p4c	1×/0×	2×/1×	1×/0×	2×/0×	1×/0×
p4rc	2×/0×	4×/0×	4×/0×	4×/0×	3×/0×
p6r	3×/0×	4×/0×	4×/0×	1×/0×	1×/0×
p6c	0×/0×	1×/1×	0×/0×	0×/1×	1×/1×
p6rc	2×/0×	5×/0×	5×/0×	4×/0×	4×/0×
	10×/1×	20×/2×	18×/0×	15×/1×	13×/1×

would yield only feasible solutions but at the same time exclude many potentially improving combinations. Finally, when injecting this solution into the VNS it is also subject to the previously mentioned 2-opt* improvement procedure. In case routes were altered during these procedures, corresponding new columns are also added to the ILP model.

What we did not mention so far is the applied route injection scheme, which is also of concern in the classical column generation approach. Basically it is possible to either add the route for the corresponding day only or for all days. The latter scheme produces significantly larger ILP models (of factor t) which might yield better solutions at the expense of longer running times to solve the ILP model. We compare these alternatives in our experimental results. For both variants we allow the ILP solver the same overall amount of CPU-time as the VNS, though it is expected that the former variant consumes only a fraction of it.

Previous Computational Results

The following results are reported in [159, 162], we present them in a concise form here. In the former works we again use the VNS as a baseline with an iteration limit of either 10^6 or $2 \cdot 10^6$. The VNS-ILP hybrid is also based on a VNS run with 10^6 iterations and $\mathcal{S}$ is set to 10, i.e. it applies ten sequences of 10^5 VNS iterations with subsequent ILP solving phases.

In [159] we performed tests on instances having a planning horizon of four and six days with three settings of ε (0%, 5%, and 10%, where 0% implies that no intermediate VNS solutions are considered) and two settings of n_{sol} (5 and 10), denoted by VNS-ILP$_{n_{\text{sol}},\varepsilon}$. The solutions' routes are mainly added for the corresponding day only (single day strategy VNS-ILP$^{\text{s}}$, but as it is the default setting we might also omit the 's'), some experiments are done with insertion for all days (all days strategy VNS-ILP$^{\text{a}}$). Each algorithm setting is run 30 times per instance and as said we report only concise results here. Among the hybrid variants there is rarely a single one significantly outperforming all others, therefore we decided to compare the number of times the methods improved over the VNS, which we first did here and performed also in later work. The results are shown in In Table 4.11, though to be "fair" only considering those runs where the runtime of VNS-ILP did not exceed those of the VNS by 20%.

As can be seen it is generally beneficial to also consider intermediate VNS solutions, yield-

Table 4.12: Concise results of VNS-ILP on Pirkwieser and Raidl instances with insertion of new routes for one or all days reported in [159].

Instances	best VNS-ILP	VNS-ILP$^{a}_{5,5\%}$
significantly better than VNS ($2 \cdot 10^6$)		
p4r	4×	4×
p4c	1×	3×
p4rc	2×	2×
p6r	4×	1×
p6c	1×	0×
p6rc	2×	1×
	14×	11×

ing twice as many significant improvements. Regarding the quality of these solutions, there is a slight advantage when using better ones ($\varepsilon = 5\%$). While the performance of the hybrid method is nearly the same when adding 5 or 10 solutions at a time for instances having a planning horizon of four days, the performance notably degrades for a planning horizon of six days. This is mainly because of the too large runtime when the ILP solver consumes more of the allotted time, leading to an exclusion of these variants for some instances (e.g. happening six times for VNS-ILP$_{10,[5\%,10\%]}$). However, for many variants and instances the overall execution time of the ILP solver itself is short (a few seconds), often solving the ILP to optimality in fractions of a second. The remaining overhead compared to solely running the VNS is mainly due to the information exchange, especially for storing the intermediate VNS solutions and avoiding duplicates. The least improvement occurred for the clustered instances.

Now we will also have a look at the all days strategy, applied with $n_{\text{sol}} = 5$ and $\varepsilon = 5\%$, since this setting yielded good results in the former tests. To be fair, its performance must be compared to the VNS with $2 \cdot 10^6$ iterations, since its time consumption is basically bounded by taking twice the run time of the VNS with 10^6 iterations, plus the overhead for the information exchange. Again concise results are shown in Table 4.12, alongside with the best performing single day strategy of each instance (see [159] for details), note that no variants need to be excluded because of a too high time consumption here. It can be observed that the all days strategy is generally worse than the single day strategy, though still yielding a significant improvement for 11 of the 30 instances (36%). Most notably the best runs of the single day strategy improve the results in 14 cases (46%), although the CPU-time consumption is considerably less than that of both competitors. All in all one of the single day strategies either performs better or at least comparable to the VNS with $2 \cdot 10^6$ iterations.

Additional test runs are reported in [162], where also the instances with a planning horizon of eight days are considered, which were not available in the former case. We present them in short in Table 4.13, again giving the number of times VNS-ILP is better or worse than VNS, also stating the average runtime consumed. As can be observed for an increasing plan-

Table 4.13: Additional concise results of VNS-ILP on Pirkwieser and Raidl instances reported in [162].

Instances	VNS-ILP$_{8,5\%}$		VNS
	sign. better/worse than VNS	t[s]	t[s]
p4r	4×/0×	28.2	24.3
p4c	2×/0×	27.2	25.0
p4rc	5×/0×	27.8	26.2
p6r	4×/0×	35.5	27.9
p6c	1×/0×	34.3	31.3
p6rc	4×/0×	33.5	29.6
p8r	2×/0×	35.1	29.5
p8c	0×/0×	36.6	32.2
p8rc	2×/0×	33.4	30.4

ning horizon the better performance of the hybrid variant decreases compared to the standard VNS approach, especially notable for the p8 instances. It seems that the derived (intermediate) VNS solutions of these larger instances are probably too diverse, so the ILP is not able to utilize meaningful solution parts (routes and visit combinations), which are kind of "overlapping" for smaller instances, in a cost-decreasing manner anymore. A second observation is that the additional runtime, mainly due to the ILP solving steps, is only quite moderate.

4.9.2 Hybridizing the Multiple VNS and the Set Covering ILP

Here we investigate a Matheuristic composed of the multiple VNS and again the set covering ILP model. This hybrid variant is conceptually similar to the previous one when using the standard VNS, though the handling of the solutions as well as some details are different. The information exchange between the multiple VNS and the ILP, which can also be regarded an integrative combination, can be seen in Figure 4.8 as well as in Algorithm 14.

Concerning the hybridization with the multiple VNS a natural and suitable way is to apply the ILP solver after a block of section-wise executions. This way the number of solutions to consider for adding to the ILP model is given by the number of VNS instances, from which we always use the actual best solutions. Due to different search trajectories these solutions' routes are further assumed to be diverse enough. Hence, we deem conditions (i)–(iii) introduced in the previous section as fulfilled. Hence, contrary to [206] and the previous VNS-ILP hybrid we restrict ourselves to the actual best solutions only, yet now we have more VNS instances available. Due to this there is no need for handling intermediate solutions. Similarly the ILP solver is allotted the same amount of CPU-time than the multiple VNS.

The application of the ILP solver can in some way be regarded as a recombination operator taking into account all available solutions provided by the "population" of the VNS instances,

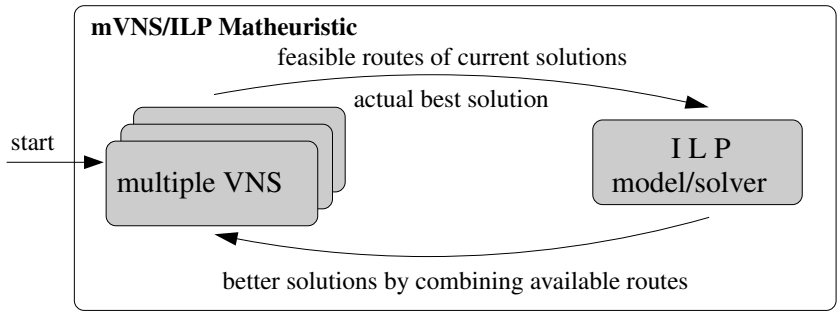


Figure 4.8: Information exchange between multiple VNS and ILP model/solver.

Algorithm 14: Multiple VNS / ILP Hybrid: $\#VNS$ refers to the number of VNS instances, $\#sec$ to the number of sections per VNS instance and to the maximal number of ILP solver applications, and $iter_{max}$ is the total number of allowed iterations.

$\Omega' \leftarrow \emptyset$; `// start with empty model`
for $i = 1$ **to** $\#VNS$ **do**
    initialize VNS[i]
$iter_{sec} \leftarrow \lceil iter_{\max}/(\#VNS \cdot \#sec) \rceil$
for $sec = 1$ **to** $\#sec$ **do**
    $\Omega'_{sec} \leftarrow \emptyset$
    for $i = 1$ **to** $\#VNS$ **do**
        execute VNS[i] for $iter_{sec}$ iterations
        add VNS[i] solutions' routes to Ω'_{sec}; `// gather columns`
    $x^* \leftarrow$ actual best solution
    $\Omega' \leftarrow \Omega' \cup \Omega'_{sec}$; `// enrich ILP model`
    $x \leftarrow$ apply ILP solver on Ω', initialized with x^*
    Replace solution of worst VNS instance by x
Return best solution of all VNS instances

hence realizing some kind of optimal merging. A novelty here is that in case a solution is not feasible as a whole, its feasible routes are added anyway, yet the ILP solver is only applied if at least one feasible solution exists. Again, the ILP solver is always initialized with the current best solution to speed up the process.

If the ILP solver is able to improve on the current best solution this new solution is transferred to the multiple VNS, where as usual the solution of the worst VNS instance is replaced. There are also two options regarding the lifetime of the routes added to the ILP model: Either we only consider the actual solutions' routes, i.e. they are discarded afterwards, or we keep all inserted routes and the ILP model gradually grows, i.e. realizing a long term memory. However, it is clear that a model of continuously increasing size in general demands more and more computation time to be solved. This could in turn lead to worse solutions when setting

a time limit as in our case. Nevertheless the larger induced search space might also contain better solutions. If routes are kept then the ILP solver is only applied if non-existing routes could be added after a multiple VNS section. Both variants are examined in the next section.

Previous Computational Results

The results presented here are mainly reported in [161], extended by tests on Pirkwieser and Raidl instances with a planning horizon of eight days. Like the multiple VNS (see Section 4.5) also the mVNS-ILP hybrid is allowed 10^6 VNS iterations in total and $\#sec$ is consistently set to 10 (as determined by preliminary tests); i.e. applying ten sequences of $\#VNS$ VNS instances, each one running for $10^6/(\#VNS \cdot 10)$ iterations per section. For solving the ILP model in the mVNS-ILP hybrid we applied the general purpose MIP solver IBM ILOG CPLEX 12.1. We experimented with 5, 8, 10, and 15 VNS instances, denoted by mVNS-ILP$_{\#VNS,\#sec}$. As before each algorithm setting was run 30 times per instance.

Table 4.14 shows concise results of all mVNS-ILP variants considered, with the setting of storing all injected routes. We state how often they performed significantly better or worse than the corresponding mVNS, as well as how many times they were significantly better or worse than the standard VNS variants, again using a Wilcoxon rank sum test with an error level of 5% for testing statistical significance. As was observed in Section 4.5 mVNS was already able to often outperform standard VNS. However, combining the mVNS with ILP techniques in the mVNS-ILP hybrid consistently yields even more satisfying results. Looking at the overall best variant, mVNS-ILP$_{8,10}$, it significantly improves upon VNS with 10^6 iterations in about 90% of all cases, and upon the corresponding mVNS as well as VNS with $2 \cdot 10^6$ iterations in about 70%. Taking the average runtimes into account, which are given in Table 4.16, we see that the runtime of mVNS-ILP$_{8,10}$ is still notably less than that of the VNS with $2 \cdot 10^6$ iterations (which is per setting the upper limit). We note that already mVNS-ILP$_{5,10}$ delivers good results, with the advantage of only a very small increase in runtime. Naturally, for an increasing number of VNS instances $\#VNS$ and especially for longer planning horizons we observe that the runtime of mVNS-ILP approaches that of VNS ($2 \cdot 10^6$), i.e. the ILP solving consumes all of the allotted time, hence justifying a comparison to this latter VNS variant. For these cases a performance degradation can be observed, too. In general, the mVNS-ILP hybrid approach achieves to a large extent significantly better results than the standard VNS variants as well as the corresponding mVNS, yet the runtime is very often below the limit.

So far we only considered the strategy to keep all routes in the model once they were added. Therefore analog results of mVNS-ILP when resetting the columns after each application of the ILP solver are given in Table 4.15, again stating the results of the statistical significance tests, yet this time more condensed per planning horizon. Comparing these results to the previous ones there is mostly no gain in solution quality observable, for neither of the planning horizons. Contrary, the results are in fact worse, i.e. it seems generally better to work with an ILP model of increasing size and hence exploit information from the search trajectory. The only exception being mVNS-ILP$_{15,10}$, especially when compared to its corresponding mVNS. Resetting the columns also leads, for obvious reasons, to shorter runtimes, which is

Table 4.14: Concise results of mVNS-ILP on all Pirkwieser and Raidl instances, partly reported in [161], stating how often mVNS-ILP performs significantly better/worse than mVNS and VNS.

Instances	mVNS-ILP$_{5,10}$	mVNS-ILP$_{8,10}$	mVNS-ILP$_{10,10}$	mVNS-ILP$_{15,10}$
significantly better/worse than corresponding mVNS				
p4r	2×/0×	5×/0×	5×/0×	5×/0×
p4c	1×/0×	3×/0×	5×/0×	5×/0×
p4rc	4×/0×	5×/0×	5×/0×	5×/0×
p6r	2×/0×	4×/0×	4×/0×	1×/0×
p6c	1×/0×	1×/0×	1×/0×	1×/0×
p6rc	1×/1×	5×/0×	4×/0×	2×/0×
p8r	2×/0×	4×/0×	4×/0×	2×/0×
p8c	1×/0×	1×/0×	1×/0×	2×/0×
p8rc	2×/0×	4×/0×	3×/0×	2×/0×
	16×/1×	32×/0×	32×/0×	25×/0×
significantly better/worse than VNS (10^6)				
p4r	5×/0×	5×/0×	5×/0×	5×/0×
p4c	5×/0×	5×/0×	5×/0×	5×/0×
p4rc	5×/0×	5×/0×	5×/0×	5×/0×
p6r	5×/0×	5×/0×	5×/0×	3×/0×
p6c	5×/0×	5×/0×	4×/0×	4×/0×
p6rc	5×/0×	5×/0×	4×/0×	2×/0×
p8r	5×/0×	4×/0×	3×/0×	2×/1×
p8c	2×/0×	3×/0×	3×/0×	2×/1×
p8rc	3×/0×	4×/0×	2×/0×	1×/1×
	40×/0×	41×/0×	36×/0×	29×/3×
significantly better/worse than VNS ($2 \cdot 10^6$)				
p4r	4×/0×	5×/0×	5×/0×	5×/0×
p4c	4×/0×	5×/0×	5×/0×	5×/0×
p4rc	4×/0×	5×/0×	5×/0×	5×/0×
p6r	5×/0×	5×/0×	5×/0×	2×/0×
p6c	3×/0×	4×/0×	2×/0×	3×/0×
p6rc	0×/0×	3×/1×	2×/1×	1×/2×
p8r	2×/0×	2×/0×	2×/1×	2×/3×
p8c	1×/2×	1×/2×	1×/2×	1×/3×
p8rc	1×/1×	1×/1×	1×/2×	1×/3×
	24×/3×	31×/4×	28×/6×	25×/11×

Table 4.15: Concise results of mVNS-ILP on all Pirkwieser and Raidl instances when resetting the columns.

Instances	mVNS-ILP$_{5,10}$	mVNS-ILP$_{8,10}$	mVNS-ILP$_{10,10}$	mVNS-ILP$_{15,10}$
significantly better/worse than corresponding mVNS				
p4	4×/0×	6×/0×	10×/0×	14×/0×
p6	2×/1×	8×/0×	4×/0×	9×/0×
p8	3×/0×	2×/0×	7×/0×	12×/0×
	9×/1×	16×/0×	21×/0×	35×/0×
significantly better/worse than VNS (10^6)				
p4	14×/0×	15×/0×	15×/0×	14×/0×
p6	14×/0×	13×/0×	12×/0×	11×/0×
p8	10×/0×	7×/0×	8×/0×	8×/0×
	38×/0×	35×/0×	35×/0×	33×/0×
significantly better/worse than VNS ($2 \cdot 10^6$)				
p4	10×/0×	13×/0×	13×/0×	13×/0×
p6	6×/1×	7×/1×	6×/1×	8×/1×
p8	2×/4×	3×/6×	2×/4×	3×/8×
	18×/5×	23×/7×	21×/5×	24×/9×

documented in Table 4.16.

Nevertheless, for other instances or settings (iterations and/or VNS instances) it might pay off to reset the columns, since due to the reduced size of the resulting model potentially more improvements could be possible in limited time.

4.9.3 Hybridizing the Column Generation Approach and the Evolutionary Algorithm

The next Matheuristic we investigated is a combination of the column generation approach and the EA (denoted as CG-EA). We were motivated by the fact that columns created when solving the LP relaxation of the problem often lend themselves to quite good primal solutions when the resulting model is subsequently solved with an ILP solver. This led us to think about ways of exploiting these columns and more generally the LP information derived when performing column generation. First we apply an artificial start of the RMP by inserting the mentioned slack variables and allowing to visit no customers yet to meet the visit constraints. Next we apply the REUSE heuristic until it either does not find new columns or the amount of new columns decreased in the last five iterations. Afterwards we switch to the dynamic programming algorithm applied in a heuristic way. Each time the RMP is solved we keep track of the LP values of the columns (routes). Since in the end we want to have a predictable

Table 4.16: Average runtimes in seconds of all VNS, mVNS and mVNS-ILP variants on Pirkwieser and Raidl instances.

Method	p4	p6	p8
VNS (10^6)	25.2	29.6	30.7
VNS ($2 \cdot 10^6$)	50.0	58.9	59.1
$\text{mVNS}_{5,10}$	25.5	30.2	29.6
$\text{mVNS}_{8,10}$	25.5	30.1	29.7
$\text{mVNS}_{10,10}$	25.8	30.4	29.8
$\text{mVNS}_{15,10}$	25.9	30.6	30.2
storing all columns			
$\text{mVNS-ILP}_{5,10}$	26.3	33.3	33.7
$\text{mVNS-ILP}_{8,10}$	29.8	50.5	51.0
$\text{mVNS-ILP}_{10,10}$	34.7	58.8	58.2
$\text{mVNS-ILP}_{15,10}$	41.7	60.9	59.3
resetting columns			
$\text{mVNS-ILP}_{5,10}$	25.5	30.0	29.9
$\text{mVNS-ILP}_{8,10}$	24.1	30.9	30.7
$\text{mVNS-ILP}_{10,10}$	26.4	32.4	32.1
$\text{mVNS-ILP}_{15,10}$	29.1	42.4	51.3

running time, we limit the runtime for solving the LP relaxation. Finally we exploit the obtained data for initializing the EA. The number of nonempty routes per day is set to the rounded down sum of the LP values of all active routes of that day. These are then set to routes corresponding to active columns of that day in the last solved LP relaxation of the RMP, applying a binary tournament selection for each route according to the accumulated LP values and preferring those having a higher value, i.e. those which have proven suitable. Currently, this procedure is applied to half of the initial chromosomes, the remaining ones are initialized as described before. Although a solution created in such a way is most likely not feasible due to over- and/or under-covering, it presumably includes high quality routes advantageous for the whole gene pool.

Since we can expect that the initially created solutions will change quite soon, it seems desirable to have an ongoing exploitation of the column generation data. Preliminary experiments turned out that a simple yet effective way of achieving this is via mutation: CG-EA can additionally replace a route by another one selected from a given pool of routes. The latter is created per day and contains all corresponding routes that were at least once active in a solution to an LP relaxation of the RMP. Again, we apply a binary tournament selection using the accumulated LP values as a decision criterion. Though more sophisticated operations would certainly be possible (e.g. a more advanced initialization, exploiting the column pool in a local search, or applying re-initializations) we rather aim here at a proof of concept showing

that the data from column generation (i.e., generated variables and their (accumulated) LP values) can be successfully used to boost a metaheuristic.
We deem this hybridization, also shown in Figure 4.9, as a high-level sequential collaborative combination where the column generation guides the EA.

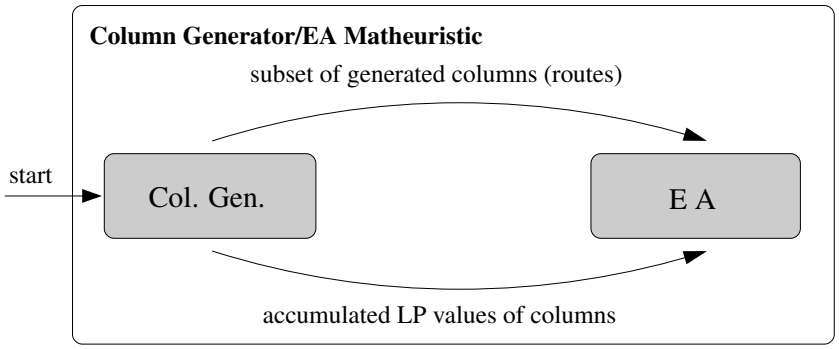


Figure 4.9: Information exchange between column generator and EA.

Column Generation Based Heuristic

To some extent the counterpart of the Matheuristics presented so far is to apply column generation and subsequently solve the final restricted master problem to integrality by a general purpose ILP solver. In both cases a certain time limit might be set. Generally this is often referred to as a *column generation based heuristic*. For comparison purposes, we examine this approach also here and denote it as CG-ILP. As for CG-EA we only consider at least once active routes. Finally having a solution to the ILP we remove redundant customers and apply our 2-opt procedure on all routes. This repair process is repeated several times ($100\times$) for the same initial solution with a randomized customer removal, naturally keeping the best solution found.

Computational Results

The results were initially reported in [162], and as we will not present further results of this method later on we give them in detail here. As mentioned in Section 4.6 we will present the results of the proposed EA together with this matheuristic variant, since the former acts as a baseline for the latter (just like [m]VNS did for [m]VNS-ILP).
The EA always applies recombination and performs mutation following a Poisson distribution with $\lambda = 1$, running for $2 \cdot 10^5$ iterations. To have equal conditions CG-EA and CG-ILP are based on the same runs of column generation, limiting the latter to 20 seconds, which suffices for many of the instances considered. Each algorithm setting is run 30 times per instance and we report average results, stating the average travel costs (avg), corresponding standard deviations (sdv), average CPU-times in seconds (t[s]), and the number of runs yielding a feasible solution (feas).

Table 4.17: Results of EA, CG-EA and CG-ILP on Pirkwieser and Raidl instances with a planning horizon of four days.

Instance	EA				CG-EA				CG-ILP			
	avg	sdv	t[s]	feas	avg	sdv	t[s]	feas	avg	sdv	t[s]	feas
p4r101	4199.14	45.00	28.7	30	4162.54	35.92	31.4	30	4119.54	15.56	28.2	30
p4r102	3784.31	33.14	27.1	30	3780.50	24.52	31.7	30	3777.63	29.37	30.0	30
p4r103	3248.05	31.50	26.9	30	3217.31	24.84	33.9	30	3258.92	44.13	33.4	30
p4r104	2691.66	36.48	28.7	30	2673.09	29.85	40.7	30	2780.10	64.70	40.2	21
p4r105	3777.90	34.10	26.7	30	3745.00	28.82	29.7	30	3801.99	33.80	29.3	30
p4c101	2918.47	12.02	30.8	30	2921.08	22.11	35.0	30	2917.91	6.32	11.9	30
p4c102	3032.23	49.41	30.1	30	2963.28	42.32	44.7	30	2925.01	49.33	41.2	30
p4c103	2874.99	54.80	31.0	30	2825.01	42.33	43.9	30	2973.94	89.65	43.6	18
p4c104	2542.46	24.39	29.2	30	2518.90	32.90	46.7	30	2479.80	24.76	46.3	30
p4c105	3072.79	86.04	29.6	30	2977.45	54.82	38.1	30	2991.24	77.13	37.6	30
p4rc101	4081.77	44.36	26.9	30	4047.87	32.44	31.4	30	4087.80	43.80	31.0	30
p4rc102	3904.33	56.09	28.3	30	3869.21	53.28	32.1	30	3870.02	35.30	31.6	30
p4rc103	3596.08	45.32	28.2	29	3549.13	33.54	34.9	29	3670.73	60.17	34.5	18
p4rc104	3142.79	37.99	29.4	30	3114.51	36.46	39.4	30	3185.14	42.46	38.9	21
p4rc105	4052.78	42.09	28.7	30	4040.32	22.11	30.8	30	4047.39	40.29	30.3	30
# sign. better		0				12						

Table 4.18: Results of EA, CG-EA and CG-ILP on Pirkwieser and Raidl instances with a planning horizon of six days.

Instance	EA				CG-EA				CG-ILP			
	avg	sdv	t[s]	feas	avg	sdv	t[s]	feas	avg	sdv	t[s]	feas
p6r101	5471.23	33.24	39.7	30	5453.07	32.60	41.4	30	5505.08	42.90	41.0	30
p6r102	5315.03	31.43	36.8	30	5318.87	25.76	43.0	30	5445.35	40.39	42.6	30
p6r103	4149.57	41.18	36.9	30	4120.37	34.46	48.5	30	4254.40	67.58	48.1	30
p6r104	3465.46	28.20	37.1	30	3441.55	22.04	53.1	30	3665.01	62.43	52.6	30
p6r105	4514.95	46.59	35.8	30	4457.93	48.46	44.0	30	4647.59	112.43	43.6	28
p6c101	4192.24	77.09	36.8	30	4162.92	68.33	50.0	30	4592.38	194.23	49.6	4
p6c102	3960.89	56.36	38.9	30	3950.54	65.92	55.2	30	4414.48	208.85	54.7	19
p6c103	3788.68	63.95	37.3	30	3719.95	82.20	55.3	30	4191.75	170.11	54.8	13
p6c104	3450.31	54.19	36.5	30	3422.22	56.05	54.5	30	3766.94	92.55	54.0	18
p6c105	4285.79	84.27	37.1	30	4181.50	56.15	53.3	30	4551.39	187.19	52.9	13
p6rc101	5932.49	46.38	34.6	30	5909.63	40.87	39.4	30	6128.02	67.00	39.0	30
p6rc102	5577.50	54.72	36.0	30	5553.47	52.54	42.3	30	5756.83	83.19	41.8	30
p6rc103	4521.57	50.34	35.4	30	4476.44	45.03	50.5	30	4699.27	67.30	50.0	23
p6rc104	4306.30	52.83	36.1	30	4267.67	41.26	50.5	30	4436.69	85.22	49.9	30
p6rc105	5467.39	58.06	35.6	30	5450.10	44.81	42.3	30	5582.21	70.66	41.8	30
# sign. better		0				10						

Table 4.19: Results of EA, CG-EA and CG-ILP on Pirkwieser and Raidl instances with a planning horizon of eight days.

Instance	EA				CG-EA				CG-ILP			
	avg	sdv	t[s]	feas	avg	sdv	t[s]	feas	avg	sdv	t[s]	feas
p8r101	6711.50	46.38	43.8	21	6696.89	75.06	53.5	27	6820.33	68.96	53.0	30
p8r102	6300.33	49.19	44.7	20	6313.65	70.20	60.8	22	6508.59	125.34	60.4	29
p8r103	4999.87	67.08	44.2	30	4930.83	43.14	61.4	30	5250.39	114.26	61.0	29
p8r104	4667.39	52.74	44.3	30	4598.77	64.23	62.4	30	5181.33	117.11	61.9	26
p8r105	5817.43	69.13	43.1	30	5744.09	53.36	58.2	30	6013.38	105.65	57.7	27
p8c101	4991.15	119.77	44.3	30	4900.84	75.41	62.0	30	5185.67	131.66	61.5	19
p8c102	5410.25	121.16	44.7	30	5308.69	114.27	64.9	30	6442.40	0.00	64.5	1
p8c103	5029.64	105.63	44.3	30	4965.95	69.92	62.4	30	6428.76	272.02	61.9	13
p8c104	5234.18	79.30	41.5	30	5202.05	91.42	60.4	30	5592.48	254.28	59.9	5
p8c105	5434.17	91.13	45.6	30	5384.95	95.36	61.7	30	6293.36	254.71	61.2	14
p8rc101	7225.04	98.40	43.2	30	7134.84	80.09	55.1	29	7432.93	152.95	54.6	20
p8rc102	6249.95	102.21	41.8	30	6163.02	76.37	60.5	28	6392.33	149.02	60.1	23
p8rc103	5847.79	95.54	44.7	30	5778.00	73.80	61.7	30	6126.24	128.82	61.3	8
p8rc104	5301.08	52.53	43.2	30	5277.23	46.59	60.3	30	5873.20	144.27	59.9	25
p8rc105	6606.78	79.69	42.9	30	6530.36	62.94	58.5	30	6727.78	120.43	58.0	30
# sign. better		0				12						

Tables 4.17, 4.18, and 4.19 give results for instances having a planning horizon of four, six, and eight days, respectively. In the bottom line we state the number of times the EA is significantly better than CG-EA and vice versa. Significantly better results on a per-instance basis are further underlined, where as usual we used a Wilcoxon rank sum test with an error level of 5% for testing statistical significance. On average CG-EA yields a significant improvement over EA in about 80% of all cases. Further, this relative improvement is even more consistent, i.e. also for longer planning horizons, than the one of [m]VNS-ILP to [m]VNS, which is particularly interesting from a methodical point of view. However, comparing the absolute results would clearly be in favor of the VNS based hybrids, since VNS is also clearly superior to the EA.

Naturally CG-EA takes more time for solving than EA. To account for this we also compared the results of CG-EA to additional pure EA runs with $3 \cdot 10^5$ iterations, i.e. an increase of 50%, which in contrast results often in (much) more allotted runtime especially for smaller instances. Nevertheless, CG-EA was still significantly better in 13, 8, and 12 cases (instead of 12, 10, and 12 cases with the shorter EA runs; see the bottom line of the tables) and the prolonged pure EA did never outperform CG-EA. We were also interested in the effect of the ongoing exploitation via mutation, so we did a comparison to test runs without this feature: utilizing it yields in 9 out of 45 times (20%) a significant improvement, and only once (2%) a significantly worse result is obtained.

Finally we compared VNS-ILP$_{8,5\%}$ (see Section 4.9.1) and CG-EA to CG-ILP: The latter only is significantly better as VNS-ILP for instance p4c102 and as CG-EA for instances p4r101, p4c102, and p4c104. Essentially, it is only reasonable when the resulting ILP model

is not too large (in fact only for p4 instances), otherwise its performance deteriorates quickly.

4.9.4 Concepts of Other Investigated Hybridizations

During our project we also investigated some other potential matheuristics. Unfortunately their results were not convincing after all, so they did not appear in any publication so far. In part we also tried some known concepts, e.g. deriving a primal heuristic via applying column generation combined with some variable fixing procedure. Nonetheless, in the following we will present two (in our opinion) novel matheuristics whose main ideas might still be of interest.

Hybridizing the VNS and the Branch-and-Cut-and-Price Approach

For this matheuristic variant we tried to hybridize the VNS and the Branch-and-Cut-and-Price (BCP) approach in a meaningful way. Since (our) BCP is generally applicable on rather small instances only we need to "truncate" it in a suitable way to apply it on reasonably sized instances as well. In a first attempt this was achieved via restricting the underlying network to arcs which occurred in feasible solutions found during a preceding VNS run. Furthermore, feasible solutions' routes can also be utilized as initial set of columns. Of course, the best incumbent can be applied as starting solution here, too. The overall approach is basically similar to the VNS-ILP based matheuristics presented in previous sections, yet with the difference and potential advantage that new routes can be generated on demand in the process of solving. In this sequential collaborative combination the VNS guides the BCP approach.

We tested this variant on the Pirkwieser and Raidl instances applying BCP for the same amount of time than VNS. Unfortunately we observed only a small and not satisfying improvement on the instances having a planning horizon of four days. For longer planning horizons no improvements could be achieved at all. Though regarding the column generation, we even did not apply the full dominance rule when solving the pricing subproblem, but start with one of the heuristics and later switch to the incomplete dominance rule instead. Nevertheless column generation is definitely the bottleneck here, paired with the increasing time to solve the master problem for larger planning horizons.

In a second attempt we further tried to apply BCP with some additional constraints similar to local branching, e.g. limiting the change of at most k visit combinations. However, the results were still not convincing enough and we therefore abandoned this line of research.

Hybridizing the Column Generation Approach and the VNS

A somewhat opposite approach than in the previous section is based on column generation and VNS. Our initial intention was to come up with a more fine grained utilization of the information gathered when solving the LP relaxation of the problem compared to the CG-EA matheuristic reported in Section 4.9.3. The latter mainly considers whole routes only, whereas here we increase the level of detail and consider single arcs. For doing so we start by (partly) solving the LP relaxation of the problem, again using a time limit as for

CG-EA, keeping track of all arcs which occurred in intermediate solutions of the master problem. These arcs are then favored in the VNS via penalizing all remaining arcs by a small amount; obtaining a sequential collaborative combination where the VNS is guided by the CG approach. Since we do not want to focus VNS too much on these arcs the penalty should not be set too high. As a baseline for comparison we decided to apply VNS with penalizing all but the 5% least cost arcs per customer. Running the same tests than previously this approach was not able to yield any improvement against the baseline. Although we observed that the number of favored arcs contained in the final solution is often around 80% (but usually quite similar even without favoring arcs), apparently solving the LP relexation gives no clue about the "remaining" arcs necessary to obtain low-cost solutions. Of course one could solve the LP relaxation to optimality (i.e. switching to the full dominance rule without time restrictions at all), but since we also wanted an approach which is competitive to pure VNS when it comes to runtime, this was not an option.

4.10 Latest Computational Results

These additional tests were performed in order to finally compare our developed methods to other recently proposed approaches which were not available when we published/presented our original work. A major reason why we did not just take our old results for comparison is that some of the new results were obtained with considerable longer runtimes. To have a fair(er) comparison we set the number of iterations such that we have similar runtimes as in the most recent tests, though clearly oriented towards the faster methods. Therefore, deriving (most likely) even better results at the cost of excessive runtimes was strictly not our goal.

Together with the previous results, more concerned with comparing different variants of our own methods, these new results should give a nice, complete picture.

First some words about common settings. All algorithms have been implemented in C++, compiled with GCC 4.5, and performed on a single core of an Intel Xenon E5540 with 2.53 GHz and 3 GB RAM dedicated per core.

Initially, and for all previous experimental results, we set the initial temperature T_0 to 10 and fixed the penalty weights to 100. Ideally these parameters should be set for each instance individually. So we opted for an automated setting and after some experiments we decided to use the following: we consider the average inter-customer travel costs c_{avg} and set $T_0 = c_{\text{avg}}/5$ and the penalty weights directly to c_{avg}. This seems to accommodate in a suitable way for the different instance characteristics. The only setting we did not (try to) automate is the probability of applying 2-opt* whenever the newly derived solution lies within 2% to the current incumbent solution, in the remainder denoted as $P_{\text{2-opt}^*}$. It is set such that 2-opt* does not consume too much runtime and is therefore subject to experimentation. The value used will be given for each instance set.

Since there is a manageable number of works to compare to we will state them already here and omit to mention them more than once later on. At first the tabu search of Cordeau et al. [41], as well as the same method when utilizing forward time slack [42] (TS), tests were run on a Sun Ultra 2 with 300 MHz and a Pentium 4 with 2 MHz, respectively. The next

three are very recent methods: a hybrid genetic algorithm by Nguyen et al. [150] (HGA), run on an Intel Core2 Duo with 2.4 GHz, a hybrid genetic search with adaptive diversity management by Vidal et al. [227] (HGSADC), run on an Intel Xeon X7350 with 2.93 GHz, and a parallel iterated tabu search heuristic by Cordeau and Maischberger [43] (ITS), the sequential algorithm run on an Intel Xeon X7350 with 2.93 GHz and the parallel variant on a Linux cluster with 128 nodes and Infiniband interconnections, each node being equipped with dual Intel Xeon E5472 with 3 GHz. According to a CPU benchmarking site[1] the environments of HGSADC, ITS and ours have quite a similar performance (at least similar enough to more or less also compare absolute runtimes), those of HGA is about 2.7 times slower, and those of the TS in [42] is about 17.7 times slower, those of TS in [41] could not be resolved.

If not stated otherwise we perform for each algorithm setting 10 runs per instance, and state the costs of the best run (min), the average costs (avg), corresponding standard deviations (sdv), and the runtimes in minutes (t[m]).

4.10.1 Cordeau et al. Instances

For the Cordeau et al. instances we chose $2 \cdot 10^7$ VNS iterations to yield a comparable runtime. For the mVNS we set 5 VNS instances and 40 sections. We also experimented with more VNS instances (similar as reported for the previous results) but here we experienced that due to the size of the Cordeau et al. instances mVNS hardly leads to a gain when choosing a too high number, it then rather yields worse results than standard VNS. In Section 4.5 we have already reported similar findings and explanations for the Pirkwieser and Raidl instances, yet here an even smaller number of VNS instances must be used. The probability $P_{\text{2-opt}^*}$ is set to $0.05 \cdot \sqrt{288/n}$, which automatically promotes to apply 2-opt* when dealing with less customers, yet retains an acceptable amount of time spent on it for the largest instances.

The results are shown in Table 4.20 for the runs without forward slack time and in Table 4.21 with forward slack time. For both variants the best found solutions over all runs are given in Table 4.22. Finally, Table 4.23 shows a comparison of all approaches we are aware of. Note that for ITS we report the parallel runs using 8 and 64 processors, denoted as ITS/8 and ITS/64, respectively. For the setting without forward slack time our methods clearly perform best, both on average and also yielding 15 new best known solutions for the 20 instances, despite HGA consuming a lot of runtime (which even when taking the different computing environments into account still takes 10 times as long). The situation with forward slack time is more competitive, again apart from the TS, which, given the time of its publication, is not surprising. For this setting our methods seem slightly superior to ITS, but are inferior to HGSADC, though the difference is not that much. However, when also taking the runtimes into account (remember that ITS, HGSADC and our methods can be roughly compared with regard to this), HGSADC and our methods are currently performing best. As seen for the per instance as well as for the average results mVNS performs better than VNS, though the difference in the resulting gaps is rather small. Finally, we observed that the performance of our methods differs to some extent depending on the type of the time windows, first looking

[1] see at `http://www.cpubenchmark.net/` [accessed September 12, 2011]

Table 4.20: Results of VNS and mVNS on Cordeau et al. instances, indicating in bold when a method is significantly better than the other.

Instance	VNS				mVNS			
	min	avg	sdv	t[m]	min	avg	sdv	t[m]
p01a	3003.68	3009.47	2.05	10.2	2989.58	**2999.19**	8.23	11.0
p02a	5111.26	5135.39	18.94	14.1	5116.92	5130.19	13.00	14.9
p03a	7164.91	7234.44	40.55	17.7	7174.10	**7199.93**	18.24	19.2
p04a	7973.73	8001.80	29.43	23.8	7947.20	**7978.28**	19.95	26.9
p05a	8542.57	8605.95	32.69	29.5	8548.77	8587.36	38.65	36.2
p06a	10814.55	10870.19	42.71	35.8	10786.32	10864.94	51.72	43.5
p07a	6900.71	6916.05	11.14	12.3	6897.06	**6909.91**	10.82	12.9
p08a	9741.41	9792.09	33.67	26.4	9736.26	9778.75	29.46	28.1
p09a	13595.43	13704.59	65.77	39.4	13650.64	13700.67	29.37	45.4
p10a	17544.93	17716.05	114.95	58.1	17621.26	17751.66	88.29	71.1
p01b	2289.17	2289.17	0.01	11.4	2289.17	2289.17	0.00	11.6
p02b	4144.02	4156.39	13.93	16.5	4143.58	4156.32	9.99	16.8
p03b	5576.24	5614.76	19.49	20.6	5559.81	**5590.34**	24.17	21.6
p04b	6477.98	6511.02	30.65	26.2	6458.35	6501.23	28.83	28.2
p05b	6870.05	**6905.79**	23.46	28.2	6890.43	6934.42	31.18	30.7
p06b	8767.55	**8799.67**	28.42	36.1	8782.21	8825.12	29.00	41.3
p07b	5564.95	5570.36	5.01	14.9	5505.54	**5531.65**	26.18	16.4
p08b	7655.80	7689.87	17.83	27.4	7663.03	7695.00	23.77	28.5
p09b	10701.65	10785.54	51.99	41.8	10714.69	10783.99	45.31	46.3
p10b	13630.11	**13728.00**	89.09	49.4	13685.63	13800.38	52.18	59.2

at the results with forward slack time. For narrow time windows (p01a–p10a) the average gap of VNS (mVNS) is 0.39% (0.74%), while for the larger time windows (p01b–p10b) it is 0.68% (1.20%), while e.g. for ITS/64 it is 1.12% and 1.09%, respectively, and for HGSADC it is 0.58% and 0.68%, respectively. However, without forward slack time it is 0.27% and 0.01% for VNS, and 0.11% and 0.01% for mVNS, so here they perform slightly better for the larger time windows. HGA yields respective average gaps of 0.35% and 0.36%. Ultimately our VNS approach seems more sensitive to the type (i.e. size) of the time windows than other approaches, though we could not find an obvious reason for it.

4.10.2 Pirkwieser and Raidl Instances

For these instances we set $P_{\text{2-opt}^*} = 0.001$ and the number of VNS iterations to 10^7. mVNS as well as mVNS-ILP is run with 20 VNS instances and the number of sections is 40. VNS-

Table 4.21: Results of VNS and mVNS on Cordeau et al. instances using forward slack time, indicating in bold when a method is significantly better than the other.

Instance	VNS				mVNS			
	min	avg	sdv	t[m]	min	avg	sdv	t[m]
p01a	2909.02	2909.73	1.01	10.2	2909.02	**2909.25**	0.75	10.1
p02a	5030.82	5033.71	3.65	14.2	5027.35	**5030.60**	1.64	14.2
p03a	7099.29	7145.57	25.27	18.1	7079.72	**7114.52**	21.51	18.6
p04a	7807.81	7846.35	22.34	25.1	7785.81	**7812.90**	15.92	25.9
p05a	8354.43	8398.74	32.10	29.5	8364.64	8404.38	32.43	30.8
p06a	10491.49	10564.94	45.65	37.8	10537.23	10573.96	26.31	39.1
p07a	6793.98	6807.24	10.04	13.1	6783.23	**6794.33**	6.08	13.3
p08a	9616.21	9652.67	28.67	27.2	9583.33	**9612.29**	18.90	28.3
p09a	13191.03	13316.99	60.75	42.6	13229.63	13292.82	35.61	45.0
p10a	17070.65	17206.95	66.59	63.2	17148.05	17255.35	68.90	66.1
p01b	2277.44	2277.44	0.00	11.1	2277.44	2277.44	0.00	11.1
p02b	4129.05	4163.33	20.96	16.0	4122.03	**4143.71**	8.99	16.2
p03b	5549.89	5576.66	14.82	20.5	5520.27	5569.34	25.71	20.8
p04b	6387.43	6428.41	26.23	26.2	6384.05	6432.29	24.36	27.2
p05b	6855.49	6911.79	36.07	27.5	6852.85	6890.37	23.01	28.7
p06b	8671.78	8716.33	29.45	37.0	8659.46	8723.71	32.67	37.4
p07b	5483.05	5509.16	15.24	15.8	5481.61	**5487.79**	8.99	16.7
p08b	7635.45	7690.02	29.21	27.5	7646.73	7675.60	18.92	29.0
p09b	10644.80	**10709.60**	30.77	41.2	10664.02	10732.57	39.25	42.5
p10b	13552.75	13651.27	74.08	48.1	13587.45	13717.26	74.02	50.1

ILP is applied similar, i.e. there are also 40 sections of VNS applications and every time 20 solutions are extracted and inserted in the ILP model as described in Section 4.9.1, considering intermediate solutions deviating at most 5% from the incumbent solution. Both VNS-ILP and mVNS-ILP reset the columns after each ILP application, otherwise the model would grow too large due to the 40 sections and hence model extensions.

The detailed results on the instances having a planning horizon of four, six, and eight days are shown in Table 4.24, 4.25, and 4.26, respectively. The best found solutions over all runs with the corresponding gaps to the BKS are reported in Table 4.27. The eagerly awaited comparison of the methods which were applied on these instances, given per planning horizon, is given in Table 4.28. Standard VNS delivers definitely usable but not yet convincing results. Fortunately all extensions of it improve upon its performance also for these testruns. We observe that mVNS yields a higher improvement upon VNS than does VNS-ILP, so we

Table 4.22: Stating the best results obtained by our algorithms over all runs for the Cordeau et al. instances without and with forward slack time and the corresponding percentage gaps to the best known solutions (BKS), indicating newly found ones in bold.

Instance	w/o forward slack time		with forward slack time	
	best	%-gap to BKS	best	%-gap to BKS
p01a	2989.58	0.00	2909.02	0.00
p02a	5111.26	0.07	5027.35	0.02
p03a	7164.91	0.09	7040.73	0.24
p04a	**7947.20**	-0.43	7785.81	0.39
p05a	**8542.57**	-0.49	8332.33	0.25
p06a	**10786.32**	-1.37	10491.49	0.17
p07a	6897.06	0.06	6783.23	0.01
p08a	**9736.26**	-0.16	9583.33	0.09
p09a	**13595.43**	-0.82	**13191.03**	-0.08
p10a	**17544.93**	-1.18	17012.41	0.54
p01b	2284.83	0.00	2277.44	0.00
p02b	**4139.20**	-0.05	4122.03	0.01
p03b	**5559.81**	-0.13	5520.27	0.56
p04b	**6458.35**	-0.21	6381.02	0.52
p05b	**6870.05**	-1.34	6820.12	0.63
p06b	**8767.55**	-1.00	8638.34	0.65
p07b	**5505.54**	-0.06	5481.61	0.00
p08b	**7647.04**	-0.40	7635.45	0.48
p09b	**10701.65**	-1.59	10634.21	0.97
p10b	**13630.11**	-1.60	13543.20	1.02
avg		-0.53		0.32

Table 4.23: Comparison to other methods applied on the Cordeau et al. instances. Note that for ITS we state the absolute computational effort of 8 and 64 search threads running for 7.60 and 11.32 minutes each, respectively.

Method	runs	w/o forward slack time			with forward slack time		
		%-gap avg	%-gap min	t[m] avg	%-gap avg	%-gap min	t[m] avg
TS [41]	11	–	3.31	651.00	–	–	–
TS [42]	10	–	–	–	–	2.74	160.00
ITS/8 [43]	10	–	–	–	1.77	–	60.80
ITS/64 [43]	5	–	–	–	1.10	–	724.27
HGA [150]	30	0.35	0.00	654.09	–	–	–
HGSADC [227]	10	–	–	–	0.63	0.22	32.74
VNS	10	0.14	-0.37	27.00	1.07	0.53	27.59
mVNS	10	0.06	-0.40	30.48	0.97	0.52	28.56

see again that the multiple VNS instances do pay off for these rather small-sized instances. Further adding the ILP model as an ingredient, obtaining mVNS-ILP, achieves the necessary boost to obtain a competitive approach in the end. As per setting the runtime is similar to HGSADC, while HGA again consumes considerable more runtime. While it can be assumed that mVNS-ILP and HGSADC perform similar on the p4 and p6 instances, a small, but noticeable gap can be observed for the p8 instances at the expense of mVNS-ILP. However, the average gap over all instances (p4, p6, and p8) of HGSADC is 0.48% and that of mVNS-ILP is 0.56%, so the difference is marginal and most likely not significant. For completeness: that of HGA is 0.65%, hence also showing a good performance.

4.10.3 Vidal et al. Instances

Finally, we also consider the new, large-scale instances of Vidal et al. Similar overall runtimes could be achieved when running the VNS for $4 \cdot 10^7$ iterations. Due to this and the fact that prematurely stopping the VNS is problematic because of the applied linear cooling we omitted to restrict the runtime of a single run to five hours as in [227]. As the instances are considerably larger, we also slightly modified 2-opt* to obtain the desired runtimes yet still taking advantage of its improvement capabilities: on the one hand one third of all route pairs is simply skipped (determined at random for each pair), and on the other hand for each pair at most five improving moves are performed as opposed to re-applying until a local optimum is reached. Further, $P_{\text{2-opt}^*}$ is set to 0.02.

Note that since these instances were so far only tackled in [227], the best solution out of their five runs is also the best known solution for the corresponding instance. Here we also only performed five runs, showing the results of VNS in Table 4.29. Unfortunately the instance

Table 4.24: Results of VNS, mVNS and corresponding hybrid variants on Pirkwieser and Raidl instances with a planning horizon of four days.

Instance	VNS				mVNS				VNS-ILP				mVNS-ILP			
	best	avg	sdv	t[m]	best	avg	sdv	t[m]	best	avg	sdv	t[m]	best	avg	sdv	t[m]
p4r101	4088.90	4097.90	11.35	5.1	4087.10	4092.54	4.99	4.7	4082.00	4087.49	4.89	4.1	4082.00	4083.19	1.92	4.2
p4r102	3732.60	3742.91	8.86	5.3	3725.60	3730.40	2.22	5.1	3725.20	3730.19	3.43	4.6	3725.60	3728.13	2.32	4.8
p4r103	3162.70	3175.66	7.63	4.9	3153.90	3161.09	4.16	5.1	3155.80	3165.05	4.74	5.2	3153.90	3158.44	5.06	5.5
p4r104	2584.40	2592.03	5.56	5.3	2576.90	2580.40	5.18	6.5	2572.20	2580.45	8.04	6.8	2574.20	2578.03	3.01	7.2
p4r105	3649.40	3662.12	8.16	4.6	3647.30	3653.03	4.77	4.6	3643.90	3651.02	4.49	4.8	3642.20	3649.64	4.08	5.5
p4c101	2910.20	2910.20	0.00	5.6	2908.10	2909.48	0.99	5.2	2907.60	2909.15	1.07	5.4	2907.40	2908.57	0.97	5.3
p4c102	2896.90	2926.84	26.85	5.5	2882.90	2889.17	8.30	6.0	2885.80	2936.70	33.10	6.0	2882.90	2890.44	4.91	6.3
p4c103	2737.90	2753.72	17.77	5.6	2736.10	2741.06	5.88	6.9	2735.60	2754.76	22.67	7.2	2734.50	2736.57	1.66	7.3
p4c104	2431.90	2450.77	9.65	5.4	2432.20	2435.00	2.10	6.7	2423.40	2437.19	11.21	7.2	2422.30	2432.49	4.82	7.1
p4c105	2889.80	2913.09	26.00	5.2	2889.20	2894.81	10.50	5.4	2889.80	2913.03	23.16	5.5	2884.10	2889.51	2.99	5.8
p4rc101	3956.80	3965.42	6.67	4.7	3956.50	3959.54	3.13	4.8	3956.50	3960.80	3.98	4.8	3956.40	3957.89	1.88	5.1
p4rc102	3760.00	3781.32	9.93	4.9	3755.80	3763.26	4.00	5.0	3755.80	3766.49	7.09	5.1	3755.80	3759.07	4.05	5.4
p4rc103	3456.50	3475.81	15.56	5.3	3450.80	3456.61	4.15	6.2	3444.60	3455.45	5.53	6.2	3448.80	3455.16	3.95	6.5
p4rc104	3009.70	3028.49	12.77	5.3	2999.40	3012.00	9.33	6.6	2991.50	3001.22	7.14	7.2	2994.80	3002.77	6.65	7.4
p4rc105	3949.80	3964.73	7.78	4.9	3951.60	3964.66	6.65	5.0	3951.50	3958.33	4.42	5.0	3941.50	3954.07	8.72	5.2

Table 4.25: Results of VNS, mVNS and corresponding hybrid variants on Pirkwieser and Raidl instances with a planning horizon of six days.

Instance	VNS				mVNS				VNS-ILP				mVNS-ILP			
	best	avg	sdv	t[m]	best	avg	sdv	t[m]	best	avg	sdv	t[m]	best	avg	sdv	t[m]
p6r101	5388.20	5403.35	10.20	6.3	5379.30	5388.72	6.74	5.5	5376.60	5380.32	3.56	4.8	5376.80	5378.62	1.01	5.1
p6r102	5210.90	5232.42	16.40	6.3	5212.60	5223.23	6.92	6.3	5203.90	5212.15	4.64	6.6	5201.60	5206.83	3.84	7.3
p6r103	3995.60	4015.34	15.30	5.7	3971.30	3981.87	7.95	5.8	3958.20	3977.35	15.73	6.4	3956.90	3966.97	5.18	7.0
p6r104	3349.10	3374.74	13.35	5.5	3344.70	3349.21	3.66	6.0	3344.80	3348.91	4.01	7.8	3342.50	3350.42	7.07	7.7
p6r105	4293.20	4328.79	23.23	5.1	4296.10	4306.65	6.92	5.2	4279.60	4315.25	23.43	5.6	4290.00	4301.25	8.12	6.6
p6c101	3988.50	4017.93	31.25	5.4	3984.30	4002.09	15.45	5.7	3981.40	4003.33	22.15	6.2	3984.30	3994.15	7.44	7.2
p6c102	3844.80	3858.23	14.03	5.7	3843.80	3847.34	3.50	6.2	3841.90	3857.74	12.79	6.9	3843.50	3846.99	1.76	7.5
p6c103	3528.50	3556.05	28.99	5.7	3531.20	3541.99	5.92	7.1	3531.00	3546.94	18.52	8.5	3528.90	3538.28	8.56	9.1
p6c104	3240.10	3262.01	20.23	5.7	3232.60	3240.74	7.64	6.9	3224.10	3244.38	15.96	9.8	3222.60	3231.88	7.40	9.3
p6c105	4053.40	4085.89	29.92	5.5	4052.10	4062.28	10.48	5.8	4054.80	4090.01	31.11	6.6	4052.10	4063.09	8.17	7.5
p6rc101	5798.20	5815.92	15.83	5.2	5780.70	5793.53	8.96	5.0	5782.30	5791.77	9.06	5.5	5781.50	5788.11	5.34	6.3
p6rc102	5380.50	5426.17	25.88	5.1	5363.70	5382.69	15.74	5.2	5355.10	5407.37	32.02	6.0	5356.70	5374.30	11.92	6.9
p6rc103	4298.00	4317.66	13.36	5.5	4299.30	4311.74	9.65	6.2	4292.00	4313.04	15.24	7.7	4286.30	4311.02	13.84	8.1
p6rc104	4099.10	4124.24	18.08	5.6	4093.70	4097.89	2.46	6.4	4084.90	4093.89	7.44	8.9	4078.80	4089.96	6.32	8.8
p6rc105	5254.90	5289.91	23.31	5.2	5252.30	5275.35	11.88	5.3	5235.30	5269.53	21.32	5.9	5258.40	5274.01	10.63	6.9

Table 4.26: Results of VNS, mVNS and corresponding hybrid variants on Pirkwieser and Raidl instances with a planning horizon of eight days.

Instance	VNS				mVNS				VNS-ILP				mVNS-ILP			
	best	avg	sdv	t[m]	best	avg	sdv	t[m]	best	avg	sdv	t[m]	best	avg	sdv	t[m]
p8r101	6493.00	6526.76	23.19	5.9	6497.30	6512.07	7.53	5.5	6477.40	6518.43	27.46	5.1	6483.30	6492.53	9.13	6.0
p8r102	6118.00	6172.88	40.24	5.8	6124.20	6136.80	9.90	5.7	6113.10	6162.90	26.46	5.8	6099.00	6111.24	9.05	6.6
p8r103	4729.30	4767.72	38.36	5.7	4718.60	4750.18	17.82	6.0	4726.80	4750.76	22.20	6.9	4707.60	4742.58	16.60	8.2
p8r104	4403.30	4446.79	25.45	5.9	4385.00	4408.98	15.29	6.4	4414.00	4436.99	11.34	7.8	4367.80	4412.85	21.15	8.6
p8r105	5512.80	5538.68	23.80	5.3	5508.70	5528.55	18.09	5.2	5505.00	5529.58	20.88	5.7	5473.20	5515.75	20.69	6.9
p8c101	4720.10	4739.22	19.93	5.5	4712.60	4721.21	6.11	5.8	4706.20	4729.64	16.17	6.2	4679.10	4704.49	15.11	7.1
p8c102	4975.10	5034.18	33.87	6.0	4935.30	5008.60	34.92	7.1	5006.40	5057.97	38.60	7.6	4978.00	5010.55	15.27	9.7
p8c103	4673.40	4713.30	27.03	6.4	4683.90	4702.19	12.87	7.2	4680.80	4710.77	32.11	8.3	4674.30	4694.94	13.85	9.8
p8c104	4651.10	4694.46	39.49	4.9	4683.80	4742.23	33.57	4.6	4632.50	4720.34	54.27	6.0	4670.20	4697.85	21.30	7.2
p8c105	5150.70	5192.89	36.52	5.5	5143.00	5159.72	13.58	5.7	5155.50	5181.64	24.62	6.1	5134.60	5149.64	11.54	7.2
p8rc101	6902.90	6963.27	53.41	5.3	6893.10	6913.55	13.80	5.1	6905.90	6992.81	68.97	6.1	6876.80	6900.55	14.69	6.7
p8rc102	5799.10	5870.50	57.81	5.2	5812.70	5847.72	20.78	5.4	5818.90	5838.16	13.02	5.9	5798.40	5818.96	22.26	7.0
p8rc103	5479.30	5506.70	19.18	5.5	5449.30	5482.21	17.34	5.9	5456.20	5477.45	17.86	6.9	5440.30	5486.73	26.29	7.8
p8rc104	4978.40	5019.73	24.02	5.6	4980.90	5017.85	18.91	6.0	4959.90	4999.00	22.14	7.6	4969.40	5009.05	22.07	8.4
p8rc105	6224.40	6272.30	28.79	5.4	6241.90	6271.82	18.87	5.3	6207.60	6269.19	36.50	5.8	6196.70	6253.47	26.85	7.0

Table 4.27: Stating the best results obtained by our algorithms over all runs for the Pirkwieser and Raidl instances and the percentage gaps to the best known solutions (BKS), indicating newly found ones in bold.

Instance	best	%-gap to BKS
p4r101	4082.0	0.02
p4r102	3725.2	0.02
p4r103	3153.1	0.00
p4r104	2572.2	0.24
p4r105	3641.7	0.08
p4c101	2907.4	0.00
p4c102	2882.9	0.00
p4c103	2734.5	0.00
p4c104	2422.3	0.14
p4c105	2884.1	0.00
p4rc101	**3955.0**	-0.02
p4rc102	3755.8	0.00
p4rc103	**3444.6**	-0.15
p4rc104	2991.5	0.00
p4rc105	3937.7	0.13
avg		0.03
p6r101	5376.6	0.01
p6r102	5201.6	0.00
p6r103	3949.8	0.24
p6r104	3341.0	0.16
p6r105	4279.6	0.16
p6c101	3981.2	0.00
p6c102	3841.7	0.00
p6c103	3528.5	0.14
p6c104	3222.6	0.51
p6c105	4052.1	0.00
p6rc101	**5780.3**	-0.02
p6rc102	5349.3	0.30
p6rc103	4278.3	0.12
p6rc104	4065.5	0.09
p6rc105	5235.3	0.16
avg		0.12
p8r101	6477.4	0.09
p8r102	6099.0	0.02
p8r103	4701.7	0.31
p8r104	4367.8	0.28
p8r105	5473.2	0.00
p8c101	4679.1	0.00
p8c102	4935.3	0.04
p8c103	4670.3	0.14
p8c104	4632.5	0.89
p8c105	5134.2	0.00
p8rc101	6870.9	0.35
p8rc102	5784.9	0.37
p8rc103	5440.3	0.28
p8rc104	4959.9	0.62
p8rc105	**6196.7**	-0.11
avg		0.22

Table 4.28: Comparison to other methods applied on the Pirkwieser and Raidl instances.

Method	runs	p4			p6			p8		
		%-gap avg	%-gap min	t[m] avg	%-gap avg	%-gap min	t[m] avg	%-gap avg	%-gap min	t[m] avg
HGA [150]	30	0.36	0.06	70.54	0.60	0.30	86.39	1.00	0.43	97.51
HGSADC [227]	10	0.25	0.05	3.21	0.47	0.17	4.44	0.71	0.19	5.06
VNS	10	0.78	0.30	5.17	1.10	0.49	5.56	1.52	0.71	5.60
mVNS	10	0.34	0.16	5.57	0.62	0.36	5.90	1.22	0.66	5.79
VNS-ILP	10	0.50	0.09	5.70	0.70	0.22	6.90	1.42	0.66	6.50
mVNS-ILP	10	0.23	0.06	5.90	0.48	0.24	7.40	0.97	0.39	7.60

sizes do not allow to apply mVNS in a meaningful way (as discussed before). On average HGSADC and VNS yield a more or less equal performance, the slightly reduced gaps of VNS are marginal and perhaps more a by-product of fluctuation. Still, it is interesting that VNS performs consistently better than HGSADC on instances having narrow time windows (p11a–p24a), but this good performance is canceled on a global view by the consistently worse performance than HGSADC on instances having larger time windows (p11b–p24b). This is also of interest as in [227] especially the good performance of HGSADC on narrow time windows was observed in general. It appears that this performance-bias is even more distinct for the VNS, which was less evident for the Cordeau et al. instances in the previous section.

4.11 Conclusions

In this line of work, which was part of a larger research project, we aimed at deriving new, mainly heuristic solution approaches for periodic vehicle routing problems (PVRPs), especially investigating hybrid methods incorporating integer linear programming techniques. PVRPs are a generalized variant of the classical VRP where some customers have to be served more than once during a given planning horizon. On our side we concentrated on the periodic vehicle routing problem with time windows (PVRPTW), which was at that time barely treated in the literature.

First we presented a variable neighborhood search (VNS) metaheuristic for which we considered other successful VNS solution approaches developed for similar problems and combined, adapted, and extended some of their concepts. A selectively applied simple inter-route improvement procedure, 2-opt*, was shown to considerably improve the performance with only moderate computational effort. Back then the good performance of our method was demonstrated when comparing our best solution values to the previously best known solution values of a tabu search heuristic, yielding an average improvement of around 1–3%, and in some cases even more. Latest results indicate that the VNS is very competitive also to the best recent approaches. On recently proposed large-scale instances it is further able to improve upon all instances having narrow time windows.

Later we extended our previously introduced (standard) VNS to a multiple VNS (mVNS) where several VNS instances are applied cooperatively in an intertwined way. This mVNS puts emphasis on the so far best solution found within a major iteration by restarting the worst performing VNS instances with it. In this way, mVNS investigates multiple search trajectories from incumbent solutions, and from a global perspective it can be seen to adaptively allocate VNS instances to promising areas of the search space. On all but the large-scale instances mVNS outperforms VNS, especially on the smaller instances almost always significantly. The problem is that with increasing instance size it is better to devote all resources (time or iterations) to a single search instance. We note that it might be interesting to parallelize the multiple VNS instances as was done in [43] with a tabu search.

Another metaheuristic solution approach is based on evolutionary algorithms, which was the first of its kind for PVRPs. It applies several variants of recombination and mutation which operate either on the visit combinations or on the routes. Though it can quite reasonably

Table 4.29: Results on the Vidal et al. large-scale instances, indicating newly derived BKS in bold.

Id	HGSADC			VNS				
	min(BKS)	avg	t[m]	min	%-gap	avg	%-gap	t[m]
p11a	20937.29	21120.94	61.74	**20821.94**	-0.55	20905.09	-0.15	98.63
p12a	26483.68	26677.56	192.16	**26268.44**	-0.81	26478.13	-0.02	143.98
p13a	31808.00	31909.12	297.03	**31507.25**	-0.95	31596.49	-0.66	190.65
p14a	36954.39	37066.65	302.25	**36562.98**	-1.06	36714.39	-0.65	214.95
p15a	41699.07	41847.30	301.05	**41456.06**	-0.58	41547.87	-0.36	300.16
p16a	48375.16	48855.14	307.29	**48317.28**	-0.12	48479.86	0.22	347.06
p17a	28818.04	28889.82	65.28	**28506.40**	-1.08	28591.99	-0.78	107.49
p18a	37385.82	37491.40	263.63	**36956.66**	-1.15	37117.37	-0.72	169.17
p19a	48993.72	49103.78	300.39	**48552.08**	-0.90	48722.98	-0.55	240.02
p20a	60144.66	60474.34	302.59	**59699.33**	-0.74	59859.09	-0.47	331.20
p21a	54257.26	54562.68	213.11	**53716.96**	-1.00	54007.93	-0.46	167.09
p22a	72978.33	73226.99	297.44	**72258.33**	-0.99	72782.13	-0.27	264.72
p23a	90951.34	91424.98	300.02	**90286.66**	-0.73	90822.07	-0.14	365.33
p24a	114712.30	114892.01	308.38	**113472.06**	-1.08	113931.60	-0.68	532.38
avg %-gap	0.00	0.46			-0.84		-0.41	
avg t[m]			250.88					248.06
p11b	15992.20	16102.27	86.18	16098.59	0.67	16170.47	1.11	95.69
p12b	20753.17	20822.71	177.36	20799.53	0.22	20854.92	0.49	129.66
p13b	24972.94	25050.30	291.83	25121.22	0.59	25222.78	1.00	165.83
p14b	29790.14	29976.52	301.40	30249.10	1.54	30394.71	2.03	196.82
p15b	41609.04	41715.58	300.02	41759.56	0.36	41927.07	0.76	269.15
p16b	49470.50	49558.36	306.31	49587.56	0.24	49897.40	0.86	310.60
p17b	22989.05	23138.63	94.09	23015.78	0.12	23113.75	0.54	95.44
p18b	32093.04	32201.55	274.33	32396.99	0.95	32498.33	1.26	156.05
p19b	42332.28	42467.74	300.53	42884.86	1.31	42990.30	1.55	206.80
p20b	52863.23	53119.63	302.15	53570.09	1.34	53706.27	1.59	295.29
p21b	43098.26	43195.88	261.51	43179.71	0.19	43427.55	0.76	158.31
p22b	58814.76	58942.49	303.13	59051.31	0.40	59208.51	0.67	236.30
p23b	74357.84	74755.07	302.99	75476.59	1.50	75611.83	1.69	353.33
p24b	94395.56	94551.24	303.18	95351.52	1.01	95920.31	1.62	457.70
avg %-gap	0.00	0.38			0.75		1.14	
avg t[m]			257.50					223.35
avg %-gap	0.00	0.42			-0.05		0.37	
avg t[m]			254.19					235.71

solve the problem, it is clearly not competitive to the VNS variants. It was mainly derived to investigate a special matheuristic variant, mentioned later.

We further investigated an ILP formulation based on a set-covering model. The LP relaxation is solved via column generation, where the pricing subproblem resembles an elementary shortest path problem with resource constraints (ESPPRC) for which we apply an exact dynamic programming algorithm. An important issue are the simultaneous restrictions on time windows as well as on route duration. For this purpose we proposed suitable label resources, their extension function, and dominance rules, incorporating the concept of forward time slack in order to minimize overall route duration. Furthermore a GRASP-based metaheuristic as well as the REUSE entitled method is proposed for heuristically solving the ESPPRC. The latter uses active routes in the previous LP solution as a starting point to find negative reduced columns in a faster way. Computational results indicate the advantage of applying a forced early stop as well as the metaheuristics–especially REUSE–in combination with the dynamic programming algorithm to often drastically reduce computation time. For many instances only small remaining gaps were achieved. In contrast to the labeling algorithm the developed ESPPRC heuristics are not sensitive to the dual variable values, which is primarily of advantage at the beginning of the process. Due to this they further more easily allow to start with no initial columns, but using slack variables instead. Building upon this we embedded this column generation approach in a branch-and-price framework via proposing suitable branching rules. It is also extended to branch-and-cut-and-price by introducing 2-path cuts which were adapted to our problem as well as the recently proposed subset-row inequalities. Tests on smaller instances with 36 and 50 customers show that the ESPPRC heuristics do not have a great impact here, and that especially the subset-row inequalities reduce both the computation times and the resulting gaps. For the 50 customer instances separating both cuts yielded the best results on average. We believe the key to tackle larger instances, or to improve the performance in general, is to tune and/or adequately enhance the labeling algorithm. In a further attempt different, more advanced branching rules as well as additional strengthening inequalities could be considered.

In the course of the project we investigated several matheuristics, three of them were presented in detail. The first two directly made use of the set-covering ILP formulation and can be classified as intertwined collaborative cooperations where the VNS or mVNS supplies the exact method with feasible solutions' routes and the current best solution, and the ILP solver takes a global view and seeks to determine better feasible route combinations. When doing so the solver might change the visit combinations. For testing we derived new PVRPTW instances with a planning horizon of four, six, and eight days from the 100 customer Solomon VRPTW benchmark instances. The VNS-ILP matheuristic yielded for two third of all considered instances a statistically significant improvement over solely applying the VNS. It was clearly beneficial to consider both improved and intermediate VNS solutions of a certain quality for enriching the ILP model. Generally, it is better to include not too many solutions to keep the runtime for solving the ILP small and increase the chance of finding an improved solution in limited time, as well as rather to include the solutions' routes for the corresponding day only, instead for all days. This way the hybrid method still performs significantly better for almost half of the instances even when compared to VNS runs

with twice the iteration limit, additionally requiring considerably less CPU-time. In case of the mVNS-ILP matheuristic the current best solutions of the VNS instances are provided to the exact method. Hence the application of the ILP solver can in some way be regarded as a recombination operator taking into account all available solutions provided by the "population" of the VNS instances. Also, if a solution is not feasible as a whole, its feasible routes are added anyway. In our previous work mVNS-ILP yielded for 80%–100% of all conducted tests a statistically significant improvement over solely applying the VNS in a standard way. It has become evident that, if possible, keeping the routes (columns) in the model once they were added is beneficial, though one has to keep in mind the longer runtimes of this setting than when considering the actual best solutions' routes only. Nevertheless, even the mVNS-ILP hybrid with this continuously increasing ILP model requires for most of the instances less CPU-time than the standard VNS with more iterations. The latest results with longer runtimes and accordingly more applications of the ILP solver show that mVNS-ILP, this time with resetting the columns, yields the best results of our methods and performs nearly as good as the currently best performing method. Only for instances having an eight day planning horizon there is a small gap of 0.26%. Potential improvements might be to include better handling of intermediate VNS solutions (reducing overhead and probably increasing overall quality and diversity), some sort of column management for the ILP to be able to include more solutions (probably for all days) without performance degradation via discarding unpromising columns over time, or adding problem specific cuts to the ILP. It might also be possible to use such a hybrid method for finding initial feasible solutions by combining several partly-feasible ones, though this task was not too hard for the given instances.

The third matheuristic is a combination of the column generation approach and the EA (CG-EA). Here the solution to the LP relaxation of the set covering model, i.e. on the one hand columns which were created and on the other hand the variable values, is successfully exploited in the subsequent EA. E.g. also an ongoing exploitation is realized via inserting routes of a pool during mutation. The initial column generation is executed with a time limit, which is inevitable to guarantee a viable runtime in total. This matheuristic, being lower-level since specific information of the column generation is exploited in the EA and classified as a sequential collaborative combination, is more interesting from a methodical point of view. Although it performs worse compared to the VNS-ILP matheuristics, it almost always significantly outperforms the pure EA, independent of the size of the planning horizon. CG-EA also clearly outperformed a column generation based heuristic.

Finally, it is to be noted that these matheuristics not only seem promising for other, possibly richer variants of routing problems, but their concept can fairly easily be applied to other classes of combinatorial optimization problems as well.

CHAPTER 5

PERIODIC VEHICLE ROUTING PROBLEM

5.1 Introduction

After tackling the PVRPTW with various hybrid methods as reported in the previous chapter we wanted to give a completely different approach a try, though this time for the problem variant not involving time windows and especially targeting large(r) instances. The idea was to combine the concept of multilevel refinement with a variant of the VNS previously applied to the PVRPTW accordingly adapted to the specifics of the PVRP. Part of this work was presented at the *10th European Conference on Evolutionary Computation in Combinatorial Optimisation* in 2010 (EvoCOP 2010) [163].

For a problem definition we refer to Chapter 4 on the PVRPTW; in case of the PVRP only the time windows are omitted or are assumed to be $[0, \infty]$. As a special case of the PVRP, when only having a single vehicle, we will also consider the *periodic traveling salesman problem* (PTSP).

Multilevel refinement strategies [230, 231] successively coarsen an initial problem instance, yielding a sequence of instances of decreasing size, representing the problem on different abstraction levels. The smallest instance is then solved by some auxiliary technique (e.g. a metaheuristic), and the obtained solution is *extended* to a solution of the previous level. This solution is possibly improved (e.g. again by some metaheuristic) and the solution extension continued until a solution for the original problem is obtained. In an *iterated multilevel refinement strategy* the whole process is iterated; hereby, a *recoarsening* is performed exploiting the last obtained solution in order to derive an eventually better hierarchy of abstraction levels. Multilevel refinement strategies have been successfully applied to several combinatorial optimization problems, including graph partitioning [231], the traveling salesman problem [229], and also the classical capacitated vehicle routing problem [197]. When ap-

plied sensible, they seem to be able to often improve the scalability of underlying heuristics significantly, either improving running times or final solution qualities.

The subject of this work therefore is to study the extension of a variable neighborhood search (VNS) metaheuristic that was already successfully applied to periodic routing problems by a multilevel refinement strategy in order to improve the performance of the VNS especially on larger instances.

In the following section we review related work. The underlying VNS and the multilevel extension are presented in Sections 5.3 and 5.4, respectively. Experimental results, in which also new larger benchmark instances are used, are discussed in Section 5.5. We also investigate a second approach in combination with our proposed multilevel refinement as well as the VNS in Section 5.6, with corresponding results given in Section 5.7. Finally, conclusions and an outlook on potential future work is given in Section 5.8.

5.2 Related Work

From a problem oriented view recent successful approaches for one or both of PVRP and PTSP are [40], [16], [7], and [105]. Cordeau et al. [40] describe a Tabu search for periodic and multi-depot routing problems, achieving for all previously known benchmark instances new best results at that time. The algorithm is based on rather simple standard neighborhoods that move single customers to different routes or change single visit combinations. New random test instances have further been introduced in this work. A well performing construction type algorithm with an embedded improvement procedure for the PTSP is presented by Bertazzi et al. [16]. Alegre et al. [7] apply a scatter search heuristic tailored especially to solve PVRP instances having a long planning horizon. Nevertheless they also obtained improved results for many standard benchmark instances. Most recently Hemmelmayr et al. [105] tackled the PVRP and PTSP with a sophisticated VNS that again yielded many new best results. As neighborhoods for diversification they utilized moving or exchanging route segments of different maximal size as well as changing several visit combinations. As improvement procedure they applied the well-known 2-opt and a restricted 3-opt. Similar to [40] they also allow infeasible solutions during the search but additionally apply an acceptance criterion similar to simulated annealing [123]. Francis et al. [83] gave a recent survey on periodic routing problems considering several variants of it.

A VNS based on the one from [105] has further been described for the PVRP with time windows in Chapter 4. The concept of multilevel refinement including an overview on applications is covered in [230, 231]. We point out the work by Rodney et al. [197] which introduces a multilevel refinement strategy for the capacitated vehicle routing problem. They coarsen the problem by building paths (segments) of customers through fixing the corresponding edges and consider such a path as an atomic unit in the following. Due to the capacity restrictions they propose to balance these paths according to the accumulated demand and systematically allow a gradually decreasing violation of the limiting constraint. Further the multilevel strategy has been previously applied to the traveling salesman problem [229, 25] in a similar way. Yet we are not aware of an approach that was designed to handle the peri-

Table 5.1: Fixed shaking neighborhood order of VNS.

k	$\mathcal{N}_k$
1–6	*Change Visit Combination* up to $\delta = k$ times
7–9	*Move Segment* of maximal length $\delta = k - 6$
10–12	*Exchange Segments* of maximal length $\delta = k - 9$
13	*Mixed*: probably apply all previous moves decided upon individually at random

odicity as well as to accomplishing such a smooth integration into a metaheuristic. Further, to our knowledge VNS has not been combined with multilevel refinement so far.

5.3 Underlying Variable Neighborhood Search

In the following, we give a rather short overview on our underlying VNS as most parts of it have already been described in more detail in the previous chapter about the PVRPTW.

To smooth the search space and help escaping local optima, the VNS relaxes the restrictions on vehicle load and route duration and adds penalties corresponding to the excess of these constraints to the cost function. As for the PVRPTW we also set a constant penalty, again corresponding to the average inter-customer travel costs. Initially a possible visit combination is selected for each customer at random. Afterwards we apply the Clarke and Wright savings algorithm [32] in case of multiple routes. If we end up with too many routes, the customers of those routes holding the least customers are relocated in a greedy way (for details we refer to [105]). By doing so the constructed routes might exceed maximal vehicle load or allowed tour duration. For instances with a single vehicle, i.e. especially for all PTSP instances, we make use of best insertion.

In the shaking phase we utilize three different neighborhood structures with increasing perturbation size per type: (i) randomly changing up to six visit combinations in a sequential way with greedy insertion for the new visit days, where for the PVRP we also allow reassigning the same visit combination, (ii) moving a random segment of up to length three of a route to another one on the same day, and (iii) exchanging two random segments of up to length three between two routes on the same day. Finally, an additional mixed neighborhood is applied where potentially all of the previous neighborhood structures are utilized. We thus have a total of 13 shaking neighborhoods (i.e. $k_{\max} = 13$) which are always considered in a fixed order that is listed in Table 5.1. These settings reflect previous experience and showed a good performance in preliminary tests. Contrary to [105] only segments of up to length three instead of six are exchanged, we recognized no performance gain when exchanging longer segments.

For intensification we apply the well-known 3-opt intra-route exchange procedure in a first improvement fashion for the PVRP and 2-opt for PTSP, only considering routes changed during shaking. According to preliminary tests on the PTSP we restrict 2-opt to segments of

length 10 in case of $n > 300$, hence limiting its computational effort. Both are applied as long as an improvement is achieved, i.e. until a local optimum is reached. Additionally each new PVRP incumbent solution is subject to a 2-opt* inter-route exchange heuristic [173]. Hereby for each pair of routes of the same day all possible exchanges of the routes' end segments are considered. In case of the PTSP we additionally apply 3-opt on all routes. This is an addition to the work of [105].

To enhance the overall VNS performance Hemmelmayr et al. [105] propose to not only accept better solutions, but sometimes also solutions having a worse objective value. This is done in a randomized fashion using the Metropolis criterion like in simulated annealing [123]. A linear cooling scheme is used in a way such that the acceptance rate of worse solutions is nearly zero in the last iterations. As for the PVRPTW in the previous chapter the initial temperature is set to one fifth of the average inter-customer travel costs and at each temperature level 100 iterations are applied, resulting in a temperature decrease by $T_{init} \cdot 100/max\_iterations$. The proposed VNS is depicted in Algorithm 15.

5.4 Multilevel Variable Neighborhood Search

In this section we extend the described VNS with ideas from the multilevel refinement strategy, hence we call it the *Multilevel VNS* (MLVNS). The basic idea of multilevel refinement [230, 231] is to suitably coarsen the problem with two goals: (i) to concentrate more on the costly parts of the problem during optimization, and (ii) at the same time to (temporarily) reduce its size, making it more tractable at the coarser levels. The problem is then (approximately) solved at the highest level and the obtained solution refined in an iterative way until a solution to the original problem is reached.

Contrary to most existing multilevel refinement approaches, our method

- does not automatically obtain one specific feasible solution at the coarsest level to start with, which is due to the periodicity in our case, and
- does not have several subsequent (and even independent) applications of the same method on different levels, but smoothly integrates the transitions from the most abstract level to the original problem in the VNS; i.e. many levels with small changes (in fact always only one) are used.

In the following we propose an initial coarsening process operating at a given problem instance, as well as a recoarsening based on an incumbent solution.

5.4.1 Initial Problem Coarsening

Two common coarsening schemes on graphs are based on either merging nodes (into "supernodes") or building segments (paths). Since we have to determine sequences of customer visits it is rather natural to follow the latter scheme here, keeping in line with previous work [229, 197]. We opt for an exact coarsening, i.e. the objective value of a coarsened

Algorithm 15: VNS for the PVRP/PTSP with initial solution x_{init}

```
1  x_best ← x_init; // incumbent solution (though most likely
   infeasible at the beginning)
2  x_vns ← x_init; // solution utilized for shaking
3  k ← 1; // shaking strength
4  initialize temperature to (average travel costs)/5
5  while VNS stopping condition is not met do
6      while k ≠ k_max do
           // shaking:
7          Select x' ∈ N_k(x_vns)
           // local search:
8          if m > 1 then PVRP instance
9              x'' ← repeated first improving 3-opt neighbor of x'
10         else
11             if n ≤ 300 then
12                 x'' ← repeated first improving 2-opt neighbor of x'
13             else
14                 x'' ← repeated first improving reduced 2-opt neighbor of x'
           // possibly update incumbent:
15         if x'' is feasible and better than x_best then
               // additional local search:
16             if m > 1 then PVRP instance
17                 apply 2-opt* improvement on x''
18             else
19                 apply 3-opt on all routes of x''
20             if x'' made infeasible then
21                 revert previous local search
22             x_best ← x''; // set new incumbent
23             k ← 1; // reset shaking
           // possibly update solution utilized by VNS:
24         if (x'' is better than x_vns) or (x'' is worse than x_vns but accepted due to
           Metropolis criterion) then
25             x_vns ← x''
26             k ← 1; // reset shaking
27         else k ← k + 1; // increase shaking
28         every 100^th iteration apply linear cooling
29         possibly do a refinement step
       // finished VNS iteration
30     k ← 1; // reset shaking
```

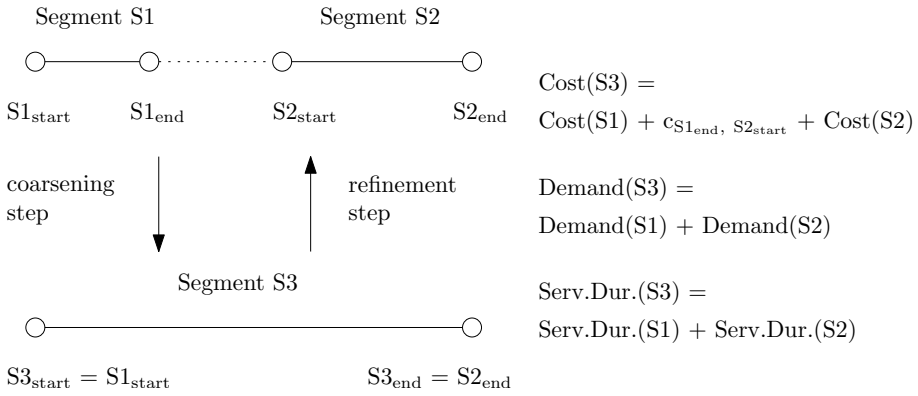


Figure 5.1: Coarsen: merge two segments into one, refine: split a segment into two.

solution always equals the one of the corresponding fully refined solution. Basically during coarsening the nodes of the graph (i.e. single customers) are merged to segments via fixing a specific edge between them. One such coarsening step of merging two segments into a longer segment and the corresponding reversed refinement step are shown in Figure 5.1. Note that a single customer can be regarded a segment with identical start and end point, too. The accumulated cost, demand, and service duration of the newly created segment must be set accordingly.

Due to the periodicity, building customer sequences on a daily basis as for the single day of the classical vehicle routing problem would not make much sense. On the one hand this would require to choose a visit day combination per customer in advance (a bad choice would potentially result in a bad coarsening) and on the other hand such segments would most likely not allow to change its visit day combination without the need of breaking it apart, thus creating only short-lived segments being inconsistent with the general idea of the multilevel refinement strategy. Therefore it is necessary to apply a coarsening process respecting the periodicity and building segments spanning the whole planning horizon. This is achieved by incorporating the customers' sets of available visit day combinations—the service frequencies alone might not be sufficient—and allow customers and subsequently segments to be merged only if their sets are equal.

At this point several initial problem coarsenings could be obtained by using different cost criteria as well as processing sequences. For example, in [197] per level they randomly choose an unmatched segment and fix an edge between it and the nearest unmatched segment according to the savings measure, whereas the actual costs of connecting the segments' free end nodes only considering segments in a defined surrounding for matching are used in [229]. We investigated several possibilities and after preliminary tests came up with these findings/settings:

- In general we observed the tendency that the greedier the selection for matching is, the better are the results on average; hence we always select the cheapest matching among all possible ones.

- Regarding the size (length) of the segments, i.e. how many customers are contained in the segment, we used two options:
  - *Setting I*: Enforce no limit at all, thus coarsen in a pure greedy fashion.
  - *Setting II*: Start with an allowed maximal length of two and gradually increase this limit by one whenever no more matching could be found. The maximum number of occurrences of one set of visit day combinations is used as an upper bound for this maximal length.
- We settled on using the actual costs of connecting the segments' free end nodes, thereby trying all four possible ways.
- Optionally we set a limit for the connection costs by multiplying the average inter-customer travel costs by a given factor δ_c, hence having an instance independent measure
- Optionally we respect given limits on vehicle capacity and route duration when building the segments.

A small schematic coarsening example for both settings I and II without limiting the connection costs is shown in Figure 5.2. The common practice of coarsening with a random factor yielded (so far) clearly worse results.

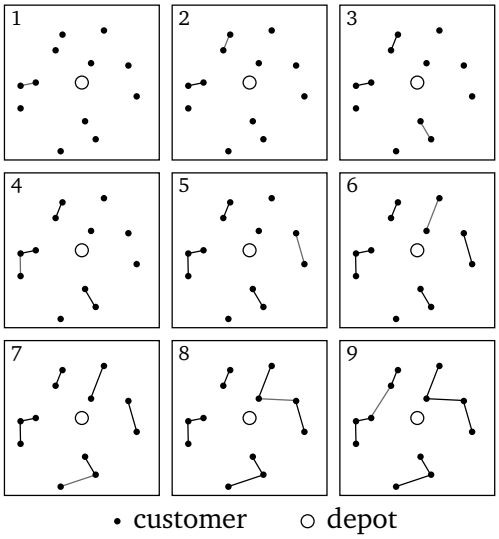
Having the coarsest problem, we generate the initial solution as described in Section 5.3 (there is no need to change this procedures). Then we interweave the iterative refinement with the VNS' execution. First, we select a fraction of the total VNS iterations in which this initial coarsening/refinement phase takes place. Then these iterations are divided by the number of present refinements (plus one to also execute some iterations on the fully refined problem) to determine the iterations per (mini-)level.

Refinement is exactly the reversed process as coarsening (in a "last in–first out" fashion), thus splitting the usually most costly matchings first and keeping the more likely (cheaper) ones accordingly longer. Hence in a refinement step (refer to Figure 5.1) we have to locate the segment in a route at all days of the actual visit day combination and split it in two segments again, thereby preserving the direction.

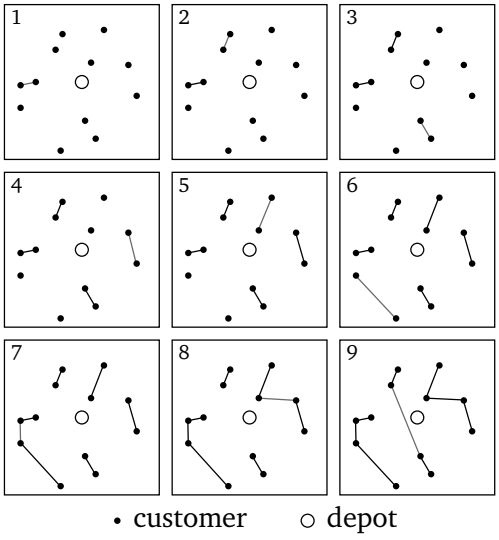
At the end of this phase we may either continue with the fully refined problem or apply a recoarsening which will be the subject of the next section.

5.4.2 Solution-Based Recoarsening

Solution-based recoarsening is a very common practice to extend the concept of multilevel refinement beyond the initial coarsening/refinement phase. For this, the problem is recoarsened on the basis of a current incumbent solution in such a way as to preserve its structure, hence neglecting/destroying no obtained information. Despite this the coarsening principle stays the same. This obviously leads to considerably less degrees of freedom during the coarsening and in our case automatically to rather short segments.



(a) Setting I



(b) Setting II

Figure 5.2: Exemplary (deterministic) initial problem coarsenings for customers having the same visit day combinations, iteratively building longer (route) segments.

This time we restrict our attention to adjacent segments in the current solution's routes, whereas again, they must have equal sets of visit day combinations. Further, such a segment pair must be adjacent on all visit days, only a whole reversed occurrence (s.t. the same end points are connected) is acceptable. As a cost criterion we directly use the present connection costs.

Regarding the recoarsening procedure we basically adhere again to the greedy approach as

used in the initial problem based coarsening, but try to prevent obtaining the same recoarsening in case we use the same solution multiple times: Using a parameter x, we always select one of the x cheapest possible matchings per step (or level) at random. Initially, x is set to one, but when we encounter the same solution to be used for recoarsening again, x is incremented. Contrary, if a different solution than in the previous iteration is used we reset x to one and turn the procedure into a pure greedy selection again. This extension further reduces the risk of getting stuck in local optima.

For recoarsening we have so far fixed the connection cost limit to $\delta_c/2$ of the average inter-customer travel costs (in case the factor δ_c is used at all). Since we utilize a feasible solution for guiding the recoarsening a violation of the vehicle capacity or route duration is impossible and must not be checked. Finally, the subsequent refinement phase is handled in the same way as the initial refinement in the preceding section. Therefore we also choose a fraction of the VNS' iterations to be allotted to a recoarsening/refinement phase.

5.4.3 Handling Segments in the VNS

Due to incorporating the multilevel refinement strategy into the VNS there are a few more things to consider when dealing with segments of merged customers:

- As already mentioned the solution construction heuristics need not to be changed, the same applies to moving or exchanging route segments and all local search procedures.
- After each refinement step we immediately apply the intra-route local search in use on the routes that contain refined segments.
- Although the segments have different start and end points, i.e. are asymmetric, when changing visit combinations during shaking we always reinsert them in the direction at the time of creation instead of the one in the previous route. We observed no gain doing otherwise nor when trying both directions.

5.5 Computational Experiments I

The following experiments were aimed at investigating the performance difference of the standard VNS and the multilevel VNS, hence we were not hunting for new best solutions on available data sets. Nevertheless, we already did find a few ones back then (which are outdated by now) but we will not elaborate on this. The algorithms have been implemented in C++, compiled with GCC 4.5 and executed on a single core of a 2.83 GHz Intel Core2 Quad Q9550 with 8 GB RAM.

For each chosen setting we performed 10 runs per instance with a limit of 10^6 iterations, i.e. solution evaluations. Preliminary tests on all considered instances suggested to ignore the given limits on vehicle capacity and tour duration during coarsening (which would only affect PVRP instances), probably because the VNS was built to cope with infeasibility. If not stated otherwise 20% of all iterations are devoted to the initial coarsening/refinement phase

Table 5.2: Summarized average results on available benchmark test data of VNS and MLVNS; giving standard deviations in paranthesis.

test data		instances	VNS		MLVNS $II_{0.5}$		
			%-gap BKS	%-gap HDH	%-gap BKS	%-gap HDH	%-time VNS
PVRP	old	32	3.02 (4.67)	-0.19 (1.43)	2.50 (3.91)	-0.66 (1.57)	89.2
	new	10	2.86 (1.73)	-0.75 (0.89)	2.68 (1.66)	-0.92 (0.90)	85.1
PTSP	old	23	0.85 (1.12)	0.05 (0.39)	0.30 (0.43)	-0.49 (0.80)	89.9
	new	10	0.34 (0.18)	0.04 (0.05)	0.28 (0.17)	-0.01 (0.07)	79.1

and recoarsening is applied four times also devoting 20% of the iterations each. The two remaining options we varied are the type of problem coarsening (setting I or II) and the cost limit factor δ_c; these will be denoted by [I,II]$_{\delta_c}$. We experienced that usually it is better to enforce a merging cost limit.

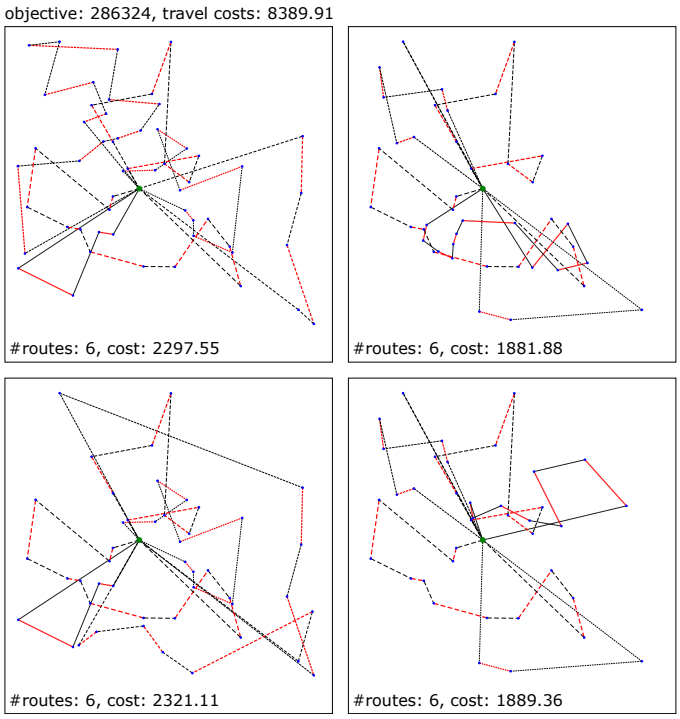
Figures 5.3 and 5.4 show PVRP solutions throughout the initial refinement process with coarsened segments highlighted in red.

5.5.1 PVRP and PTSP Instances Used in the Literature

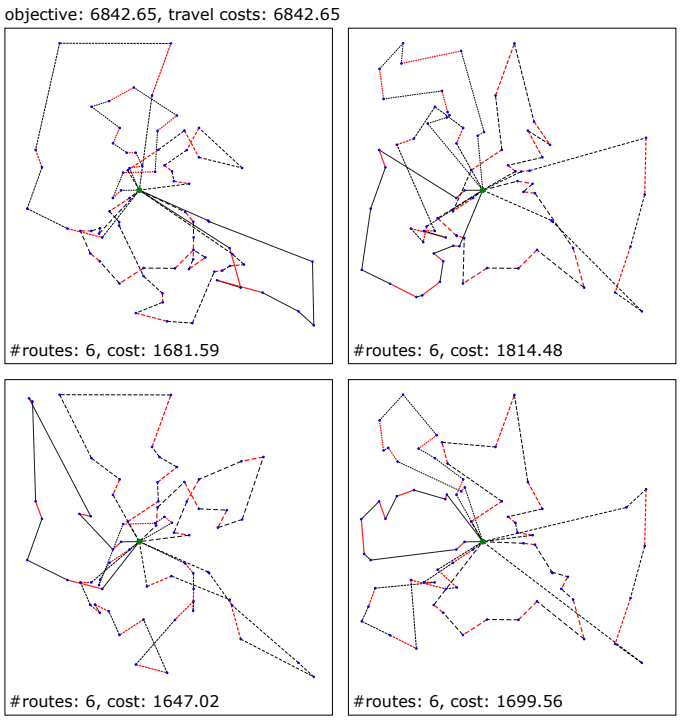
We utilize available "old" and "new" benchmark test data[1] both for the PVRP and the PTSP. Here we will only present results on them in concise form and refer to [40] for further details and origins of these instances. Back then relevant and recent results for comparison are reported in [105]. Meanwhile Vidal et al. [226] proposed a hybrid genetic algorithm outperforming previous approaches in terms of solution quality with comparable computing effort.

Results are presented in concise form in Table 5.2. Here we state the percentage gap to the so far best known solutions at that time (%-gap BKS) as well as to the average results of the VNS described by Hemmelmayr et al. [105] using the same iteration limit of 10^6 (%-gap HDH), all these values are given in [105]. The corresponding standard deviations are written in parentheses. For the MLVNS we further state the amount of CPU time spent given in percentage of the VNS' CPU time (%-time VNS). As can be seen the MLVNS generally performs better than the standard VNS, achieving especially on the old data sets a notable improvement with regard to solution quality. Contrary, on the new data sets no such clear improvement can be observed, especially not for the PTSP. However, for all these data sets there is a CPU time reduction between 10 and 20 percent on average. As it turned out coarsening setting II gave consistently better results, though the differences were sometimes negligible.

[1] Available at `http://neumann.hec.ca/chairedistributique/data/pvrp` [accessed on October 22, 2011]

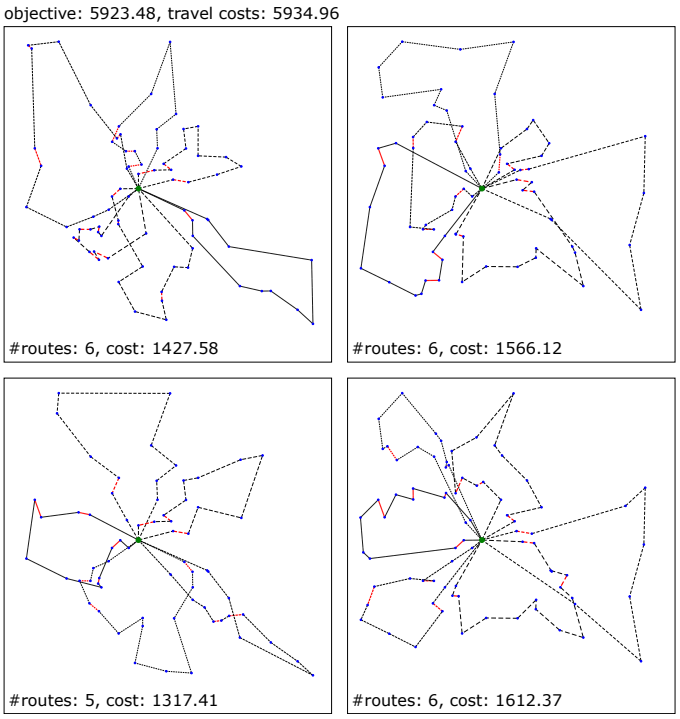


(a) initial coarse solution

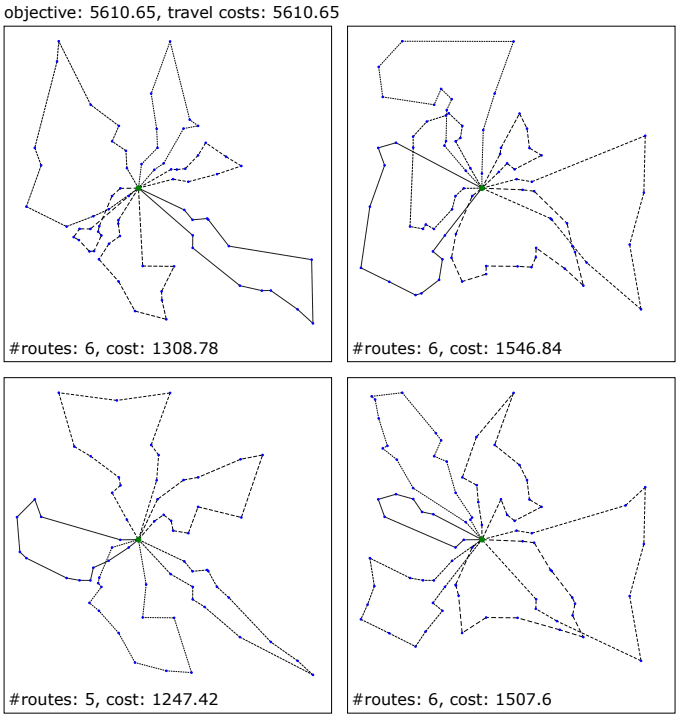


(b) solution after about 1/3 of all initial refinement steps

Figure 5.3: Exemplary (coarsened) PVRP solutions during the VNS search process.



(a) solution after about 2/3 of all initial refinement steps



(b) solution at the end of the initial refinement process

Figure 5.4: Exemplary (coarsened) PVRP solutions during the VNS search process.

Table 5.3: New larger PVRP and PTSP (setting $m = 1$ and ignoring D and Q) instances using the generation method introduced in [40], additionally stating our best found solutions' values over all runs.

Id	n	m	t	D	Q	service frequencies					best found solutions	
						f_1	f_2	f_3	f_4	f_6	PVRP	PTSP
pr11	336	14	4	480	185	112	112		112		11224.40	6618.53
pr12	384	16	4	475	195	128	128		128		11320.99	7062.96
pr13	432	18	4	450	185	144	144		144		10429.50	6757.86
pr14	480	20	4	475	185	160	160		160		13463.25	7740.23
pr15	528	22	4	470	190	176	176		176		15237.87	8377.39
pr16	576	24	4	455	185	192	192		192		13315.25	7640.57
pr17	360	15	6	445	165	90	90	90		90	15888.55	9459.08
pr18	432	18	6	450	170	108	108	108		108	19564.87	11314.85
pr19	504	21	6	440	160	126	126	126		126	22949.66	11344.27
pr20	576	24	6	450	165	144	144	144		144	24004.13	11660.07

5.5.2 Additional PVRP and PTSP Instances Similar to Cordeau et al.'s

Since the multilevel refinement strategy is in general especially appealing for large(r) instances where it may unfold its full potential, but the available test data lack them, we created some PVRP and PTSP instances on our own by applying the generation method described in [40]. They can be regarded a continuation of the instances introduced in this latter work, except that we evenly distributed the visit frequencies among the customers for all instances; more details, already including our best found solutions' values over all conducted runs (partly also with more iterations), are given in Table 5.3.

When doing preliminary tests we recognized that the VNS' acceptance decision using the Metropolis criterion in some way weakens the potential gain of the multilevel extension. Hence we also performed tests with only accepting improved solutions, whose results are given in Table 5.4 and 5.6 for the PVRP and the PTSP, respectively. The corresponding results using the default acceptance decision are given in Table 5.5 and 5.7. For both variants and problems we tested coarsening setting I and II with a cost limit factor δ_c of 0.5 and 0.33 (which was narrowed down in preliminary tests), as well as devoting all iterations to the initial coarsening/refinement phase and doing no recoarsening at all, where we always state the results of the setting yielding the best solutions on average. The tables show average travel costs (avg), corresponding standard deviations in percent (sdv[%]), CPU-times in seconds (t[s]), as well as the percentage gaps to the VNS (%-gap) and the CPU-times given in percent of the VNS (%-time) for the MLVNS. In Tables 5.4–5.7 average values of the MLVNS are printed bold whenever a statistically significant improvement compared to the VNS has been achieved or vice versa, according to a Wilcoxon rank sum test with an error level of 5%. For these new larger instances the greedy coarsening setting I turned out to achieve consistently better results than setting II. Comparing Table 5.4 and 5.5, as well as Table 5.6

Table 5.4: Average results of standard and multilevel VNS on new larger PVRP instances only accepting improved solutions.

Id	VNS				MLVNS $I_{0.5}$					
	best	avg	sdv[%]	t[s]	best	avg	sdv[%]	t[s]	%-gap	%-time
pr11	11988.17	12135.04	0.92	40.6	11764.62	12066.05	1.59	31.4	-0.57	77.3
pr12	11935.12	12099.76	0.76	42.8	11913.75	**12019.06**	0.68	33.3	-0.67	77.8
pr13	11291.21	11484.99	0.86	47.3	11123.96	**11265.22**	1.18	37.2	-1.91	78.6
pr14	14457.39	14572.72	0.61	52.4	14217.40	**14352.14**	0.80	40.3	-1.51	76.9
pr15	16329.00	16498.72	0.68	59.7	15878.62	**16078.02**	0.66	42.1	-2.55	70.5
pr16	14716.46	14815.95	0.69	67.3	13832.13	**14064.44**	1.00	47.3	-5.07	70.3
pr17	16864.64	17072.72	0.69	41.5	16525.56	**16804.22**	0.79	34.4	-1.57	82.9
pr18	20778.33	21057.55	0.83	48.8	20345.62	**20613.84**	0.65	39.4	-2.11	80.7
pr19	24898.62	25148.96	0.42	57.7	24330.25	**24795.71**	1.19	44.4	-1.40	76.9
pr20	25716.70	25875.35	0.52	70.4	25075.55	**25273.14**	0.59	53.4	-2.33	75.9
avg				50.6				39.8	-1.97	77.1

and 5.7 it can be clearly seen that the MLVNS yields a higher relative improvement when only accepting improved solutions, nevertheless also when using the default acceptance decision the improvement is notable, especially in case of the PTSP. Interestingly, for the PTSP using no recoarsening and extending the initial problem coarsening/refinement phase to all iterations turned out to be usually better (which only holds for the new larger instances, we checked it for the available test data, too). This has the additional positive effect that the required CPU-time is more than halved. Contrary, for the PVRP the decrease in CPU-time is about 20%. For the latter problem we also observed that improvements during refinement after a recoarsening occur more often when also accepting worse solutions, otherwise the customer segments are probably too coarse and do not allow an improvement in a direct way. In summary, for almost all instances and both acceptance decisions the MLVNS yields on average significantly better results than the VNS.

The relative usage, acceptance and success rate of the shaking neighborhoods for both problems averaged over all corresponding runs is shown in Table 5.8 and 5.9. In general changing visit combinations is by far the most important neighborhood, even more evident for the PTSP, yet the others make significant contributions as well. Looking at the rates of the multilevel VNS shows that they are smoothed to some extent, i.e. the neighborhoods' usage and subsequently their acceptance and success rates are more uniformly than before, again more so for the PTSP. This shows that by using the multilevel refinement allows to better utilize also the later (in terms of a higher k value) neighborhoods.

5.6 Multilevel Variable Neighborhood Descent and Embedment in VNS

Due to the smaller performance gain of the proposed multilevel refinement strategy when applying the Metropolis criterion in the previous VNS we decided to investigate a more

Table 5.5: Average results of standard and multilevel VNS variants on new larger PVRP instances using the acceptance decision with the Metropolis criterion.

Id	VNS				MLVNS $I_{0.5}$					
	best	avg	sdv[%]	t[s]	best	avg	sdv[%]	t[s]	%-gap	%-time
pr11	11777.81	11925.97	0.73	40.6	11534.90	**11791.76**	1.00	32.1	-1.13	79.1
pr12	11599.29	11662.50	0.35	42.5	11557.46	11654.71	0.51	34.0	-0.07	80.0
pr13	10775.92	10893.52	0.63	46.7	10679.85	**10789.19**	0.79	37.5	-0.96	80.3
pr14	13843.26	13968.70	0.59	51.9	13854.73	**13894.75**	0.19	40.8	-0.53	78.6
pr15	15481.21	15600.60	0.44	58.2	15544.47	15659.44	0.62	43.6	0.38	74.9
pr16	13693.90	13840.69	0.54	65.1	13701.18	**13762.91**	0.49	48.4	-0.56	74.3
pr17	16203.55	16287.59	0.47	41.2	16220.30	16324.32	0.46	34.8	0.23	84.5
pr18	19954.18	20132.26	0.55	48.2	19921.72	20046.51	0.51	40.4	-0.43	83.8
pr19	24157.00	24358.89	0.61	56.5	23921.31	**24047.23**	0.42	44.9	-1.28	79.5
pr20	24473.60	24708.36	0.57	67.2	24282.46	24583.69	0.59	53.0	-0.50	78.9
avg				50.0				40.6	-0.48	79.3

Table 5.6: Average results of standard and multilevel VNS variants on new larger PTSP instances only accepting improved solutions.

Id	VNS				MLVNS $I_{0.5}$ (no recoarsening)					
	best	avg	sdv[%]	t[s]	best	avg	sdv[%]	t[s]	%-gap	%-time
pr11	6827.31	6846.83	0.26	202.5	6704.23	**6759.25**	0.44	95.1	-1.28	47.0
pr12	7222.48	7305.13	0.60	261.0	7149.85	**7170.01**	0.27	117.5	-1.85	45.0
pr13	6916.30	6968.14	0.57	341.7	6794.93	**6852.11**	0.44	151.9	-1.67	44.5
pr14	7969.23	8040.07	0.66	426.0	7800.44	**7839.22**	0.40	183.7	-2.50	43.1
pr15	8545.86	8681.43	0.74	535.5	8482.55	**8527.30**	0.29	211.6	-1.78	39.5
pr16	7874.67	7993.22	0.82	677.0	7718.15	**7759.57**	0.32	264.9	-2.92	39.1
pr17	9748.70	9858.72	0.66	221.5	9575.85	**9640.96**	0.36	107.6	-2.21	48.6
pr18	11577.94	11703.62	0.63	318.5	11457.79	**11549.69**	0.48	148.9	-1.32	46.8
pr19	11716.97	11788.73	0.44	452.7	11519.13	**11594.03**	0.44	192.9	-1.65	42.6
pr20	12023.09	12114.09	0.43	623.2	11793.31	**11875.43**	0.33	256.3	-1.97	41.1
avg				383.9				167.8	-1.91	43.8

standard approach, too. In order to better observe the different search capabilities when utilizing the multilevel refinement a *variable neighborhood descent* (VND) seemed promising. Since it is a straightforward VND we will only mention the neighborhoods in use, which are listed in Table 5.10. The first is to change a visit combination of either a single customer or a customer set if in a coarsened state (*Change Visit Combination*). The entities as well as the possible visit combinations are enumerated in a random way to avoid any bias and the first improving change is accepted, followed by an immediate subsequent improvement via 2-opt* or 3-opt, depending on the problem as before. We also tried a variant with changing two or more visit combinations at a time, but the size of these neighborhoods and hence the computational effort grows very quickly and renders them not viable anymore. The second and third neighborhoods were proposed by Osman [153] and are the local search based variants of the corresponding shaking neighborhoods of the VNS: shift a segment of length one

Table 5.7: Average results of standard and multilevel VNS variants on new larger PTSP instances using the acceptance decision with the Metropolis criterion.

Id	VNS				MLVNS $I_{0.5}$ (no recoarsening)					
	best	avg	sdv[%]	t[s]	best	avg	sdv[%]	t[s]	%-gap	%-time
pr11	6714.47	6744.34	0.28	176.9	6648.71	**6698.56**	0.37	87.3	-0.68	49.3
pr12	7166.25	7207.09	0.38	226.2	7119.42	**7145.22**	0.24	108.8	-0.86	48.1
pr13	6840.12	6888.41	0.68	292.3	6775.80	**6824.10**	0.61	142.4	-0.93	48.7
pr14	7841.78	7893.10	0.30	363.8	7763.72	**7816.93**	0.34	173.0	-0.97	47.6
pr15	8463.47	8539.77	0.60	457.3	8432.79	**8475.92**	0.27	194.4	-0.75	42.5
pr16	7781.59	7820.54	0.50	567.1	7666.86	**7717.41**	0.35	240.9	-1.32	42.5
pr17	9575.55	9609.36	0.24	213.1	9464.30	**9533.57**	0.38	106.4	-0.79	49.9
pr18	11406.03	11479.46	0.39	299.2	11364.40	**11407.04**	0.29	143.1	-0.63	47.8
pr19	11507.13	11578.74	0.48	411.1	11408.89	**11463.24**	0.27	186.1	-1.00	45.3
pr20	11817.31	11865.51	0.26	551.8	11696.71	**11745.97**	0.27	246.6	-1.01	44.7
avg				331.5				158.1	-0.89	47.7

Table 5.8: Relative usage, improvement of x_{vns} and improvement of x_{best} in percent of shaking neighborhoods for new larger PVRP instances.

k	VNS			MLVNS $I_{0.5}$		
	%-use	%-improved	%-new best	%-use	%-improved	%-new best
1	14.01	26.45	20.35	11.94	22.92	19.17
2	11.91	23.03	19.68	10.69	21.47	19.28
3	10.07	17.90	18.02	9.51	18.10	17.62
4	8.62	13.48	15.12	8.51	14.61	15.53
5	7.52	10.15	13.23	7.70	11.76	13.54
6	6.69	7.75	10.87	7.04	9.44	11.51
7	6.05	0.19	0.16	6.51	0.23	0.27
8	5.97	0.12	0.13	6.44	0.15	0.17
9	5.92	0.10	0.10	6.40	0.13	0.14
10	5.87	0.22	0.80	6.36	0.31	1.01
11	5.82	0.15	0.48	6.32	0.21	0.64
12	5.79	0.11	0.42	6.29	0.17	0.38
13	5.76	0.35	0.63	6.27	0.49	0.75

to three from one route to another one (*Lambda-shift*) as well as interchange two segments of length one to three between two routes (*Lambda-interchange*). Due to the necessity of having at least two routes the latter two neighborhoods are only applicable in case of the PVRP. It is obvious that a high degree of coarsening allows for major changes in the solution, hence the neighborhoods are somewhat themselves successively refined. Contrary to the VNS before where a refinement occurs rather arbitrarily after a fixed amount of iterations, here, whenever

Table 5.9: Relative usage, improvement of x_{vns} and improvement of x_{best} in percent of shaking neighborhoods for new larger PTSP instances.

k	VNS			MLVNS $I_{0.5}$ (no recoarsening)		
	%-use	%-improved	%-new best	%-use	%-improved	%-new best
1	51.85	67.75	41.46	29.94	49.02	34.72
2	17.90	19.94	21.33	16.76	22.08	19.66
3	7.91	6.90	12.62	10.80	11.31	11.38
4	4.45	2.87	9.05	7.74	6.47	7.51
5	3.01	1.41	7.00	5.98	4.05	5.87
6	2.30	0.79	5.87	4.88	2.69	4.20
7	1.91	0.05	0.37	4.14	0.90	3.45
8	1.86	0.05	0.43	3.85	0.73	2.70
9	1.83	0.04	0.33	3.61	0.62	2.44
10	1.80	0.06	0.34	3.41	0.75	2.77
11	1.76	0.05	0.39	3.16	0.64	2.47
12	1.73	0.05	0.37	2.95	0.56	2.18
13	1.69	0.05	0.44	2.77	0.21	0.65

Table 5.10: Neighborhoods utilized by the VND.

l	$\mathcal{N}_l$	applied for
1	*Change Visit Combination*	PVRP, PTSP
2–4	*Lambda-shift* of a segment with length $\delta = l - 1$	PVRP
5–7	*Lambda-interchange* of segments with length $\delta = l - 4$	PVRP

the VND terminates at a local optimum a refinement step is applied.

This proposed VND naturally lends itself to be embedded into a VNS as the local search component. Basically the VNS of the preceding sections can be used for this, though with one modification regarding the shaking neighborhood where visit combinations are changed (with k ranging from 1 to 6): it turned out to be beneficial to insert the customer into a route selected at random for the new visit day as opposed to choosing the best insertion position among all daily routes. In this way a more disruptive shaking can be realized which contributes to the success of the subsequent VND. Also the VNS only accepts improved solutions. An outline of the algorithm variants is given in Algorithm 16, which also holds for the case when no coarsening is applied at all. As for the previous VNS the local search of choice is applied after each refinement step, again restricting 2-opt if a PTSP instance having more than 300 customers is encountered.

Algorithm 16: Multilevel VND/VNS for the PVRP/PTSP with initial solution x_{init} and *refine* refinement steps

```
1  x ← x_init; // incumbent solution (though most likely
     infeasible at the beginning)
2  iter ← 0
3  while iter < refine + 1 do
4      run VND or VNS with embedded VND on x
5      if not yet fully refined then
6          do one refinement step
           // Apply additional (standard) local search:
7          if m > 1 then PVRP instance
8              apply repeated first improving 3-opt on x
9          else
10             if n ≤ 300 then
11                 apply repeated first improving 2-opt on x
12             else
13                 apply repeated first improving reduced 2-opt on x
14     iter ← iter + 1
```

5.7 Computational Experiments II

For evaluating the VND, the multilevel VND (MLVND) as well as the multilevel VND embedded into the VNS (MLVNS+VND) we only consider the newly created larger instances which are more interesting for highlighting the differences. For the latter variant we set three iterations (shakings) of the VNS, which might seem rather limited but the runtime drastically increases with higher numbers. The results for the PVRP and the PTSP are given in Table 5.11 and Table 5.12, respectively. Again we tested several coarsening settings and in the end similar ones than before achieved the best results; the actual setting is denoted in the tables. For the MLVND and the MLVNS+VND we also state per instance the average percentage gap to the "classic" VND without multilevel refinement, as well as for all new methods the average percentage gap over all instances and the percentage of the required CPU time compared to the (ML)VNS. To have a somewhat fairer comparison to the previous (ML)VNS we consider the variants where only improved solutions are accepted. For both problems in each case the MLVND and the MLVNS+VND yielded significantly better results (according to a Wilcoxon rank sum test with an error level of 5%) than VND and in the same way MLVNS+VND always outperformed MLVND. Interestingly although VND is many more times applied in the multilevel variant the increase in runtime is only moderate, in fact taking about twice for the PVRP and less than three times for the PTSP. Contrary to the previous (ML)VNS a larger improvement can be noticed for the PVRP. With regard to solution quality MLVND consistently shows a performance that is comparable to MLVNS, but takes less runtime. The new MLVNS+VND yields even better results than MLVNS but

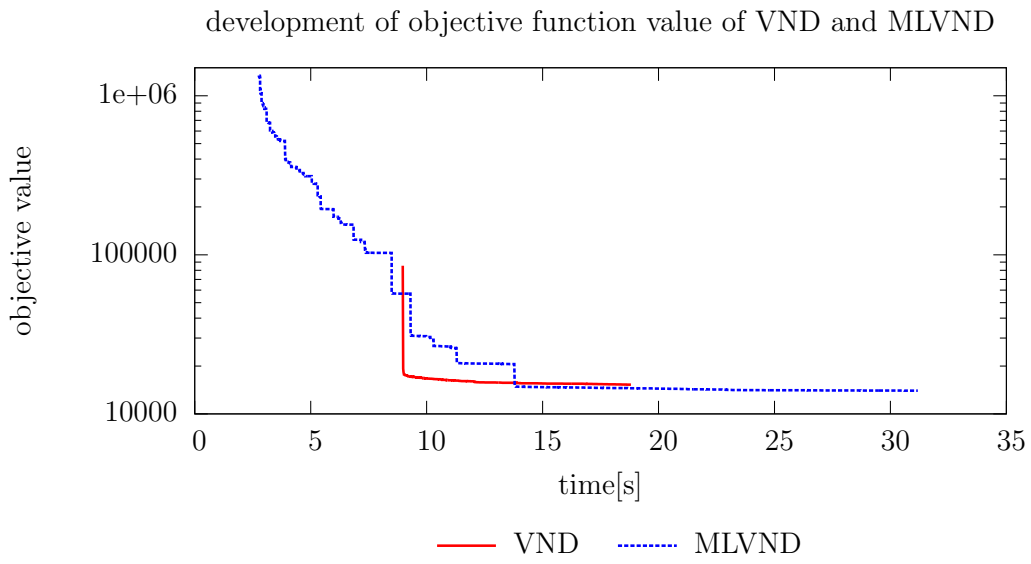


Figure 5.5: Development of the objective value of VND and MLVND on PVRP instance pr16.

this time at the expense of longer runtimes, especially in case of the PVRP (but here it is even slightly better than the MLVNS using the Metropolis criterion). An exemplary development of the objective value over time for a run of the VND and the MLVND on PVRP instance pr16 (obtaining average results each) is shown in Figure 5.5. The VND needs more time to come up with a solution at all due to the more time consuming initialization process, then instantly a feasible solution is found and improved. In contrast, MLVND starts quickly yet with a highly infeasible solution, in the following the refinement steps are observable where the objective value rapidly improves, halfway of the VND already finding a better solution and improving it a bit further. Due to the logarithmic vertical axis the improvement over VND is rather hard to spot: the VND finishes with 15275.33 as opposed to the MLVND with 14008.33, the latter achieving an improvement of 8.30%.

Additionally we also tried to reduce the number of shaking steps for the MLVNS+VND which are allowed in early stages of the search, i.e., when the problem is rather coarse. The result was only a moderate decrease of runtime but unfortunately also a decrease in solution quality, so it seems to pay off to also promote optimization at high levels. Another point was to experiment with recoarsening which was sometimes beneficial for the previous MLVNS. However, using it here only led to longer runtimes with merely no improvements at all.

Finally, we also give here statistics of the VND neighborhoods for the PVRP with and without multilevel refinement; see Table 5.13 (since in case of the PTSP solely the first neighborhood is used). Similar to before but even more evident is the increased average utilization of the neighborhoods when using the multilevel refinement. Though not only the usage rate increases but also the rate of finding better solutions. The rates when embedding the VND

Table 5.11: Average results of standard and multilevel VND as well as multilevel VNS with embedded VND on new larger PVRP instances, also compared to the previous (ML)VNS only accepting improved solutions.

Id	VND				MLVND $I_{0.33}$					MLVNS+VND $I_{0.33}$				
	best	avg	sdv[%]	t[s]	best	avg	sdv[%]	t[s]	%-gap$_{VND}$	best	avg	sdv[%]	t[s]	%-gap$_{VND}$
pr11	12634.56	13067.70	2.23	3.9	12142.60	12429.45	2.01	8.5	-4.88	11594.70	11796.89	1.27	141.3	-9.72
pr12	12328.28	12482.03	1.04	4.5	11781.20	12019.13	1.16	9.9	-3.71	11554.90	11622.37	0.54	193.0	-6.89
pr13	11728.60	11944.96	1.14	7.5	11006.50	11242.60	0.99	15.4	-5.88	10770.90	10845.56	0.43	324.8	-9.20
pr14	15030.89	15163.39	0.76	9.8	14075.50	14357.17	1.19	20.4	-5.32	13864.00	13951.85	0.38	483.2	-7.99
pr15	16960.80	17109.20	0.55	12.6	15732.00	16015.47	1.02	25.4	-6.39	15529.20	15635.65	0.37	588.3	-8.61
pr16	15042.60	15294.12	0.76	17.0	13840.90	14016.95	1.12	33.8	-8.35	13418.90	13532.33	0.52	892.0	-11.52
pr17	17511.40	17695.16	0.65	5.2	16812.50	16953.37	0.44	11.2	-4.19	16304.50	16417.29	0.48	192.3	-7.22
pr18	21358.30	21657.83	0.70	9.0	20320.30	20542.77	0.62	21.5	-5.15	19959.90	20142.09	0.57	385.6	-7.00
pr19	25496.30	25824.67	0.97	18.7	24470.40	25003.55	1.44	36.6	-3.18	23539.90	23732.15	0.63	829.3	-8.10
pr20	26071.27	26288.21	0.65	22.6	24787.80	24925.14	0.48	46.3	-5.19	24241.20	24444.04	0.42	1119.7	-7.02
avg				11.1				22.9	-5.22				515.0	-8.33
%-gap$_{VNS}$		3.66				-1.75					-4.98			
%-time$_{VNS}$				21.9				45.2					1017.8	
%-gap$_{MLVNS}$		5.75				0.21					-3.07			
%-time$_{MLVNS}$				27.9				57.5					1294.0	

Table 5.12: Average results of standard and multilevel VND as well as multilevel VNS with embedded VND on new larger PTSP instances, also compared to the previous (ML)VNS only accepting improved solutions.

Id	VND				MLVND $I_{0.66}$					MLVNS+VND $I_{0.66}$				
	best	avg	sdv[%]	t[s]	best	avg	sdv[%]	t[s]	%-gap$_{VND}$	best	avg	sdv[%]	t[s]	%-gap$_{VND}$
pr11	6935.61	7007.02	0.64	4.5	6809.01	6844.67	0.41	14.2	-2.32	6724.82	6764.44	0.32	73.7	-3.46
pr12	7330.27	7438.38	0.83	7.9	7173.00	7264.08	0.58	24.0	-2.34	7151.94	7195.01	0.50	124.3	-3.27
pr13	7049.83	7162.79	0.78	13.3	6869.75	6925.05	0.51	37.2	-3.32	6810.81	6851.70	0.38	204.3	-4.34
pr14	8174.15	8259.66	0.60	21.5	7863.06	7913.39	0.37	54.6	-4.19	7778.52	7834.57	0.47	306.0	-5.15
pr15	8805.48	8913.84	0.73	32.6	8579.72	8627.60	0.35	77.5	-3.21	8467.17	8506.31	0.33	431.2	-4.57
pr16	8104.44	8163.57	0.48	51.4	7756.60	7794.63	0.47	112.7	-4.52	7680.65	7721.30	0.23	624.2	-5.42
pr17	9936.62	10085.59	0.87	6.1	9647.28	9705.98	0.49	26.5	-3.76	9583.69	9619.17	0.27	123.4	-4.62
pr18	11844.70	11961.05	0.51	13.7	11512.10	11626.64	0.63	54.1	-2.80	11415.20	11468.00	0.30	244.4	-4.12
pr19	11936.75	12041.04	0.63	26.8	11681.70	11719.25	0.29	88.6	-2.67	11458.00	11537.61	0.36	433.2	-4.18
pr20	12275.69	12344.65	0.44	45.3	11916.00	11985.51	0.42	150.2	-2.91	11765.70	11820.05	0.36	738.4	-4.25
avg				22.3				63.9	-3.20				330.3	-4.34
%-gap$_{VNS}$		2.30				-0.97					-2.14			
%-time$_{VNS}$				5.8				16.6					86.0	
%-gap$_{MLVNS}$		4.30				0.96					-0.23			
%-time$_{MLVNS}$				13.3				38.1					196.8	

Table 5.13: Relative usage and success rate of VND neighborhoods for new larger PVRP instances.

l	VND		MLVND $I_{0.33}$		MLVNS+VND $I_{0.33}$	
	%-use	%-new best	%-use	%-new best	%-use	%-new best
1	70.17	89.93	31.74	83.68	53.39	85.48
2	7.22	0.00	12.63	0.52	10.17	0.14
3	7.22	0.18	12.51	0.54	10.09	0.28
4	7.09	0.03	12.39	0.12	9.95	0.05
5	7.07	8.89	12.36	13.22	9.93	12.75
6	0.85	0.65	9.34	1.37	3.48	0.98
7	0.39	0.32	9.03	0.54	2.99	0.32

inside the VNS are somewhat between the two others, i.e., in contrast to the MLVND the search process concentrates again more on the earlier neighborhoods.

5.8 Conclusions

We extended a recently proposed leading variable neighborhood search (VNS) with the multilevel refinement strategy to a multilevel VNS (MLVNS) for better solving periodic routing problems. A path based coarsening scheme is used that builds fixed (route) segments of customers accounting for the periodicity. The refinement process, i.e. starting at the coarsest level and iteratively refining until the original problem is reached again, is smoothly integrated into the VNS. Furthermore a suitable solution-based recoarsening is proposed that respects the structure of a given solution during coarsening.

We presented results on available benchmark test data as well as on newly generated larger instances that show the advantage of the multilevel VNS compared to the standard VNS, often yielding significantly better results in usually less CPU time. In general the performance gain on the PTSP instances is higher. To note is that the gain for both problems is smaller when also accepting worse solutions using the Metropolis criterion, yet the final solution quality is still better.

Further we also proposed a variable neighborhood descent (VND) which builds upon neighborhoods similar to those utilized in the shaking of the VNS. We subsequently derived a multilevel VND (MLVND) which represents a more classical multilevel refinement approach according to previous literature, i.e., improve until getting stuck at a local optimum and refine afterwards, executed in an iterated manner. At high levels the neighborhoods are able to impose major changes on the solution structure, while with increasing refinement the focus of the optimization lies automatically on the details. The MLVND clearly outperformed the VND with only a moderate increase in runtime, this time yielding a better performance for the PVRP. Finally, embedding the MLVND in a slightly changed VNS improved the results

once more, though clearly at the expense of longer runtimes. The initial assumption holds that these new approaches are especially appealing for large instances.

Potential Future Work

It might be interesting to perform longer test runs to further analyze the performance of the new approaches utilizing the multilevel refinement strategy. Another interesting point would be to create even larger instances having different characteristics. It might also be worthwhile to think about alternative ways of interweaving the refinement with the VNS, such as refining whenever the VNS gets stuck for a while; similar thoughts apply to the recoarsening. Also the interrelation of the multilevel extension and the Metropolis criterion could be a future topic for investigation, probably also considering to accept worse solutions for the MLVNS+VND approach. Finally, it should be relatively easy to adopt this promising multilevel refinement strategy for periodic routing problems also for other underlying algorithms.

CHAPTER 6

(PERIODIC) LOCATION-ROUTING PROBLEM

6.1 Introduction

Having obtained good results with matheuristics for the PVRPTW in Chapter 4 in course of the funded FWF project we were motivated to follow this promising line of research in the context of another problem. We deemed the location-routing problem a promising candidate for this: it is in many aspects similar to the former problem yet it is even "richer". Part of this work, at an earlier stage, was presented at the *7th International Workshop on Hybrid Metaheuristics* in 2010 (HM 2010) [164].

The *location-routing problem* (LRP) combines two classical $\mathcal{NP}$-hard problems: the *facility location problem* (FLP) and the *vehicle routing problem* (VRP). The LRP occurs when it is necessary to place some facilities at given locations, assign customers to them, and serve these customers by a fleet of vehicles, often imposing a limit on the maximal load (capacity) of a vehicle. Additionally, also capacity constraints on the facilities can and will be considered in the following. Hence, contrary to the simple out-and-back routes visiting single customers in the classical FLP, one is faced here with multi-stop routes. Solving the LRP demands a strategic (facility placement) and tactical (routing) planning task at the same time, contributing to the potential practical relevance of the LRP. Considering both aspects simultaneously is in favor of solving them in a subsequent way (usually starting with placing the facilities first), since the latter is more prone to yield suboptimal solutions [204].

An additional interesting strategic level can be incorporated by considering the LRP for a given planning period, resulting in the *periodic LRP* (PLRP). Here, specific customers must be served several times during the planning period.

The following definitions are targeted to the PLRP only since the LRP can be considered as

special case when having only a single day planning horizon. A planning horizon of t days, referred to by $T = \{1, \ldots, t\}$, is considered and it is defined on a complete, undirected, weighted graph $G = (V, E)$, with $V = V_C \cup V_D$ being the set of nodes, composed of n customers $V_C = \{0, \ldots, n-1\}$ and m potential depots $V_D = \{n, \ldots, n+m\}$, and $E = \{\{i, j\} \mid i, j \in V_C,\, i \neq j\} \cup \{\{i, j\} \mid i \in V_C,\, j \in V_D\}$ being the set of edges. Travel cost $c_{ij} \geq 0$, independent of the day, are given for any pair of nodes $i, j \in V$. Each depot $i \in V_D$ has an associated capacity W_i and opening costs O_i. Further, a homogeneous fleet of K vehicles, each having capacity Q, is available per depot and day. The fixed cost of using a single vehicle at least once during T is given by F, and each vehicle is limited to perform one single route per day. Further, each customer $j \in V_C$ has defined a total demand d_j, a visit frequency f_j, and a non-empty set $C_j \subseteq \{T' \mid T' \subseteq T,\, |T'| = f_j\}$ of allowed combinations of visit days. The actual demand of customer j on day l using visit combination $r \in C_j$ is assumed to be given by d_{jlr}.

The PLRP then aims at selecting facilities (depots) to be opened as well as a single visit combination per customer, and finding (at most) $N \leq K$ vehicle routes on each of the t days on G such that:

- Each route starts and ends at the same opened depot within the same day,
- each customer j belongs to f_j routes over the planning horizon at those days belonging to the selected visit day combination, overall satisfying its demand d_j and respecting the given d_{jlr} values,
- the total load of each route does not exceed vehicle capacity limit Q,
- for each opened depot i the total load of each route assigned to it on any day does not exceed depot capacity limit W_i,
- the total costs of opening depots, fixed costs for used vehicles, and corresponding travel costs are minimized.

A visualization of a solution (which is in fact a new best known one) for PLRP instance 200-10-1b is shown in Figure 6.1.

In this chapter we present a variable neighborhood search (VNS) suited for the periodic as well as the non-periodic LRP. For this we combine successful VNS variants/concepts for similar problems. To further improve the overall performance and at the same time continue investigating matheuristics, very large neighborhood searches based on integer linear programming are introduced, which are combined with the VNS in a fruitful way.

The remainder is organized as follows: Related work is presented in the next section, the VNS is the topic of Section 6.3, and the very large neighborhood searches are detailed in Section 6.4. Experimental results for both, LRP and PLRP, are given in Section 6.5. Section 6.6 finishes this chapter with concluding remarks.

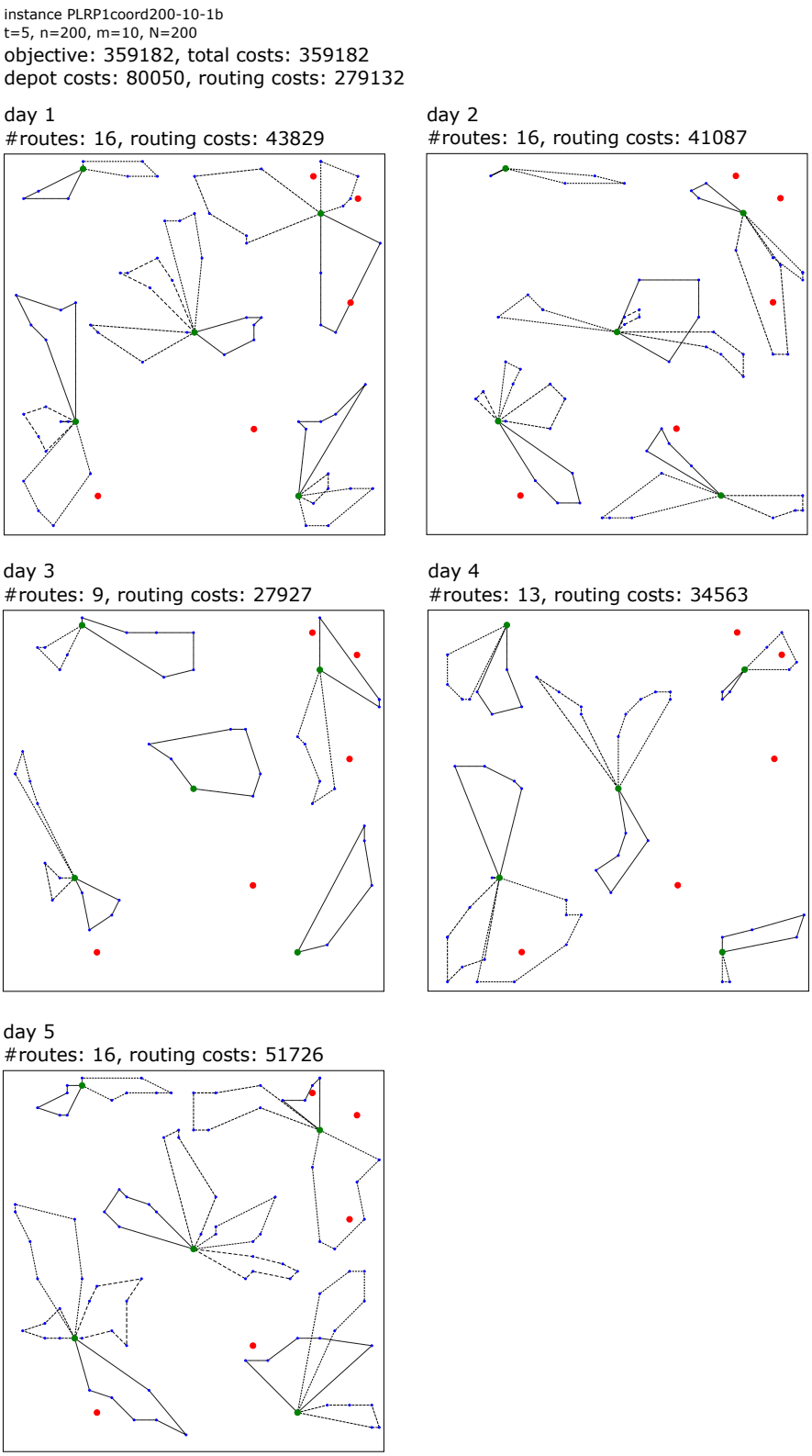


Figure 6.1: Visualized solution of PLRP instance 200-10-1b.

6.2 Related Work

The LRP with capacitated vehicles and depots was primarily dealt with in recent years only, earlier work often considered one of both restrictions exclusively. Among the currently leading methods for the so-called *general LRP* are (hybrid) metaheuristics by Prins et al. [176, 175] and Duhamel et al. [68, 69] as well as combinations of exact techniques and heuristics, e.g. [177]. Another recent approach based on simulated annealing is presented by Yu et al. [241]. As pointed out by them their heuristic is fairly standard, yet the applied solution encoding scheme was specially designed for this problem: basically a string consisting of (a permutation of) the customers and the potential depots is interspersed with so-called dummy zeros which terminate the current route by visiting the current depot again (besides terminating a route due to capacity constraints). All customers between two depot entries are served by the former depot, also placing such an entry at the beginning of the string. Such schemes have also been applied for other routing problems in the past.

We further mention two solution approaches which appeared very recently and in fact after our work has been published. Our former (but now outdated) results are included there. The first is an adaptive large neighborhood search by Hemmelmayr et al. [104]. Though they actually tackle the richer two-echelon VRP (2E-VRP), where two levels of facilities exist, also results for the LRP—being a special case of the former problem—are given. In their method they use several schemes for removing and reinserting customers as well as changing facilities. To our knowledge the most current work is by Contardo et al. [35] and proposes a GRASP plus ILP-based metaheuristic, where similar to our work the capability of the metaheuristic is enhanced by an ILP-based large neighborhood search. More details on it will be presented later in the corresponding section.

A survey of different LRPs and solution methods can be found in Nagy and Salhi [148]. The authors emphasize the high practical relevance of the LRP in supply chain management as well as a substantially increasing interest in the last years.

The PLRP was introduced only recently by Prodhon together with an iterative algorithm for solving it [179]. At the Hybrid Metaheuristics 2008 workshop, the same author(s) presented a sophisticated genetic algorithm [182] yielding improved results. Finally, at the Hybrid Metaheuristics 2009 workshop the so far best performing algorithm for the PLRP was again introduced by Prodhon [180], being a hybrid of evolutionary local search and path relinking, even more concentrating on the periodic aspect. A more recent article of the latter method is [181], where also other results are presented.

A similar VNS as the one applied here has been used for the periodic VRP (PVRP) in [105] and in Chapter 5, the PVRP with time windows (PVRPTW) in Chapter 4, as well as for the related multi-depot VRP with time windows [172]. As already mentioned, we draw upon the experience described therein and obtained by ourselves.

Furthermore, a very large neighborhood search similar to the one we present in Section 6.4.2 was introduced by De Franceschi et al. [58] for VRPs in general. The very large neighborhood search of Section 6.4.1 is related to it, but operates on a coarser level of the problem. A variant of it was recently successfully applied to the PVRPTW by us [161].

6.3 Variable Neighborhood Search for the (P)LRP

To smooth the search space to more easily overcome local optima our *variable neighborhood search* (VNS) relaxes the vehicle load as well as the depot load restrictions and adds penalties corresponding to the excess of these constraints to the cost function; more about setting them will be described in the results section. Though the method is typically able to find feasible solutions and improve upon them in short time, allowing infeasible solutions might help the overall search process.

An initial solution is created in the following way:

1. Choose a single visit day combination $r \in C_j$ per customer $j \in V_C$ at random.
2. Determine the actual lower bound for the accumulated depot capacity based on the previously selected visit combinations: $W_{\text{LB}} = \underset{l \in T}{\operatorname{argmax}} \sum_{j \in V_C} d_{jlr}$, for the LRP this equals the sum of the customer demands: $\sum_{j \in V_C} d_j$.
3. Select depots to be opened at random one by one until their combined capacity is greater than or equal to W_{LB}.
4. Repeat the following steps for each day l in T:
   a) Assign each customer to be visited on day l to its nearest opened depot, thereby considering penalized depot load violations.
   b) Apply the Clarke and Wright savings algorithm [32] until no more routes can be feasibly merged due to the vehicle load restriction.
   c) In the event of ending up with more than N routes, least customer routes are selected and customers contained therein are re-added in a greedy way on that day, allowing (penalized) excess of vehicle load. [Note that due to the structure of the considered instances this never happened in our tests.]

We deem the initialization procedure rather straightforward, being not that sophisticated as those described in [180]. Nevertheless, we felt (and also experienced in preliminary tests) that having initial solutions of high(er) quality is not necessarily beneficial for the subsequent optimization process, and we apply the procedure solely for initialization purposes, too.

In the shaking phase we utilize five different neighborhood structures, each with several moves of increasing perturbation size (denoted by δ), yielding a total of 18 shaking neighborhoods (i.e. $k_{\max} = 18$) for both LRP variants. Next, these five basic neighborhood structures are described.

Exchange-segments: Exchange two random segments of variable length between two routes at the same depot and on the same day. One of the segments might be empty, realizing a customer relocation.

Exchange-segments-two-depots: In principle similar to the previous *exchange-segments* but both segments are located at two distinct depots. This neighborhood structure facilitates

Table 6.1: (Fixed) Shaking neighborhood order used for the LRP.

k	$\mathcal{N}_k$
1–5	*Exchange-segments* of maximal length $\delta = k$
6	*Exchange-segments* of maximal lengths bounded by corresponding route size ($\delta = 6$)
7–11	*Exchange-segments-two-depots* of maximal length $\delta = k - 6$
12	*Exchange-segments-two-depots* of maximal lengths bounded by corresponding route size ($\delta = 6$)
13–15	*Change-two-depots* is applied up to $\delta = k - 12$ times
16–18	*Change-depot* is applied up to $\delta = k - 15$ times

the partitioning of customers to the opened depots. *Exchange-segments* is applied in case only a single depot is opened.

Change-two-depots: A previously closed depot is opened and subsequently another opened depot is closed such that the actual lower bound regarding the depot capacity W_{LB} is still satisfied. After opening the selected depot it is tried to relocate routes to it in a cost saving manner, which means to place the new depot at minimum cost between two consecutive customers. Afterwards the routes of the depot to be closed are relocated to an opened depot one by one in the least expensive way (as before also using the penalized objective function). After such a neighborhood move the number of opened/closed depots stays the same. A prerequisite is that at least one of all the available depots is not opened yet, the following *change-depot* is applied otherwise.

Change-depot: Here the status of a single depot is changed, i.e. from closed to opened or vice versa. The necessary route relocation operations are applied as for *change-two-depots*. Finally, this neighborhood move allows to alter the number of opened/closed depots. However, also here it is guaranteed that the actual lower bound regarding the depot capacity W_{LB} is still satisfied.

Change-visit-combinations: For tackling the periodic aspect of the PLRP it is necessary to enable the VNS to change the selected visit combination per customer. As for the periodic VRPs, it turned out that randomly changing several visit combinations with greedy insertion for the new visit days (also allowing to reassign the same visit combination) performs very well.

In this work we only consider a fixed shaking neighborhood order, which is detailed in Table 6.1 for the LRP and in Table 6.2 for the PLRP. Apart from the obvious difference of using *change-visit-combinations* for the PLRP only, a greater focus is laid on exchanging segments for the LRP instead.

For intensification we apply the well-known 3-opt intra-route exchange procedure in a best improvement fashion, only considering routes changed during shaking, and re-applying the operator until no more improvement is possible. Afterwards, each new incumbent solution

Table 6.2: (Fixed) Shaking neighborhood order used for the PLRP.

k	$\mathcal{N}_k$
1–6	*Change-visit-combinations* for up to $\delta = k$ customers
7–9	*Exchange-segments* of maximal length $\delta = k - 6$
10–12	*Exchange-segments-two-depots* of maximal length $\delta = k - 9$
13–15	*Change-two-depots* is applied up to $\delta = k - 12$ times
16–18	*Change-depot* is applied up to $\delta = k - 15$ times

is also subject to a 2-opt* inter-route exchange heuristic [173]. Hereby for each pair of routes of the same day and depot all possible exchanges of the routes' end segments are tried, also applied repeatedly and in a best improvement fashion. For the LRP this is further applied with a probability of 0.2 to each newly derived solution lying within 3% to the current incumbent.

To often enhance the overall VNS performance quite substantially, sometimes also solutions having a worse objective value are accepted in addition to better solutions. As in [105, 158] this is done in a randomized way using the Metropolis criterion like in simulated annealing [123]. A linear cooling scheme is used in a way such that the acceptance rate of worse solutions is nearly zero in the last iterations. Though this is somewhat of a hybrid variant on its own, we still denote it as VNS. For completeness the proposed VNS is depicted in Algorithm 17. We want to note that similar to [105] we also tried other acceptance schemes here. Among them were threshold accepting and skewed VNS as in [105] but also the sequence heuristic and great deluge approach. Although especially the skewed VNS showed some potential none of them could yield a consistent gain in solution quality.

6.4 ILP-based Very Large Neighborhood Searches

The general idea of *very large(-scale) neighborhood search* (VLNS) was already outlined in Section 2.5.6. In the following we will introduce two VLNS procedures based on *integer linear programming* (ILP) applicable to the LRP as well as to the PLRP.

6.4.1 VLNS Operating on Routes

The first VLNS deals with (re-)locating whole routes to depots, as well as opening/closing depots in the course of the application. In fact, we implemented a simpler version (denoted as V_1) and a more sophisticated one (denoted as V_2) of it, the latter building upon a set covering formulation which is to some degree similar to the one used for the PVRPTW in Section 4.7 and being especially appealing for the PLRP in principle. V_1 is a special case of V_2 and basically resembles the procedure applied in [177] for the LRP. There the authors apply a Lagrangian relaxation approach on the FLP subproblem via considering the aggregated

Algorithm 17: VNS for the (P)LRP with initial solution x_{init}

$x_{best} \leftarrow x_{init}$; `// incumbent solution (though most likely infeasible at the beginning)`
$x_{vns} \leftarrow x_{init}$; `// solution utilized for shaking`
$k \leftarrow 1$; `// shaking strength`
initialize temperature to (average travel costs)/5
while *VNS stopping condition is not met* **do**
  while $k \neq k_{\max}$ **do**
    `// shaking:`
    Select $x' \in \mathcal{N}_k(x_{vns})$
    `// local search:`
    $x'' \leftarrow$ repeated first improving 3-opt neighbor of x'
    `// probably apply additional local search:`
    if *tackling the LRP and* $x'' < 1.03 \cdot x_{best}$ *and with probability 0.2* **then**
      apply 2-opt* improvement on x''
    `// possibly update incumbent:`
    if x'' *is feasible and better than* x_{best} **then**
      `// additional local search:`
      apply 2-opt* improvement on x'' if not already done before
      if x'' *made infeasible* **then**
        revert previous local search
      $x_{best} \leftarrow x''$; `// set new incumbent`
      $k \leftarrow 1$; `// reset shaking`
    `// possibly update solution utilized by VNS:`
    if *(*x'' *is better than* x_{vns}*) or (*x'' *is worse than* x_{vns} *but accepted due to Metropolis criterion)* **then**
      $x_{vns} \leftarrow x''$
      $k \leftarrow 1$; `// reset shaking`
    else $k \leftarrow k + 1$; `// increase shaking`
    every 100^{th} iteration apply linear cooling
  `// finished VNS iteration`
  $k \leftarrow 1$; `// reset shaking`

routes as super-customers. For this several subproblems need to be solved many times, as well as a lower and upper bound must be computed. In contrast, we directly solve the resulting ILP model, which is presented in the following and is also applicable to the PLRP:

$$(\mathrm{V}_1) \qquad \min \quad \sum_{i \in V_D} O_i \, y_i + \sum_{l \in T} \sum_{i \in V_D} \sum_{j \in R(l)} C^L_{ij} \, x_{ij} + \sum_{i \in V_D} F \, z_i \tag{6.1}$$

$$\text{subject to} \quad \sum_{i \in V_D} x_{ij} = 1 \qquad \forall l \in T;\, \forall j \in R(l) \tag{6.2}$$

$$\sum_{j \in R(l)} x_{ij} \leq z_i \qquad \forall i \in V_D;\, \forall l \in T \tag{6.3}$$

$$\sum_{j \in R(l)} L_j \, x_{ij} \leq W_i \qquad \forall i \in V_D;\, \forall l \in T \tag{6.4}$$

$$\sum_{i \in V_D} W_i \, y_i \geq W_{\mathrm{LB}} \tag{6.5}$$

$$z_i \geq y_i \qquad \forall i \in V_D \tag{6.6}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in V_D;\, \forall l \in T;\, \forall j \in R(l) \tag{6.7}$$

$$y_i \in \{0, 1\} \qquad \forall i \in V_D \tag{6.8}$$

$$z_i \in \mathbb{N} \qquad \forall i \in V_D \tag{6.9}$$

The objective (6.1) is to minimize costs for opening depots, routing costs (here only route location costs), as well as fixed costs for vehicles. The set of all considered (aggregated) routes per day l is denoted by $R(l)$, the least cost of locating it at depot j is C^L_{ij}. We introduce following binary variables: x_{ij} (6.7) indicating whether or not depot i hosts route j, y_i (6.8) if depot i is opened, as well as integer variables z_i (6.9) stating the maximum number of routes located at depot i of all days, used for the vehicle fixed costs in (6.1). The following restrictions are applied: Each route must be located at one depot (6.2), the value of the z_i variables is determined by (6.3), and the load of a depot must be respected (6.4), with L_j denoting the load of route j. The last two constraints are to strengthen the model: the accumulated capacity of the selected depots must be at least the actual corresponding lower bound (6.5), and the depot with its corresponding maximum number of routes are coupled via (6.6); refer to [6], though they used the minimal number of depots in (6.5). A visualization of an application of V_1 is shown in Figure 6.2.

The previous model is built for a given feasible solution and the solution's routes are used for the corresponding day only.

As already mentioned, the more sophisticated variant formulates a similar, yet potentially much larger neighborhood as a set covering model. Therefore, the main difference between V_1 and V_2 is that although both operate on whole routes, the latter takes the single customers into account. The whole model including constraints for both the LRP and the PLRP can be stated as (also refer to [6] for a similar model for the LRP only):

$$(\mathrm{V}_2) \qquad \min \sum_{i \in V_D} O_i \, y_i + \sum_{l \in T} \sum_{i \in V_D} \sum_{j \in R(l)} (C^L_{ij} + C^R_j) \, x_{ij} + \sum_{i \in V_D} F \, z_i \tag{6.10}$$

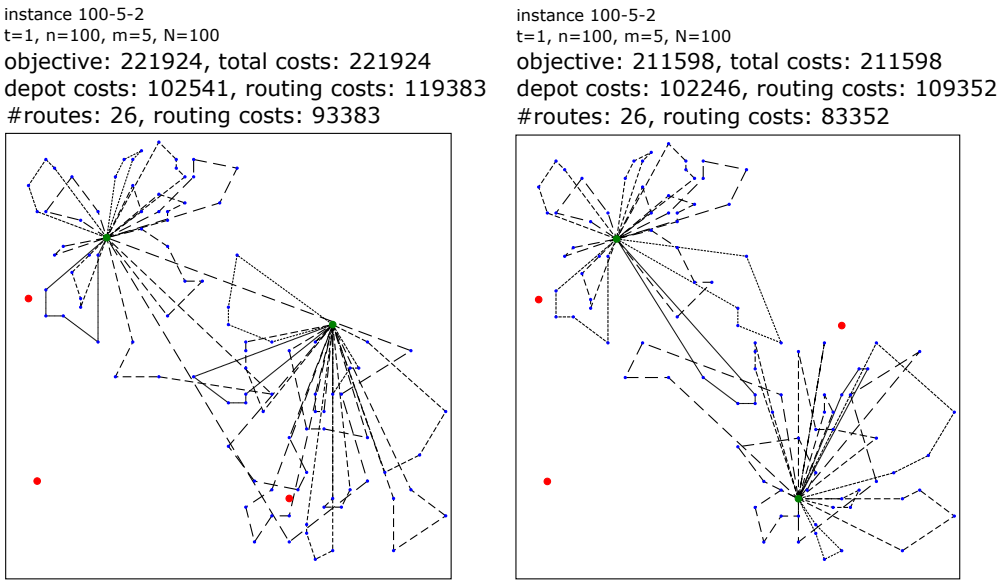


Figure 6.2: Visualized application of ILP neighborhood V_1 on a solution of LRP instance 100-5-2 (left: before, right: after application).

subject to (6.3)–(6.4)

$$\sum_{r \in C_n} p_{nr} \geq 1 \qquad \forall n \in V_C \qquad (6.11)$$

$$\sum_{i \in V_D} \sum_{j \in R(l)} a_{nj}\, x_{ij} - \sum_{r \in C_n} b_{nrl}\, p_{nr} \geq 0 \qquad \forall n \in V_C;\, \forall l \in T \qquad (6.12)$$

$$\sum_{i \in V_D} \sum_{j \in R(l)} x_{ij} \leq N \qquad \forall l \in T \qquad (6.13)$$

$$\sum_{j \in R(l)} a_{nj}\, x_{ij} - y_i \sum_{j \in R(l)} a_{nj} \leq 0 \qquad \forall n \in V_C; \forall i \in V_D; \forall l \in T \qquad (6.14)$$

$$\sum_{i \in V_D} W_i\, y_i \geq W_{\text{LB}} \qquad (6.15)$$

$$\sum_{i \in V_D} W_i\, y_i - \sum_{n \in V_C} \sum_{r \in C_n} b_{nrl}\, p_{nr}\, d_{nlr} \geq 0 \qquad \forall l \in T \qquad (6.16)$$

$$\sum_{l \in T} \sum_{i \in V_D} \sum_{j \in R(l)} x_{ij} - \left\lceil \frac{\sum_{n \in V_C} d_n}{Q} \right\rceil \geq 0 \qquad (6.17)$$

$$\sum_{l \in T} \sum_{i \in V_D} \sum_{j \in R(l)} d'_{nj}\, x_{ij} \geq d_n \qquad \forall n \in V_C \qquad (6.18)$$

$$\sum_{i \in V_D} \sum_{j \in R(l)} d'_{nj}\, x_{ij} - d_{jlr}\, b_{nrl}\, p_{nr} \geq 0 \qquad \forall n \in V_C;\, \forall r \in C_n;\, \forall l \in T \qquad (6.19)$$

(6.6)–(6.9)

$$p_{nr} \in \{0,1\} \qquad \forall n \in V_C;\ \forall r \in C_n \qquad (6.20)$$

Beside the adopted constraints and variables from V_1 following additions were made: For each customer $n \in V_C$, binary variables p_{nr} (6.20) indicate whether or not visit combination $r \in C_n$ is chosen. The objective function (6.10) now also includes routing costs C_j^R for visiting the customers in route j (without the depot connection). Cover constraints (6.11) guarantee that at least one visit day combination is selected per customer, visit constraints (6.12) link the routes and the visit combinations, whereat a_{nj} and b_{irl} are binary constants indicating whether or not route j visits customer n and if day l belongs to visit combination $r \in C_n$ of customer n, respectively, and the number of routes per day may not exceed N (6.13). Again, constraints (6.14)–(6.17) strengthen the model: the routes containing customer n located at depot j and the depot itself are coupled in (6.14), (6.15) is like for V_1 and only used in case of the LRP, further (6.16) is a special variant instead of the latter for the PLRP, incorporating the periodic aspect, and finally, the minimal amount of vehicles (routes) necessary is set by (6.17). The motivation for a set covering model was to be able to exploit the routes of more than one feasible solution. A consequence with respect to the PLRP is that the selected visit combination of several customers might (or better need) to eventually change. However, the fact that the daily demand of a customer depends on the chosen visit combination does not "suit the model very well": In order to build a feasible PLRP solution out of the ILP solution it might be necessary to change the amount delivered to a customer, which is a potential problem if the amount has to be increased due to given vehicle load constraints. At least we alleviate this problem by introducing constraints (6.18) and (6.19), reducing the chance of a conflict by forcing a certain amount to be delivered per customer (and chosen visit combination), with d'_{nj} denoting the amount delivered to customer n in route j. If all fails, the customer is tried to be added in a feasibly way via greedy insertion. Finally, also over-covered customers need to be dealt with: we simply remove all but their first occurrence.

The model of V_2 is created for a given set of feasible solutions, again using the solution's routes for the corresponding day only. This solution set always contains the current incumbent solution as well as a preferred number of optional solutions which are selected via binary tournament from the set of all improved solutions (found during the run up to that time). This way of handling it has the advantage of keeping a certain diversity (assuming enough available solutions) yet still favoring good solutions, as well as having to store no additional solutions. The current incumbent further acts as a starting solution for the ILP solver.

Basically, this model could be applied as the master problem of a classical column generation approach as well (see Section 2.3.1), of course a suitable subproblem for generating new columns would have to be defined. However, our intention is to have a (relatively) fast supplementary neighborhood. Therefore we restrict ourselves to producing the columns (routes) with the VNS only, i.e. having a pure metaheuristic column generation.

To control (limit) the effort and hence runtime for solving an instance of V_1 and V_2, there is the possibility to set an upper bound on the number of depots the routes might be located to. In this case the depots are selected per route in the order of increasing costs for locating this

route. The only exception being the best solution in the given set, its routes are allowed to be located at any depot.

6.4.2 VLNS Operating on Customers

Our second type of VLNS (denoted by V_3) operates on the customer level, it is therefore responsible for finer-grained optimization, yet it bears quite some resemblance with V_2 in that it also makes use of a set covering formulation. It is similar to the so-called *ILP-based refinement heuristic* presented by De Franceschi et al. [58]. The idea is to extract sequences of customers from given routes, re-connect the disconnected route parts, and then find an optimal allocation of these sequences to possible insertion points, i.e. between any two (remaining) consecutive customers. Whenever the customer sets of sequences are not disjoint, this neighborhood is hard to solve and an ILP formulation as a set covering model is appropriate. A suitable one for our setting is the following:

$$(V_3) \qquad \min \quad \sum_{j \in R} C_j^R + \sum_{s \in S} \sum_{i \in I_s} C_{si}^I \, x_{si} \tag{6.21}$$

$$\text{subject to} \quad \sum_{s \in S : n \in s} \sum_{i \in I_s} x_{si} \geq 1 \qquad \forall n \in V_C \tag{6.22}$$

$$\sum_{s \in S : i \in I_s} x_{si} \leq 1 \qquad \forall i \in I \tag{6.23}$$

$$L_j + \sum_{s \in S} \sum_{i \in I_s \cap I(j)} L'_s \, x_{si} \leq Q \qquad \forall j \in R \tag{6.24}$$

$$\sum_{j \in R(i)} L_j + \sum_{s \in S} \sum_{i \in I_s \cap I(j)} L'_s \, x_{si} \leq W_i \qquad \forall i \in V_D \tag{6.25}$$

$$x_{si} \in \{0, 1\} \qquad \forall s \in S; \forall i \in I_s \tag{6.26}$$

The objective (6.21) is to minimize the insertion costs C_{si}^I of the sequences $s \in S$ in their possible or allowed insertion points $i \in I_s$. Each extracted customer must be covered by at least one of the sequences containing it (6.22). Further, each insertion point can hold at most one sequence (6.23). Constraints (6.24) and (6.25) are responsible to enforce limits on vehicle and depot loads, respectively. Here, L_j again denotes the load of route j and L'_s the load of sequence s, $I(j)$ are all insertion points provided by route j, and $R(i)$ are all routes located at depot i.

Several methods were proposed in the literature for similar or related neighborhoods to extract customers from given routes. Among them to pick the customers inducing the greatest detour [203], select a seed node and neighbored nodes, to select all odd or even customers of a route, or just select them at random on a per route basis; refer to [58] for more details. Here we settle for a simple model: Select the customers independent of the actual routes, since it is to be expected that no selection scheme is generally better as all others, and preliminary results showed no great difference, too. Therefore we proceed in the following way: Choose a preferred number of consecutive iterations $iter_{V_3}$ of V_3, randomly partition all customers

into equally sized sets N_i with $|N_i| = n/iter_{V_3}$, and finally perform V_3 $iter_{V_3}$ times, for each iteration i using set N_i as customers to be extracted as well as considering a specific set of feasible solutions. The latter is created as for V_2 described in Section 6.4.1. The routes of the current incumbent represent the considered set of routes R, and together with the sequences contained therein they are used as a starting solution for the ILP solver. Initially all possible sequences from the given solutions' routes are extracted adhering to the pre-selected customers, and the potential insertion points are determined while processing routes of set R. This way we omit the costly generation of sequences via making use of the linear programming dual values as in [58], yet (most likely) also obtain non-disjoint sequences, hence like for V_2 we again exploit the information of several solutions. Additionally, for each sequence of length more than one we create all possible sequences containing one of the customers each. Again, a customer is only accepted in the resulting solution on its first occurrence. Note that V_3 is applied on a per day basis only, which has to be considered when using it for the PLRP. The possible insertion points I_s per sequence s are determined by taking the x% least costly ones, also omitting infeasible pairings.

To conclude the VLNS section, in [35] basically a mixture of both presented ILP-based very large neighborhoods is used. I.e. they simultaneously locate facilities and reallocate customers to routes assigned to these facilities. In another step the same ILP is iteratively solved by column generation to further improve the solution. Naturally this also yields a notable increase in runtime. In the following section we will compare, among others, the results of both methods.

6.5 Experimental Results

The algorithms have been implemented in C++, compiled with GCC 4.5 and executed on a single core of a 2.53 GHz Intel Xeon E5540 with 3 GB RAM dedicated per core, though the memory consumption is not an issue here. The general purpose MIP solver IBM ILOG CPLEX version 12.2 is used to solve the VLNS' ILP models.

For both problem variants we took freely available benchmark data sets which were collectively provided by Caroline Prodhon [178]. They comprise 79 and 30 instances for the LRP and the PLRP, respectively. The *Prodhon instances* differ in the number of customers n, depots m, and clusters, as well as in the vehicle capacity ('a' denotes low, and 'b' high) and are named: n-m-#clusters[a,b]. The PLRP instances further span a working week, i.e. five working days and two idle days and customers must be visited one, two, or three times, offering five, three, and one visit combinations, respectively. We refer to [178, 180] for computing the daily demands d_{jlr}. For the *Tuzun and Burke LRP instances* we give the identifier, the number of customers is one of $\{100, 150, 200\}$ and the number of depots either 10 or 20. Finally, the *Barreto LRP instances* are named as: identifier-nxm. More details about the instances can be found in [177] regarding the LRP and in [180] for the PLRP.

Initially we settled to use for all Prodhon instances a constant penalty weighting of 10000 and an initial temperature of 500 for the Metropolis acceptance criterion, applying linear cooling every 100 iterations, thereby reducing the temperature by $500 \cdot 100/max\_iterations$; as was

done in [164]. Later on we also did tests on other instance sets and it became evident that the penalty weights as well as the temperature need to be properly adjusted according to the instance at hand to yield the desired effect. Choosing wrong values might otherwise even compromise the method. Due to this a simple scheme was sought allowing to set both values automatically. In the end we applied a similar strategy as for the PVRP(TW) in the previous chapters: the penalty weighting is set consistently to the averaged inter-customer distances and the initial temperature is set to one tenth of this value for the LRP and to one fifth for the PLRP. Linear cooling is still applied every 100 iterations with the same temperature decrease than before.

The combination with the VLNS variants is performed in the following way: V_1 is applied on each new incumbent solution since it can be solved quite fast (denoted by VNS+V_1), hence in a dynamic way. The other two, more demanding VLNS are applied in a static way by deciding a-priori how often they should be applied and executing them after fixed intervals of the VNS, e.g. running 10^6 VNS iterations in total and setting 10 applications of the VLNS, the latter is applied after every 10^5 iterations of the VNS. Thereby we either solely apply V_2 (resulting in VNS+$V_{1,2}$) or V_3 (resulting in VNS+$V_{1,3}$), or V_2 directly followed by V_3 (denoted by VNS+$V_{1,2,3}$). The number of additional feasible solutions used for V_2 is ten and for V_3 nine, $iter_{V_3}$ is set to four. As an upper bound for considered depots in V_2 we chose five, and in V_3 only the least costly 50% insertion points are considered. Note again, the initial depot location is always retained. All these values were determined in preliminary test runs. Basically, the runtime per application of each VLNS is restricted to two seconds, although this limit is seldom reached. Finally, each new VLNS incumbent is subject to 2-opt*, those of V_3 also to 3-opt. These are all high-level integrative combinations, where the VLNS methods are incorporated in the VNS.

For each algorithm setting we perform 10 runs per instance, and state the average costs (avg), corresponding coefficients of variation (i.e. standard deviations divided by average values) in percent (CV [%]), and the runtimes in seconds (t[s]). For the more detailed, longer runs also the costs of the best run out of these 10 is given (min). Furthermore, we state the average percentage gap to the so far best known solutions including all (mostly recent) previous results in the literature of the average solution values (%-gap $_{\text{avg}}$) as well as of the best solutions obtained in the 10 runs each (%-gap $_{\text{min}}$).

We will proceed by comparing our enhanced VNS methods to solely applying VNS, followed by a comparison to leading methods of the corresponding problem variant in terms of solution quality and runtime.

6.5.1 Results on the PLRP

At first we did some preliminary test runs to investigate the effect of the different VNS plus VLNS combinations. VNS is applied for 10^5 iterations in total, optionally applying V_2 and/or V_3 after every 10^4 iterations (10 times); the results are shown in Table 6.3. Only using V_1 already yields a notable improvement, additionally applying V_2, i.e. the combined variant VNS+$V_{1,2}$, further improves the results, regarding both the average and the best solution values. So it seems incorporating the periodicity in V_2 pays off. Interestingly, V_3 leads

Table 6.3: Results of VNS and VNS plus VLNS combinations on the Prodhon PLRP instances using 10^5 iterations.

Instance	VNS			VNS+V_1			VNS+$V_{1,2}$			VNS+$V_{1,3}$			VNS+$V_{1,2,3}$		
	avg	CV [%]	t [s]	avg	CV [%]	t [s]	avg	CV [%]	t [s]	avg	CV [%]	t [s]	avg	CV [%]	t [s]
20-5-1	79079.20	0.61	1.7	79546.50	1.10	1.8	79790.70	1.28	2.1	79875.30	0.53	2.1	79522.50	0.81	2.4
20-5-1b	78829.70	2.73	1.8	79206.20	2.56	1.9	77791.50	2.09	2.1	78844.70	2.62	2.3	79065.50	2.39	2.5
20-5-2	80202.90	0.82	1.7	80162.50	0.72	1.7	79490.90	1.42	2.2	79997.40	1.16	2.0	79868.80	1.11	2.4
20-5-2b	63300.30	0.88	1.7	63408.40	1.58	1.6	63037.70	0.89	2.1	63514.60	0.89	2.1	63625.60	1.83	2.4
50-5-1	153411.60	1.66	3.4	152451.70	0.88	3.7	152238.00	1.73	4.7	151268.80	1.31	4.5	151653.70	1.00	5.4
50-5-1b	142500.50	1.52	3.7	141690.80	0.85	4.0	140947.50	1.94	5.5	141766.30	1.34	4.7	141973.10	1.27	6.0
50-5-2	144327.70	1.29	3.4	143676.00	0.83	3.5	142436.40	0.56	5.6	143274.70	0.95	4.1	142478.80	0.97	5.8
50-5-2b	115400.00	1.68	3.4	116741.30	2.07	3.5	115926.40	1.86	5.1	115378.40	1.51	4.2	116264.80	1.54	5.8
50-5-2BIS	175304.80	1.71	2.7	174386.20	2.14	3.0	174825.20	1.75	4.5	174458.10	1.50	3.5	174767.70	2.72	4.8
50-5-2bBIS	104751.70	2.13	3.1	105168.50	2.35	3.2	104134.40	1.14	4.1	105950.20	2.66	4.0	104515.40	2.16	4.9
50-5-3	158870.20	1.95	3.3	157284.60	0.76	3.6	157376.70	1.68	4.6	158006.80	1.33	4.2	156496.70	0.89	5.1
50-5-3b	113800.20	1.52	3.5	113261.90	2.14	3.7	112314.80	1.59	5.0	113960.10	2.24	4.3	112235.00	1.08	5.7
100-5-1	356657.30	1.00	5.3	357192.10	0.88	5.3	352195.40	1.16	7.1	355113.40	1.37	6.9	354109.40	1.54	8.1
100-5-1b	240643.10	1.69	6.6	237007.80	1.32	6.7	236916.90	1.45	8.6	238111.00	1.73	8.5	235487.20	1.36	9.7
100-5-2	271884.40	1.35	5.3	271378.50	0.84	5.3	270047.00	1.13	8.3	270476.50	1.03	7.0	269231.70	1.25	8.9
100-5-2b	170139.20	1.60	5.2	168906.80	1.29	5.3	169266.20	1.76	7.8	172698.70	1.41	6.9	167610.50	1.02	8.5
100-5-3	228007.90	1.20	5.2	227561.90	1.02	5.4	225608.50	1.17	8.9	228104.60	1.25	6.8	225191.80	1.11	9.4
100-5-3b	178278.10	1.16	6.5	177974.10	1.40	6.8	176035.60	1.73	10.1	177392.20	1.21	8.6	177111.70	1.45	11.1
100-10-1	270383.30	1.19	9.2	265443.10	1.42	10.9	263635.70	1.08	22.3	265233.90	0.73	12.4	264774.90	0.95	23.2
100-10-1b	217477.30	2.08	8.6	214547.10	1.30	10.5	215792.10	1.68	20.0	217726.20	1.70	12.2	216959.50	1.33	19.1
100-10-2	268995.80	0.94	6.2	267025.00	1.23	7.7	267754.90	1.01	15.5	268647.80	1.31	9.2	266471.70	1.27	15.0
100-10-2b	175366.70	1.74	6.7	174171.20	1.36	7.9	175383.00	1.26	11.9	175722.90	1.41	9.7	176397.80	2.25	13.0
100-10-3	266926.60	1.25	7.2	263543.90	1.35	8.9	264155.70	1.44	17.0	266119.60	0.95	10.3	263964.50	0.78	16.0
100-10-3b	200396.20	2.03	8.6	197000.20	1.57	9.6	199448.70	1.15	15.4	201774.10	1.26	10.8	198643.50	1.15	16.0
200-10-1	454664.50	0.58	14.3	448954.70	0.35	17.1	451892.20	1.21	28.2	454979.10	1.41	19.8	451096.30	1.34	27.3
200-10-1b	385855.90	1.30	15.7	384340.40	0.92	18.7	386983.00	1.04	30.3	387602.70	1.61	24.1	385048.90	1.60	33.3
200-10-2	397318.90	0.58	11.9	393381.80	0.78	13.0	388849.60	1.16	23.2	399891.70	0.89	16.5	393020.80	0.77	21.4
200-10-2b	326984.60	1.64	12.0	325399.40	0.94	14.3	324441.70	1.42	21.8	328909.00	0.92	19.1	328219.80	1.70	24.9
200-10-3	558626.40	0.68	11.4	559491.50	1.06	14.6	555431.90	1.40	25.2	567518.10	1.60	18.3	559833.00	1.28	24.9
200-10-3b	364258.00	1.03	12.3	360919.70	1.02	14.9	360204.00	0.84	22.4	365983.20	1.38	18.7	358196.80	0.94	24.6
avg	224754.77	1.38	6.4	223374.13	1.27	7.3	222811.74	1.37	11.7	224943.34	1.37	8.9	223127.91	1.36	12.3
%-gap $_{avg}$	-0.62			-1.11			**-1.42**			-0.61			-1.26		
%-gap $_{min}$	-2.62			-2.98			**-3.34**			-2.75			-3.28		

to a decrease of solution quality on average and obviously is not well suited for the PLRP in this constellation. Especially when combined with V_1 it neutralizes the potential gain when solely using V_1. Seemingly V_3 works against the optimization process and probably promotes getting stuck in (unfavorable) local optima. However, the bad performance of V_3 for the PLRP is most likely also related to the structure of the instances considered, since they rather promote many small routes, offering less potential for more sophisticated intra-route improvement.

Naturally, when applying the VLNS variants an increase in runtime occurs, in fact it roughly doubles it for these runs, but this factor is approximately constant, i.e. does not strongly depend on the size of the instance.

The second round of test runs devotes more runtime and we apply the VNS and the previously best performing VNS plus VLNS variant VNS+$V_{1,2}$ (which will be denoted as VNS+ILP in the following): VNS is run for 10^6 iterations and V_2 is applied after every $2{\cdot}10^4$ iterations (50 times). Table 6.4 shows the corresponding results, as usual the statistical significance tests were performed with a Wilcoxon rank sum test with an error level of 5%. Also in these tests VNS+ILP yields a consistent and notable improvement over VNS, for nine out of 30 instances (30%) even a significant improvement. This time the increase in runtime is on average about 50%, which is less than before (due to fewer applications of V_2)

Table 6.4: More detailed results of VNS and best performing VNS plus VLNS combination on the Prodhon PLRP instances using 10^6 iterations, significant improvements are marked bold.

Instance	VNS				VNS+ILP (VNS+$V_{1,2}$)			
	min	avg	CV [%]	t [s]	min	avg	CV [%]	t [s]
20-5-1	78477.00	78670.20	0.30	16.7	78477.00	78828.00	0.45	18.9
20-5-1b	76102.00	77119.00	1.59	19.0	76102.00	77196.10	1.26	19.7
20-5-2	77784.00	78369.60	0.89	17.8	77784.00	78347.00	0.60	18.6
20-5-2b	62133.00	62473.70	0.48	16.5	62133.00	62536.40	0.42	18.9
50-5-1	147621.00	148713.10	0.81	36.0	145639.00	148465.00	1.02	43.9
50-5-1b	134997.00	138621.00	1.42	37.3	136412.00	138618.90	1.06	45.7
50-5-2	139556.00	141328.50	0.68	32.6	138486.00	**140028.50**	1.00	43.0
50-5-2b	111413.00	112768.10	1.02	34.0	110526.00	112028.60	0.73	42.9
50-5-2BIS	169350.00	170377.70	0.28	27.5	169075.00	170160.10	0.31	34.3
50-5-2bBIS	101951.00	102749.90	0.54	30.8	101501.00	**102100.40**	0.47	36.4
50-5-3	153687.00	154270.40	0.28	33.4	152530.00	154099.60	0.90	40.7
50-5-3b	109631.00	110234.80	0.42	34.7	109047.00	110331.90	0.73	42.0
100-5-1	335702.00	341514.00	0.90	52.3	337892.00	342945.90	0.70	65.7
100-5-1b	224085.00	229494.10	1.17	56.6	223757.00	227701.30	1.19	77.2
100-5-2	261091.00	263952.10	0.73	50.3	257710.00	**261394.90**	0.79	66.0
100-5-2b	162963.00	164853.90	0.99	48.6	162799.00	164494.60	0.70	62.5
100-5-3	218750.00	220264.10	0.53	50.5	214118.00	219215.00	0.92	71.0
100-5-3b	171363.00	173188.70	0.68	66.8	166865.00	**171381.90**	1.05	85.8
100-10-1	255630.00	259677.60	0.84	92.0	257114.00	259201.30	0.49	159.4
100-10-1b	208410.00	210398.10	0.78	91.3	207574.00	**208568.50**	0.38	146.7
100-10-2	258611.00	262216.30	0.78	57.8	255345.00	**259468.80**	1.40	105.7
100-10-2b	166703.00	169311.50	1.05	64.9	167817.00	169623.40	0.66	93.6
100-10-3	253280.00	258223.40	1.21	69.3	254472.00	256907.60	1.08	131.5
100-10-3b	189370.00	191316.80	1.37	85.7	189557.00	192022.60	1.19	120.5
200-10-1	431183.00	436251.40	0.70	126.7	431131.00	**433030.70**	0.46	216.2
200-10-1b	364583.00	370378.20	0.79	152.8	359182.00	367650.00	1.49	240.1
200-10-2	380989.00	384230.40	0.67	98.9	374016.00	**378350.90**	0.56	168.0
200-10-2b	309586.00	313302.70	1.03	99.8	308316.00	311796.90	0.53	182.5
200-10-3	526244.00	534405.30	0.75	97.4	521229.00	**528412.10**	0.76	179.4
200-10-3b	339142.00	346259.70	0.98	114.1	339321.00	343923.60	0.91	180.2
avg	214012.90	216831.14	0.82	60.4	212864.20	215627.68	0.81	91.9
%-gap $_{avg}$		-3.73				-4.13		
%-gap $_{min}$		-4.88				-5.32		

and deemed acceptable by us. Due to the dependence of V_2's ILP model size also on the depots the runtime increase is more evident when $m = 10$.

Now we will contrast our solution approaches for the PLRP with all previous ones we are aware of: an iterated metaheuristic [179] (IM), a memetic algorithm with population management [182] (MAPM), a evolutionary local search with path relinking [180], and the hybrid evolutionary local search approach [181] (HELS). To be able to also roughly compare the runtimes we determined the speedup of our machine stated at the beginning of this section to their Intel Core Duo with 1.2 GHz. According to a CPU benchmark[1] our machine is 7.25 times faster. Results are shown in Table 6.5. We also give the number of runs performed per instance, if stated in the original work. In addition to the gap of the average and best solutions we need to occasionally state the gaps of single runs here ($\%\text{-gap}_{1\times\text{run}}$) as unfortunately only these were provided. As we feel it is not fair to directly compare average results to results of single runs, a separate column is used for these single run results. Despite this comparison issues our approaches clearly outperform the previous ones, already the preliminary results which require less runtime are better. Yet even the better results of the second test run are still obtained within the same order of runtime.

Speaking of the latter, a further interesting comparison is made with regard to the dependence of runtime on instance size. The results of this are presented in Table 6.6. Here each algorithm's average runtime is normalized by its average runtime on instances of type $n = 50$, $m = 5$, yielding a somewhat better picture as using the smallest instances with $n = 20$, $m = 5$ for it. Looking at the truly moderate increase of relative runtime of our methods compared to the others, they also seem very promising with respect to tackling even larger instances without a substantial increase in runtime. See Figure 6.3 for a more appealing way of presenting this data, note that the vertical axis is logarithmic.

The comparison of the best results obtained over all our runs—also including previous [164] and further preliminary runs—to the so far best known solutions is given in Table 6.7. On all but one of the 30 instances we were able to obtain a (mostly considerable) improvement, which is 5.48% on average.

Finally, in Table 6.8 we give statistics of the shaking neighborhoods for both approaches considering both test runs at once. At first sight most of the values seem nearly unchanged, which is more or less even true for the usage, improvement and acceptance rates. The ability to derive new incumbent solutions is, however, reduced for many of the neighborhoods, especially those operating on the depots, in case of applying the large neighborhood searches. This is because the VLNS itself finds better solutions, partly instead of the VNS neighborhoods.

6.5.2 Results on the LRP

For testing our approaches on the LRP we will proceed basically in the same way as for the PLRP and use the same notation. First results for the VNS and all considered VNS plus VLNS variants on the Prodhon instances with only 10^5 VNS iterations and again optionally

[1] see at `http://www.cpubenchmark.net/`

Table 6.5: Comparison of our approaches to previous methods applied to the PLRP. Originally given average runtimes (t^o[s]) are approximately normalized to our computing environment.

Method	runs	Prodhon instances				
		%-gap avg	%-gap min	%-gap 1×run	t[s] avg	t^o[s] avg
IM	–			13.73	31.2	226.0
MAPM	–			10.72	23.0	166.7
ELS+PR	5	2.71			16.1	116.4
HELS	10		0.16		22.8	165.3
VNS 10^5	10	-0.62	-2.62		6.4	
VNS+ILP 10^5	10	-1.42	-3.34		11.7	
VNS 10^6	10	-3.73	-4.88		60.4	
VNS+ILP 10^6	10	**-4.13**	**-5.32**		91.9	

Table 6.6: Average relative runtimes of PLRP solution approaches on Prodhon instances.

Method	n, m			
	50, 5	100, 5	100, 10	200, 10
IM	1.00	8.01	17.97	145.14
MAPM	1.00	11.26	13.45	250.78
ELS+PR	1.00	5.50	10.15	60.95
HELS	1.00	5.25	9.87	58.98
VNS	1.00	1.63	2.31	3.45
VNS+ILP	1.00	1.74	3.07	4.73

applying V_2 and/or V_3 after every 10^4 iterations (10 times) are given in Table 6.9. Again, V_1 yields the highest improvement, even with nearly no increase in runtime. Regarding the performance of V_2 and V_3 we encounter almost the opposite here as for the PLRP: additionally applying V_2 slightly worsens the solution quality, while V_3 is this time obviously able to realize an intra-route improvement, profiting from the now longer routes which offer more room for improvement. With this setting the runtime is about twice for some instances, by trend rather for the larger instances. To conclude this first results, VNS+$V_{1,3}$ is the method of choice for the LRP, and will in the remainder for simplicity also be denoted as VNS+ILP in the context of the LRP. We also did preliminary tests on the other available LRP instance sets, yet the outcomes were the same, so we omit these results and concentrate on the "final" test runs instead. As a side note, we experienced that solely applying V_2 would often lead to

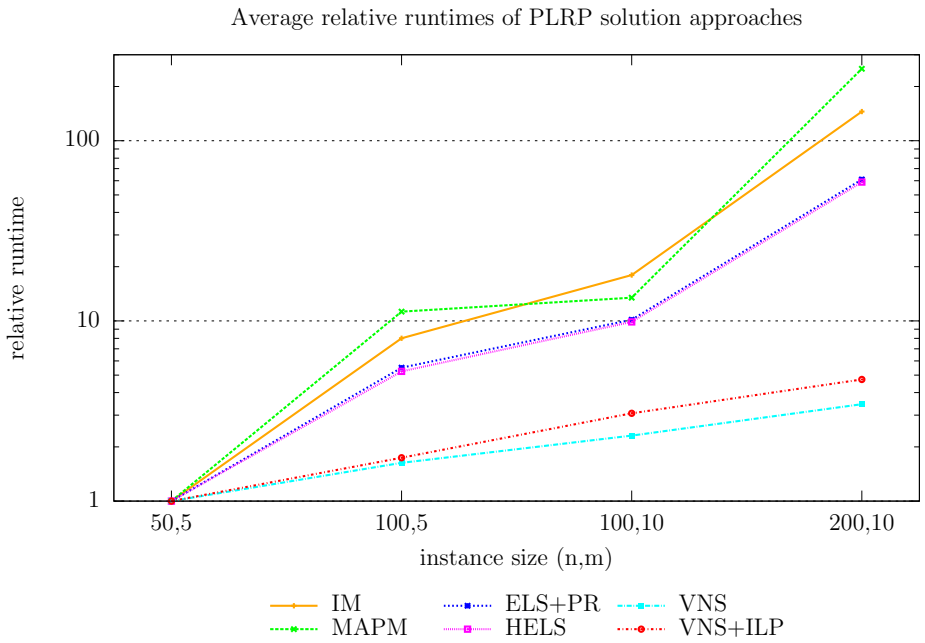


Figure 6.3: Plot of average relative runtimes of PLRP solution approaches on Prodhon instances.

a considerably larger increase in runtime, especially for the LRP (but also for the PLRP). So it seems that V_1 achieves important preliminary work.

The second test runs are consistently performed with $4{\cdot}10^6$ VNS iterations in total with optionally applying V_3 after every $2{\cdot}10^4$ iterations (200 times). For the Prodhon instances the results are shown in Table 6.10. Though VNS+ILP only yields a significant improvement in six out of 30 cases (20%) the overall average gap is more than halved, yet only consuming about 20% more runtime on average as the VNS. Also, the best solution values of the runs are very close to the so far best known solutions (0.02%). Table 6.11 gives the results on the Tuzun and Burke instances, this time showing a significant improvement of VNS-ILP over VNS for six out of 36 instances (16.6%). The results on the Barreto set is shown in Table 6.12, where the enhanced method is only once of 13 times significantly better (6.7%). Anyway, for both instance sets the average as well as the minimal gap were further reduced, so an overall performance gain is clearly noticeable, with an increase in runtime of 30% and 13%, respectively.

As for the PLRP we also compare our methods to previous solution approaches for the LRP, some of them were proposed very recently: a GRASP based approach [176] (GRASP), a memetic algorithm with population management [175] (MAPM), a Lagrangean relaxation-granular tabu search heuristic [177] (LRGTS), a GRASP with evolutionary local search [69] (GRASP+ELS), a simulated annealing heuristic [241] (SALRP), an adaptive large neighborhood search [104] (ALNS) and a GRASP plus ILP-based metaheuristic [35] (GRASP+ILP).

Table 6.7: Comparison of previous best known solutions (BKS) and best results obtained by our algorithms over all runs for the LRP and PLRP instances of Prodhon.

Instance	LRP			PLRP		
	BKS	best	%-gap	BKS	best	%-gap
20-5-1a	54793	54793	0.00	78741	78477	**-0.34**
20-5-1b	39104	39104	0.00	**75554**	76102	0.73
20-5-2a	48908	48908	0.00	80386	77784	**-3.24**
20-5-2b	37542	37542	0.00	62987	62133	**-1.36**
50-5-1a	90111	90111	0.00	156577	145639	**-6.99**
50-5-1b	63242	63242	0.00	142359	134997	**-5.17**
50-5-2a	88298	88298	0.00	143934	138486	**-3.79**
50-5-2b	67308	67308	0.00	117416	110526	**-5.87**
50-5-2aBIS	84055	84055	0.00	177529	168721	**-4.96**
50-5-2bBIS	51822	51822	0.00	102775	100836	**-1.89**
50-5-3a	86203	86203	0.00	157929	152530	**-3.42**
50-5-3b	61830	61830	0.00	114622	108790	**-5.09**
100-5-1a	274814	275079	0.10	362452	335702	**-7.38**
100-5-1b	213615	213568	**-0.02**	234762	223757	**-4.69**
100-5-2a	193671	193671	0.00	281699	257710	**-8.52**
100-5-2b	157095	157095	0.00	168754	162799	**-3.53**
100-5-3a	200079	200079	0.00	233108	214118	**-8.15**
100-5-3b	152441	152441	0.00	175365	166726	**-4.93**
100-10-1a	287695	287692	**-0.001**	278508	255630	**-8.21**
100-10-1b	230989	230989	0.00	220133	207326	**-5.82**
100-10-2a	243590	243590	0.00	280447	255345	**-8.95**
100-10-2b	203988	203988	0.00	176408	166703	**-5.50**
100-10-3a	250882	250882	0.00	272463	253280	**-7.04**
100-10-3b	204317	204567	0.12	204897	188678	**-7.92**
200-10-1a	475294	474850	**-0.09**	464596	431131	**-7.20**
200-10-1b	377043	376509	**-0.14**	398611	359182	**-9.89**
200-10-2a	449115	448421	**-0.15**	403962	374016	**-7.41**
200-10-2b	374280	373885	**-0.11**	324121	308316	**-4.88**
200-10-3a	469433	469471	0.01	567336	521229	**-8.13**
200-10-3b	362653	362645	**-0.002**	356618	339142	**-4.90**
avg			**-0.01**			**-5.48**

The comparative results are given in Table 6.13 for each of the considered instance sets, adhering to the same notation as before. Unfortunately quite often the methods' performance is only documented by the results of single runs, again not allowing a really fair comparison. The most recent methods clearly yield the best performance, but also devote more computing effort than previous approaches (not even taking the outdated hardware into account). The most comprehensive comparison can be made to ALNS and GRASP+ILP, where our methods, especially VNS+ILP, show clearly a similar behavior with respect to solution quality. Due to this we additionally state in Table 6.14 average results over all instance sets, i.e., considered as one large set, since just averaging over the already averaged gaps would neglect the number of instances. We see that VNS performs on average as ALNS, and VNS+ILP as GRASP+ILP (actually differing at the third digit and only after rounding at the second), especially noting for VNS+ILP the very small value of 0.04 of %-gap_{min}. So much for the solution quality, but it is also interesting to take the absolute runtime into account for achieving it, which is often easier said than done. ALNS was run on a 2.2,GHz AMD Opteron 275

Table 6.8: Relative usage, improvement of x_{vns}, acceptance of worse solutions and improvement of x_{best} in percent of shaking neighborhoods averaged over all 30 PLRP instances.

k	VNS				VNS+ILP			
	%-usage	%-impr.	%-accept	%-best	%-usage	%-impr.	%-accept	%-best
1	7.90	24.85	21.57	22.20	7.89	24.78	21.50	23.90
2	7.20	18.90	16.72	17.63	7.19	18.84	16.68	18.90
3	6.67	13.80	12.39	14.47	6.66	13.75	12.34	14.64
4	6.27	10.19	9.24	10.73	6.27	10.19	9.20	11.32
5	5.98	7.78	7.12	8.80	5.98	7.76	7.10	9.03
6	5.76	6.08	5.62	7.22	5.76	6.06	5.61	7.28
7	5.58	7.59	11.13	6.84	5.59	7.70	11.25	6.08
8	5.30	5.40	8.05	4.53	5.31	5.47	8.08	4.19
9	5.10	4.50	6.85	3.97	5.10	4.54	6.91	3.43
10	4.93	0.17	0.31	0.26	4.94	0.16	0.31	0.23
11	4.93	0.14	0.25	0.36	4.93	0.13	0.24	0.27
12	4.92	0.14	0.25	0.41	4.92	0.13	0.24	0.30
13	4.91	0.16	0.16	0.69	4.92	0.17	0.17	0.15
14	4.91	0.07	0.08	0.39	4.91	0.07	0.08	0.05
15	4.91	0.04	0.05	0.18	4.91	0.04	0.05	0.05
16	4.91	0.09	0.08	0.61	4.91	0.08	0.08	0.08
17	4.90	0.06	0.08	0.45	4.91	0.08	0.09	0.04
18	4.90	0.05	0.06	0.29	4.90	0.06	0.07	0.05

and GRASP+ILP on an Intel Xeon E5472 with 3.0,GHz. The former is definitely slower than ours, so ALNS would probably take the same or even less time if run on our machine, whereas the latter is even a slightly faster but comparable machine. Ultimately considering both runtime and solution quality our methods currently achieve the best performance.

Additionally we will show the relative increase of runtime dependent on instance size in the same way as before, again using the Prodhon instance set, which provides instances of several different sizes; see Table 6.15 and Figure 6.4. As expected VNS and VNS+ILP behave the same as for the PLRP, and fortunately yield again the least increase of all methods. Those obtaining the most similar performances are ALNS and SALRP, which is not surprising since they also bear the most resemblance from a methodical point of view. Contrary, especially the GRASP based methods show quite a huge increase and hence seem less suited for potentially larger instances.

Looking at the best results obtained by our methods for the Prodhon (see Table 6.7), Tuzun and Burke, as well as Barreto instance sets (both in Table 6.16) again highlights the satisfying performance they achieve. Despite the fierce competition, for 20 of the 79 instances a new best known solution is found, 45 times the best known solution could be reproduced and only in 14 cases a slightly worse solution was achieved.

Table 6.9: Results of VNS and VNS plus VLNS combinations on the Prodhon LRP instances using 10^5 iterations.

Instance	VNS			VNS+V_1			VNS+$V_{1,2}$			VNS+$V_{1,3}$			VNS+$V_{1,2,3}$		
	avg	CV [%]	t [s]	avg	CV [%]	t [s]	avg	CV [%]	t [s]	avg	CV [%]	t [s]	avg	CV [%]	t [s]
20-5-1	55054.20	1.08	1.2	54903.90	0.28	1.2	54861.40	0.20	1.3	54815.80	0.13	1.3	54881.10	0.28	1.4
20-5-1b	39104.00	0.00	1.3	39104.00	0.00	1.2	39104.00	0.00	1.3	39104.00	0.00	1.4	39104.00	0.00	1.5
20-5-2	48985.70	0.50	1.1	48908.00	0.00	1.2	48908.00	0.00	1.2	48908.00	0.00	1.3	48908.00	0.00	1.4
20-5-2b	37542.00	0.00	1.3	37542.00	0.00	1.3	37542.00	0.00	1.4	37542.00	0.00	1.4	37542.00	0.00	1.5
50-5-1	90111.00	0.00	1.6	90751.60	1.22	1.6	90343.80	0.81	1.9	90892.60	1.82	2.1	90140.10	0.10	2.2
50-5-1b	64220.20	2.68	2.4	64019.20	0.99	2.4	65077.90	3.63	2.5	63464.60	0.58	3.0	63509.40	0.56	3.1
50-5-2	90685.40	0.71	1.7	89989.20	1.36	1.7	89707.00	0.96	1.9	90129.90	1.43	2.1	89982.80	1.28	2.5
50-5-2b	68775.20	1.24	3.7	68041.50	0.53	3.3	67887.30	0.19	3.4	68103.20	0.98	3.8	68449.90	1.35	4.0
50-5-2BIS	84407.00	0.37	2.2	84316.70	0.19	2.3	84174.00	0.16	2.4	84163.60	0.15	2.7	84199.20	0.09	2.9
50-5-2bBIS	52219.80	0.53	3.9	51977.50	0.36	4.1	52149.70	0.39	4.2	51991.90	0.22	4.6	51896.60	0.18	4.9
50-5-3	87190.70	1.02	1.3	86851.10	0.59	1.5	86615.70	0.27	1.8	87045.10	0.72	1.8	86713.50	0.53	2.1
50-5-3b	62105.90	0.81	2.2	62133.70	0.96	2.3	61969.50	0.42	2.6	61961.40	0.52	3.0	62062.30	0.71	3.1
100-5-1	279094.20	0.40	4.1	278530.50	0.25	4.1	278201.00	0.29	4.6	277644.70	0.22	5.6	277663.00	0.29	5.8
100-5-1b	217476.90	0.56	6.7	216280.80	0.54	6.7	217378.70	0.62	7.0	215680.30	0.35	7.9	216246.40	0.43	8.5
100-5-2	195832.60	0.60	3.3	195566.50	0.28	3.4	195495.30	0.31	3.9	194859.90	0.37	4.9	195219.00	0.46	5.1
100-5-2b	158535.70	0.92	3.4	158247.80	0.41	3.5	158405.10	0.60	3.8	157462.70	0.12	5.0	157784.60	0.30	5.3
100-5-3	202811.70	0.52	2.7	201540.70	0.47	2.9	201604.50	0.20	3.7	201645.20	0.42	4.3	201819.60	0.25	4.9
100-5-3b	155785.70	0.64	3.2	155183.60	0.80	3.2	155888.30	0.87	3.5	154652.90	0.70	4.7	154414.40	0.39	5.1
100-10-1	314374.80	2.43	4.4	293069.20	0.56	6.6	292108.40	0.55	9.2	292023.30	0.59	10.4	292506.00	0.66	15.0
100-10-1b	260432.90	5.83	6.5	239751.00	1.02	6.4	239743.90	1.38	7.1	236627.10	0.47	9.3	237965.40	0.46	10.9
100-10-2	246980.20	0.34	3.8	245817.60	0.38	4.5	245976.70	0.51	7.5	245550.60	0.40	6.0	245577.90	0.52	8.5
100-10-2b	206721.10	0.66	5.4	205822.80	0.46	5.6	206117.40	0.55	6.3	205575.90	0.36	7.1	205939.80	0.65	8.0
100-10-3	257457.40	0.46	3.9	254869.80	0.55	4.6	254803.70	0.24	11.6	255658.70	0.47	6.4	255146.90	0.54	11.1
100-10-3b	209678.10	0.99	5.1	207029.00	0.37	5.3	207159.20	0.28	6.3	206363.50	0.50	7.0	207546.80	0.72	9.1
200-10-1	487561.30	0.95	12.0	481822.70	0.20	13.4	481776.90	0.31	16.8	480800.90	0.19	17.7	480737.00	0.21	21.3
200-10-1b	386651.50	0.84	8.9	383746.40	0.91	9.9	382662.00	0.33	12.4	379859.20	0.34	16.5	381209.30	0.75	18.1
200-10-2	457560.30	0.38	11.6	453159.40	0.28	12.6	452449.40	0.21	15.1	451533.60	0.16	16.9	451753.80	0.27	18.6
200-10-2b	383981.20	0.96	8.7	377852.80	0.26	9.4	378544.90	0.32	11.2	376051.20	0.14	17.1	376414.10	0.33	17.7
200-10-3	478663.50	0.57	11.5	475215.40	0.34	13.3	475144.30	0.23	18.4	475435.20	0.13	17.5	474901.80	0.28	21.3
200-10-3b	372324.50	0.54	8.8	366853.90	0.23	9.1	366263.80	0.23	10.9	365284.90	0.20	16.7	365164.20	0.14	17.4
avg.	201744.16	0.92	4.6	198963.28	0.49	5.0	198935.46	0.50	6.2	198361.06	0.42	7.0	198513.30	0.42	8.1
%-gap $_{avg}$	2.10			1.05			1.08			**0.82**			0.87		
%-gap $_{min}$	0.90			0.49			0.47			**0.32**			0.40		

Finally, we will also inspect the VNS neighborhoods statistics, given in Table 6.17. Apparently also for the LRP only very little changes occur when additionally applying the ILP-based VLNS. Perhaps most noticeable is again the reduction of %-best of the neighborhoods dealing with the depots, most likely due to V_1, yet this time V_3 also has a slight influence on the remaining route segment-based neighborhoods.

6.6 Conclusions

We presented a variable neighborhood search (VNS) for the periodic location-routing problem (PLRP) with capacitated vehicles and depots, which is also directly applicable to the LRP, i.e. the special case when having only a single day planning horizon. For the core part, the shaking, we apply neighborhood structures exchanging route segments between routes of the same depot and between routes of two depots, changing the status (opened/closed) of two

Table 6.10: More detailed results of VNS and best performing VNS plus VLNS combination on the Prodhon LRP instances using $4{\cdot}10^6$ iterations, significant improvements are marked bold.

Instance	VNS				VNS+ILP			
	min	avg	CV [%]	t [s]	min	avg	CV [%]	t [s]
20-5-1	54793.00	54793.00	0.00	47.7	54793.00	54793.00	0.00	48.1
20-5-1b	39104.00	39104.00	0.00	49.7	39104.00	39104.00	0.00	54.4
20-5-2	48908.00	48908.00	0.00	47.5	48908.00	48908.00	0.00	47.7
20-5-2b	37542.00	37542.00	0.00	52.0	37542.00	37542.00	0.00	55.0
50-5-1	90111.00	90140.10	0.10	62.8	90111.00	90111.00	0.00	72.4
50-5-1b	63242.00	63254.60	0.04	92.6	63242.00	63253.20	0.04	105.9
50-5-2	88298.00	88712.40	0.81	64.8	88298.00	88640.90	0.73	74.8
50-5-2b	67449.00	67913.70	0.39	141.4	67308.00	**67408.80**	0.24	147.9
50-5-2BIS	84055.00	84055.00	0.00	89.6	84055.00	84055.00	0.00	95.0
50-5-2bBIS	51822.00	51830.70	0.03	155.9	51822.00	51842.90	0.05	168.9
50-5-3	86203.00	86349.50	0.18	53.1	86203.00	86370.80	0.17	62.4
50-5-3b	61830.00	61830.00	0.00	90.7	61830.00	61830.00	0.00	101.9
100-5-1	275079.00	275689.00	0.15	137.5	275441.00	276091.50	0.25	165.4
100-5-1b	213671.00	214364.30	0.20	241.4	213671.00	214349.40	0.23	273.6
100-5-2	193671.00	193839.80	0.09	101.4	193671.00	193750.70	0.06	122.8
100-5-2b	157129.00	157163.70	0.01	120.4	157129.00	157201.30	0.08	150.5
100-5-3	200114.00	200480.30	0.16	95.5	200202.00	200565.00	0.18	116.8
100-5-3b	152441.00	153240.30	0.38	119.6	152441.00	153165.70	0.41	145.3
100-10-1	288480.00	310418.70	3.39	162.6	287692.00	**289365.80**	0.41	182.2
100-10-1b	232041.00	234710.90	0.64	181.3	230989.00	233564.10	0.53	216.8
100-10-2	243590.00	244105.20	0.23	131.6	243590.00	244422.80	0.29	159.2
100-10-2b	204217.00	204992.00	0.24	200.0	204178.00	**204560.00**	0.21	235.6
100-10-3	252670.00	253823.40	0.56	139.8	250971.00	**252872.10**	0.40	162.9
100-10-3b	204631.00	205077.30	0.25	194.1	204567.00	204799.60	0.13	228.6
200-10-1	476402.00	477690.80	0.22	422.8	475165.00	477088.30	0.20	486.6
200-10-1b	377011.00	377699.80	0.16	327.5	376876.00	377752.10	0.20	435.3
200-10-2	449173.00	450345.40	0.25	413.4	448985.00	449481.50	0.05	523.4
200-10-2b	374070.00	374927.10	0.14	316.9	374411.00	**374691.10**	0.10	458.9
200-10-3	470627.00	471986.40	0.23	383.5	469471.00	471558.90	0.23	456.4
200-10-3b	364214.00	365689.30	0.26	314.7	363129.00	**363653.40**	0.13	424.2
avg	196752.90	198022.56	0.30	165.1	196526.50	197093.10	0.18	199.3
%-gap $_{avg}$		0.58				0.24		
%-gap $_{min}$		0.10				0.02		

Table 6.11: Results of VNS and best performing VNS plus VLNS combination on the Tuzun and Burke LRP instances using $4{\cdot}10^6$ iterations, significant improvements are marked bold.

Instance	VNS				VNS+ILP			
	min	avg	CV [%]	t [s]	min	avg	CV [%]	t [s]
111112	1469.43	1481.18	0.56	151.8	1467.68	1479.60	0.43	184.7
111122	1449.20	1458.12	0.69	179.6	1449.20	1452.53	0.36	210.6
111212	1396.59	1403.96	0.63	145.3	1394.80	**1399.55**	0.44	176.7
111222	1432.29	1441.91	0.69	172.0	1432.29	1435.00	0.53	197.4
112112	1167.16	1186.61	0.88	152.4	1167.16	1185.10	0.98	184.5
112122	1102.24	1104.92	0.40	203.8	1102.24	1103.33	0.20	221.2
112212	793.09	795.28	0.27	151.2	791.74	794.20	0.22	178.6
112222	728.40	729.33	0.12	183.9	728.30	**728.72**	0.13	208.8
113112	1238.49	1247.29	0.88	188.9	1239.22	1253.72	0.74	208.2
113122	1247.68	1249.56	0.21	199.2	1247.27	1249.70	0.19	226.8
113212	902.26	913.85	2.63	173.2	902.26	902.65	0.07	226.3
113222	1018.29	1031.84	0.98	446.5	1018.29	1025.51	0.67	467.3
131112	1922.27	1927.74	0.24	247.1	1898.39	1921.31	0.62	305.3
131122	1825.00	1854.30	0.97	254.1	1820.94	**1830.15**	0.53	340.2
131212	1969.92	1982.61	0.49	217.4	1966.07	1989.04	0.71	285.3
131222	1804.33	1817.18	0.53	249.8	1798.13	1819.94	0.85	308.7
132112	1445.13	1461.12	1.14	168.8	1443.32	1455.24	1.40	231.9
132122	1433.82	1443.65	0.46	213.5	1431.43	1447.77	0.75	270.4
132212	1204.86	1205.45	0.03	203.3	1204.42	1206.06	0.26	265.4
132222	931.58	932.03	0.04	234.0	927.67	**930.81**	0.17	303.0
133112	1722.37	1726.51	0.24	245.8	1718.55	1723.94	0.30	314.2
133122	1393.55	1409.32	0.76	251.4	1392.69	1406.32	0.68	313.9
133212	1199.02	1207.18	0.53	187.8	1198.31	1208.13	0.58	238.4
133222	1152.81	1156.99	0.19	316.0	1151.83	**1155.59**	0.16	388.4
121112	2250.83	2270.21	0.64	235.4	2251.56	2275.44	0.65	364.9
121122	2145.17	2164.49	0.49	369.5	2144.91	2163.56	0.52	484.7
121212	2206.39	2227.53	0.45	327.8	2227.23	2237.17	0.40	532.5
121222	2240.58	2259.81	0.42	291.2	2229.27	2253.22	0.66	448.5
122112	2086.57	2097.67	0.30	206.3	2090.34	2100.05	0.35	325.5
122122	1695.94	1713.40	0.78	276.5	1685.91	1705.20	0.75	394.1
122212	1460.99	1469.24	0.43	209.0	1453.78	1465.67	0.64	322.9
122222	1083.02	1086.32	0.17	264.5	1082.85	1085.95	0.15	377.5
123112	1959.52	1968.71	0.25	325.6	1951.02	1973.91	0.85	438.8
123122	1925.86	1947.06	0.47	355.1	1920.99	1937.94	0.70	470.7
123212	1767.83	1770.65	0.30	202.4	1763.98	1788.05	1.03	338.9
123222	1391.71	1393.70	0.09	443.4	1391.45	**1392.74**	0.06	570.4
avg	1504.56	1514.91	0.54	240.1	1502.37	1513.41	0.52	314.6
%-gap $_{avg}$		0.87				0.75		
%-gap $_{min}$		0.21				0.07		

Table 6.12: Results of VNS and best performing VNS plus VLNS combination on the Barreto LRP instances using $4{\cdot}10^6$ iterations, significant improvements are marked bold.

Instance	VNS				VNS+ILP			
	min	avg	CV [%]	t [s]	min	avg	CV [%]	t [s]
Christ.69-50x5	570.50	576.11	1.26	122.2	565.60	573.24	1.41	133.5
Christ.69-75x10	848.85	855.23	0.66	188.0	848.85	853.25	0.38	201.9
Christ.69-100x10	833.43	835.03	0.35	196.0	833.43	835.83	0.41	221.8
Daskin95-88x8	355.78	356.96	1.00	276.6	355.78	355.78	0.00	312.6
Daskin95-150x10	44167.18	44398.07	0.59	358.9	43919.90	**44126.12**	0.40	453.2
Gaskell67-21x5	424.90	424.90	0.00	49.1	424.90	424.90	0.00	51.9
Gaskell67-22x5	585.11	585.11	0.00	66.9	585.11	585.11	0.00	72.9
Gaskell67-29x5	512.10	512.10	0.00	59.2	512.10	512.10	0.00	64.7
Gaskell67-32x5	562.22	562.22	0.00	68.5	562.22	562.22	0.00	72.1
Gaskell67-32x5b	504.33	504.33	0.00	101.9	504.33	504.33	0.00	104.6
Gaskell67-36x5	460.37	460.37	0.00	73.1	460.37	460.37	0.00	77.8
Min92-27x5	3062.02	3062.02	0.00	66.6	3062.02	3062.02	0.00	73.3
Min92-134x8	5720.09	5746.90	0.75	320.0	5711.49	5769.73	1.17	368.6
avg	4508.22	4529.18	0.35	149.8	4488.16	4509.62	0.29	169.9
%-gap $_{\text{avg}}$		0.38				0.28		
%-gap $_{\text{min}}$		0.13				0.003		

depots, such that one is closed and another is opened, and finally also changing the status of a single depot. For the PLRP also the selected visit combinations can be changed during shaking. The VNS is subsequently combined with integer linear programming-based very large neighborhood searches (VLNS). Two of them operate on a higher level via relocating whole routes to depots, considering all days at once, with one being a more sophisticated version using a set covering model. The third one deals with the location of customer sequences to insertion points in routes of a single day. Two of the VLNS are further designed to exploit the information contained in several solutions provided by the VNS. Experimental results on available benchmark test data show the excellent performance of our methods especially on the PLRP but also on the LRP (where quite a competition exists) when compared to corresponding leading approaches, both in terms of solution quality, the absolute computing time needed for achieving it as well regarding the dependence of runtime on instance size. For the latter our methods show the least increase and seem very promising to tackle even larger instances. The results further indicate almost always a gain in solution quality when applying the proper combination of VLNS, sometimes yielding significantly better results as applying VNS alone. Mostly at the expense of only a moderate increase in runtime, especially when considering the longer (final) runs.

Table 6.13: Comparison to previous methods applied to the LRP.

Method	runs	Prodhon instances				Tuzun and Burke instances				Barreto instances			
		%-gap avg	%-gap min	%-gap 1×run	t[s] avg	%-gap avg	%-gap min	%-gap 1×run	t[s] avg	%-gap avg	%-gap min	%-gap 1×run	t[s] avg
GRASP	–			3.65	96.5			3.42	159.6			1.59	21.1
MAPM	–			1.43	76.7			1.78	203.1			2.02	37.8
LRGTS	–			0.78	17.5			1.76	21.2			1.62	18.2
GRASP+ELS	5		1.11		258.2		1.22		606.6		0.04		187.6
SALRP	–			0.46	422.4			1.41	826.4			0.25	140.5
ALNS	5	0.74	0.44		451.0	0.81	0.36		830.0	**0.21**	0.12		174.7
GRASP+ILP	10	0.26	0.05		1129.2	**0.59**	0.27		2255.0	0.64	0.15		255.0
VNS	10	0.58	0.10		165.1	0.87	0.21		240.1	0.38	0.13		149.8
VNS+ILP	10	**0.24**	**0.02**		199.3	0.75	**0.07**		314.6	0.28	**0.003**		169.9

Table 6.14: Most recent and also best performing methods applied to the LRP.

Method	runs	all three instance sets		
		%-gap avg	%-gap min	t[s] avg
ALNS	5	0.68	0.35	578.2
GRASP+ILP	10	0.47	0.17	1498.4
VNS	10	0.68	0.16	196.8
VNS+ILP	10	0.48	0.04	247.0

Table 6.15: Average relative runtimes of LRP solution approaches on Prodhon instances.

Method	n, m			
	50, 5	100, 5	100, 10	200, 10
GRASP	1.00	10.87	17.33	207.79
MAPM	1.00	9.10	8.46	87.88
LRGTS	1.00	5.11	16.00	73.11
GRASP+ELS	1.00	29.76	32.52	188.26
SALRP	1.00	3.88	3.60	21.25
ALNS	1.00	6.12	1.79	10.33
GRASP+ILP	1.00	7.18	78.40	236.63
VNS	1.00	1.45	1.79	3.87
VNS+ILP	1.00	1.57	1.91	4.48

Potential Future Work

It would be possible to incorporate the periodic aspect of the PLRP in a suitably enhanced version of the third VLNS, which might increase its performance and make it a viable candidate for the PLRP again. Especially in the light of the latter addition it would probably make sense to also generate more diverse PLRP instances to allow for a broader performance evaluation and also a potential better comparison to other approaches in the future.

Due to the very satisfying results also on the LRP we deemed it a natural (and promising) step to adapt/extend our solution approach to the richer two-echelon variant of the problem (2E-LRP) [24]. In the meantime we realized a VNS for this problem and began to experiment also with appropriate variants of the ILP neighborhoods. In [208] we report on the successful VNS approach, showing that it is highly competitive to the leading approaches, obtaining several new best solutions.

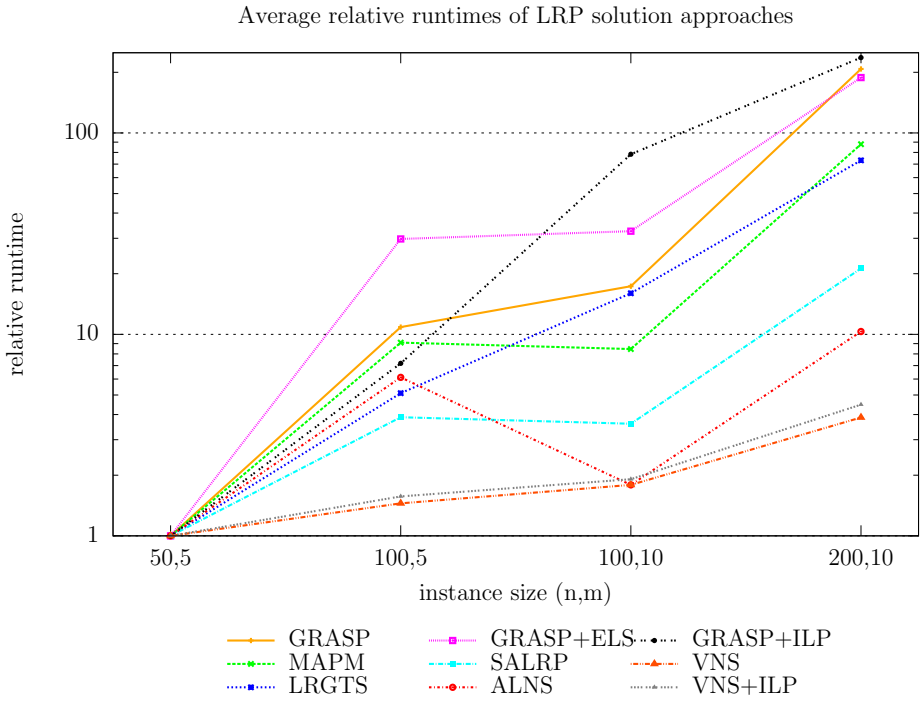


Figure 6.4: Plot of average relative runtimes of LRP solution approaches on Prodhon instances.

Table 6.16: Comparison of previous best known solutions (BKS) and best results obtained by our algorithms over all runs for the LRP instances of Tuzun and Burke as well as Barreto.

Tuzun and Burke				Barreto			
Instance	BKS	best	%-gap	Instance	BKS	best	%-gap
111112	1467.68	1467.68	0.00	Christ.69-50x5	565.60	565.6	0.00
111122	1449.20	1449.20	0.00	Christ.69-75x10	848.85	848.85	0.00
111212	1394.80	1394.80	0.00	Christ.69-100x10	833.40	833.43	0.00
111222	1432.29	1432.29	0.00	Daskin95-88x8	355.78	355.78	0.00
112112	1167.16	1167.16	0.00	Daskin95-150x10	43919.90	43919.90	0.00
112122	1102.24	1102.24	0.00	Gaskell67-21x5	424.90	424.90	0.00
112212	791.66	791.74	0.01	Gaskell67-22x5	585.11	585.11	0.00
112222	728.30	728.30	0.00	Gaskell67-29x5	512.10	512.10	0.00
113112	1238.49	1238.49	0.00	Gaskell67-32x5	562.22	562.22	0.00
113122	1245.31	1247.18	0.15	Gaskell67-32x5b	504.33	504.33	0.00
113212	902.26	902.26	0.00	Gaskell67-36x5	460.37	460.37	0.00
113222	1018.29	1018.29	0.00	Min92-27x5	3062.02	3062.02	0.00
131112	1866.75	1898.39	1.69	Min92-134x8	5709.00	5711.49	0.04
131122	1823.53	1820.32	**-0.18**				
131212	1965.12	1966.07	0.05				
131222	1796.45	1792.77	**-0.20**				
132112	1443.33	1443.32	**-0.001**				
132122	1434.63	1431.43	**-0.22**				
132212	1204.42	1204.42	0.00				
132222	930.99	925.14	**-0.63**				
133112	1694.18	1694.22	0.001				
133122	1392.01	1392.69	0.05				
133212	1198.28	1198.31	0.00				
133222	1151.80	1151.80	0.00				
121112	2251.93	2247.07	**-0.22**				
121122	2159.93	2139.64	**-0.94**				
121212	2220.01	2206.39	**-0.61**				
121222	2230.94	2224.17	**-0.30**				
122112	2073.73	2086.57	0.62				
122122	1692.17	1685.65	**-0.39**				
122212	1453.18	1453.78	0.04				
122222	1082.74	1082.46	**-0.03**				
123112	1960.30	1947.62	**-0.65**				
123122	1918.93	1920.99	0.11				
123212	1762.03	1762.54	0.03				
123222	1391.68	1390.99	**-0.05**				
avg			**-0.05**				0.003

Table 6.17: Relative usage, improvement of x_{vns}, acceptance of worse solutions and improvement of x_{best} in percent of shaking neighborhoods averaged over all 79 LRP instances.

k	VNS				VNS+ILP			
	%-usage	%-impr.	%-accept	%-best	%-usage	%-impr.	%-accept	%-best
1	7.01	35.77	40.64	26.99	6.84	35.10	39.79	27.67
2	6.34	20.04	19.85	16.15	6.26	19.77	19.65	16.64
3	5.99	13.64	12.62	12.66	5.95	13.61	12.64	13.04
4	5.76	10.20	9.21	10.57	5.75	10.29	9.32	10.71
5	5.59	8.13	7.35	9.48	5.60	8.29	7.51	9.76
6	5.46	6.24	5.79	8.23	5.47	6.41	5.94	8.50
7	5.35	0.39	0.42	1.71	5.38	0.38	0.42	1.66
8	5.35	0.26	0.26	1.04	5.37	0.25	0.26	1.14
9	5.34	0.19	0.19	0.93	5.37	0.19	0.19	1.08
10	5.34	0.16	0.16	0.81	5.36	0.16	0.16	0.78
11	5.34	0.15	0.15	0.80	5.36	0.15	0.15	0.74
12	5.33	0.13	0.13	0.69	5.36	0.13	0.14	0.63
13	5.33	1.14	1.22	2.09	5.36	1.33	1.44	1.34
14	5.31	0.77	0.66	1.11	5.34	0.90	0.78	0.70
15	5.30	0.60	0.47	0.71	5.32	0.69	0.57	0.52
16	5.29	0.87	0.12	2.96	5.31	0.90	0.15	2.44
17	5.28	0.69	0.31	1.76	5.30	0.74	0.37	1.47
18	5.27	0.63	0.43	1.32	5.30	0.70	0.51	1.19

CHAPTER 7

VEHICLE ROUTING PROBLEM WITH COMPARTMENTS

7.1 Introduction

In this chapter we investigate another special vehicle routing problem, combining the classical *(capacitated) vehicle routing problem* ((C)VRP) and a packing subproblem caused by having more than one compartment in which the customers' orders have to be placed considering potential incompatibilities. We adhere to the definition of a rather general variant given by Derigs et al. [60] and hence also denote it as the *vehicle routing problem with compartments* (VRPC).

As usual some words about the history: The VRPC was introduced to us by Jens Gottlieb in the event of the *7th International Workshop on Hybrid Metaheuristics* in October 2010, which was organized by us and took place in Vienna. Having gained experience for other special routing problems in the past we decided to tackle also this problem, thereby trying to focus on new aspects of it.

Part of this work was presented at the *13th International Conference on Computer Aided Systems Theory* in 2011 (Eurocast 2011) [165], as well as at the *9th Metaheuristic International Conference* in 2011 (MIC 2011) [166]. A post-conference proceedings article of the Eurocast 2011 is also available [167].

7.1.1 Problem Description

Besides having vehicles of limited capacity and the goal to minimize the total travel costs, which resembles a classical VRP, the specialties of the VRPC are:

- vehicles have more than one compartment,
- more than one product is delivered,
- the customer demand per product might be satisfied via several smaller orders,
- all goods delivered on a tour must be assigned to compartments (in a feasible way),
- there might be incompatible products and compartments, or incompatibilities between products.

7.1.2 Considered Scenarios

Similarly to [60] we will primarily consider the cases of having compartments which are flexible in size/capacity but bounded by the total vehicle capacity, together with products that are only compatible with specific compartments, as well as fixed compartment capacities and product groups that might not be placed together in the same compartment.

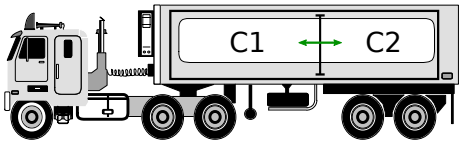


Figure 7.1: Truck of food scenario with a movable dividing wall between the compartments.

The first setting occurs in practice for food retail when delivering frozen and dry goods, or when collecting harmful substances (depicted in Figure 7.1), whereas the second—and from a computational point of view more challenging and thus interesting—setting occurs when distributing petrol involving different fuel types (depicted in Figure 7.2). In fact, in case of the latter setting, the packing subproblem is $\mathcal{NP}$-hard, which will be discussed later.

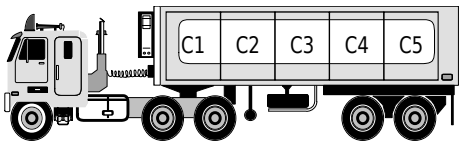


Figure 7.2: Truck of petrol scenario with compartments of fixed size.

A further scenario involves two compartments of fixed size and two types of products, each one dedicated to a single compartment type in advance. Due to the pre-selected compartment per customer order there evidently occurs no packing subproblem, hence we basically face two superposed classical VRPs in this case. A real-life scenario of distributing cattle food to farms, probably involving several such compartment-product pairs, is mentioned in [72].

7.1.3 Outline

Related work is presented in the next section, followed by an examination of the packing subproblem in Section 7.3. A variable neighborhood search and an adaptive large neighborhood search is introduced in Section 7.4 and 7.5, respectively. Experimental results are shown and discussed in Section 7.6, giving conclusions in Section 7.7.

7.2 Related Work

Vehicle routing problems involving several compartments have been tackled in the literature only very recently. Note that in previous work it is mostly denoted as the *multi-compartment vehicle routing problem* (MC-VRP). El Fallahi et al. [72] propose a memetic algorithm and a tabu search for a cattle food delivery scenario. Muyldermans and Pang [147] consider the (co-)collection of waste and present solution approaches based on local search as well as on guided local search. Mendoza et al. [136, 137] tackled a problem variant with stochastic demands via several construction heuristics and a memetic algorithm. All these works considered the previously mentioned simpler scenario, comprising two compartments with fixed capacities and two product groups, each being compatible with only one compartment. As already mentioned, in Derigs et al. [60] a more general variant is proposed. Instead of a single solution algorithm they derive a metaheuristic framework and solve the problem with several instances of it, aiming at identifying a good setting.

7.3 The VRPC Packing Subproblem: The Compartment Assignment Problem

We will start with the core of the VRPC, essentially differentiating it from the classical VRP: the packing (sub-)problem, in the following denoted as the *compartment assignment problem* (CAP). When solving the VRPC we are either given an empty route or one with some orders already assigned to compartments, and a set of new orders to insert. The question is then if we can find a feasible (re-)assignment of the new (or all) orders. Naturally we are not only interested whether such a (re-)assignment exists but also in the concrete assignment of the orders. The problem resembles a decision problem since any feasible solution will satisfy our needs, no matter what the actual packing looks like, although we will later favor some assignments over others when it comes to heuristically solving the problem. Since we are filling one-dimensional entities the CAP is similar to the bin packing problem, yet different in some aspects as in the classical bin packing problem there is usually an unlimited number of bins and there are no restrictions (incompatibilities) regarding the placing of items in the bins. In Table 7.1 we give several settings of compartments and products which can possibly occur together with the resulting packing problem. The latter ranges from a simple capacity check over an $\mathcal{NP}$-hard bin packing-like problem to a problem related to the $\mathcal{NP}$-hard bin packing problem with conflicts [90].

Table 7.1: Compartment/product settings and resulting packing problem.

Setting	**Required feasibility check or resulting packing problem**
only overall capacity limited, compartment partition entirely flexible; covered by instances of type food	capacity check on vehicle level is sufficient
product types pre-assigned to compartment(s) and one compartment per type; covered by instances of type food as well as Christofides and Eilon based instances (see Section 7.6.1)	capacity check on compartment level is sufficient
product types pre-assigned to compartment(s) and multiple fixed compartments per type	$\mathcal{NP}$-hard bin packing-like problem
incompatibilities between products; instances of type petrol	$\mathcal{NP}$-hard bin packing-like problem
incompatibilities between orders	similar to related $\mathcal{NP}$-hard bin packing problem with conflicts [90]

7.3.1 Straightforward ILP Formulation

We give a straightforward (naive) integer linear programming (ILP) model of the previously termed "$\mathcal{NP}$-hard bin packing-like problem"; a similar variant is present in the ILP formulation of the whole VRPC given in [60]. In the following the set of all orders is denoted by O, the set of all product types by P, the set of all compartments by C, the capacities of compartments $c \in C$ by Q_c, the demands of orders $o \in O$ by d_o, all orders of product type $p \in P$ by $ordProd(p)$, the incompatibilities between products and compartments by Inc_{PC}, given as set of pairs (p, c) with $p \in P$ and $c \in C$, as well as the incompatibilities between different products by Inc_{PP}, given as set of pairs (p, q) with $p, q \in P$. We assume a subset $O' \subseteq O$ of all orders is considered for packing in currently empty compartments:

$$\max \quad \sum_{o \in O'} \sum_{c \in C} x_{oc} \tag{7.1}$$

$$\text{s.t.} \quad \sum_{c \in C} x_{oc} = 1 \qquad \forall\, o \in O' \tag{7.2}$$

$$\sum_{o \in O'} d_o\, x_{oc} \leq Q_c \qquad \forall\, c \in C \tag{7.3}$$

$$\sum_{o \in O' : o \in ordProd(p)} x_{oc} - M_p\, y_{cp} \leq 0 \qquad \forall\, p \in P;\, c \in C \tag{7.4}$$

$$y_{cp} = 0 \qquad \forall\,(p,c) \in Inc_{PC} \tag{7.5}$$
$$y_{cp} + y_{cq} \leq 1 \qquad \forall\,(p,q) \in Inc_{PP} \tag{7.6}$$
$$x_{oc} \in \{0,1\} \qquad \forall\, o \in O';\, c \in C \tag{7.7}$$
$$y_{cp} \in \{0,1\} \qquad \forall\, p \in P;\, c \in C \tag{7.8}$$

Binary variables x_{oc} and y_{cp} denote whether order o is placed in compartment c and whether an order of product type p is placed in compartment c, respectively. Equalities (7.2) ensure that each order $o \in O'$ is placed in exactly one of the compartments. The capacity restrictions per compartment are ensured by inequalities (7.3). Next, inequalities (7.4) link the x_{oc} and y_{cp} variables by forcing y_{cp} to one as soon as an order of product type p is placed in compartment c. Here the "big-M"-like constants M_p can either be all set consistently to $|O'|$ or more accurately per product p to $|\{o|o \in O' : o \in ordProd(p)\}|$. Incompatibilities between products and compartments Inc_{PC} are considered via equalities (7.5), while incompatibilities between different products Inc_{PP} are taken into account via equalities (7.6). Last and in this case even least the objective function (7.1) basically maximizes the number of placed orders, though equalities (7.2) render it a dummy objective function since every feasible solution has the value $|O'|$. The latter is mainly because we are only interested in feasible solutions where all orders are placed and want this condition to be included the model.

This model is merely to formalize the CAP, its practical applicability will be discussed in Section 7.3.3.

Complexity of the CAP

Here we investigate the complexity of the CAP when having (at least) two compartments of fixed capacity. We can show its $\mathcal{NP}$-hardness via reduction from the *subset sum problem* (SSP) to the CAP. The SSP is defined as follows: Given a set of n integer numbers and a number s, does a subset of the integer numbers exist which sums up to s? The construction of a CAP that solves the SSP is as follows: the CAP has n orders, with order sizes equivalent to the integer numbers of the SSP, assuming a vehicle with capacity Q equals the sum of all integers and incorporating two compartments with capacities s and $Q - s$. A feasible solution for the CAP exists if and only if a feasible solution to the SSP exists. Note that CAP assumes non-negative order sizes, but SSP may contain negative numbers. However, such a SSP can be transformed to an equivalent SSP with only positive integers. Therefore, we can assume a SSP with only positive integer numbers.

The SSP is, however, after all only of limited applicability, as it effectively only covers CAPs involving two compartments. Further, to be more precise, we have so far only shown that the CAP is at least *weakly* $\mathcal{NP}$-hard, as the SSP belongs to this class. We will reconsider its complexity at the end of Section 7.3.3.

7.3.2 Cascaded CAP Solving Approach

Having described the different scenarios and characteristics of the CAP we explain how it will actually be tackled in the solution approaches to follow, either deriving a feasible

solution for it or determining that it is unsolvable. At some points we differentiate between adding a single order or several orders to an existing (possibly empty) assignment. These sequence of solving techniques is applied:

1. Perform a simple check considering the current load and the total vehicle capacity. If the additional demand exceeds the remaining capacity the CAP is infeasible.
2. To speed up the packing process and hence save computation time at a first attempt we try to solve an incremental CAP by keeping the possible already existing assignments and trying to insert the additional orders:
   a) For a single order: try to insert it in an allowed compartment (respecting the incompatibilities) either using *first-fit* or *best-fit* (FF or BF). If no single compartment with enough residual space exists we check whether the total residual space for this product type scattered over all admissible compartments is less than the demand of this order. If so, the CAP is unsolvable.
   b) For several orders: determine whether the simple continuous lower bound

$$\#min\_compartments = \sum_{p \in P} \left\lceil \frac{\sum_{o \in O' : o \in ordProd(p)} d_o}{\max_{c \in C} Q_c} \right\rceil$$

   exceeds the number of compartments. If not, try to insert the orders via first fit or *best-fit decreasing* (BFD).
3. For several orders: remove all previously assigned orders and try to solve the CAP from scratch via FF or BFD.
4. *Optionally* for several orders: finally try to solve it exactly as described in the next section.
5. *Optionally* for a single order: also try to solve the CAP from scratch via FF or BFD, if unsuccessful try to solve it exactly as described in the next section.

For completeness we also describe the simple and well-known (bin-)packing heuristics which were mentioned before:

- first-fit: place the order in the first compartment where it fits (respecting capacity and incompatibilities),
- best-fit: place the order in the viable compartment resulting in the minimal residual space,
- best-fit decreasing: consider the orders in the sequence of decreasing demand and pack each one via best-fit.

For more information about them we refer to [33].

7.3.3 Exactly Solving the CAP

Here we will describe the exact solution approaches for the CAP, which are two based on integer linear programming and one on constraint programming.

Applying the Straightforward ILP Approach

Here the straightforward model from Section 7.3.1 is utilized, which can be directly applied to solve the problem. However, the required effort (runtime) for solving it is not appealing. This is mainly due to considering all product types at once, a circumstance that will be considered for improvement in the next ILP model.

Nevertheless we also tried a variant of it where equalities (7.2) are changed to inequalities (i.e., replacing "=" by "$\leq$"). This time the model is actually an optimization problem, packing as many orders as possible, afterwards checking if all could be assigned to a compartment to identify feasible solutions. Not surprisingly the runtime is even higher as before in general, as often a substantial effort is devoted to pack, say, all but one order yet still resulting in an infeasible solution.

Improved ILP Approach

Taking a closer look at the problem and considering the incompatibilities between products, reveals that one can in fact split the CAP into several independent bin packing problems. Whenever orders of a certain product type are involved in the CAP a bin packing problem needs to be solved for them, always minimizing the number of compartments used ("opened"). We will state an appropriate ILP model for this, assuming that O'_p contains only orders belonging to the same product type and C' is the set of currently available empty compartments:

$$\min \quad \sum_{c \in C'} z_c \tag{7.9}$$

$$\text{s.t.} \quad \sum_{c \in C'} x_{oc} = 1 \qquad \forall\, o \in O'_p \tag{7.10}$$

$$\sum_{o \in O''} d_o x_{oc} \leq Q_c z_c \qquad \forall\, c \in C' \tag{7.11}$$

$$z_{c+1} \leq z_c \qquad \forall\, c \in \{1, \ldots, |C'| - 1\} \tag{7.12}$$

$$x_{oc} \in \{0, 1\} \qquad \forall\, o \in O'_p;\ c \in C' \tag{7.13}$$

$$z_c \in \{0, 1\} \qquad \forall\, c \in C' \tag{7.14}$$

Binary variables x_{oc} (7.13) are defined as before, while binary variables z_c (7.14) indicate whether compartment c is used. We ensure that each order must be placed in exactly one compartment (7.10), and that the capacity of the compartments might not be exceeded (7.11), conveniently linking both variables via the latter constraints. Finally, constraints (7.12) are introduced for symmetry breaking, forcing that compartments with lower index are filled

first. Note that the latter constraints are usually added to reduce the effective search space and hence to potentially speed up the solution approach. Though we did not observe this effect in practice, it would perhaps be more evident given a larger number of compartments. Also note that when the minimal amount of required compartments exceeds $|C'|$ then there exists no feasible packing.

An outline of this procedure is shown in Algorithm 18, also stating the variant with constraint programming explained in the next section.

Constraint Programming Approach

The third solution approach is based on constraint programming (CP), otherwise being quite similar to the previous ILP approach. So also here we consider the orders per product type and solve the corresponding bin packing problem. Fortunately we encountered a bin packing example (`bin-packing.cpp`) distributed with Gecode [1], which could more or less be utilized out-of-the-box. In this example the lower bound L_2 according to Martello and Toth [135] is applied together with a simple upper bound obtained via FFD; both were adopted by us in Algorithm 18. The CP model uses three types of variables: one per order holding the selected compartment, one per compartment representing the load, and a single variable stating how many compartments are currently in use. The domain of the latter decision variable is initially set to $[LB, UB]$. We always use the (optional) more sophisticated components: a problem specific bin-packing constraint introduced by Shaw [209], as well as the complete decreasing best-fit branching by Gent and Walsh [93] with some improvements also given in [209]. Contrary to the previous ILP model we cannot directly limit the number of compartments to those which are currently available, but have to check afterwards. We will not further go into detail here, but refer to Chapter 17 of the Gecode manual [207]. Again, in Algorithm 18 the outline of the solution procedure also when using the CP model is shown.

Complexity of the CAP Revisited

As we have seen the CAP can be considered a multiple bin packing problem, where one bin packing problem arises per product type. Since the bin packing problem is a strongly $\mathcal{NP}$-hard problem, we might be tempted to categorize the CAP as a strongly $\mathcal{NP}$-hard problem as well. Yet this point of view of the CAP is perhaps too strict. It might be an unnecessary effort to obtain per product type the minimal number of bins possible, as this might eventually lead to some spare compartments, although even a solution where all of them are opened would be equally good (as the CAP is a decision problem).

Very recently we discovered the work of O'Neil [152] in which a sub-exponential time algorithm for the bin packing problem with a fixed number of bins is proposed. The algorithm is a so-called *dynamic dynamic programming* (DDP) algorithm where the pool of solutions is dynamically allocated, additionally pruning this pool depending on problem semantics and the set of the remaining items to be considered. Since the CAP involving an arbitrary number

Algorithm 18: Solve the CAP for order subset O' with ILP or CP via several bin-packing problems, returning *infeasible* or a feasible solution

```
numFreeCompartments ← |C|
foreach p ∈ P do
    O'_p ← {o|o ∈ O' ∧ o ∈ ordProd(p)}
    if O'_p ≠ ∅ then
        LB ← determine lower bound L_2 for O'_p according to [135]
        if LB > numFreeCompartments then
            return infeasible
        UB ← determine upper bound for O'_p via FFD
        if LB == UB then
            store feasible and optimal FFD sub-solution for product p
            numFreeCompartments ← numFreeCompartments −
                compartments used in FFD solution
        else
            if apply ILP then
                exactSolution ← solve ILP model for O'_p using |C'| =
                    numFreeCompartments
                if exactSolution is feasible then
                    numFreeCompartments ← numFreeCompartments −
                        compartments used in exactSolution
                    store feasible and optimal sub-solution for product p
                else
                    return infeasible
            else // apply CP
                exactSolution ← solve CP model for O'_p using bounds
                    LB and UB
                if compartments used in exactSolution ≤ numFreeCompartments then
                    numFreeCompartments ← numFreeCompartments −
                        compartments used in exactSolution
                    store feasible and optimal sub-solution for product p
                else
                    return infeasible
return feasible solution combining all sub-solutions
```

of compartments could be solved by several applications of this algorithm it can again and finally be classified as weakly $\mathcal{NP}$-hard.

7.3.4 CAP Solution Cache

When solving the VRPC we can expect that some CAP instances appear several times. Hence in order to save unnecessary computations we use a simple solution cache, holding the results of the exact CAP solution approach. This cache is realized with a hash map, using as key the array of the orders sorted according to increasing index and stores whether the solution is feasible plus the assignment of the orders to the compartments. In preliminary tests this variant outperformed a static bit set approach (one bit per order). Of course more sophisticated cache variants could be implemented, very promising could be to use index structures that also offer subset and superset queries. The latter were successfully applied to a two-dimensional routing and loading problem in [217].

7.3.5 Density as Packing Measure

So far we treated each feasible solution to the CAP equally, since we are essentially only interested in whether we can find a packing or not. If each CAP would always be solved in an exact way this would still be the case. However, since the CAP or even the incremental variant of it will mainly be solved heuristically to significantly reduce the computational effort we might prefer some packings over others. Indeed, as will be shown later in the results, it is beneficial to increase the *density*, i.e. the efficiency, of the packing. Opting for a high packing density directly corresponds to maximizing the utilization of the vehicles, eventually enabling a more (cost) efficient delivery, e.g. allowing a customer to be visited by only one instead of two vehicles. The basic idea is adopted from a concept introduced by Falkenauer and Delchambre for the one-dimensional bin packing and line balancing problem [73]. This density measure is the average squared loading ratio (load divided by capacity) on a per compartment basis for fixed capacities:

$$\frac{\sum_{c \in C} \left(\frac{L_c}{Q_c}\right)^2}{|C|},$$

where L_c and Q_c denote the load and capacity of compartment c, and on a per vehicle basis otherwise:

$$\left(\frac{L}{Q}\right)^2,$$

where L and Q denote the vehicle load and capacity, respectively. The value of the density lies within the range $[0, 1]$. These formulas are applicable to single routes, for the whole solution the overall density is just the average of all routes.

7.3.6 Local Search to Improve the Packing

When applying the heuristic insertion as well as solution procedures for the CAP mentioned in Section 7.3.2 it is expected that the packing "degrades" over time, i.e. the density decreases, and one fails with increasing probability to add orders to an existent packing. This is because the residual space tends to get scattered over the compartments and it becomes less likely that a great portion of it remains in a single compartment. Hence, in order to maintain a rather favorable packing we additionally apply a local search specifically aiming at the packing and using the density as objective function to be maximized:

1. Re-insert all orders of a route using BFD similar to the fallback strategy when the incremental CAP cannot be solved. Keep an improved packing, in any case go to step 2. Note: a possibility would be to (partly) change the deterministic order sequence—according to decreasing demand—at random, but we did not consider this here.

2. Iteratively empty single compartments and re-insert these orders via BF. The compartments are considered according to increasing index. Repeat this procedure as long as an improvement is obtained, else go to step 3.

3. Iteratively apply several order exchange moves, trying to exchange two by two, two by one, and one by one order. Consider the orders as they appear in the corresponding vehicle route. If this procedure yielded an improvement go to step 2 again.

Moves 2 and 3 are similar to the heuristic for bin packing presented in [130], they are further applied in a variable neighborhood descent fashion (as pointed out by the "go to" statements). This local improvement procedure, in the remainder denoted as repacking heuristic, is only applied on improved solutions (to be more precise: on routes which changed), as they are the basis of the subsequent iterations of the metaheuristic in use. Applying it on each newly derived solution would yield no further gain, as the travel costs are not changed in any way. An alternative would be to apply it also on intermediate solutions, e.g. after removing some of the orders in course of a neighborhood move, which might allow an easier re-insertion and could eventually lead to minimized costs. However, a significant overhead would most likely be the result of this, so we did not actually apply it that way.

7.4 Variable Neighborhood Search for the VRPC

Our first metaheuristic solution approach is primarily based on *variable neighborhood search* (VNS) and includes some of the problem-specific techniques from [60] which were reported to yield good performance. In the following we give an overview on our VNS for the VRPC.

7.4.1 Objective Function

Besides trying to minimize the total routing costs, which still is the ultimate goal, we also aim at increasing the overall packing density described in Section 7.3.5. Hence the density

Table 7.2: Shaking neighborhoods and their order as applied by the VNS, newly proposed ones are marked bold.

k	$\mathcal{N}_k$
1	randomly remove orders
2	**remove random customers' orders**
3	remove orders of random route
4	remove orders of longest route
5	**remove orders of least density route**
6	**remove random compartments' orders**
7	**remove orders of non-empty least loaded compartment (having most orders)**
8	remove most costly orders (**as sets**) based on detour
9	remove orders based on similarity
10–15	exchange segments with lengths up to $k - 7$

is applied as a minor objective for tie-breaking when having two solutions with equal costs, basically preferring a smaller amount of well-filled compartments (routes) over many equally and hence less-filled ones. We further only consider feasible solutions during search, hence no sort of repair operations or penalty terms for violations are necessary.

7.4.2 Initial Solution

As initial solution for the classical single-trajectory VNS we select the best solution out of several generated with variants of best insertion, the savings algorithm, and the sweep algorithm. We implemented two sweep-like algorithms: the first is similar to that of [60], even though we did not search for the largest (radial) gap among all customers to obtain the beginning of the order sequence for insertion. In contrast we select a customer at random, consider the successive one tenth of the customers and select the largest gap among them (denoted as "sweep 1"). In the second variant we do not only insert the orders in the current single open route but insert them in a greedy fashion considering all potential routes (denoted as "sweep 2"). As expected, and also experienced in [60], the performance of these initialization procedures depend on the characteristics of the problem instance. In the results section we will state how often the individual methods could obtain the best initial solution for the instances considered.

7.4.3 Shaking Neighborhoods

In the shaking phase we utilize several move operations, i.e. we remove and reinsert a certain number of orders selected according to different criteria. On the one hand, we choose the orders either at random, based on the induced costs (or detour), or on their similarity to a randomly chosen seed order taking into account the product type, the demand and the

customer location as in [60]. Regarding the removal based on the induced costs, unlike in [60] we consider orders belonging to the same customer as a set, otherwise only the first and last order of such a route sub-sequence would be "misplaced" (since the distance between orders of the same customer is zero). Whole sets of orders are also selected either via considering orders belonging to a certain customer, or being contained in a route which is itself selected at random, having the highest routing costs, or the least density. Similarly, such sets of orders might belong to a randomly selected compartment or the compartment with the least load. In the latter case we use the number of orders as a tie-breaking criterion and prefer a higher number, since several smaller orders are easier to reinsert. On the other hand, we also try to exchange route segments of limited size between two different routes as is often done in the context of VRPs. An overview of the shaking neighborhoods $\mathcal{N}_k,\ k \in [1, 15]$, as well as their order is shown in Table 7.2.

7.4.4 Insertion of Orders

The insertion of single orders in a route's sequence is either done in a purely greedy and thus myopic way or using a regret-k heuristic [170, 60] which acts more foresighted. The latter takes the k cheapest routes' insertion costs into account for selecting the next order. We randomly select k to be between two and five or equal to the number of all routes. All six insertion variants are selected with equal probability.

Local Improvement

To improve upon the travel costs after the insertion we apply the well-known 3-opt intra-route exchange procedure, where in terms of sequences a-b-c we exclusively apply the move resulting in a-c-b. Additionally, all improved solutions as well as with a small probability $P^{\text{new}}_{\text{2-opt}^*}$ (concrete value will be given in the results section) also newly derived solutions lying within two percent to the current incumbent solution are subject to the 2-opt* inter-route exchange heuristic [173]. In 2-opt* all routes' end segments of all route pairs are tried to be exchanged, hence contrary to 3-opt also the packing needs to be checked and solved. Both exchange procedures are applied repeatedly and in a first improvement fashion. So also for the VRPC we basically rely on the same local improvement methods which proved successful in previous chapters.

7.5 Adaptive Large Neighborhood Search Based on VNS Components

As second metaheuristic we apply *adaptive large neighborhood search* (ALNS) [199, 171]. Our motivation was twofold: (i) already having implemented a VNS we can realize an ALNS with only little additional effort as basically only the selection of the (shaking) neighborhoods differs, and (ii) ALNS is especially appropriate when the neighborhoods are only partly or even not overlapping at all, which is in contrast to the VNS where for consecutive shaking neighborhoods it is usually assumed that $\mathcal{N}_i \cap \mathcal{N}_{i+1} \neq \emptyset$ or even $\mathcal{N}_i \subset \mathcal{N}_{i+1}$.

Since in our case the number of orders to be removed is selected at random whenever one of the neighborhoods is applied and they are further conceptually quite different, we deemed ALNS a viable candidate.

For ALNS in each iteration one of the neighborhoods is selected at random with a probability directly proportional to its previous success, stated as a score value. This success, or contribution to the search process, is in general not necessarily restricted to cover improved solutions only, e.g. also the ability for diversification might be accounted for. Several schemes have been proposed in the literature regarding the update of the score value. After some preliminary tests we settled with a quite simple variant, which was also used in [104]: whenever an improved solution is found the corresponding score is increased by one, initially setting all scores to one. All other components of this ALNS are the same as for our VNS.

7.6 Experimental Results

The algorithm was implemented in C++, compiled with GCC 4.5 and executed on a single core of a 2.53 GHz Intel Xeon E5540 with 24 GB RAM, 3 GB RAM dedicated per core. We apply the general purpose MIP solver IBM ILOG CPLEX 12.2 for the ILP packing models, whereas the constraint programming based bin packing approach is implemented in Gecode 3.7.1 [1]. For all settings and instances we performed 10 runs. The number of orders to be removed and reinserted during shaking is chosen at random between two and one third of all orders. If not stated otherwise, newly derived solutions (lying within two percent to the current incumbent solution) will be subject to 2-opt* with a probability of 1%, i.e. $P^{\text{new}}_{\text{2-opt}^*} = 0.01$. Regarding the packing we distinguish between solely applying first-fit (denoted by VNS-FF or ALNS-FF), or using best-fit and best-fit decreasing when inserting single orders and several orders, respectively (denoted as VNS-BFD or ALNS-BFD). Since first-fit is used for inserting single as well as several orders, we denote the respective sub-methods as FF^1 and FF^+ if they need to be distinguished. When a method applies the density measure, including the shaking neighborhoods related to the packing, this is denoted by adding "d" as superscript. In case the repacking heuristic is used we add a "r" as superscript. Note that the latter are only effective when facing a hard packing problem (CAP), so they will only be considered for some instance sets.

7.6.1 Christofides and Eilon Based Instances

The first set of instances is based on well-known VRP instances introduced by Christofides and Eilon, available in the VRPLIB [228] denoted as *symmetric CVRP instances*. Some of them were extended to two compartments by introducing two product types and assigning the original requests to both of them (i.e. doubling them), where the capacity of each compartment corresponds to the original vehicle capacity. On the positive this scheme allows for an easy regeneration of the instances (they are not publicly available) but on the negative their very special structure limits the usefulness for the VRPC, especially for such a general variant as considered here, a fact that was also pointed out in [60]. Since an optimal solution to these instances directly corresponds to an optimal VRP solution where the original

Table 7.3: Considered adapted Christofides and Eilon instances (only those without route length restrictions), also giving the costs of the corresponding best known VRP solution.

Id	BKS (of VRP)
Subset 1	
E051-05e	524.61
E076-10e	835.26
E101-08e	826.14
E151-12c	1028.42
E200-17c	1291.29
E121-07c	1042.11
E101-10c	819.56
E241-22k	707.79
E484-19k	1107.19
Subset 2	
E072-04f	237.00
E076-08s	735.00
E076-07u	682.00
E135-07f	1162.00
Subset 3	
E253-27k	859.11
E256-14k	583.39
E301-28k	998.73
E321-30k	1081.31
E324-16k	742.03
E361-33k	1366.86
E397-34k	1345.23
E400-18k	918.45
E421-41k	1821.15
E481-38k	1622.69

requests appear in both compartments (so to say “mirrored”), they are far more appropriate for testing the ability to solve VRP-like VRPC instances, and in fact not facing any special packing/partitioning subproblem. It has to be noted that because of this in [72] a second set was used where the demands of the requests are not alike but differ in their amount between the the two products. Unfortunately no direct comparison is possible since on the one hand their creation involves randomness and they were not made publicly available, and on the other hand they used real numbers for the demands, which is at present not incorporated in our implementation. Basically in line with this, also in [147] an additional set was created, this time not based on any VRP instances and using integer demands. However, it was not

Table 7.4: Results of VNS for the Christofides and Eilon based instances for different CPU-times.

Instance	5 min		10 min		20 min		60 min	
	min	avg	min	avg	min	avg	min	avg
E051-05e	524.61	526.47	524.61	526.47	524.61	526.47	524.61	526.47
E076-10e	836.78	841.12	835.77	840.92	835.26	840.72	835.26	840.41
E101-08e	831.71	836.06	831.71	835.21	831.71	833.94	830.79	832.62
E151-12c	1047.64	1052.94	1047.64	1052.27	1047.64	1052.21	1047.64	1051.14
E200-17c	1319.91	1335.11	1318.74	1332.97	1318.74	1330.48	1315.22	1325.65
E121-07c	1042.12	1043.85	1042.12	1043.68	1042.12	1043.68	1042.12	1042.47
E101-10c	819.56	819.56	819.56	819.56	819.56	819.56	819.56	819.56
E241-22k	723.77	730.33	723.77	729.00	720.36	726.68	715.94	722.01
E484-19k	1166.90	1170.30	1165.01	1168.79	1161.46	1165.02	1151.23	1159.55
%-gap$_{BKS}$	1.40	1.90	1.36	1.82	1.26	1.70	1.05	1.49
E072-04f	241.97	241.97	241.97	241.97	241.97	241.97	241.97	241.97
E076-08s	742.04	749.14	742.04	748.53	742.04	747.68	742.04	746.91
E076-07u	691.51	698.34	687.60	695.78	687.60	694.53	687.60	694.05
E135-07f	1162.96	1167.57	1162.96	1167.57	1162.96	1167.57	1162.96	1165.82
%-gap$_{BKS}$	1.13	1.72	0.99	1.61	0.99	1.53	0.99	1.45
E253-27k	885.70	891.95	884.07	890.10	880.54	887.63	878.17	885.41
E256-14k	600.28	604.00	599.59	602.07	599.17	601.59	597.03	600.07
E301-28k	1036.19	1042.98	1036.19	1041.52	1033.07	1039.68	1028.70	1035.73
E321-30k	1122.77	1131.42	1119.12	1128.88	1118.34	1126.48	1111.73	1122.94
E324-16k	773.71	778.56	772.92	776.31	769.37	775.15	768.16	773.74
E361-33k	1415.40	1420.81	1412.09	1418.39	1411.09	1415.64	1402.08	1411.50
E397-34k	1395.60	1407.86	1386.23	1403.13	1383.63	1400.42	1383.63	1395.37
E400-18k	956.54	964.30	952.14	961.57	951.40	959.38	950.11	956.70
E421-41k	1904.04	1909.89	1896.92	1906.18	1894.34	1902.82	1889.10	1898.67
E481-38k	1697.38	1703.17	1686.36	1694.57	1682.05	1689.45	1673.95	1684.24
%-gap$_{BKS}$	3.84	4.48	3.52	4.20	3.31	3.99	2.97	3.70

made public, too, and they further used it to highlight the benefits of co-collection in general and not presented detailed results. Nevertheless, we still consider the few VRP-like VRPC instances to obtain a comparison as meaningful as possible; they are given in Table 7.3 with the costs of the corresponding best known solutions (BKS), which are derived from the best known VRP solutions, as well as their partition into subsets as considered later on.

Similar to [60] we will report on results of our methods for CPU-times of 5, 10, 20, and 60 minutes, stating the minimal costs obtained in the 10 runs (min) as well as the average costs (avg). Due to the simple packing problem both methods rely on first-fit for placing the orders into the compartments, and apply the density measure. So, adhering to our naming convention they should be denoted as VNS-FF^d and ALNS-FF^d, though for simplicity we only use the abbreviations VNS and ALNS as we only apply these variants here. The results of VNS and ALNS are shown in Table 7.4 and Table 7.5, respectively.

Table 7.5: Results of ALNS for the Christofides and Eilon based instances for different CPU-times.

Instance	5 min		10 min		20 min		60 min	
	min	avg	min	avg	min	avg	min	avg
E051-05e	524.61	526.47	524.61	526.47	524.61	526.47	524.61	526.47
E076-10e	835.26	841.80	835.26	841.60	835.26	841.60	835.26	841.60
E101-08e	828.34	834.11	828.34	833.49	828.34	833.36	828.34	833.00
E151-12c	1047.38	1053.61	1047.38	1053.15	1046.55	1051.41	1045.58	1048.44
E200-17c	1320.02	1331.14	1317.23	1327.66	1317.23	1327.15	1317.23	1324.60
E121-07c	1042.12	1043.26	1042.12	1042.95	1042.12	1042.82	1042.12	1042.65
E101-10c	819.56	819.56	819.56	819.56	819.56	819.56	819.56	819.56
E241-22k	718.00	726.43	716.58	722.86	714.77	720.79	714.62	716.68
E484-19k	1161.81	1172.05	1160.91	1168.12	1156.93	1164.46	1150.59	1159.90
%-gap$_{BKS}$	1.19	1.80	1.13	1.66	1.06	1.56	0.98	1.39
E072-04f	241.97	241.97	241.97	241.97	241.97	241.97	241.97	241.97
E076-08s	742.56	748.68	742.56	748.24	742.56	746.52	742.56	746.38
E076-07u	687.60	695.48	687.60	695.39	687.60	693.84	687.60	693.75
E135-07f	1162.96	1164.78	1162.96	1164.65	1162.96	1164.01	1162.96	1164.00
%-gap$_{BKS}$	1.01	1.54	1.01	1.52	1.01	1.39	1.01	1.39
E253-27k	881.63	890.20	876.52	887.04	876.52	885.04	876.52	881.99
E256-14k	598.02	602.23	597.62	600.79	592.72	599.65	591.63	598.33
E301-28k	1033.03	1037.56	1031.86	1035.19	1027.53	1032.23	1023.79	1028.83
E321-30k	1115.92	1127.96	1115.25	1125.32	1113.61	1123.02	1113.16	1122.09
E324-16k	771.92	777.19	771.51	775.52	771.23	773.97	768.00	772.52
E361-33k	1412.63	1420.21	1404.58	1414.52	1401.17	1410.51	1399.26	1405.09
E397-34k	1390.37	1404.22	1384.25	1399.14	1383.94	1394.75	1379.11	1389.40
E400-18k	955.93	962.92	952.22	958.76	949.26	956.15	946.91	952.88
E421-41k	1896.25	1911.17	1893.23	1906.92	1886.98	1900.23	1876.01	1887.15
E481-38k	1688.33	1696.54	1684.27	1692.33	1678.08	1685.91	1670.72	1681.00
%-gap$_{BKS}$	3.47	4.24	3.20	3.94	2.92	3.65	2.64	3.32

In Table 7.6 we state the relative usage of the neighborhoods, as well as how often they yielded an improved solution. VNS does not focus on specific neighborhoods and applies them all alike. Although VNS returns to the first shaking neighborhood in case of an improvement, this has no notable effect here as the number of improvements is negligible compared to the overall iterations. In contrast, ALNS effectively reinforces neighborhoods showing a good performance, and interestingly neighborhoods 10–15 are less often applied as in case of VNS yet show a higher success rate. However, for both methods these neighborhoods only play a minor role.

Finally, a comparison to previous methods is shown in Table 7.7. The memetic algorithm (MA) and tabu search (TS) of El Fallahi et al. [72] were run on a PC Pentium 4 at 2.4 GHz, the guided local search (GLS) of Muyldermans and Pang [147] was run with different iteration limits on a PC Pentium M740 at 1.73 GHz, and the method of Derigs et al. [60] was run with different time limits on a not further defined 3 GHz PC with 2 GB RAM. While both

Table 7.6: Relative usage and contribution to success of shaking neighborhoods of VNS and ALNS on Christofides and Eilon based instances.

k	VNS		ALNS	
	%-use	%-success	%-use	%-success
1	6.67	17.92	5.50	9.83
2	6.67	60.27	34.27	70.66
3	6.67	1.68	5.44	0.79
4	6.67	0.52	3.94	0.29
5	6.67	1.11	4.90	0.54
6	6.67	1.01	4.21	0.84
7	6.67	0.92	4.64	0.63
8	6.67	13.46	8.74	12.31
9	6.67	2.87	4.87	3.20
1–9	60.00	99.76	76.52	99.09
10	6.67	0.01	3.72	0.11
11	6.67	0.05	3.97	0.14
12	6.67	0.02	3.94	0.18
13	6.67	0.05	3.85	0.11
14	6.67	0.05	3.82	0.11
15	6.67	0.07	4.18	0.25
10–15	40.00	0.24	23.48	0.91

former computing environments are definitely slower than ours, a comparison to the last one is not really possible. Nevertheless, against better judgement we chose the same CPU-time limits. None of these works reports average results and no details are given about how many runs were performed in total. As a consequence of this we regard their single runs as their best runs in the following, and at least when comparing our best runs to the results of Derigs et al. this seems reasonable, albeit comparing best (or also single runs in general) has to be done with care. We can observe that the MA, TS and GLS seem inferior in solution quality to the latest approaches, although keeping in mind that they were mostly given less runtime. Comparing the results obtained by Derigs et al. to ours reveals that there is hardly a difference when given one hour runtime. However, looking at the results of the 5, 10, and 20 minutes runs, both VNS and ALNS yield a better solution quality. Opposing VNS and ALNS is in favor of the latter: ALNS almost always obtains better results on average as well as with regard to the best found solutions.

7.6.2 Instances of Derigs et al.

The second set of instances was introduced in [60] and is available online at `http://www.ccdss.org/vrp/` together with the best known solutions. The instances differ in type

Table 7.7: Comparison to previous methods applied on the adapted Christofides and Eilon instances.

Method	runs	Subset 1			Subset 2			Subset 3		
		%-gap avg	%-gap min or 1×run	t[s] avg	%-gap avg	%-gap min or 1×run	t[s] avg	%-gap avg	%-gap min or 1×run	t[s] avg
MA	–		2.82	369.3		1.72	58.0			
TS	–		2.12	968.7		1.63	55.2			
GLS (300k)	–		1.88	152.7					4.60	184.1
GLS (600k)	–		1.45	305.0					4.31	361.3
GLS (1200k)	–		1.32	626.9					3.94	737.3
Derigs et al. (5 min)	–		1.74	300.0		1.21	300.0		4.81	300.0
Derigs et al. (10 min)	–		1.51	600.0		1.16	600.0		3.92	600.0
Derigs et al. (20 min)	–		1.24	1200.0		1.11	1200.0		3.33	1200.0
Derigs et al. (60 min)	–		0.95	3600.0		1.06	3600.0		2.64	3600.0
VNS (5 min)	10	1.90	1.40	300.0	1.72	1.13	300.0	4.48	3.84	300.0
VNS (10 min)	10	1.82	1.36	600.0	1.61	0.99	600.0	4.20	3.52	600.0
VNS (20 min)	10	1.70	1.26	1200.0	1.53	0.99	1200.0	3.99	3.31	1200.0
VNS (60 min)	10	1.49	1.05	3600.0	1.45	0.99	3600.0	3.70	2.97	3600.0
ALNS (5 min)	10	1.80	1.19	300.0	1.54	1.01	300.0	4.24	3.47	300.0
ALNS (10 min)	10	1.66	1.13	600.0	1.52	1.01	600.0	3.94	3.20	600.0
ALNS (20 min)	10	1.56	1.06	1200.0	1.39	1.01	1200.0	3.65	2.92	1200.0
ALNS (60 min)	10	1.39	0.98	3600.0	1.39	1.01	3600.0	3.32	2.64	3600.0

(petrol or food), number of customers (10 to 200, either clustered or not), number of products (2 or 3), vehicle capacity (600 to 9000), and the maximal order demand. This maximal order demand d_{max} represents an upper bound on the amount of single demands: the total customer demand of a specific product d_p is split into $\lfloor d_p/d_{max} \rfloor$ orders with a demand of d_{max} each, plus an additional order of demand $d_p \mod d_{max}$ (if this value is different from zero). For type petrol there are five compartments of fixed capacity $Q/5$ and d_{max}/Q_c is either 0.5 or 1, whereas for type food there are flexible compartments where each one might occupy the total storage, yet their total capacities sum up to the vehicle capacity and d_{max}/Q_c is either 0.25 or 0.5; also see Section 7.1.2 about these scenarios. There are 200 instances in total, 125 of type petrol and 75 of type food. Though they did not use and hence not explicitly state the packing density of the best solutions it can be calculated given the actual assignment of orders to compartments. We consistently set a CPU-time limit of 10 minutes as was done in [60] to allow for a more or less direct comparison. Three algorithm variants are tested: VNS-FF, VNS-BFD$^{d[r]}$, and ALNS-BFD$^{d[r]}$, where we apply the repacking heuristic only for instances of type petrol. Again 10 runs are performed per instance and setting.

The results on the instances of type food are given in Table 7.8, those on the instances of type petrol in Table 7.9, where we averaged them for instances with the same number of customers n and products p. As expected the VNS benefits more from the extensions for the instances of type petrol. However, also for the food instances where we are faced with a considerably simpler packing subproblem a slight gain can be observed. We can see that improved costs are accompanied by an improved (i.e. increased) density, suggesting that utilizing the density measure, along with the packing-related neighborhoods and the repacking heuristics, is beneficial. Since the average performances, when comparing to the best solutions of [60], are not varying much, we examine whether statistically significant differences exist. Corresponding results in Table 7.10 confirm our former impression that the performance gap between VNS-BFD$^{d[r]}$ and VNS-FF is larger on petrol type instances. We can further conclude that ALNS-BFD$^{d[r]}$ outperforms VNS-BFD$^{d[r]}$, though only for food type instances there is a notable difference.

Altogether, the performance of our algorithmic framework is very encouraging as highlighted in Table 7.11: the best method, ALNS, could obtain for 146 out of 200 instances (73.0%) a new best known solution, reach the same objective value for 37 instances (18.5%), and only for 17 instances (8.5%) the solution quality is slightly inferior. Yet remarkably, all methods perform quite similar with respect to the best solutions obtained. For a complete listing of the overall best found solution values we refer to Section A.2 in the appendix.

Finally, Tables 7.12 and 7.13 show the usage and success of the neighborhoods for instances of type food and petrol, respectively. We see again the focus of ALNS on better performing neighborhoods and that our newly introduced neighborhoods (number 2, 5, 6, and 7) contribute a lot to the total improvements in general.

Examining the Packing of the Petrol Type Instances

We want to take a closer look at the CAP regarding the petrol type instances, as in contrast to the food type instances they involve the $\mathcal{NP}$-hard packing subproblem. Therefore Ta-

Table 7.8: Average results of VNS variants and ALNS on instances of type food compared to so far best solutions obtained by Derigs et al. [60].

n	p	VNS-FF			VNS-BFD^d			ALNS-BFD^d		
		%-gap min cost	%-gap avg cost	%-gap avg density	%-gap min cost	%-gap avg cost	%-gap avg density	%-gap min cost	%-gap avg cost	%-gap avg density
10	2	-0.16	0.35	5.85	-0.16	-0.14	6.86	-0.16	0.02	5.64
	3	0.00	0.02	-0.05	0.00	0.00	0.35	0.00	0.00	0.35
25	2	-0.18	0.09	0.57	-0.23	-0.04	1.11	-0.42	0.01	0.94
	3	-0.10	0.16	0.18	-0.26	-0.08	0.83	-0.26	-0.10	0.83
50	2	-0.35	-0.13	0.35	-0.36	-0.12	0.83	-0.24	-0.05	0.43
	3	-0.12	0.17	0.65	-0.37	-0.11	1.56	-0.63	-0.25	1.72
100	2	-0.30	-0.04	0.17	-0.24	0.07	0.23	-0.18	0.08	-0.06
	3	-0.45	-0.23	0.11	-0.75	-0.47	1.27	-0.78	-0.43	1.08
200	2	-0.23	0.43	-0.86	-0.12	0.28	-0.29	-0.15	0.67	-1.22
	3	-0.30	-0.04	-0.07	-0.24	-0.02	-0.06	-0.37	-0.22	-0.08
avg		-0.25	0.03	0.34	-0.34	-0.09	0.93	-0.38	-0.07	0.70

Table 7.9: Average results of VNS variants and ALNS on instances of type petrol compared to so far best solutions obtained by Derigs et al. [60].

n	p	VNS-FF			VNS-BFD^{dr}			ALNS-BFD^{dr}		
		%-gap min cost	%-gap avg cost	%-gap avg density	%-gap min cost	%-gap avg cost	%-gap avg density	%-gap min cost	%-gap avg cost	%-gap avg density
10	2	0.00	0.08	-0.42	0.00	0.02	1.98	0.00	0.03	1.97
	3	0.00	0.37	0.04	0.00	0.39	0.79	0.00	0.32	0.71
25	2	-0.14	0.11	-0.18	-0.15	-0.01	0.90	-0.16	0.04	1.00
	3	-0.37	0.57	-1.68	-0.45	0.43	-0.30	-0.44	0.50	-0.46
50	2	-0.56	-0.01	0.46	-0.54	-0.18	1.24	-0.67	-0.14	1.09
	3	-0.70	0.16	0.22	-0.77	0.01	0.70	-0.86	0.02	0.77
100	2	-0.56	0.10	0.74	-0.87	-0.26	1.87	-0.89	-0.30	1.67
	3	-0.90	-0.06	1.98	-1.22	-0.27	2.29	-1.24	-0.13	1.56
200	2	-0.74	-0.31	-0.62	-1.47	-0.97	1.77	-1.61	-1.10	1.60
	3	-3.34	-1.63	2.50	-3.67	-2.32	4.79	-3.68	-2.35	4.06
avg		-0.62	0.06	0.26	-0.77	-0.16	1.29	-0.82	-0.14	1.12

Table 7.10: Pairwise Wilcoxon rank sum tests of results on instances of Derigs et al. with an error level of 5%, stating how often significantly better / worse.

	VNS-BFD$^{d[r]}$	ALNS-BFD$^{d[r]}$
type food		
VNS-FF	10 (13.3%) / 17 (22.7%)	9 (12.0%) / 21 (28.0%)
VNS-BFD^{d}	–	7 (9.3%) / 20 (26.7%)
type petrol		
VNS-FF	6 (4.8%) /26 (20.8%)	8 (6.4%) / 30 (24.0%)
VNS-BFD^{dr}	–	9 (7.2%) / 12 (9.6%)

Table 7.11: Comparison of best solutions obtained by our methods to best known solutions derived by [60].

	better	draw	worse
type food			
VNS-FF	48 (64.0%)	16 (21.3%)	11 (14.6%)
VNS-BFD^{d}	50 (66.7%)	15 (20.0%)	10 (13.3%)
ALNS-BFD^{d}	50 (66.7%)	18 (24.0%)	7 (9.3%)
type petrol			
VNS-FF	93 (74.4%)	18 (14.4%)	14 (11.2%)
VNS-BFD^{d}	95 (76.0%)	17 (13.6%)	13 (10.4%)
ALNS-BFD^{d}	96 (76.8%)	19 (15.2%)	10 (8.0%)
total			
VNS-FF	141 (70.5%)	34 (17.0%)	25 (12.5%)
VNS-BFD^{d}	145 (72.5%)	32 (16.0%)	23 (11.5%)
ALNS-BFD^{d}	146 (73.0%)	37 (18.5%)	17 (8.5%)

bles 7.14 and 7.15 show the amount of undecided packings, which are those (presumably hard) CAP instances where the heuristics fail to find a feasible solution yet we are also not able to prove the infeasibility via the applied checks (see Section 7.3). The amount is of small to moderate size. The difficulty of inserting single orders correlates to the number of customers, whereas we are seemingly more successful at inserting several orders. Table 7.15 shows that instances where the vehicle capacity Q is larger the both packing scenarios tend to get harder, which is not surprising as also the size of the route is increasing. There is only quite a small difference between VNS-FF and VNS-BFD^{dr}, more noticeable is the difference of both to ALNS-BFD^{dr}. As the latter two methods exhibit an equal performance on this instances this might confuse at first. The reason is the different usage of the neighborhoods as shown in Table 7.13, as ALNS implicitly also takes the packing outcomes into account

Table 7.12: Relative usage and contribution to success of shaking neighborhoods of VNS variants and ALNS on food type instances of Derigs et al.

k	VNS-FF		VNS-BFD^d		ALNS-BFD^d	
	%-use	%-success	%-use	%-success	%-use	%-success
1	7.69	39.21	6.67	35.28	17.93	45.97
2	7.69	16.01	6.67	12.19	12.77	11.71
3	7.69	3.03	6.67	2.37	6.27	0.86
4	7.69	1.50	6.67	1.36	5.29	0.75
5	–	–	6.67	4.00	5.13	2.30
6	7.69	9.48	6.67	9.12	7.99	5.08
7	–	–	6.67	11.41	7.21	10.68
8	7.69	2.99	6.67	2.23	4.79	2.09
9	7.69	24.97	6.67	16.90	9.98	19.34
1–9	53.85	97.19	60.00	94.86	77.34	98.79
10	7.69	0.53	6.67	0.87	3.64	0.16
11	7.69	0.53	6.67	0.89	3.80	0.20
12	7.69	0.50	6.67	0.83	3.82	0.22
13	7.69	0.42	6.67	0.79	3.81	0.21
14	7.69	0.41	6.67	0.90	3.81	0.21
15	7.69	0.41	6.67	0.85	3.78	0.21
10–15	46.15	2.81	40.00	5.14	22.66	1.21

and hence automatically prefers neighborhoods where the packing is more often solvable, leading to less undecided packings in general.

Although we have seen that usually not many packings are undecided, it would be interesting to know whether they could still lead to a better routing. Therefore we have tried to solve them in an exact way using one of our developed methods. The interesting finding was that we did not encounter a single CAP which was previously undecided and could be feasibly solved. All of them were shown to be infeasible, hence although we only applied heuristics for the packing, they seem to almost always yield a feasible solution whenever one exists for these instances. The reason has to be the strict pre-splitting according to the maximal order demand, producing a uniform demand distribution with only very few orders having a demand different from the maximal order demand, in fact only one per customer and product type (as a side note: the median absolute deviation is always zero).

7.6.3 Modified Derigs et al. Petrol Instances

To finally be able to test our methods on instances which actually provide a harder packing subproblem we modified the Derigs et al. instances of type petrol. Our goal was to make the order demands less uniform by introducing some randomness, yet we wanted them to still relate to the original instances. In the end we decided to choose their values at random in the range $[0.4 \cdot d_{max}, 0.6 \cdot d_{max}]$. Except for instances `vrpc_p_n100_p2_k06_120.vrp`,

Table 7.13: Relative usage and contribution to success of shaking neighborhoods of VNS variants and ALNS on petrol type instances of Derigs et al.

k	VNS-FF		VNS-BFD^d		ALNS-BFD^d	
	%-use	%-success	%-use	%-success	%-use	%-success
1	7.69	24.12	6.67	24.39	13.10	23.39
2	7.69	13.84	6.67	11.33	10.96	7.85
3	7.69	1.68	6.67	1.23	5.21	0.50
4	7.69	0.85	6.67	0.76	3.85	0.28
5	–	–	6.67	1.24	4.76	0.78
6	7.69	31.75	6.67	22.51	14.48	24.03
7	–	–	6.67	16.22	10.37	17.77
8	7.69	3.08	6.67	2.38	4.87	1.99
9	7.69	23.07	6.67	18.76	11.33	23.11
1–9	53.85	98.39	60.01	98.81	78.94	99.71
10	7.69	0.32	6.67	0.28	3.44	0.05
11	7.69	0.29	6.67	0.21	3.56	0.05
12	7.69	0.26	6.67	0.18	3.56	0.07
13	7.69	0.24	6.67	0.19	3.48	0.04
14	7.69	0.25	6.67	0.17	3.50	0.03
15	7.69	0.25	6.67	0.16	3.51	0.05
10–15	46.15	1.61	39.99	1.19	21.06	0.29

`vrpc_p_n10_p3_k1_c_200.vrp`, `vrpc_p_n50_p2_k06_120.vrp`, and `vrpc_p_n50_p3_k06_c_120.vrp`, where we had to choose $[0.2 \cdot d_{max}, 0.8 \cdot d_{max}]$ instead to realize a harder packing.

Computational Effort of Exact Packing Methods

In case the heuristics fail to solve a CAP instance and we have not already proven its infeasibility we occasionally want to solve it to optimality. Hence the different methods which were developed were analyzed in preliminary tests. As we are not able to integrate the density measure into these models there is not really a possibility to prefer one of the optimal solution over another, hence there is just a single important factor for to differentiate the methods: the runtime they need for solving. One scenario is to apply the exact methods on CAPs when inserting several orders, i.e. when building initial solutions with the savings heuristic, exchanging segments during shaking, and applying 2-opt*. The fraction of the overall runtimes which are devoted to exactly solving the CAPs for setting VNS-BFD^{dr} are shown in Table 7.16 for the two ILP approaches and the CP based approach. The alternative ILP formulation (ILP-2) more than halves the runtime needed for the straightforward ILP formulation (ILP-1), but the CP formulation is able to yield an incredible additional speedup of about 155. So the latter method is clearly superior and will be the only one considered in the following tests.

Table 7.14: Average results of undecided packings on petrol type instances of Derigs et al., differentiating between inserting single orders (via FF^1 or BF) or several orders (via FF^+ or BFD).

n	p	VNS-FF		VNS-BFD^{dr}		ALNS-BFD^{dr}	
		FF^1[%]	FF^+[%]	BF[%]	BFD[%]	BF[%]	BFD[%]
10	2	0.30	0.07	0.07	0.00	0.04	0.00
	3	0.36	0.07	0.17	0.05	0.07	0.01
25	2	1.38	0.70	0.54	0.12	0.29	0.06
	3	1.19	3.09	1.13	3.26	1.00	4.19
50	2	2.84	4.00	2.46	3.60	1.76	0.98
	3	2.42	2.87	2.52	2.48	1.84	0.70
100	2	4.63	1.05	4.66	0.89	2.83	0.08
	3	4.05	0.52	4.08	0.36	2.80	0.05
200	2	13.29	0.85	11.85	0.72	7.55	0.31
	3	10.71	0.40	11.64	0.31	7.66	0.20
avg		3.40	1.47	3.21	1.27	2.04	0.57

Table 7.15: Results of Table 7.14 given per vehicle capacity.

Q	VNS-FF		VNS-BFD^{dr}		ALNS-BFD^{dr}	
	FF^1[%]	FF^+[%]	BF[%]	BFD[%]	BF[%]	BFD[%]
600	0.35	0.01	0.24	0.00	0.09	0.00
800	0.80	0.07	0.43	0.04	0.19	0.01
1000	0.76	0.03	0.70	0.03	0.36	0.01
3000	4.20	0.88	2.66	0.32	1.55	0.09
9000	24.90	18.70	27.13	17.91	18.62	13.18

Testing Various Methods on the Modified Instances

For these instances we more thoroughly examine the effects of the proposed extensions, namely the density measure, the repacking heuristics, and the chosen exact approach for solving the CAP. We do this by iteratively enhancing the solution method, beginning with VNS-FF. There are two variants of exactly solving the CAP, as was already explained in Section 7.3.2: only when inserting several orders due to exchanging segments during shaking and applying 2-opt* (denoted as CP), as well as when also inserting single orders excluding insertions occurring in the repacking heuristics (denoted as CP^{full}). In total we tested six settings: VNS-FF, VNS-FF^d, VNS-BFD^d, VNS-BFD^{dr}, VNS-BFD^{dr}-CP, and VNS-BFD^{dr}-CP^{full}. In preliminary tests we also tried the ALNS, but the results were at most competitive to the VNS, so we do not apply it here. We use the same test settings as for the Derigs et

Table 7.16: Fraction of overall runtime devoted to exactly solving harder CAPs when inserting several orders using different methods.

n	p	ILP-1	ILP-2	CP
10	2	27.37	16.70	0.02
	3	27.43	15.78	0.04
25	2	13.51	3.81	0.02
	3	7.98	3.30	0.01
50	2	4.22	1.30	<0.01
	3	1.85	0.67	0.10
100	2	1.70	0.40	<0.01
	3	1.26	0.51	<0.01
200	2	0.78	0.21	<0.01
	3	0.45	0.05	<0.01
avg		6.77	3.10	0.02

Table 7.17: Average results of VNS variants on new instances of type petrol taking best solutions of VNS-FF as baseline.

Method	%-gap min cost	%-gap avg cost	%-gap avg density	t[s] avg
(1) VNS-FF	0.00	1.34	-1.18	602.9
(2) VNS-FF^{d}	-0.76	0.52	2.72	602.8
(3) VNS-BFD^{d}	-1.10	0.17	3.77	602.9
(4) VNS-BFD^{dr}	-1.18	0.10	4.54	602.9
(5) VNS-BFD^{dr}-CP	-1.42	-0.26	4.59	602.9
(6) VNS-BFD^{dr}-CP^{full}	-1.71	-0.47	5.50	1244.1

al. instances, i.e. a CPU-time limit of 10 minutes and 10 runs per setting and instance. The only exception being VNS-BFD^{dr}-CP^{full} where we examined in preliminary tests that the solution quality is not competitive when also allotting a 10 minute runtime, which is due to the effort of packing many CAPs in an exact way. Since we were interested in the potential of setting CP^{full} anyway, we decided to apply VNS-BFD^{dr}-CP^{full} with the same amount of iterations as VNS-BFD^{dr}-CP. Concise average results are shown in Table 7.17. Together with Table 7.18, stating the statistical significance results using the numbering introduced in Table 7.17, we see that most extensions have a notable and significant impact on the solution quality. Lesser improvements occur when directly comparing VNS-BFD^{d} and VNS-BFD^{dr}, as well as VNS-BFD^{dr}-CP and VNS-BFD^{dr}-CP^{full}. It is to note that for some instances we experienced that the repacking heuristic in its current form consumes quite a lot of runtime, so it would be wise to limit its computational effort. Looking at VNS-BFD^{dr}-CP^{full} shows that solving the CAP when also inserting single orders is beneficial in principle. However,

Table 7.18: Pairwise Wilcoxon rank sum tests on all 125 new petrol type instances with an error level of 5%, stating how often sign. better/worse.

Method	(2)	(3)	(4)	(5)	(6)
(1)	1/55	0/83	2/84	1/91	1/97
(2)	–	1/35	5/42	6/57	2/69
(3)	–	–	9/15	8/33	3/39
(4)	–	–	–	1/26	4/45
(5)	–	–	–	–	6/19

Table 7.19: More detailed average results of VNS-FF and VNS-BFD^{dr}-CP on new instances of type petrol taking best solutions of VNS-FF as baseline.

n	p	VNS-FF			VNS-BFD^{dr}-CP		
		%-gap min cost	%-gap avg cost	%-gap avg density	%-gap min cost	%-gap avg cost	%-gap avg density
10	2	0.00	0.72	-1.83	-2.33	-2.19	9.92
	3	0.00	1.59	-1.93	-1.54	-0.24	5.06
25	2	0.00	1.36	-1.74	-1.31	-0.43	3.90
	3	0.00	1.61	-2.59	-0.92	0.45	3.00
50	2	0.00	1.29	-0.68	-1.23	-0.20	5.04
	3	0.00	1.49	-0.36	-1.68	-0.15	4.81
100	2	0.00	1.02	-0.44	-1.57	-0.67	4.94
	3	0.00	1.21	-0.04	-1.65	-0.53	5.03
200	2	0.00	0.96	-1.07	-1.20	0.12	3.90
	3	0.00	1.73	-3.06	-1.22	0.30	3.06
avg		0.00	1.34	-1.18	-1.42	-0.26	4.59

we have to keep in mind that the actual improvements were achieved with roughly twice the amount of runtime and we would have to apply the other methods with a longer runtime, too, to allow a fair comparison. Finally, in Table 7.19 we give more detailed results of the baseline method, VNS-FF, and the most enhanced method with equal runtime, VNS-BFD^{dr}-CP.

Examining the Packing

As for the Derigs et al. instances we will also investigate the packing more closely here. Like before we state the fraction of the packings which remain undecided in view of the heuristics, and in case of the CP-enriched variants for how many of the CAPs which were subject to the exact solving approach a feasible solution could be obtained. In the latter case we additionally state the amount of solutions (either feasible ones or the proof that none exists) which could be retrieved from the hash map and were thus already solved before. We

Table 7.20: Average results of undecided packings of heuristics on new petrol type instances, differentiating between inserting single orders (via FF^1 or BF) or several orders (via FF^+ or BFD).

n	p	VNS-FF		VNS-FF^d		VNS-BFD^d		VNS-BFD^{dr}	
		FF^1[%]	FF^+[%]	FF^1[%]	FF^+[%]	BF[%]	BFD[%]	BF[%]	BFD[%]
10	2	14.09	0.12	5.29	0.11	4.33	0.10	4.24	0.11
	3	12.49	0.38	4.78	0.34	4.20	0.16	3.94	0.20
25	2	14.71	0.28	9.47	0.28	8.20	0.20	7.17	0.21
	3	12.01	0.30	8.14	0.30	6.68	0.19	5.13	0.19
50	2	14.40	0.26	11.50	0.28	10.02	0.18	7.66	0.18
	3	12.85	0.28	10.60	0.28	9.07	0.14	5.39	0.14
100	2	13.30	0.26	11.18	0.25	9.93	0.14	8.12	0.14
	3	10.70	0.33	9.17	0.31	8.13	0.14	5.34	0.14
200	2	12.65	0.15	12.00	0.16	9.90	0.10	8.50	0.11
	3	4.46	0.19	4.04	0.19	2.99	0.10	1.89	0.10
avg		12.80	0.29	9.50	0.28	8.17	0.17	6.17	0.18

use the same notation for CP as in previous sections for FF to state the concrete results, i.e. denote it as CP^1 when inserting a single order and as CP^+ othwerwise.

In Table 7.20 the results of the pure heuristic methods are shown, whereas Table 7.21 shows that of the hybrid methods. We see that the number of undecided packings for single orders is decreasing with the enhancement status of the given methods. Contrary, the number of undecided packings for several orders seems only to depend on whether FF or BFD is applied. Further, on average for about one fourth of all CAPs when inserting several orders (CP^+[%]) a feasible solution could be obtained, whereas in case of inserting single orders (CP^1[%]) it is about 12% on average. Not very surprisingly, we can conclude that it is definitely more "profitable" to solve the CAPs to optimality when inserting several orders. Especially when also taking the runtime into account, which was investigated already in the previous section. Further, as we have seen in Table 7.16 the time for solving the CP model for VNS-BFD^{dr}-CP is on average 0.02% of the overall runtime, whereas for VNS-BFD^{dr}-CP^{full} it is 13.98%. The large number of solutions which were retrieved from the hash map indicates that very often the same CAPs are considered. Without implementing such a solution archive, even more runtime would have to be devoted to solving the CAPs, in fact the whole method would not be viable anymore. Note that in case of VNS-BFD^{dr}-CP^{full} quite a huge number of packings are stored, leading to an according memory consumption which has to be considered (for these tests less than 3 GB RAM were used).

What was not mentioned so far are the absolute number of times the different packing methods were applied, which are interesting, too. Therefore these numbers are given in Table 7.22 as average per applied method, where for the CP-based packings we also count those cases

Table 7.21: Average results on new petrol type instances: undecided packings of heuristics, considered CAPs where a feasible solution could be obtained with the CP approach, as well as the fraction of solutions retrieved from the map ("hit"), differentiating between inserting single orders (via BF or CP^1) and several orders (via BFD or CP^+).

n	p	VNS-BFD^{dr}-CP				VNS-BFD^{dr}-CP^{full}					
		BF[%]	BFD[%]	CP^+[%]	hit[%]	BF[%]	BFD[%]	CP^1[%]	hit[%]	CP^+[%]	hit[%]
10	2	4.12	0.08	74.94	56.03	3.36	0.09	34.02	94.22	79.93	69.76
	3	4.01	0.20	30.76	85.31	3.22	0.18	22.90	95.43	35.15	87.54
25	2	7.28	0.19	19.01	89.09	6.43	0.19	11.36	97.36	21.78	91.24
	3	5.08	0.19	25.05	92.26	4.36	0.18	13.24	96.70	25.49	93.47
50	2	7.42	0.19	15.07	92.17	6.63	0.19	10.82	97.84	21.65	93.09
	3	5.25	0.13	20.22	91.41	4.65	0.14	12.35	97.49	23.33	92.50
100	2	7.94	0.13	14.04	89.59	7.25	0.14	8.61	98.39	16.07	90.98
	3	5.38	0.13	9.60	88.87	4.75	0.14	8.18	97.96	13.14	89.77
200	2	8.46	0.10	21.34	86.71	7.89	0.11	7.97	98.85	27.06	88.24
	3	1.92	0.10	10.32	87.91	1.65	0.10	19.75	97.39	13.44	86.63
avg		6.10	0.17	21.56	89.20	5.39	0.17	11.57	97.58	24.83	90.90

Table 7.22: Average applications of the different packing methods on new instances of type petrol taking.

Method	FF^1/BF	FF^+/BFD	CP^1	CP^+
VNS-FF	1938948918	19365950	–	–
VNS-FF^d	1880783649	18688275	–	–
VNS-BFD^d	1700816334	18333503	–	–
VNS-BFD^{dr}	1638908180	18348678	–	–
VNS-BFD^{dr}-CP	1624697163	18114419	–	30872
VNS-BFD^{dr}-CP^{full}	1572424316	18474309	72801609	31226

where the solution could be retrieved from the hash map (as we consider this as part of the method itself). Due to the way our metaheuristics are designed adding a single order to an existing packing occurs more often, the difference is roughly two orders of magnitudes for the heuristic packing methods and even larger for the exact packing methods. The latter are way less often applied as their heuristic counterparts, as they are only a fallback.

7.6.4 Performance of Initial Solution Construction Procedures

In this last results section we look at the performance of the initial solution construction heuristics. Remember that we generate several solutions using the different methods and pick the best one as starting solution. The evaluation for each instance set considered is shown

Table 7.23: Number of times in percent the respective construction heuristic yielded the best initial solution.

heuristic	times best initial solution[%]				
	Eilon	food	petrol	new petrol	avg
best insertion	0.00	20.76	20.26	0.00	10.26
savings	100.00	32.00	17.14	31.88	45.25
sweep 1	0.00	19.56	23.49	0.00	10.76
sweep 2	0.00	27.69	39.11	68.12	33.73

in Table 7.23, highlighting that the success of the methods highly depends on the instance characteristics. On average the savings method as well as our additional sweep heuristic (sweep 2), which was not applied in [60], perform best. However, we have to keep in mind that it is not generally the case that using the best initial solution ultimately yields the best solution after the optimization process, as e.g. a worse solution might allow more changes (with regard to a getting stuck in a local optimum) and be better suited as initial solution. So a more thorough investigation would be necessary to identify the real potential/benefit of the different construction procedures.

7.7 Conclusions

In this chapter we considered the vehicle routing problem with compartments (VRPC), which is basically the classical VRP incorporating vehicles offering more than one compartment, delivering goods of different type, and possible incompatibilities between products and between products and compartments. For the first time we specifically focused on the packing aspect of the problem, initially showing different scenarios and the resulting packing problem. In some cases, which were mostly considered in previous work, there is no packing problem at all, hence a simple capacity check is sufficient. In case of at least two compartments having a fixed capacity we are faced with an $\mathcal{NP}$-hard packing problem, which we denote as the compartment assignment problem (CAP). This problem is solved in a cascaded way, mainly relying on the first-fit, best-fit, and best-fit decreasing heuristics, but optionally also apply an exact approach to solve it. For the latter we proposed a straightforward overall integer linear programming (ILP) model and a simplified approach that considers one bin packing problem per product type. This alternative is realized either also by an ILP model or with constraint programming (CP). The CP approach turned out to be clearly faster and is thus the method of choice. We also implemented a solution cache via a hash map to avoid unnecessary re-computations for the exact approach. Further, we applied a measure to distinguish packings and ultimately to favor solutions with denser packings. For the developed variable neighborhood search (VNS), which uses some concepts from previous work, we introduced additional problem-tailored neighborhood structures for shaking, which also partly incorporate the packing, e.g. emptying single weakly packed compartments or clear-

ing whole routes with a bad packing. These neighborhoods were shown to contribute a lot to the overall improvements. Building upon the VNS, we also developed an adaptive large neighborhood search (ALNS), where, in contrast, the (shaking) neighborhood to be applied is always chosen at random with a probability according to its previous success. The initial solutions are constructed using different procedures, as their performance depends on the characteristics of the instance at hand.

Several instance sets were used to evaluate the methods. The most common used instances, based on VRP instances of Christofides and Eilon, have a very special structure and offer no real packing problem. Both VNS and ALNS performed at least competitive to previously applied methods, especially yielding good results in short time. ALNS is able to outperform VNS here. As second set we considered those introduced by Derigs et al. where two scenarios, one of delivering food and another one of delivering petrol, are modeled. Only the petrol scenario involves the $\mathcal{NP}$-hard CAP, at least in principle, but in all these instances we did never encounter a single CAP instance which could not be solved via the heuristic packing methods. The results are very encouraging, for nearly two third of all instances a new best known solution was found. As expected, the algorithms performed especially well on the petrol instances. Overall ALNS yields slightly better results than VNS. The potential of our extensions could, however, not be fully assessed yet. For this we modified the petrol type instances to show less uniform order demands than before. On this last set we could observe the benefits of our extensions when gradually adding them to the methods. Also tackling some CAPs with the exact approach, which would otherwise remain undecided, is of benefit. At first only CAPs when inserting several orders were solved, which was possible with only a very small computational effort, but yielded a notable improvement. In a second attempt we also solved those CAPs to optimality when inserting several orders, which again lead to an improvement. However, this was more to investigate the potential of it, as this was only possible with roughly twice the amount of runtime. This is because these cases occur for several orders of magnitude more often and hence the overall computational effort is significantly increasing (including the memory consumption of the hash map).

Potential Future Work

Meaningful future work could be to consider real-world scenarios where hard(er) packing problems involving compartments occur. In these cases our cascaded CAP solving approach and the problem-tailored neighborhoods would clearly be of advantage and allow an interesting evaluation. In the course of this also other packing solution archives could be tried, e.g. like the index structures we have already mentioned. Another future topic could be to investigate methods for determining (even) stronger lower bounds, in order to avoid some more cases of wasting time by handing an infeasible packing problem to the exact method. We also tried two ILP-based very large neighborhood search approaches (basically similar to those for the location-routing problem described in Section 6.4), but their performance with regard to runtime and/or improvement was not satisfying so far. However, a further consideration might still be worthwhile.

CHAPTER 8

CONCLUSIONS

I set out on this "journey" with the aim to thoroughly investigate and devise *hybrid methods* in order to come up with highly effective solution approaches for certain classes of $\mathcal{NP}$-hard combinatorial optimization problems. In general, such hybrids try to combine in various ways the strengths of two or more methods from possibly different streams. It was further intended to focus in particular on combining exact and (meta-)heuristic algorithms, especially exploiting the power of mathematical programming techniques, yielding so-called *matheuristics* (or *model-based metaheuristics*). Although we did not decide on the problems which would be tackled right from the start—as I was more interested in the methodical aspect—it eventually turned out that we dealt with problems that are not only interesting from an academic perspective but highly relevant in practical application areas, too. More specifically, one of them is from the bioinformatics domain, whereas all others arise in the field of transportation and are extensions of the capacitated vehicle routing problem motivated by important real-world aspects. So it happened that over the years I began to develop a particular interest in transportation problems, and in logistic problems in general.

In retrospect, we can actually confirm that hybrid methods, if appropriately conceived, implemented, and eventually applied, can be very powerful and one might even obtain leading solution approaches; something we achieved for most of the problems we considered.

In the following we will shortly mention the hybrid methods that were developed, following the use cases documented in Section 2.5. Note that several other extensions were proposed which we do not mention here again.

Finding Initial or Improved Solutions. In Chapter 3 we developed sequential and intertwined collaborative combinations for the consensus tree problem (CTP). For this we combined an evolutionary algorithm (EA) as well as a memetic algorithm with a variable neighborhood descent (VND) and a variable neighborhood search (VNS). The individual methods benefit from each other via exchanging whole solutions. Also for the CTP, in Section 3.10.4

an integer linear programming (ILP) approach, starting with a heuristically pruned set of variables, is enriched by an iterative rounding and repair procedure applied to each non-integer candidate solution in order to find a feasible tree. On the one hand, such a solution acts as new primal bound, but on the other hand variables that appear in this solution but are not present in the current model are added to it. In this way a heuristic column generation is realized. Next, a cooperative search is realized via several VNS instances running in an intertwined way in Section 4.5 for solving the periodic vehicle routing problem with time windows (PVRPTW). This *multiple VNS* (mVNS) puts emphasis on the so far best solution found within a major iteration by restarting the worst performing VNS instances with it. In this way, mVNS investigates multiple search trajectories from incumbent solutions, and from a global perspective it can be seen to adaptively allocate VNS instances to promising areas of the search space.

Multi-Stage Approaches. For the periodic vehicle routing problem in Chapter 5 we proposed a novel, smooth integration of the multilevel refinement strategy into a VNS, which could be applied to other metaheuristics as well. We further implemented a multilevel VND representing a more classical approach according to previous literature. Multilevel refinement enhanced the scope of the neighborhoods involved and thus also shaking and local improvement, and led to improved solutions and shorter runtimes in general, making the methods especially suited for large instances.

Strategic Guidance of a Method by Another. In Chapters 4–6 the concept of simulated annealing is tightly integrated into a VNS in order to better escape local optima, hence realizing a low-level integrative combination. In contrast, a high-level sequential collaborative combination of a column generation approach and an EA is proposed in Section 4.9.3 for the PVRPTW. Here the solution to the LP relaxation of a set covering model, i.e. on the one hand columns which were created and on the other hand the variable values, is successfully exploited in the subsequent EA.

Very Large Neighborhood Search (VLNS). ILP-based VLNS applied within a VNS were presented in Section 6.4 for the (periodic) location routing problem ((P)LRP). V_1 is handed a single solution and operates on a higher level via relocating whole routes to depots, considering all days at once in case of the PLRP, while V_3 deals with the (re-)location of customer sequences to insertion points in routes (of a single day each). Note that V_3 can exploit the information contained in several solutions provided by the VNS via extracting admissible customer sequences, thus omitting a possibly more costly generation.

Combination of *Very Large Neighborhood Search* and *Optimal Merging*. The search space of the VLNS mentioned right before is defined by a single solution. Although V_3 can use several solutions, the insertion points are still derived of a single solution only. Contrary, the approaches for the PVRPTW in Section 4.9.1 and 4.9.2 as well as that for the (P)LRP in Section 6.4.1 (V_2, being a more sophisticated variant of V_1) are primarily devised to directly exploit the information of several VNS solutions. For both problems they are based on a set covering ILP formulation, but instead of considering only attributes of several solutions, which would be done in case of optimal merging, dedicated problem parts might freely change and adapt to the information offered. This results in a potentially much larger search space and therefore can be considered a VLNS, too. In case of the PVRPTW the visit

combinations are free to change, in case of the LRP it is the placement of the facilities, and for the PLRP even both. In general, this approach can also be considered a purely metaheuristic column generation, as the set covering ILP model, being the master problem, is solely enriched by columns provided by VNS solutions.

Heuristically Solving Subproblems in Exact Approaches. In Section 4.7 we faced the $\mathcal{NP}$-hard elementary shortest path problem with resource constraints as pricing problem in a column generation approach for the PVRPTW. Therefore we applied several heuristic measures before switching to an exact algorithm to proof optimality. They included construction and improvement (meta-)heuristics as well as variants of the exact algorithm applied in a heuristic way. Later in Section 4.8 a heuristic, which is similar to VND, is involved in separating 2-path cuts for a branch-and-cut-and-price approach.

Exactly Solving Subproblems in Heuristic Approaches. Finally, in Chapter 7 the $\mathcal{NP}$-hard packing subproblem of the vehicle routing problem with compartments (VRPC) might be solved to optimality by different exact methods. Here constraint programming clearly turned out to be the method of choice. Of course, the exact method is only applied whenever a cascade of heuristics fails to find a solution.

We have shown that for all considered problems a skillfull hybridization of the developed exact and heuristic methods, or of several heuristics, led to a significant improvement in general. In fact, the exact add-ons for heuristics and vice versa, representing an integrative combination, gave in our cases almost always a considerable performance boost to the main (host) method. Thereby either heuristic components were able to notably reduce the required runtime or exact components could significantly increase the solution quality. Moreover, the collaborative combinations could clearly benefit from the diverse algorithms in use. However, it is clear that for hybrids where an exact solution method is involved, the size of the instances for which they are viable strongly depends on the performance of this exact method. For example, our devised hybrids for the PVRPTW could not be applied to the larger instances in a meaningful way, as opposed to the (P)LRP and VRPC, where we were able to tackle all instances.

In addition, the role of the individual methods or the single underlying method is not to be underestimated. In our case variants of variable neighborhood search were the most prominent metaheuristics applied, and for all but one problem a solution approach based on VNS was presented for the first time. The simple elegance of VNS offered a great flexibility when it came to extension as well as specialization, as neighborhood structures can be added like building blocks in order to assemble a powerful solution method. Especially meaningful problem-tailored neighborhood structures, which vary on the level/part of the problem they operate, contributed a lot to the overall success. Combined with an appropriate embedded local search component we always achieved a good balance of exploration and intensification. Note that in some cases, as mentioned above, the ability to escape local optima was improved by tightly integrating the concept of simulated annealing, i.e. accepting worse solutions with a certain probability.

In thorough comparisons to previous solution approaches we almost always achieved at least competitive results. In many cases they were even clearly better, hence obtaining currently

leading approaches. This is also documented by numerous new best known solutions obtained. However, the improvement was not only in solution quality, but often our methods also exhibited much better runtime behaviors and thus scalability to larger instances. As a consequence of this, already competitive results could often be obtained with considerably less runtime. Beside obvious methodical differences it is likely that implementation specifics such as appropriate data structures, efficient neighborhood evaluations, as well as incremental updates and copying also contributed to this (although it is, for better or worse, not common to report on such things).

Since the means to compare to other approaches are after all quite limited, we were all the more concerned with rigorously comparing our "baseline methods" to the subsequently enhanced hybrid methods whenever meaningful. Overall, it turned out that our hybrid methods almost always exhibited statistically significant better results, in some cases for whole instance sets. Although this could be at the expense of an also significant increase in runtime, this was not the case here, as we either strictly allowed the same runtime or only a moderate increase occurred otherwise.

Note that each of these hybrid variants has its strengths and weaknesses, which have been addressed in this work. Not surprisingly, we encountered none that clearly dominates all others and should be the preferred variant for each possible problem – also for hybrid methods there is "no free lunch". Nevertheless, our work provides additional guidelines concerning under which conditions which hybridization schemes can be promising. For one thing, our devised matheuristics not only seem promising in particular for other, possibly even richer variants of routing problems, but their concept can fairly easily be applied to other classes of combinatorial optimization problems as well. Especially the applied combination of very large neighborhood search and optimal merging is recommendable for problems exhibiting a similar structure.

Despite all their potential benefits, hybrid methods generally also have some drawbacks which one should be aware of: they have a higher complexity, they require more effort for design and implementation, to combine algorithms/concepts from different streams an appropriate knowledge of each individual stream is a prerequisite, and they are likely to be harder to tune. However, if one copes with these issues such hybridizations might give rise to promising solution approaches for many problems.

Since their advent hybrid solution approaches gained an ever increasing interest and popularity, which is clear when looking at the many success-stories, where we were also able to contribute some. Especially combining possibly complementary methods from different streams, e.g. matheuristics, which in some way exploit the mathematical model of a problem in conjunction with a heuristic, is very promising and likely to produce further effective solution approaches. Although hybridization might not be *the* holy grail of (combinatorial) optimization, it surely is worth of continuous research as well as application. Moreover, it will be exciting to see its future development – which I hope not only to observe but in some way to participate in, too.

BIBLIOGRAPHY

[1] Gecode: generic constraint development environment. `http://www.gecode.org`. Version 3.7.1 [Online; accessed February 11, 2012].

[2] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[3] E. N. Adams. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21(4):390–397, 1972.

[4] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from the lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

[5] 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(1-3):75–102, 2002.

[6] Z. Akca, R. T. Berger, and T. K. Ralphs. A branch-and-price algorithm for combined location and routing problems under capacity restrictions. In J. W. Chinneck et al., editors, *Operations Research and Cyber-Infrastructure*, volume 47 of *Operations Research/Computer Science Interfaces Series*, pages 309–330. Springer, 2009.

[7] J. Alegre, M. Laguna, and J. Pacheco. Optimizing the periodic pick-up of raw materials for a manufacturer of auto parts. *European Journal of Operational Research*, 179(3):736–746, 2007.

[8] B. L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001.

[9] A. A. Andreatta and C. C. Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8:429–447, 2002.

[10] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[11] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[12] R. Baldacci, E. Bartolini, A. Mingozzi, and A. Valetta. An exact algorithm for the period routing problem. *Operations Research*, 59(1):228–241, 2011.

[13] R. Barták. Theory and practice of constraint propagation. In J. Figwer, editor, *Proceedings of the Third Workshop on Constraint Programming in Decision and Control (CPDC2001)*, pages 7–14. Gliwice, Poland, 2001.

[14] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, USA, 1957.

[15] V. Berry and D. Bryant. Faster reliable phylogenetic analysis. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB'99)*, pages 59–68, New York, 1999. ACM Press.

[16] L. Bertazzi, G. Paletta, and M. G. Speranza. An improved heuristic for the period traveling salesman problem. *Computers & Operations Research*, 31(8):1215–1222, 2004.

[17] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[18] D. Bertsimas and R. Weismantel. *Optimization Over Integers*. Dynamic Ideas, 2005.

[19] C. Blum. Beam-aco - hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & OR*, 32:1565–1591, 2005.

[20] C. Blum and M. Blesa. Combining ant colony optimization with dynamic programming for solving the k-cardinality tree problem. In J. Cabestany, A. Prieto, and F. Sandoval, editors, *Computational Intelligence and Bioinspired Systems*, volume 3512 of *LNCS*, pages 219–241. Springer Berlin Heidelberg, 2005.

[21] C. Blum, M. J. Blesa Aguilera, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2008.

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

[23] C. Blum and A. Roli. Hybrid metaheuristic: An introduction. In Blum et al. [21], chapter 1, pages 1–30.

[24] M. Boccia, T. G. Crainic, A. Sforza, and C. Sterle. Location-routing models for designing a two-echelon freight distribution system. Technical Report CIRRELT-2011-06, Université de Montréal, 2011.

[25] N. Bouhmala. Combining local search with the multilevel paradigm for the traveling salesman problem. In C. Blum et al., editors, *Proc. 1st Intl. Workshop on Hybrid Metaheuristics*, pages 51–58, 2004.

[26] J. Bramel and D. Simchi-Levi. On the effectiveness of set covering formulations for the vehicle routing problem with time windows. *Operations Research*, 45:295–301, 1997.

[27] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[28] O. Bräysy, W. Dullaert, and M. Gendreau. Evolutionary algorithms for the vehicle routing problem with time windows. *Journal of Heuristics*, 10(6):587–611, 2004.

[29] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[30] D. Bryant. A classification of consensus methods for phylogenies. In M. Janowitz et al., editors, *Bioconsensus*, DIMACS, pages 163–184. AMS, 2003.

[31] A. Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33(10):2972–2990, 2006.

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

[33] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[34] R. K. Congram, C. N. Potts, and S. L. van de Velde. An iterated Dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

[35] C. Contardo, J.-F. Cordeau, and B. Gendron. A grasp + ilp-based metaheuristic for the capacitated location-routing problem. Technical Report CIRRELT-2011-52, Université de Montréal, 2011.

[36] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[37] W. Cook and J. L. Rich. A parallel cutting-plane algorithm for the vehicle routing problem with time windows. Technical Report TR-99-04, Rice University, 1999.

[38] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. The VRP with time windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, chapter 7, pages 157–193. SIAM, Philadelphia, 2002.

[39] J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany. New heuristics for the vehicle routing problem. In A. Langevin and D. Riopel, editors, *Logistics Systems: Design and Optimization*, pages 279–297. Springer US, 2005.

[40] J.-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

[41] J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.

[42] J.-F. Cordeau, G. Laporte, and A. Mercier. Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows. *Journal of the Operational Research Society*, 55(5):542–546, 2004.

[43] J.-F. Cordeau and M. Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050, 2012.

[44] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

[45] C. Cotta. On the application of evolutionary algorithms to the consensus tree problem. In J. Gottlieb and G. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3248 of *LNCS*, pages 58–67. Springer, 2005.

[46] C. Cotta. Scatter search with path relinking for phylogenetic inference. *European Journal of Operational Research*, 169(2):520–532, 2006.

[47] C. Cotta and P. Moscato. Inferring phylogenetic trees using evolutionary algorithms. In J. Merelo et al., editors, *Parallel Problem Solving From Nature VII*, volume 2439 of *LNCS*, pages 720–729. Springer, 2002.

[48] C. Cotta and P. Moscato. A memetic-aided approach to hierarchical clustering from distance matrices: Application to gene expression clustering and phylogeny. *BioSystems*, 72(1–2):75–97, 2003.

[49] C. Cotta and J. M. Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18(2):137–153, 2003.

[50] E. Danna and C. Le Pape. Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In Desaulniers et al. [61], chapter 4, pages 99–129.

[51] E. Danna, E. Rothberg, and C. Le Pape. Integrating mixed integer programming and local search: A case study on job-shop scheduling problems. In *Fifth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'2003)*, pages 65–79, 2003.

[52] G. B. Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

[53] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

[54] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[55] G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction*. Springer-Verlag New York, Inc., 1997.

[56] G. B. Dantzig and M. N. Thapa. *Linear Programming 2: Theory and Extensions*. Springer-Verlag New York, Inc., 2003.

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

[58] R. De Franceschi, M. Fischetti, and P. Toth. A new ILP-based refinement heuristic for vehicle routing problems. *Math. Program*, 105(2-3):471–499, 2006.

[59] J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In W. Porto et al., editors, *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 2317–2324. IEEE Press, 1999.

[60] U. Derigs, J. Gottlieb, J. Kalkoff, M. Piesche, F. Rothlauf, and U. Vogel. Vehicle routing with compartments: applications, modelling and heuristics. *OR Spectrum*, 33(4):885–914, 2011.

[61] G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors. *Column Generation*. Springer, 2005.

[62] G. Desaulniers, F. Lessard, and A. Hadjar. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404, 2008.

[63] J. Desrosiers and M. E. Lübbecke. A primer in column generation. In Desaulniers et al. [61], chapter 1, pages 1–32.

[64] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[65] K. Dörner and V. Schmid. Survey: Matheuristics for rich vehicle routing problems. In M. J. Blesa et al., editors, *Proceedings of Hybrid Metaheuristics – Seventh International Workshop, HM 2010, Vienna, Austria, October 1–2, 2010*, volume 6373 of *LNCS*, pages 206–221. Springer Berlin Heidelberg, 2010.

[66] S. Dreyfus. Richard bellman on the birth of dynamic programming. *Operations Research*, 50(1):48–51, 2002.

[67] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1–3):229–237, 1999.

[68] C. Duhamel, P. Lacomme, C. Prins, and C. Prodhon. A memetic approach for the capacitated location routing problem. In C. Prodhon et al., editors, *Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing*, Troyes, France, 2008.

[69] C. Duhamel, P. Lacomme, C. Prins, and C. Prodhon. A GRASP×ELS approach for the capacitated location-routing problem. *Computers & OR*, 37(11):1912–1923, 2010.

[70] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[71] M. El-Abd and M. Kamel. A taxonomy of cooperative search algorithms. In M. J. Blesa et al., editors, *Proceedings of Hybrid Metaheuristics – Second International Workshop, HM 2005, Barcelona, Spain, August 29–30, 2005*, volume 3636 of *LNCS*, pages 32–41. Springer, 2005.

[72] A. El Fallahi, C. Prins, and R. W. Calvo. A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem. *Computers & Operations Research*, 35(5):1725–1741, 2008.

[73] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, volume 2, pages 1186–1192, May 1992.

[74] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

[75] J. Felsenstein. PHYLIP (Phylogeny Inference Package) version 3.6, 2005. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle.

[76] T. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[77] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[78] M. Fischetti and A. Lodi. Local Branching. *Mathematical Programming, Series B*, 98:23–47, 2003.

[79] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345–, 1962.

[80] G. Fogel. Evolutionary computation for the inference of natural evolutionary histories. *IEEE Connections*, 3(1):11–14, 2005.

[81] L. R. Foulds and R. L. Graham. The steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

[82] P. Francis, K. Smilowitz, and M. Tzur. The period vehicle routing problem with service choice. *Transportation Science*, 40(4):439–454, 2006.

[83] P. M. Francis, K. R. Smilowitz, and M. Tzur. The period vehicle routing problem and its extensions. In B. Golden et al., editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 73–102. Springer, 2008.

[84] G. Fuellerer, K. F. Doerner, R. F. Hartl, and M. Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers & Operations Research*, 36(3):655–673, 2009.

[85] P. Galinier, A. Hertz, S. Paroz, and G. Pesant. Using local search to speed up filtering algorithms for some NP-hard constraints. *Annals of Operations Research*, 184:121–135, 2011.

[86] J. E. Gallardo, C. Cotta, and A. J. Fernández. Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. In J. Mira and J. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *LNCS*, pages 21–30. Springer, 2005.

[87] J. E. Gallardo, C. Cotta, and A. J. Fernández. On the hybridization of memetic algorithms with branch-and-bound techniques. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(1):77–83, 2007.

[88] F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, 8(3):375–388, 2002.

[89] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[90] M. Gendreau, G. Laporte, and F. Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, 31:347–358, 2004.

[91] M. Gendreau and J.-Y. Potvin, editors. *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer US, second edition, 2010.

[92] M. Gendreau, J.-Y. Potvin, O. Bräysy, G. Hasle, and A. Løkketangen. Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In Golden et al. [98], pages 143–169.

[93] I. P. Gent and T. Walsh. From approximate to optimal solutions: constructing pruning and propagation rules. In *Proceedings of the Fifteenth international joint conference on Artifical intelligence - Volume 2*, IJCAI'97, pages 1396–1401, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[94] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

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

[96] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.

[97] A. Goëffon, J.-M. Richer, and J.-K. Hao. Progressive tree neighborhood applied to the maximum parsimony problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(1):136–145, 2008.

[98] B. Golden et al., editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer US, 2008.

[99] O. Goldreich. *P, NP, and NP-Completeness: The Basics of Computational Complexity*. Cambridge University Press, 2010.

[100] C. Groër, B. L. Golden, and E. A. Wasil. A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing*, 23(2):315–330, 2011.

[101] M. Gruber and G. R. Raidl. (meta-)heuristic separation of jump cuts in a branch&cut approach for the bounded diameter minimum spanning tree problem. In V. Maniezzo et al., editors, *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*, chapter 8, pages 209–229. Springer, 2010.

[102] S. Gualandi and F. Malucelli. Constraint programming-based column generation. *4OR: A Quarterly Journal of Operations Research*, 7(2):113–137, 2009.

[103] P. Hansen, N. Mladenović, J. Brimberg, and J. A. Moreno Pérez. Variable neighborhood search. In Gendreau and Potvin [91], chapter 3, pages 61–86.

[104] V. C. Hemmelmayr, J.-F. Cordeau, and T. G. Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. Technical Report CIRRELT-2011-42, Université de Montréal, 2011.

[105] V. C. Hemmelmayr, K. F. Doerner, and R. F. Hartl. A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, 195(3):791–802, 2009.

[106] J. Holland. *Adaptation In Natural and Artificial Systems*. University of Michigan Press, 1975.

[107] S. Holmes. Phylogenies: An overview. In M. Halloran and S. Geisser, editors, *Statistics and Genetics*, pages 81–119. Springer, 1999.

[108] J. N. Hooker. *Integrated Methods for Optimization*. Springer, 2007.

[109] B. Hu, M. Leitner, and G. R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, 14(5):473–499, 2008.

[110] B. Hu and G. R. Raidl. Variable neighborhood descent with self-adaptive neighborhood-ordering. In C. Cotta, A. J. Fernandez, and J. E. Gallardo, editors, *Proceedings of the 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics*, Malaga, Spain, 2006.

[111] B. Hu and G. R. Raidl. Effective neighborhood structures for the generalized traveling salesman problem. In J. van Hemert and C. Cotta, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2008*, volume 4972 of *LNCS*, pages 36–47. Springer, 2008.

[112] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In Desaulniers et al. [61], chapter 2, pages 33–65.

[113] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[114] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

[115] M. Jünger, T. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors. *50 Years of Integer Programming 1958–2008*. Springer, 2009.

[116] B. Kallehauge. Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, 35(7):2307–2330, 2008.

[117] D. R. Karger. Random sampling in cut, flow and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

[118] D. R. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996.

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

[120] R. M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[121] L. Khachiyan. A polynomial algorithm in linear programming (english translation). *Soviet Mathematics Doklady*, 20:191–194, 1979.

[122] J. Kim and T. Warnow. Tutorial on phylogenetic tree estimation. In T. Lengauer et al., editors, *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology*, pages 196–205. AAAI Press, 1999.

[123] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[124] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities*, volume III, pages 159–175. Academic Press, New York, 1972.

[125] N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon, and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[126] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer Berlin Heidelberg, fifth edition, 2012.

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

[128] J. Larsen. *Parallelization of the Vehicle Routing Problem with Time Windows*. PhD thesis, Technical University of Denmark, 1999.

[129] M. Leitner, M. Ruthmair, and G. R. Raidl. On stabilized branch-and-price for constrained tree problems. Technical Report TR-186-1-11-01, Vienna University of Technology, 2011. (submitted to a journal).

[130] J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55:705–716, 2004.

[131] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[132] I. J. Lustig and J.-F. Puget. Program does not equal program: Constraint programming and its relationship to mathematical programming. *Interfaces*, 31(6):29–53, 2001.

[133] V. Maniezzo, T. Stützle, and S. Voß, editors. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, 2010.

[134] S. Martello and P. Toth. *Knapsack Algorithms: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[135] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete and Applied Mathematics*, 28(1):59–70, 1990.

[136] J. E. Mendoza, B. Castanier, C. Guéret, A. L. Medaglia, and N. Velasco. A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands. *Computers & Operations Research*, 37(11):1886–1898, 2010.

[137] J. E. Mendoza, B. Castanier, C. Guéret, A. L. Medaglia, and N. Velasco. Constructive heuristics for the multicompartment vehicle routing problem with stochastic demands. *Transportation Science*, 45(3):346–363, 2011.

[138] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[139] B. Meyer. Hybrids of constructive metaheuristics and constraint programming: A case study with aco. In Blum et al. [21], pages 151–183.

[140] M. Milano and P. Van Hentenryck, editors. *Hybrid Optimization: The Ten Years of CPAIOR*, volume 45 of *Springer Optimization and Its Applications*. Springer, 2010.

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

[142] A. Moilanen. Searching for most parsimonious trees with simulated evolutionary optimization. *Cladistics*, 15(1):39–50, 1999.

[143] J. A. Moreno-Pérez, P. Hansen, and N. Mladenović. Parallel variable neighborhood search. In E. Alba, editor, *Parallel Metaheuristics: A New Class of Algorithms*, chapter 11, pages 247–266. John Wiley & Sons, NJ, USA, 2005.

[144] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Publishers, 2003.

[145] P. Moscato and C. Cotta. A modern introduction to memetic algorithms. In Gendreau and Potvin [91], chapter 6, pages 141–183.

[146] M. Mourgaya and F. Vanderbeck. Column generation based heuristic for tactical planning in multi-period vehicle routing. *European Journal of Operational Research*, 183(3):1028–1041, 2007.

[147] L. Muyldermans and G. Pang. On the benefits of co-collection: Experiments with a multi-compartment vehicle routing algorithm. *European Journal of Operational Research*, 206(1):93–103, 2010.

[148] G. Nagy and S. Salhi. Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177(2):649–672, 2007.

[149] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, NY, USA, 1988.

[150] P. K. Nguyen, T. G. Crainic, and M. Toulouse. A hybrid genetic algorithm for the periodic vehicle routing problem with time windows. Technical Report CIRRELT-2011-25, Université de Montréal, 2011.

[151] A. G. Nikolaev and S. H. Jacobson. Simulated annealing. In Gendreau and Potvin [91], chapter 1, pages 1–39.

[152] T. E. O'Neil. Sub-exponential algorithms for 0/1 knapsack and bin packing. In H. R. Arabnia, G. A. Gravvanis, and A. M. G. Solo, editors, *Proceedings of the 2011 Intl. Conf. on Foundations of Computer Science*, pages 209–214. CSREA Press, 2011.

[153] I. H. Osman. *Metastrategy simulated annealing and tabu search for combinatorial optimization problems*. PhD thesis, The Management School, Imperial College of Science and Medicine, University of London, London, UK, 1991.

[154] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, NY, USA, 1998.

[155] C. A. Phillips and T. Warnow. The asymmetric median tree - A new model for building consensus trees. *Discrete Applied Mathematics*, 71(1-3):311–335, 1996.

[156] S. Pirkwieser. A Lagrangian decomposition approach combined with metaheuristics for the knapsack constrained maximum spanning tree problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Oct. 2006. Supervised by G. Raidl and J. Puchinger.

[157] S. Pirkwieser and G. R. Raidl. Finding consensus trees by evolutionary, variable neighborhood search, and hybrid algorithms. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 323–330, New York, NY, USA, 2008. ACM.

[158] S. Pirkwieser and G. R. Raidl. A variable neighborhood search for the periodic vehicle routing problem with time windows. In C. Prodhon et al., editors, *Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing*, Troyes, France, 2008.

[159] S. Pirkwieser and G. R. Raidl. Boosting a variable neighborhood search for the periodic vehicle routing problem with time windows by ILP techniques. In S. Voss and M. Caserta, editors, *Proceedings of the 8th Metaheuristic International Conference (MIC 2009)*, Hamburg, Germany, 13–16 July 2009.

[160] S. Pirkwieser and G. R. Raidl. A column generation approach for the periodic vehicle routing problem with time windows. In M. G. Scutellà et al., editors, *Proceedings of the International Network Optimization Conference 2009*, Pisa, Italy, 26–29 Apr. 2009.

[161] S. Pirkwieser and G. R. Raidl. Multiple variable neighborhood search enriched with ILP techniques for the periodic vehicle routing problem with time windows. In M. J. Blesa et al., editors, *Proceedings of Hybrid Metaheuristics – Sixth International Workshop, HM 2009, Udine, Italy, October 16–17, 2009*, volume 5818 of *LNCS*, pages 45–59. Springer, 2009.

[162] S. Pirkwieser and G. R. Raidl. Matheuristics for the periodic vehicle routing problem with time windows. In *Proceedings of Matheuristics 2010: third international workshop on model-based metaheuristics, Vienna, Austria, June 28–30*, 2010.

[163] S. Pirkwieser and G. R. Raidl. Multilevel variable neighborhood search for periodic routing problems. In P. Cowling and P. Merz, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2010*, volume 6022 of *LNCS*, pages 226–238. Springer, 2010.

[164] S. Pirkwieser and G. R. Raidl. Variable neighborhood search coupled with ILP-based very large neighborhood searches for the (periodic) location-routing problem. In M. J. Blesa et al., editors, *Proceedings of Hybrid Metaheuristics – Seventh International Workshop, HM 2010, Vienna, Austria, October 1–2, 2010*, volume 6373 of *LNCS*, pages 174–189. Springer Berlin Heidelberg, 2010.

[165] S. Pirkwieser, G. R. Raidl, and J. Gottlieb. Improved packing and routing of vehicles with compartments. In A. Quesada-Arencibia et al., editors, *Proceedings of EUROCAST 2011 – 13th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 6–11, 2011*, pages 302–304, 2011.

[166] S. Pirkwieser, G. R. Raidl, and J. Gottlieb. Tackling the loading aspect of the vehicle routing problem with compartments. In L. Di Gaspero et al., editors, *Proceedings of the 9th Metaheuristic International Conference (MIC 2011)*, pages 679–681, Udine, Italy, 25–28 July 2011.

[167] S. Pirkwieser, G. R. Raidl, and J. Gottlieb. Improved packing and routing of vehicles with compartments. In R. Moreno-Díaz et al., editors, *Computer Aided Systems Theory – EUROCAST 2011: 13th International Conference, Las Palmas de Gran Canaria, Spain, February 6–11, 2011, Revised Selected Papers, Part I*, volume 6927 of *LNCS*, pages 392–399. Springer, 2012.

[168] S. Pirkwieser, G. R. Raidl, and J. Puchinger. A Lagrangian decomposition/evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem. In C. Cotta and J. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 69–85. Springer Berlin Heidelberg, 2008.

[169] S. Pirkwieser, R. Ruiz-Torrubiano, and G. R. Raidl. Exact methods and metaheuristic approaches for deriving high quality fully resolved consensus trees. In J. Küng, K. Schneider, and R. Wagner, editors, *Proceedings of BIRD'08, 2nd International*

Conference on Bioinformatics Research and Development, Poster Presentations, volume 26 of *Schriftenreihe Informatik*, pages 115–124, Technical University of Vienna, Austria, 7–9 July 2008. Trauner Verlag.

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

[171] D. Pisinger and S. Ropke. Large neighborhood search. In Gendreau and Potvin [91], chapter 13, pages 399–419.

[172] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10:613–627, 2004.

[173] J.-Y. Potvin and J.-M. Rousseau. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 46:1433–1446, 1995.

[174] M. Prandtstetter and G. R. Raidl. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *European Journal of Operational Research*, 191(3):1004–1022, 2008.

[175] C. Prins, C. Prodhon, and R. W. Calvo. A memetic algorithm with population management (MA|PM) for the capacitated location-routing problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2006*, volume 3906 of *LNCS*, pages 183–194. Springer, 2006.

[176] C. Prins, C. Prodhon, and R. W. Calvo. Solving the capacitated location-routing problem by a GRASP complemented by a learning process and a path relinking. *4OR*, 4(3):221–238, 2006.

[177] C. Prins, C. Prodhon, A. Ruiz, P. Sorianoa, and R. W. Calvo. Solving the capacitated location-routing problem by a cooperative Lagrangean relaxation-granular tabu search heuristic. *Transportation Science*, 41(4):470–483, 2007.

[178] C. Prodhon. `http://prodhonc.free.fr/`. [Online; accessed October 14, 2011].

[179] C. Prodhon. A metaheuristic for the periodic location-routing problem. In J. Kalcsics and S. Nickel, editors, *Operations Research Proceedings 2007*, pages 159–164. Springer, 2007.

[180] C. Prodhon. An ELS×path relinking hybrid for the periodic location-routing problem. In M. J. Blesa, C. Blum, L. D. Gaspero, A. Roli, M. Sampels, and A. Schaerf, editors, *Proceedings of Hybrid Metaheuristics – Sixth International Workshop, HM 2009, Udine, Italy, October 16–17, 2009*, volume 5818 of *LNCS*, pages 15–29. Springer, 2009.

[181] C. Prodhon. A hybrid evolutionary algorithm for the periodic location-routing problem. *European Journal of Operational Research*, 210(2):204–212, 2011.

[182] C. Prodhon and C. Prins. A memetic algorithm with population management (MA|PM) for the periodic location-routing problem. In M. J. Blesa, C. Blum, C. Cotta, A. J. Fernández, J. E. Gallardo, A. Roli, and M. Sampels, editors, *Proceedings of Hybrid Metaheuristics – Fifth International Workshop, HM 2008, Málaga, Spain, October 8–9, 2008*, volume 5296 of *LNCS*, pages 43–57. Springer, 2008.

[183] J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira and J. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *LNCS*, pages 113–124. Springer Berlin Heidelberg, 2005.

[184] J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183(3):1304–1327, 2007.

[185] J. Puchinger and G. R. Raidl. Bringing order into the neighborhoods: Relaxation guided variable neighborhood search. *Journal of Heuristics*, 14(5):457–472, 2008.

[186] J. Puchinger, G. R. Raidl, and U. Pferschy. The core concept for the multidimensional knapsack problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2006*, volume 3906 of *LNCS*, pages 195–208. Springer, 2006.

[187] J. Puchinger, G. R. Raidl, and S. Pirkwieser. MetaBoosting: Enhancing integer programming techniques by metaheuristics. In V. Maniezzo et al., editors, *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*, chapter 3, pages 71–102. Springer, 2010.

[188] G. R. Raidl. A unified view on hybrid metaheuristics. In F. Almeida et al., editors, *Proceedings of Hybrid Metaheuristics – Third International Workshop, HM 2006, Gran Canaria, Spain, October 13–14, 2006*, volume 4030 of *LNCS*, pages 1–12. Springer, 2006.

[189] G. R. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In Blum et al. [21], chapter 2, pages 31–62.

[190] G. R. Raidl, J. Puchinger, and C. Blum. Metaheuristic hybrids. In Gendreau and Potvin [91], chapter 16, pages 469–496.

[191] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer, 2003.

[192] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In Gendreau and Potvin [91], chapter 10, pages 283–319.

[193] M. Resende, C. Ribeiro, F. Glover, and R. Martí. Scatter search and path-relinking: Fundamentals, advances, and applications. In Gendreau and Potvin [91], chapter 4, pages 87–107.

[194] C. C. Ribeiro and D. S. Vianna. A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. *International Transactions in Operational Research*, 12:325–338, 2005.

[195] Y. Richter, A. Freund, and Y. Naveh. Generalizing alldifferent: The somedifferent constraint. In F. Benhamou, editor, *Principles and Practice of Constraint Programming – CP 2006*, volume 4204 of *LNCS*, pages 468–483. Springer Berlin Heidelberg, 2006.

[196] S. Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3:92–94, 2006.

[197] D. Rodney, A. Soper, and C. Walshaw. Multilevel refinement strategies for the capacity vehicle routing problem. *International Journal of Information Technology & Intelligent Computing*, 2(3), 2007.

[198] S. Ropke and J.-F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[199] 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.

[200] F. Rossi, P. Beek, and T. Walsh, editors. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[201] E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

[202] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer Berlin Heidelberg, second edition, 2006.

[203] L.-M. Rousseau, M. Gendreau, and G. Pesant. Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, 8(1):43–58, 2002.

[204] S. Salhi and G. K. Rand. The effect of ignoring routes when locating depots. *European Journal of Operational Research*, 39(2):150 – 156, 1989.

[205] M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4:146–154, 1992.

[206] V. Schmid, K. F. Doerner, R. F. Hartl, M. W. P. Savelsbergh, and W. Stoecher. A hybrid solution approach for ready-mixed concrete delivery. *Transportation Science*, 43(1):70–85, 2009.

[207] C. Schulte, G. Tack, and M. Z. Lagerkvist. Modeling and programming with gecode. `http://www.gecode.org/doc/3.7.1/MPG.pdf`. Version 3.7.1 [Online; accessed February 11, 2012].

[208] M. Schwengerer, S. Pirkwieser, and G. R. Raidl. A variable neighborhood search approach for the two-echelon location-routing problem. In J. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2012*, volume 7245 of *LNCS*, pages 13–24. Springer, 2012.

[209] P. Shaw. A constraint for bin packing. In M. Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *LNCS*, pages 648–662. Springer, 2004.

[210] L. Sheneman and J. A. Foster. Estimating the destructiveness of crossover on binary tree representations. In *Proceedings of the 8th Conference on Genetic and Evolutionary Computation*, pages 1427–1428. ACM Press, 2006.

[211] M. Sipser. The history and status of the P versus NP question. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, STOC'92, pages 603–618, New York, NY, USA, 1992. ACM Press.

[212] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, second edition, 2005.

[213] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958.

[214] C. Solnon. *Ant Colony Optimization and Constraint Programming*. Wiley, 2010.

[215] M. M. Solomon. VRPTW instances. `http://web.cba.neu.edu/~msolomon/problems.htm`. [Online; accessed September 12, 2011].

[216] Y. S. Song. On the combinatorics of rooted binary phylogenetic trees. *Annals of Combinatorics*, 7:365–379, 2003.

[217] J. Strodl, K. Doerner, F. Tricoire, and R. Hartl. On index structures in hybrid metaheuristics for routing problems with hard feasibility checks: An application to the 2-dimensional loading vehicle routing problem. In M. J. Blesa et al., editors, *Proceedings of Hybrid Metaheuristics – Seventh International Workshop, HM 2010, Vienna, Austria, October 1–2, 2010*, volume 6373 of *LNCS*, pages 160–173. Springer Berlin Heidelberg, 2010.

[218] E.-G. Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, Hoboken, New Jersey, USA, 2009.

[219] S. Talukdar, L. Baeretzen, A. Gove, and P. de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4:295–321, 1998.

[220] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, Philadelphia, 2002.

[221] Tree Of Life Web Project (ToL). `http://tolweb.org/tree/phylogeny.html`. [Online; accessed July 16, 2010].

[222] TreeBASE: A Database of Phylogenetic Knowledge. `http://www.treebase.org`. [Online; accessed July 19, 2010].

[223] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*, volume 114 of *International Series in Operations Research & Management Science*. Springer, third edition, 2008.

[224] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[225] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[226] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. Technical Report CIRRELT-2011-05, Université de Montréal, 2011.

[227] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time windows. Technical Report CIRRELT-2011-61, Université de Montréal, 2011.

[228] D. Vigo. VRPLIB: A Vehicle Routing Problem LIBrary. `http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html`. [Online; accessed February 9, 2012].

[229] C. Walshaw. A multilevel approach to the travelling salesman problem. *Operations Research*, 50(5):862–877, 2002.

[230] C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research*, 131:325–372, 2004.

[231] C. Walshaw. Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In C. Blum et al., editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *SCI*, pages 261–289. Springer, 2008.

[232] J. T. L. Wang, H. Shan, D. Shasha, and W. H. Piel. TreeRank: a similarity measure for nearest neighbor searching in phylogenetic databases. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, pages 171–180. IEEE Press, 2003.

[233] T. Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9:11–12, 1962.

[234] J. Weyer-Menkhoff, C. Devauchelle, A. Grossmann, and S. Grünewald. Integer linear programming as a tool for constructing trees from quartet data. *Computational Biology and Chemistry*, 29(3):196–203, 2005.

[235] D. Weyland. A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology. *International Journal of Applied Metaheuristic Computing*, 1(2):50–60, 2010.

[236] Wikispecies: free species directory. `http://species.wikimedia.org/wiki/Main_Page`. [Online; accessed July 16, 2010].

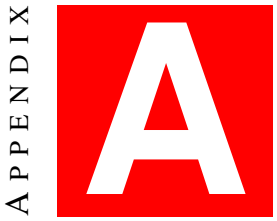
[237] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[238] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[239] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, NY, USA, 1998.

[240] B. Y. Wu. Constructing the maximum consensus tree from rooted triples. *Journal of Combinatorial Optimization*, 8(1):29–39, 2004.

[241] V. F. Yu, S.-W. Lin, W. Lee, and C.-J. Ting. A simulated annealing heuristic for the capacitated location routing problem. *Computers & Industrial Engineering*, 58:288–299, 2010.



SUPPLEMENTARY MATERIAL

A.1 Conference Poster on Consensus Tree Problem

In Figure A.1 we show our poster presented at the *2nd International Conference on Bioinformatics Research and Development* in 2008 (BIRD'08), accompanying our conference contribution [169].

A.2 Best Found Solution Values on VRPC Instances of Derigs et al.

Here we report on the costs of the best found solutions considering all test runs reported in Section 7.6.2 as well as a few preliminary test runs where also a CPU-time limit of 10 minutes was set. The instance names hold the characteristics, where we directly took the file name: vrpc followed by 'f' (food) or 'p' (petrol), then 'n' followed by the number of customers, next 'p' followed by the number of products, 'k' followed by an encoded vehicle capacity (06 means 600, whereas 3 means 3000), optionally 'c' denoting a clustered customers, as well as finally giving the maximal order demand, yielding the scheme: vrpc_[f|p]_n$\#customers$_p$\#products$_k$vehicleCapacity$_[c_]$maximalOrderDemand$.
The results for instances of type food and petrol are given in Table A.1 and A.2, respectively, stating the costs of the previous best known solution (prev. BKS), the costs of our best found solution, as well as the resulting percentage gap (i.e. $(bestFound - prevBKS)/prevBKS * 100\%$). For the 75 instances of type food we obtain 56 times (74.7%) an improved solution and 19 times (25.3%) a solution having equal costs, the overall gap is -0.498%. Investigating the performance on the 125 petrol type instances, we obtain an improved solution for 105 times (84.0%) while a solution having equal costs for 20 times (16.0%), yielding an

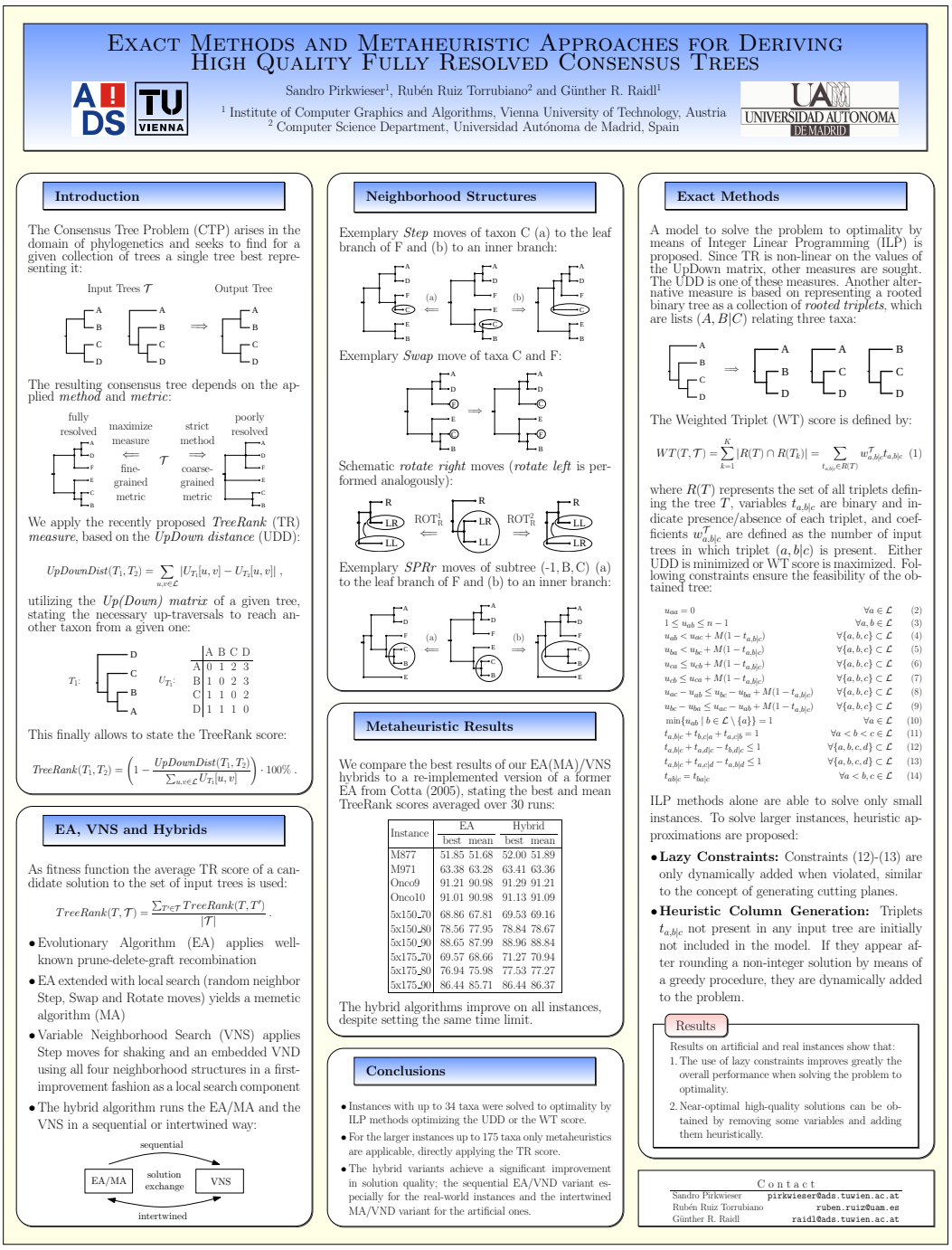


Figure A.1: Poster presented at the *2nd International Conference on Bioinformatics Research and Development* in 2008 (BIRD'08).

overall gap of -0.988%. So for both instance types the best found solutions where at least of the same quality, mostly even better. Especially in Table A.2 we can observe a larger improvement with increasing instance size in general, culminating in -7.738% for instance `vrpc_p_n200_p3_k9_c_900`.

Table A.1: Comparison of costs of previous best known solutions to our best found solutions on **food type** instances of Derigs et al. [60].

Instance	prev. BKS	best found	%-gap
vrpc_f_n10_p2_k06_150	11456.73	11456.73	0.000
vrpc_f_n10_p2_k06_c_150	8934.37	8905.86	-0.319
vrpc_f_n10_p3_k06_150	14197.32	14197.32	0.000
vrpc_f_n10_p3_k06_300	14231.80	14231.80	0.000
vrpc_f_n25_p2_k06_150	32825.67	32333.69	-1.499
vrpc_f_n25_p2_k06_c_150	21471.16	21212.50	-1.205
vrpc_f_n25_p2_k08_400	27039.70	27039.70	0.000
vrpc_f_n25_p3_k06_150	33524.81	33248.39	-0.825
vrpc_f_n25_p3_k06_c_150	19158.18	19070.64	-0.457
vrpc_f_n25_p3_k08_200	25293.84	25293.84	0.000
vrpc_f_n25_p3_k08_c_200	12837.50	12837.50	0.000
vrpc_f_n25_p2_k06_300	32950.05	32950.05	0.000
vrpc_f_n25_p2_k06_c_300	22015.68	22015.68	0.000
vrpc_f_n25_p2_k08_c_400	19278.13	19278.13	0.000
vrpc_f_n25_p3_k06_300	33404.46	33288.80	-0.346
vrpc_f_n25_p3_k06_c_300	19085.20	19077.71	-0.039
vrpc_f_n25_p3_k08_400	26485.08	26485.08	0.000
vrpc_f_n25_p3_k08_c_400	13595.43	13531.28	-0.472
vrpc_f_n50_p2_k06_150	68625.52	67347.28	-1.863
vrpc_f_n50_p2_k08_200	51700.25	51698.40	-0.004
vrpc_f_n50_p2_k3_750	15389.21	15389.21	0.000
vrpc_f_n50_p3_k06_300	69145.39	68532.43	-0.886
vrpc_f_n50_p3_k08_400	53896.39	53203.32	-1.286
vrpc_f_n50_p3_k3_c_750	10782.88	10782.88	0.000
vrpc_f_n50_p2_k06_300	69113.32	69037.63	-0.110
vrpc_f_n50_p2_k08_400	51678.11	51655.78	-0.043
vrpc_f_n50_p2_k3_c_750	7408.67	7408.67	0.000
vrpc_f_n50_p3_k06_c_150	41556.88	40817.69	-1.779
vrpc_f_n50_p3_k08_c_200	30674.70	30616.86	-0.189
vrpc_f_n50_p3_k9_4500	8469.54	8469.54	0.000
vrpc_f_n50_p2_k06_c_150	50188.20	48917.75	-2.531
vrpc_f_n50_p2_k08_c_200	39048.54	38971.99	-0.196
vrpc_f_n50_p2_k9_c_4500	3615.55	3615.55	0.000
vrpc_f_n50_p3_k06_c_300	41132.19	40866.95	-0.645
vrpc_f_n50_p3_k08_c_400	32283.54	31873.92	-1.269
vrpc_f_n50_p2_k06_c_300	50923.35	50833.40	-0.177

continued on next page

Table A.1 – continued from previous page

Instance	prev. BKS	best found	%-gap
vrpc_f_n50_p2_k08_c_400	39029.48	38971.99	-0.147
vrpc_f_n50_p3_k06_150	69575.34	68623.86	-1.368
vrpc_f_n50_p3_k08_200	51454.34	51286.57	-0.326
vrpc_f_n50_p3_k3_750	16531.02	16531.02	0.000
vrpc_f_n100_p2_k06_150	141560.56	140593.09	-0.683
vrpc_f_n100_p2_k08_200	103565.03	103036.14	-0.511
vrpc_f_n100_p2_k3_750	29676.06	29676.06	0.000
vrpc_f_n100_p3_k06_150	139297.91	136632.99	-1.913
vrpc_f_n100_p3_k08_200	92274.76	91459.81	-0.883
vrpc_f_n100_p3_k3_750	28181.98	28179.02	-0.010
vrpc_f_n100_p2_k06_300	142506.12	142082.23	-0.297
vrpc_f_n100_p2_k08_400	103534.48	103030.03	-0.487
vrpc_f_n100_p2_k3_c_750	18366.51	18354.27	-0.067
vrpc_f_n100_p3_k06_300	138137.05	135863.90	-1.646
vrpc_f_n100_p3_k08_400	96844.83	96340.95	-0.520
vrpc_f_n100_p3_k3_c_750	19145.10	19141.43	-0.019
vrpc_f_n100_p2_k06_c_150	76540.90	75668.96	-1.139
vrpc_f_n100_p2_k08_c_200	75006.44	74621.86	-0.513
vrpc_f_n100_p2_k9_4500	13076.57	13064.65	-0.091
vrpc_f_n100_p3_k06_c_150	87838.23	86197.75	-1.868
vrpc_f_n100_p3_k08_c_200	68355.19	67904.66	-0.659
vrpc_f_n100_p3_k9_4500	13095.44	13095.44	0.000
vrpc_f_n100_p2_k06_c_300	77268.61	76921.93	-0.449
vrpc_f_n100_p2_k08_c_400	75034.32	74624.65	-0.546
vrpc_f_n100_p2_k9_c_4500	6763.30	6757.10	-0.092
vrpc_f_n100_p3_k06_c_300	87615.28	85711.27	-2.173
vrpc_f_n100_p3_k08_c_400	71929.07	71429.02	-0.695
vrpc_f_n100_p3_k9_c_4500	7386.82	7386.82	0.000
vrpc_f_n200_p2_k06_300	537151.14	536911.78	-0.045
vrpc_f_n200_p2_k3_750	112659.90	112551.11	-0.097
vrpc_f_n200_p2_k9_4500	47773.07	47454.79	-0.666
vrpc_f_n200_p3_k08_400	439891.72	437381.48	-0.571
vrpc_f_n200_p3_k3_c_750	94515.40	94338.87	-0.187
vrpc_f_n200_p3_k9_c_4500	37707.03	37464.49	-0.643
vrpc_f_n200_p2_k08_400	399830.11	397508.69	-0.581
vrpc_f_n200_p2_k3_c_750	88506.35	88255.39	-0.284
vrpc_f_n200_p2_k9_c_4500	39753.17	39653.74	-0.250
vrpc_f_n200_p3_k3_750	112636.55	112395.97	-0.214
vrpc_f_n200_p3_k9_4500	46188.89	45939.85	-0.539

Table A.2: Comparison of costs of previous best known solutions to our best found solutions on **petrol type** instances of Derigs et al. [60].

Instance	prev. BKS	new BKS	%-gap
vrpc_p_n10_p2_k06_60	11299.13	11299.13	0.000
vrpc_p_n10_p2_k08_80	13717.15	13717.15	0.000
vrpc_p_n10_p3_k06_60	11502.25	11502.25	0.000
vrpc_p_n10_p3_k08_160	13111.39	13111.39	0.000
vrpc_p_n10_p3_k08_c_160	7034.47	7034.47	0.000
vrpc_p_n10_p3_k1_100	10008.83	10008.83	0.000
vrpc_p_n10_p3_k1_c_100	2968.06	2968.06	0.000
vrpc_p_n10_p2_k06_c_60	9298.64	9298.64	0.000
vrpc_p_n10_p2_k08_c_80	3726.27	3726.27	0.000
vrpc_p_n10_p3_k06_c_60	5558.19	5557.86	-0.006
vrpc_p_n10_p3_k08_80	12127.13	12127.13	0.000
vrpc_p_n10_p3_k08_c_80	6619.40	6619.09	-0.005
vrpc_p_n10_p3_k1_200	10021.58	10021.58	0.000
vrpc_p_n10_p3_k1_c_200	2968.59	2968.59	0.000
vrpc_p_n25_p2_k06_120	34239.36	34158.85	-0.235
vrpc_p_n25_p2_k08_80	26877.06	26701.27	-0.654
vrpc_p_n25_p2_k1_c_100	16655.57	16576.10	-0.477
vrpc_p_n25_p3_k06_120	34869.26	34864.43	-0.014
vrpc_p_n25_p3_k08_80	26681.67	26227.52	-1.702
vrpc_p_n25_p3_k1_c_100	14342.57	14211.84	-0.911
vrpc_p_n25_p3_k3_c_600	5810.59	5810.59	0.000
vrpc_p_n25_p2_k06_60	34161.05	34138.55	-0.066
vrpc_p_n25_p2_k08_c_160	18259.73	18241.79	-0.098
vrpc_p_n25_p2_k1_c_200	16993.28	16991.37	-0.011
vrpc_p_n25_p3_k06_60	34219.86	33530.50	-2.015
vrpc_p_n25_p3_k08_c_160	12584.29	12542.30	-0.334
vrpc_p_n25_p3_k1_c_200	14267.07	14211.60	-0.389
vrpc_p_n25_p3_k9_900	5602.74	5602.74	0.000
vrpc_p_n25_p2_k06_c_120	19124.78	19124.03	-0.004
vrpc_p_n25_p2_k08_c_80	18358.89	18251.55	-0.585
vrpc_p_n25_p2_k3_300	10826.72	10807.92	-0.174
vrpc_p_n25_p3_k06_c_120	26430.79	26430.79	0.000
vrpc_p_n25_p3_k08_c_80	12212.12	11918.06	-2.408
vrpc_p_n25_p3_k3_300	10540.55	10524.77	-0.150
vrpc_p_n25_p2_k06_c_60	19145.98	19115.13	-0.161
vrpc_p_n25_p2_k1_100	21301.84	21240.02	-0.290
vrpc_p_n25_p2_k3_c_300	6264.54	6264.21	-0.005
vrpc_p_n25_p3_k06_c_60	25423.34	24982.95	-1.732
vrpc_p_n25_p3_k1_100	21536.35	21402.91	-0.620
vrpc_p_n25_p3_k3_600	10578.49	10578.49	0.000
vrpc_p_n25_p2_k08_160	26701.33	26701.27	-0.000
vrpc_p_n25_p2_k1_200	21895.59	21892.80	-0.013

continued on next page

Table A.2 – continued from previous page

Instance	prev. BKS	new BKS	%-gap
vrpc_p_n25_p2_k3_c_600	6979.93	6979.93	0.000
vrpc_p_n25_p3_k08_160	27472.19	27414.74	-0.209
vrpc_p_n25_p3_k1_200	21539.70	21329.76	-0.975
vrpc_p_n25_p3_k3_c_300	5799.42	5790.32	-0.157
vrpc_p_n50_p2_k06_120	69543.35	69077.11	-0.670
vrpc_p_n50_p2_k08_c_160	33417.74	33206.55	-0.632
vrpc_p_n50_p2_k3_300	17639.70	17537.85	-0.577
vrpc_p_n50_p3_k06_120	59296.26	59251.75	-0.075
vrpc_p_n50_p3_k08_c_160	31193.75	30773.49	-1.347
vrpc_p_n50_p3_k3_300	19123.75	18942.81	-0.946
vrpc_p_n50_p2_k06_60	69703.29	69285.69	-0.599
vrpc_p_n50_p2_k08_c_80	33941.72	33290.48	-1.919
vrpc_p_n50_p2_k3_600	18466.70	18464.95	-0.009
vrpc_p_n50_p3_k06_60	58931.43	58049.52	-1.496
vrpc_p_n50_p3_k08_c_80	30540.11	29899.62	-2.097
vrpc_p_n50_p3_k3_600	19050.70	18950.91	-0.524
vrpc_p_n50_p2_k06_c_120	38264.58	37980.87	-0.741
vrpc_p_n50_p2_k1_100	44110.75	43640.29	-1.067
vrpc_p_n50_p2_k3_c_300	9324.11	9262.55	-0.660
vrpc_p_n50_p3_k06_c_120	43166.79	43077.55	-0.207
vrpc_p_n50_p3_k1_100	42593.88	42157.15	-1.025
vrpc_p_n50_p3_k3_c_300	12214.62	12090.77	-1.014
vrpc_p_n50_p2_k06_c_60	38453.62	38066.04	-1.008
vrpc_p_n50_p2_k1_200	44718.46	44579.50	-0.311
vrpc_p_n50_p2_k3_c_600	9654.59	9654.59	0.000
vrpc_p_n50_p3_k06_c_60	42670.99	41698.75	-2.278
vrpc_p_n50_p3_k1_200	42403.02	41651.78	-1.772
vrpc_p_n50_p3_k3_c_600	12095.53	12030.97	-0.534
vrpc_p_n50_p2_k08_160	49324.23	48820.47	-1.021
vrpc_p_n50_p2_k1_c_100	27986.95	27497.60	-1.748
vrpc_p_n50_p2_k9_900	9094.30	9094.30	0.000
vrpc_p_n50_p3_k08_160	54581.30	53985.31	-1.092
vrpc_p_n50_p3_k1_c_100	31403.05	30989.43	-1.317
vrpc_p_n50_p3_k9_900	9297.92	9211.97	-0.924
vrpc_p_n50_p2_k08_80	50497.70	49170.04	-2.629
vrpc_p_n50_p2_k1_c_200	28529.33	28380.61	-0.521
vrpc_p_n50_p2_k9_c_900	4288.61	4288.61	0.000
vrpc_p_n50_p3_k08_80	53846.58	52751.20	-2.034
vrpc_p_n50_p3_k1_c_200	31039.89	30618.72	-1.357
vrpc_p_n50_p3_k9_c_900	5931.20	5929.39	-0.031
vrpc_p_n100_p2_k06_120	136367.37	135442.97	-0.678
vrpc_p_n100_p2_k08_c_80	62390.24	60935.03	-2.332
vrpc_p_n100_p2_k3_300	34071.60	33489.13	-1.710
vrpc_p_n100_p2_k9_c_900	9627.42	9598.55	-0.300
vrpc_p_n100_p3_k08_c_80	58195.58	57515.23	-1.169

continued on next page

Table A.2 – continued from previous page

Instance	prev. BKS	new BKS	%-gap
vrpc_p_n100_p3_k3_300	35346.80	34605.59	-2.097
vrpc_p_n100_p3_k9_c_900	11759.74	11422.03	-2.872
vrpc_p_n100_p2_k06_c_120	78615.94	77916.09	-0.890
vrpc_p_n100_p2_k1_100	89671.50	88177.68	-1.666
vrpc_p_n100_p2_k3_600	35215.70	35153.94	-0.175
vrpc_p_n100_p3_k06_120	142984.55	142552.32	-0.302
vrpc_p_n100_p3_k1_100	87827.08	87627.10	-0.228
vrpc_p_n100_p3_k3_600	34868.96	34073.00	-2.283
vrpc_p_n100_p2_k08_160	111130.81	109941.72	-1.070
vrpc_p_n100_p2_k1_200	90946.34	89992.50	-1.049
vrpc_p_n100_p2_k3_c_300	21200.61	20769.87	-2.032
vrpc_p_n100_p3_k06_c_120	85658.52	85417.49	-0.281
vrpc_p_n100_p3_k1_200	86893.43	85282.74	-1.854
vrpc_p_n100_p3_k3_c_300	23623.33	23288.19	-1.419
vrpc_p_n100_p2_k08_80	113599.58	111774.99	-1.606
vrpc_p_n100_p2_k1_c_100	49877.31	48597.03	-2.567
vrpc_p_n100_p2_k3_c_600	22055.41	22023.80	-0.143
vrpc_p_n100_p3_k08_160	110606.77	109181.52	-1.289
vrpc_p_n100_p3_k1_c_100	56518.88	55313.72	-2.132
vrpc_p_n100_p3_k3_c_600	23314.19	22818.67	-2.125
vrpc_p_n100_p2_k08_c_160	60498.25	59876.71	-1.027
vrpc_p_n100_p2_k1_c_200	50678.36	50269.36	-0.807
vrpc_p_n100_p2_k9_900	15913.75	15900.42	-0.084
vrpc_p_n100_p3_k08_c_160	59037.40	58466.41	-0.967
vrpc_p_n100_p3_k1_c_200	55438.01	54588.56	-1.532
vrpc_p_n100_p3_k9_900	17578.58	17322.81	-1.455
vrpc_p_n200_p2_k3_300	136456.65	132074.12	-3.212
vrpc_p_n200_p2_k3_c_300	104902.77	101592.71	-3.155
vrpc_p_n200_p2_k9_900	57033.68	56283.36	-1.316
vrpc_p_n200_p3_k3_300	132669.63	128124.07	-3.426
vrpc_p_n200_p3_k3_c_300	116198.19	114159.88	-1.754
vrpc_p_n200_p3_k9_900	62121.91	59259.26	-4.608
vrpc_p_n200_p2_k3_600	139285.23	138384.29	-0.647
vrpc_p_n200_p2_k3_c_600	107486.95	106889.54	-0.556
vrpc_p_n200_p2_k9_c_900	36935.56	36357.12	-1.566
vrpc_p_n200_p3_k3_600	127928.67	123976.62	-3.089
vrpc_p_n200_p3_k3_c_600	115811.24	111464.37	-3.753
vrpc_p_n200_p3_k9_c_900	47562.86	43882.29	-7.738

APPENDIX B

CURRICULUM VITAE

Personal Information

Name	Sandro Pirkwieser
Date of birth	December 11th, 1979
Place of birth	Innsbruck, Austria
Nationality	Austria
Resident	Vienna, Austria
Family status	Married to Marlene Pirkwieser
Children	Johanna (May 6th, 2010)
Languages	German (native), English (fluent)

Education

since 11/2006	Doctorate (PhD) studies at the Vienna University of Technology under supervision of Günther Raidl in the field of algorithms and combinatorial optimization, mainly considering hybridizations of exact and heuristic methods (so-called *matheuristics*), in particular for special vehicle routing problems.
04/2005 – 10/2006	Computer science studies in the master program *Intelligent Systems* at the Vienna University of Technology with graduation to *Diplom-Ingenieur* (comparable to MSc). Master's thesis: "A Lagrangian decomposition approach combined with metaheuristics for the knapsack constrained maximum spanning tree problem". *Passed with distinction.*

10/2002 – 03/2005	Computer science studies in the bachelor program *Software & Information Engineering* at the Vienna University of Technology with graduation to *Bakkalaureus technicae* (comparable to BSc). *Passed with distinction.*
10/2001 – 06/2002	Computer science bachelor studies at the University of Innsbruck.
09/1994 – 06/1999	Higher technical school for electrical engineering with training program *Industrial Electronics and Power Engineering*, in Innsbruck, Austria. *Passed with merits.*

Work Experience

Academic

03/2010 – 10/2011	Teaching assistant, Algorithms and Data Structures Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria.
01/2008 – 02/2010	Employed in the FWF project *Matheuristics: Hybrid Algorithms for Transportation Problems* (P20342-N13) where we developed successful hybrid methods for solving vehicle routing problems with multiple visits. This was a joint project with partners from the University of Vienna: Karl F. Dörner (project leader) from the Department of Business Administration and Walter J. Gutjahr from the Department of Statistics and Decision Support Systems.
11/2006 – 03/2008	Doing research in the area of bioinformatics within the project *Hybridizing Branch-and-Bound with Metaheuristics for Solving Tree-Structured Combinatorial Optimization Problems* (project number 13/2006) supported by the Austrian exchange service (ÖAD) within "Acciones Integradas 2006-2007", cooperating with Carlos Cotta and colleagues from the University of Málaga.
11/2006 – 10/2011	Research assistant, Algorithms and Data Structures Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria.
10/2003 – 01/2006	Student assistant, Knowledge Based Systems Group, Institute of Information Systems, Vienna University of Technology, Austria.

Other

since 12/2010	Co-founder (together with my colleague Andreas Chwatal) and CTO of Destion – IT Consulting OG, Vienna, Austria.

07/2002 – 08/2002	Employed as intern, Swarovski Optik, Swarovski Group, Wattens, Austria.
10/2000 – 09/2001	Fully employed, Datacon Semiconductor Ges.m.b.H, Radfeld, Austria.
07/1999 – 03/2000	Military service as driver in Lienz, Austria.

Other Scientific Activities

2012	Reviewer for *Journal of Heuristics*
2011	Reviewer for *Transportation Science*
2011	Reviewer for *Central European Journal of Operations Research*
2011	Reviewer for Track *Genetic Algorithms* at GECCO 2011 – 13th Genetic and Evolutionary Computation Conference
2010	Reviewer for Workshop *Heuristic Problem Solving* at Eurocast 2011 – 13th International Conference on Computer Aided Systems Theory
2010	Local organizing committee member at the HM 2010 – 7th International Workshop on Hybrid Metaheuristics, Vienna, Austria.
2010	Reviewer for WIND 2010 – The Third International Workshop on Information Network Design
2010	Reviewer for Track *Genetic Algorithms* at GECCO 2010 – 12th Genetic and Evolutionary Computation Conference
2009	Reviewer for Track *Genetic Algorithms* at GECCO 2009 – 11th Genetic and Evolutionary Computation Conference
2008	Reviewer for Workshop *Heuristic Problem Solving* at Eurocast 2009 – 12th International Conference on Computer Aided Systems Theory

Awards and OR/Optimization Contests

02/2012	Currently participating in the *ROADEF/EURO Challenge 2012: Machine Reassignment* together with Roman Steiner. We are among the 30 teams out of 82 which passed the qualification phase in February 2012 and entered the final phase.
07/2010	Participated in the *ROADEF/EURO Challenge 2010: A large-scale energy management problem with varied constraints* together with my colleagues Roman Steiner and Matthias Prandtstetter. We successfully qualified and finally achieved the 1^{st} place in the junior category and the 2^{nd} place in total.

07/2008	Participated in the *GECCO 2008 Contest Problem: A 2D Packing Problem* and was ranked 3rd of 9 regarding the participating teams.
2005	Academic Excellence Scholarship (Leistungsstipendium) received from the Vienna University of Technology.

List of Publications

19. Martin Schwengerer, Sandro Pirkwieser, and Günther R. Raidl.
A variable neighborhood search approach for the two-echelon location-routing problem. In J.-K. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2012*, volume 7245 of *LNCS*, pages 13–24. Springer, 2012.

18. Sandro Pirkwieser, Günther R. Raidl, and Jens Gottlieb.
Improved packing and routing of vehicles with compartments. In R. Moreno-Díaz et al., editors, *Computer Aided Systems Theory – EUROCAST 2011: 13th International Conference, Las Palmas de Gran Canaria, Spain, February 6–11, 2011, Revised Selected Papers, Part I*, volume 6927 of *LNCS*, pages 392–399. Springer, 2012.

17. Andreas M. Chwatal and Sandro Pirkwieser.
Solving the two-dimensional bin-packing problem with variable bin sizes by greedy randomized adaptive search procedures and variable neighborhood search. In R. Moreno-Díaz et al., editors, *Computer Aided Systems Theory – EUROCAST 2011: 13th International Conference, Las Palmas de Gran Canaria, Spain, February 6–11, 2011, Revised Selected Papers, Part I*, volume 6927 of *LNCS*, pages 456–463. Springer, 2012.

16. Sandro Pirkwieser, Günther R. Raidl, and Jens Gottlieb.
Tackling the loading aspect of the vehicle routing problem with compartments. In L. Di Gaspero et al., editors, *Proceedings of the 9th Metaheuristic International Conference (MIC 2011)*, Udine, Italy, 25–28 July 2011.

15. Sandro Pirkwieser, Günther R. Raidl, and Jens Gottlieb.
Improved packing and routing of vehicles with compartments. In A. Quesada-Arencibia et al., editors, *Proceedings of EUROCAST 2011 – 13th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 6–11, 2011*, pages 302–304, 2011.

14. Andreas M. Chwatal and Sandro Pirkwieser.
Solving the two-dimensional bin-packing problem with variable bin sizes by greedy randomized adaptive search procedures and variable neighborhood search. In A. Quesada-Arencibia et al., editors, *Proceedings of EUROCAST 2011 – 13th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 6–11, 2011*, pages 320–321, 2011.

13. Sandro Pirkwieser and Günther R. Raidl.
Variable neighborhood search coupled with ILP-based very large neighborhood searches for the (periodic) location-routing problem. In M. J. Blesa et al., editors, *Proceedings of Hybrid Metaheuristics – Seventh International Workshop, HM 2010, Vienna, Austria, October 1–2, 2010*, volume 6373 of *LNCS*, pages 174–189. Springer, 2010.

12. Sandro Pirkwieser and Günther R. Raidl.
Matheuristics for the periodic vehicle routing problem with time windows. In *Proceedings of Matheuristics 2010: third international workshop on model-based metaheuristics, Vienna, Austria, June 28–30*, 2010.

11. Jakob Puchinger, Günther R. Raidl, and Sandro Pirkwieser.
MetaBoosting: Enhancing integer programming techniques by metaheuristics. In V. Maniezzo et al., editors, *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*, chapter 3, pages 71–102. Springer, 2010.

10. Sandro Pirkwieser and Günther R. Raidl.
Multilevel variable neighborhood search for periodic routing problems. In P. Cowling and P. Merz, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2010*, volume 6022 of *LNCS*, pages 226–238. Springer, 2010.

9. Sandro Pirkwieser and Günther R. Raidl.
Multiple variable neighborhood search enriched with ILP techniques for the periodic vehicle routing problem with time windows. In M. J. Blesa et al., editors, *Proceedings of Hybrid Metaheuristics – Sixth International Workshop, HM 2009, Udine, Italy, October 16–17, 2009*, volume 5818 of *LNCS*, pages 45–59. Springer, 2009.

8. Sandro Pirkwieser and Günther R. Raidl.
Boosting a variable neighborhood search for the periodic vehicle routing problem with time windows by ILP techniques. In M. Caserta and S. Voß, editors, *Proceedings of the 8th Metaheuristic International Conference (MIC 2009)*, Hamburg, Germany, 13–16 July 2009.

7. Sandro Pirkwieser and Günther R. Raidl.
A column generation approach for the periodic vehicle routing problem with time windows. In M. G. Scutellà et al., editors, *Proceedings of the International Network Optimization Conference 2009*, Pisa, Italy, 26–29 April 2009.

6. Sandro Pirkwieser and Günther R. Raidl.
A variable neighborhood search for the periodic vehicle routing problem with time windows. In C. Prodhon et al., editors, *Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing*, Troyes, France, 23–24 October 2008.

5. Sandro Pirkwieser and Günther R. Raidl.
   Finding consensus trees by evolutionary, variable neighborhood search, and hybrid algorithms. In M. Keijzer et al., editors, *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 323–330, Atlanta, GA, USA, 12–16 July 2008. ACM.

4. Sandro Pirkwieser, Rubén Ruiz-Torrubiano, and Günther R. Raidl.
   Exact methods and metaheuristic approaches for deriving high quality fully resolved consensus trees. In J. Küng et al., editors, *Proceedings of BIRD'08, 2nd International Conference on Bioinformatics Research and Development, Poster Presentations*, volume 26 of *Schriftenreihe Informatik*, pages 115–124, Technical University of Vienna, Austria, 7–9 July 2008. Trauner Verlag.

3. Sandro Pirkwieser, Günther R. Raidl, and Jakob Puchinger.
   A Lagrangian decomposition/evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem. In C. Cotta and J. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 69–85. Springer, 2008.

2. Sandro Pirkwieser, Günther R. Raidl, and Jakob Puchinger.
   Combining Lagrangian decomposition with an evolutionary algorithm for the knapsack constrained maximum spanning tree problem. In C. Cotta and J. van Hemert, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2007*, volume 4446 of *LNCS*, pages 176–187. Springer, 2007.

1. Sandro Pirkwieser.
   A Lagrangian decomposition approach combined with metaheuristics for the knapsack constrained maximum spanning tree problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, October 2006.
   Supervised by Günther R. Raidl and Jakob Puchinger.