A flexible metaheuristic framework for solving rich vehicle routing problems

Inauguraldissertation
zur Erlangung des Doktorgrades
der Wirtschafts- und Sozialwissenschaftlichen Fakultät
der Universität zu Köln

2011

vorgelegt von

Dipl.-Wirt.-Inf. Ulrich Vogel

aus Hürth

Referent:	Prof. Dr. Dr. Ulrich Derigs
Korreferent:	Prof. Dr. Dirk Briskorn
Tag der Promotion:	16.12.2011

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations and Symbols

2L-CVRP	Capacitated vehicle routing problem with two-dimensional loading constraints
3L-CVRP	Capacitated vehicle routing problem with three-dimensional loading constraints
ABHC	Attribute based hill climber
AMP	Adaptive memory programming
ARP	Arc routing problem
BKS	Best known solution(s)
CMT	Set of testing instances by Christofides et al. (1979)
CV	Coefficient of variation
CVRP	Capacitated vehicle routing problem
DDM	Dialog, data, models
DSS	Decision support system
DVRP	Dynamic vehicle routing problem
FSMVRP	Fleet size and mix vehicle routing problem
GDA	Great deluge algorithm
GIS	Geographic information system
GIST	Greedy indirect search technique
GLS	Guided local search
GPS	Global positioning system

GWKC	Set of testing instances by Golden et al. (1998)
HVRP	Heterogeneous vehicle routing problem
HYBRID-ABHC	ABHC heuristic with LS neighborhoods and LNS moves
HYBRID-RRT	RRT heuristic with LS neighborhoods and LNS moves
LNS	Large neighborhood search
LNS-RRT	RRT heuristic with LNS moves
LS	Local search
LS-ABHC	ABHC heuristic with LS neighborhoods
LS-RRT	RRT heuristic with LS neighborhoods
MDVRP	Multi-depot vehicle routing problem
Mflop/s	Million floating point operations per second
MP-VRP	Multi-pile vehicle routing problem
NV	Number of vehicles
OOP	Object-oriented programming
OR	Operations research
OVRP	Open vehicle routing problem
PDP	Pickup and delivery problem
PDPTW	Pickup and delivery problem with time windows
pkm	Passenger-kilometer
PVRP	Periodic vehicle routing problem
RPDPTW	Rich pickup and delivery problem with time windows
RRT	Record-to-record travel
SA	Simulated annealing
SD	Steepest descent

SDVRP	Split delivery vehicle routing problem
TA	Threshold accepting
TD	Total distance
TDVRP	Time-dependent vehicle routing problem
tkm	Tonne-kilometer
TS	Tabu search
TSP	Traveling salesman problem
TTRP	Truck and trailer routing problem
VNS	Variable neighborhood search
VRP	Vehicle routing problem
VRPB	Vehicle routing problem with backhauls
VRPC	Vehicle routing problem with compartments
VRPM	Vehicle routing problem with multiple use of vehicles
VRPPD	Vehicle routing problem with pickups and deliveries
VRPTT	Vehicle routing problem with trailers and transshipments
VRPTW	Vehicle routing problem with time windows
A	Set of ABHC attributes
$A(S)$	Set of ABHC attributes of a solution
A^e	Set of entering ABHC attributes
$\bar{A}^e$	Set of entering ABHC attributes (problem-specific)
A^l	Set of leaving ABHC attributes
$\bar{A}^l$	Set of leaving ABHC attributes (problem-specific)
$amem$	ABHC attribute memory
C	Set of compartments

C_i	Set of customer visit combinations
c_{ij}	Arc cost
$cost(S)$	Solution cost
$customers(t)$	Set of customers visited during a tour
d	Depot
e_i	Begin of customer time window
$\mathcal{F}$	Set of feasible solutions
f_i	Customer service frequency
$\mathcal{I}$	Set of insertion heuristics
I^{pc}	Set of product/compartment incompatibilities
I^{pp}	Set of product/product incompatibilities
l_i	End of customer time window
$locations(t)$	Set of locations visited during a tour
m	Maximum number of vehicles
m_k	Maximum number of trucks
m_l	Maximum number of trailers
$\mathcal{N}$	Set of neighborhoods
N	Set of nodes/locations
N_c	Set of customers
$N(S)$	Set of neighbors of a solution
$N(S, i)$	Set of neighbors reachable by a move involving a specific customer
O_i	Set of orders placed by a customer
P	Set of products
p^{LS}	Probability of LS move in a hybrid method

p_o	Order product type
Q	Vehicle capacity
Q_c	Compartment capacity
Q_k	Truck capacity
Q_l	Trailer capacity
q	Number of customers removed in LNS
q_i	Customer demand
q_o	Order quantity
$\mathcal{R}$	Set of removal heuristics
$R(i,j)$	Shaw removal relatedness between two customers
r	Removal percentage in LNS
s_i	Customer service time
T	Maximum travel duration
t	Number of periods in planning horizon
t_{ij}	Arc travel time
$tours(S)$	Set of tours of a solution
V	Set of vehicles
β	LS-RRT randomization
β^{SR}	Shaw removal randomization
β^{STR}	Subtour removal randomization
β^{WR}	Worst removal randomization
Δ	Cost increase
$\bar{\Delta}$	Cost increase (problem-specific)
δ	RRT deviation

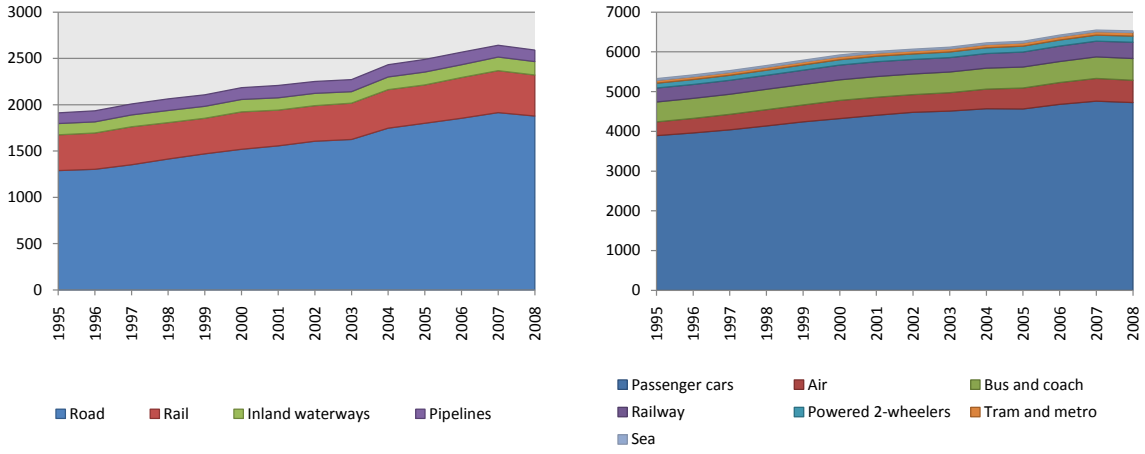
Φ	Problem-specific information
Φ^s	Problem-specific sequence-level information
Φ^t	Problem-specific tour-level information
φ	Shaw relatedness weight for distance
χ	Shaw relatedness weight for time windows
ψ	Shaw relatedness weight for demand
ω	Shaw relatedness weight for product type

Chapter 1

Introduction

Road carriage accounts for an integral part of freight transportation within the European Union. In fact, it is the dominant mode of inner-EU transports: Eurostat (2011a) reports a total volume of 1,690 billion tonne-kilometers (tkm) for the 27 member states in the year 2009, of which around 70 percent are contributed by vehicles registered in Germany, Spain, Poland, France, Italy, and the United Kingdom. Compared to 359 billion tonne-kilometers by rail and 130 billion tonne-kilometers by inland waterways the road share of inland freight transport is 77.6 percent. Air freight is less important for inner-EU transports, amounting to only 12.3 million tonnes compared to 15,123 million tonnes of road freight; yet, the average value of one tonne of air cargo is usually much higher than in other modes of transport. Between European airports a significant amount of freight declared as air cargo is actually transported on the ground by so-called *road feeder services* today.

For the years 1995 to 2008 the European Commission (2010) reports an average annual increase of freight transport (in tkm) within the EU-27 countries of 2.3 percent. In Eurostat (2011b) this growth is explained by the "rapid increase in global trade [...] and the deepening integration of the enlarged EU, alongside a range of economic practices (including the concentration of production in fewer sites to reap economies of scale, delocalisation, and just-in-time deliveries)". The growth of freight transport over the years is shown in figure 1.1(a) for the individual transportation modes: road carriage accounts for most of the growth, with its share rising from 67.4 to 72.5 percent. Note that the recent global economic and financial crisis had a heavy impact on the logistics sector as well, and the decline is already evident in the year 2008.



(a) Freight transport (in billion tkm)

(b) Passenger transport (in billion pkm)

Figure 1.1: EU-27 road transportation by transportation modes (European Commission, 2010).

Road transportation does not only refer to freight but also to passenger transport: figure 1.1(b) displays the growth of passenger transportation modes for the years 1995 to 2008, measured in passenger-kilometers (pkm). In 2008 public transport on the road by buses and coaches accounted for 8.4 percent of passenger transportation.

Route planning The logistics sector is well-known to be a market with small profit margins for competitors; cost is usually the major decision factor for a customer choosing a carrier. On the side of trucking companies this cost pressure requires an efficient planning of operations to increase resource utilization and reduce empty mileage, thereby reducing variable costs such as fuel and toll costs on the one hand, and also fixed costs related to the company's resources such as drivers' wages and acquisition and maintenance costs of the vehicle fleet on the other hand. At the same time companies need to maintain the quality of service promised to their customers which is, first of all, punctuality and reliability.

Route planning is the operational task of a trucking company to create tours and schedules for vehicles and drivers to accomplish its customers' transportation-related demands. The aim is to determine a minimum cost plan which complies with agreed services and observes legal restrictions and other side-constraints. Unfortunately, feasible and especially cost-optimal plans can hardly be generated manually in real-world scenarios. To cite an example, a major German supermarket chain operates a fleet of 2,300 vehicles located at 28 distribution centers; replenishing the stores involves the planning of more than 100

tours per day for some distribution centers (DVZ, 2011b). Such complex scenarios require the use of information and communication technology, supported by *operations research* methods.

Operations research methods Hillier and Lieberman (1980) describe operations research (OR) as a "scientific approach to decision making" which is "applied to problems that concern how to conduct and coordinate the operations or activities within an organization" and which employs techniques from multiple disciplines such as mathematics, statistics, economics, business administration, electronic computing, engineering, and behavioral sciences. Applying the OR approach to a specific planning problem typically includes the construction of a mathematical model abstracting the essential elements of the problem, the development of systematic procedures to obtain solutions, and the determination of (preferably) optimal solutions, which indicate the best possible course of action. According to Hillier and Lieberman (1980), being concerned with practical management OR "must also provide positive, understandable conclusions to the decision maker(s)".

The generation of transportation plans is a classical application of OR methods. The most common model formulation for route planning is the *vehicle routing problem* (VRP). Introduced more than fifty years ago it is one of the most intensively studied problems in the field of OR today. Its basic variant considers a set of geographically dispersed customers, each having a certain demand, who are served by a fleet of vehicles located at one depot. Minimizing the total distance traveled the customers are partitioned into a set of clusters assigned to one vehicle each, and within each cluster the shortest tour starting and ending at the depot is determined. Constraints such as vehicle capacity or maximal tour length restrain the possibilities of routing.

Many solution methods have been designed for VRPs over time: *exact* solution methods, which always generate the optimal solution of a problem, and *heuristic* approaches, which waive the guarantee of optimality and aim at producing high-quality solutions within a short time instead. Often, due to unacceptably long and inconsistent running times exact methods are not applicable in real-world contexts which involve the solution of large, complex VRPs and which only allow a short time to generate a plan and make a decision. Consequently, heuristics such as *neighborhood search* methods are common practice in route planning.

To provide the best possible support for decision makers and to generate high-quality transportation plans good routing software must be tailored to the specific planning problems and business rules of a trucking company or a certain industrial sector. For instance, the distribution of groceries to supermarkets involves time windows for delivery, different replenishment frequencies for stores of different sizes, refrigerated trucks to maintain the cold chain for certain products, and changes of demand at short notice, e.g. in the case of weather changes (DVZ, 2011a). In an OR-based approach such specific requirements affect both the model formulation and the solution method embedded into a planning system. If business rules are not modeled adequately, solutions and recommendations generated by optimization are suboptimal or even cannot be translated into feasible real-world operations.

Design of a VRP framework Software vendors and consultancies offering individual decision support software for the road transportation sector are faced with the task of designing and implementing solution methods for the specific vehicle routing scenarios of their customers. Here, previously developed methods for similar VRPs can often be reused to a certain degree. Many successful neighborhood search methods in the literature developed for vehicle routing problems share a good deal of their ideas and differ only rather slightly in few problem-specific aspects. This observation provided motivation to develop a software framework to facilitate the reuse of design ideas and code. Providing a reusable set of classes for neighborhoods search heuristics, the purpose of this framework is to accelerate the process of developing high-quality solution methods for the wide class of real-world *rich vehicle routing problems*. Yet, the focus is not only on the reuse of code, but also on defining the structure of a unified solution approach and implicitly providing a guideline for development.

In the remainder of the introduction to this thesis we give further motivation for the topic in section 1.1 by characterizing planning problems that occur in road transportation. Then, we present a brief overview of components and features of common decision support systems for road transportation in section 1.2. Section 1.3 clarifies certain aspects regarding the purpose of our framework. Finally, section 1.4 presents a summary of the main contributions of this thesis and section 1.5 sketches the outline.

1.1 Planning Problems in Road Transportation

Problems related to route planning arise in a diverse field of economic sectors and businesses. To name a few examples, consider the distribution of consumer products, e.g. supermarket chains replenishing their stores from central distribution centers, the coordination of service crews, the routing and scheduling of buses in public transportation, or operations in emergency services. Fulfilling requests for transportation efficiently is the core of the business of shippers and carriers in freight logistics. Companies outside the logistics sector have parts of their supply chains managed by third-party logistics providers, or they maintain their own networks for the transportation of goods or persons. Here, we present some typical planning problems in the field of road transportation and classify them according to the three classical levels of decision-making:

- **Strategic planning** has a deep impact on the operations and the success of a company; it aims at long time horizons and often involves large capital investments. This is the case for decisions concerning the design of a transportation network, for instance the decision at which location to open a new hub to improve services in a certain region or whether to close an existing hub. Planning resources typically affects the dimension and the composition of the vehicle fleet, the number of drivers employed, or the distribution of vehicles and drivers over multiple hubs. In this context questions might arise how to dimension resources to maintain a certain service level and to which degree to cope with (seasonal) demand fluctuations by falling back on subcontractors.

- **Tactical planning** aims at allocating the existing resources of a company effectively and efficiently over a medium-term horizon rather than on a day-to-day level. Consider, for example, the design of service/delivery areas assigned to the same vehicle/personnel or the creation of weekly master plans. Potentially, there is some degree of freedom in choosing customers' delivery days or delivery rhythms, which can help to level out strong daily fluctuations in resource utilization.

- **Operational planning** deals with short-term problems, most notably the dispatching for the current day or the next days, dealing with the concrete routing and scheduling of vehicles and drivers. Often, the related decisions involve dynamic changes to the transportation plan and require short reaction times: incoming transportation requests must be compatible with the current schedule, or requests must be declined when not profitable. The rerouting of vehicles may be necessary on occasions such as cancellations or changes in the traffic situation.

Modeling these scenarios, the heart of most problems is some variation of a VRP, although a common VRP does not directly address questions such as which fleet to maintain or where to open a new hub. Even if the specific decision to be made is not formulated within the VRP itself, its solution using OR methods can provide valuable decision support nevertheless, for example within a simulation study or by answering *what-if questions*. In the location planning case such a what-if question can be, "to which extent can the total variable delivery costs be reduced if a new hub is opened at a certain location?" Generating and solving multiple VRP instances reflecting different hub locations gives an insight into the consequences of potential scenarios. A *monte carlo simulation* can help when dealing with uncertainties, for example when dimensioning the fleet: randomly generating a significant number of instances from given assumptions (distributions) concerning the number of future transportation requests at certain points in time, solving these instances, and aggregating the results helps to estimate a fleet size which allows to maintain a specific service level.

1.2 Decision Support for Road Transportation

According to a study of the German-speaking market by Drexl (2011) 50 software vendors are currently offering commercial products including VRP algorithms to support organizations in route planning. Another recent survey of routing software is given by Partyka and Hall (2010) who evaluate 22 products by 12 North American and 4 European vendors. Routing software is used in a diverse set of industries, and products are often specialized for one sector, e.g. courier services, service fleets, or emergency services.

Route planning systems essentially belong to the class of *decision support systems* (DSS), which Scott Morton (1971) defines as "interactive computer-based systems which help decision makers utilize data and models to solve unstructured problems". As explained by Keen and Scott Morton (1978) the central characteristic of a DSS is that it can be applied to "decisions in which there is sufficient structure for computer and analytic aids to be of value but where managers' judgment is essential", which implies that decisions are prepared in a process of manager/machine interaction, improving the effectiveness of decision making rather than its efficiency. For managers a DSS serves as a "supportive tool, under their own control, which does not attempt to automate the decision process, predefine objectives, or impose solutions". Sprague and Carlson (1982) define the main components of a typical decision support system as *dialog*, *data*, and *models*, known as the *DDM paradigm*. In brief, modern routing software has the following basic features in terms of the DDM paradigm:

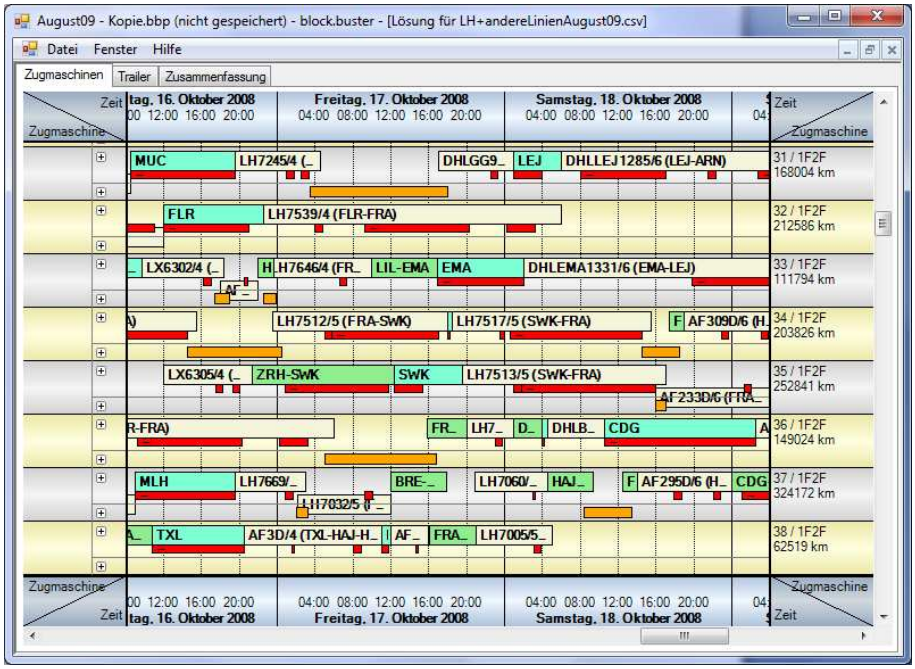


Figure 1.2: Example of a road transportation DSS user interface.

- **Dialog component**: The user interface displays the current status of routing and scheduling, usually including a digital map view of the tours and stops and a *Gantt* chart illustrating the activities of the company's resources – vehicles and drivers – on a timeline. Tours generated by the internal algorithms can be modified by drag&drop editing, which is an essential aspect of manager/machine interaction. Plans and reports can be prepared in a tabular form, e.g. for communication purposes. See figure 1.2 for the user interface of a DSS which addresses specific planning problems arising in road feeder services described in Derigs et al. (2011b). Here, activities of vehicles and drivers such as regular trips, empty trips, or breaks are indicated on a timeline using bars of different colors.

- **Data component**: Besides maintaining information on customers, transportation requests, vehicle fleet, drivers etc. routing software today can usually access several rich data sources: detailed historical traffic data combined with real-time traffic information helps to forecast travel times for different times of the day, vehicle positions can easily be tracked via the *global positioning system* (GPS), and road databases providing comprehensive street attributes (for instance accessibility for the transport of hazardous goods) are used to determine the best paths through road networks.

- **Model component**: Proprietary algorithms, mostly heuristic methods, are used to solve the underlying VRPs that represent planning problems. A crucial feature with respect to dispatching is real-time optimization and rerouting, enabled by the availability of real-time data. Computation times must be appropriately short: according to Partyka and Hall (2010) most software vendors state running times ranging between one and five minutes for an average-sized problem (50 routes, 1000 stops, two-hour hard time windows). A special situation in this context is the quick scheduling of a new stop while the requesting customer is still on the phone.

Partyka and Hall (2010) point out that optimization (the model component) is actually only one aspect of successful routing software and similarly, Drexl (2011) lists modules for *telematics*, statistics, and *geographic information systems* (GIS) as separate components alongside DDM. Telematics features are closely related to *fleet management* functions and include GPS-enabled real-time tracking, navigation, and communication with drivers. Integration with planning software allows tour information to be transferred to the driver, and feedback on vehicle position and trip status (e.g. estimated time of arrival) is transmitted in return to update route planning. GIS provide all information related to digital maps, which refers to both the data and the dialog component of the DSS; for example, GIS provide the road database and translate address information into coordinates (*geocoding*).

Packaged software for route planning usually focuses on the operational planning level and offers a basic functionality which aims at fulfilling the needs for a wide range of applications. Typical business rules and planning options of advanced products include customer time windows or opening hours, prioritization of orders, consideration of vehicle capacities and equipment, driving time regulations, multi-depot planning, and support for both *full truckload* (FTL) and *less-than truckload* (LTL) planning. According to Drexl (2011) all software vendors offer additional customization of their products.

1.3 Metaheuristic VRP Framework

To clarify the designated purpose and role of our framework in helping to develop customized route planning software we address the questions, what a framework actually is, for which types of problems our framework is designed, for which it is *not* designed, and which type of user it targets.

The framework approach Voss and Woodruff (2002) classify optimization software libraries, which are "intended to make it relatively easy and cost effective to incorporate advanced planning methods in application-specific software systems", into *callable packages*, *numerical libraries*, and *component libraries*:

- Callable packages typically include a "black box" solver with a classical functional interface that allows to set up a model and feed it into the solver. Certain solver parameters can be manipulated by the user.
- Numerical libraries provide mathematical optimization methods on a lower level of abstraction, dealing with vectors and matrices rather than with "speaking" model elements.
- Component libraries provide algorithmic components, usually at source code level and following the object-oriented programming (OOP) paradigm, and mechanisms to manipulate and combine these algorithms or parts of them. While *class libraries* offer collections of adaptable classes that can be reused flexibly to develop custom algorithms integrated into other software systems, *frameworks* "impose a broader structure on the whole system".

There is often no clear distinction between class libraries and frameworks. As a possible definition Voss and Woodruff (2002) state that a framework is "a set of classes that embody an abstract design for solutions to a family of related problems [...], and thus provides us with abstract applications in a particular domain, which may be tailored for individual applications." Defining a "reference application architecture ("skeleton"), providing not only reusable software elements but also some type of reuse of architecture and design patterns", frameworks may simplify software development considerably.

In our case of a metaheuristic framework for the solution of rich vehicle routing problems this idea of a reference architecture is realized through a set of predefined metaheuristics for the standard VRP. Serving as unified solution methods, their algorithmic principles form design patterns which are reused for different VRP variants. Customization of these methods is required only for the specific aspects of the rich VRP under consideration.

Focus on vehicle routing problems Our framework aims at facilitating the development of solution methods to solve vehicle routing problems. There are two other particular problem classes which are relevant to route planning but which receive slightly less attention than VRPs: *pickup and delivery problems* (PDP) and *arc routing problems* (ARP). PDPs involve transportation requests with a pickup of goods at a certain location and

their delivery at another location. This is a common scenario in courier services. We do not support PDPs explicitly since we consider the routing of such location pairs, with both locations being assigned to the same tour and the pickup preceding the delivery, as a structural difference to the VRP class, which is more profound than a mere set of additional constraints and affects solution methods to a large extent. Yet, we believe that a PDP framework can be designed in a very similar way to our VRP framework presented in this thesis. ARPs are defined on arcs instead of nodes, i.e. they model the service of street segments instead of customers, speaking in practical applications. Here, we do not consider ARPs either.

Users The potential *user* of this framework must not be confused with an *end-user* of a decision support system or the like. Users in our context have profound knowledge in OR combined with software development skills to design and implement VRP algorithms. To use the framework properly it is important for users to know which algorithms are provided, how they work conceptionally, and which options they offer to be modified or extended. Users must understand some classes and methods they get in touch with directly, but they do not require complete knowledge of all implementation details of the framework.

1.4 Main Contributions

The thesis centers on the development of a metaheuristic framework to solve rich VRPs. Demonstrating the use of the framework we also contribute to the research in the rich VRP area by creating and evaluating new solution algorithms which have not been investigated before. We consider the following three aspects to be the main contributions of this thesis.

A flexible metaheuristic VRP framework The prime contribution of the thesis is the conception and design of a framework for the development of heuristic solution methods for rich VRPs. We assume that such a framework is of particular interest for vendors of customized vehicle routing software offering optimization which is tailored to the specific planning scenarios of their customers.

The framework aims at facilitating and accelerating the development process on two levels: first, it introduces a structure and standardization of development by providing a set of base heuristics together with a set of options for specific adaptations and modifications.

Second, it allows the reuse of solver code to a high degree, enforcing that only problem-specific aspects need to be reimplemented for a new VRP solver. Instead of proposing new general algorithmic concepts the focus of the thesis is on structuring neighborhood search techniques in a way to improve flexibility and customizability. The metaheuristic concepts upon which the framework is based are among the most widely used techniques in routing software according to the survey of Drexl (2011).

New solution methods for five rich VRPs The framework provides five different neighborhood search heuristics, two of them based on *local search*, one based on *large neighborhood search*, and two hybrid approaches combining the search methods. We customize these heuristics for a set of five rich VRPs, each of which has its own characteristic that makes it difficult to be solved: the vehicle routing problem with time windows (VRPTW), the vehicle routing problem with compartments (VRPC), the split delivery vehicle routing problem (SDVRP), the periodic vehicle routing problem (PVRP), and the truck and trailer routing problem (TTRP). A few heuristic/problem combinations have already been proposed and evaluated in similar ways in the literature by other authors, but several applications have not been examined, yet. Especially hybrid solution methods combining local search and large neighborhood search as well as applications of the so-called *attribute based hill climber* are still relatively rare in the literature.

Structured computational evaluations Alongside with enriching the "portfolio" of solution approaches for the five rich VRPs under consideration we present new numerical results. As noted above, a few of the heuristic/problem combinations presented in this thesis have already been examined in the literature in similar ways. Yet, solution methods by different authors often cannot be compared easily since despite using similar techniques they incorporate their own individual aspects. Also, due to different programming languages, implementation skills, and testing environments it is difficult to judge running times. We present computational results for several different problems, which allow conclusions on the general behavior of heuristic methods since all heuristics share the same code basis and we apply a structured, consistent testing scheme. Our results include a few improvements of best known solutions for testing instances listed in the literature, which help the research community to judge the quality of solutions in a field where exact solution approaches generating optimal solutions are rare.

1.5 Outline

Being divided into three main parts, the thesis is structured to provide background information on the relevant scientific fields to the reader first, then to present the concepts and the design of the proposed metaheuristic framework in detail, and finally to report on a set of heuristics developed for a selection of rich VRPs using the framework, including computational results.

Part I: Vehicle Routing Problems and Solution Procedures The first part serves as an introduction into the field of VRPs and metaheuristic solution methods to solve these problems.

Section 2 starts with defining the standard VRP, which is the basic, classical variant of the VRP class. Many real-world extensions have been reported in the literature over time, and we present an overview of these *rich VRPs* to give an idea of the diversity of problems. A selection of five problems, used for defining framework requirements in part II and for evaluation in part III, is covered in more detail.

Metaheuristics are the most common and widely-used solution methods to solve VRPs; some of these specific methods define the foundation of our framework. In section 3 we convey a broad understanding, going into detail with those techniques which are part of our framework. Brief attention is also given to other well-known metaheuristic methods.

Part II: Metaheuristic Framework The second part is dedicated to the concepts and the design of our metaheuristic VRP framework.

Section 4 starts with a review of publications dealing with aspects such as flexibility and reusability of VRP solvers. In our opinion there is a certain shortage in this field, and we formulate requirements for a flexible VRP framework which motivate our own choices regarding framework concept and design.

Section 5 presents the main concepts and the architecture of our framework. On a pseudocode level we describe five heuristics for the standard VRP provided by the framework, and we define mechanisms to adapt these heuristics to the specifics of rich VRPs.

The most relevant classes we have designed for our framework implementation are covered in section 6. Data structure classes to represent a VRP solution and algorithmic classes which are designed for adaptability are especially relevant. Having used the Microsoft .NET framework for our implementation we also provide a list of special techniques applied.

Part III: Customizing The last part demonstrates the use of the framework for developing problem-specific solutions methods, and computational results are presented to show the effectiveness and efficiency of these methods.

Section 7 exemplifies specific modifications of the standard VRP heuristics for five rich VRPs: adaptations refer to data structures and neighborhood search methods. This section can serve as a source of inspiration to the reader when designing algorithms for new, so far unstudied VRPs based on our framework.

Section 8 summarizes the computational results obtained for the rich VRP heuristics. After determining a standard parametrization of the framework we fine-tune our heuristics for each individual problem and compare the results with state-of-the-art solvers of the literature. Special attention is given to the overall robustness of heuristics, which is an important aspect for practical application.

Section 9 finally gives a critical review of our work and especially discusses whether the requirements defined are met by our framework. We also give some ideas for further research.

Part I

Vehicle Routing Problems and Solution Procedures

Chapter 2

Rich Vehicle Routing Problems

The vehicle routing problem is one of the most intensively studied problems in the field of operations research. More than fifty years ago Dantzig and Ramser (1959) introduced the "truck dispatching problem" to find the optimal routing of a fleet of gasoline delivery trucks to supply a certain number of service stations from a terminal. Since then, a vast number of books and articles on problem variants and solution approaches has been published. For instance, a classical bibliography on routing problems is presented by Laporte and Osman (1995), and Golden et al. (2008) report on recent developments in the VRP area. A survey of the most important contributions of exact mathematical programming algorithms and metaheuristics for the VRP is provided by Laporte (2009).

In this chapter we present an overview of well-known vehicle routing problems from the literature, putting the emphasis on problem variants that we specifically address later in this thesis. After introducing the *standard vehicle routing problem* in section 2.1, which is the essence of all VRPs, section 2.2 covers extensions and modifications that reflect real-world aspects, referred to as *rich vehicle routing problems*: delivery time windows in section 2.2.1, vehicles with compartments for heterogeneous products in section 2.2.2, splitting of deliveries in section 2.2.3, periodic delivery schedules in section 2.2.4, and vehicle fleets with separate trucks and trailers in section 2.2.5. Section 2.2.6 completes the chapter by briefly stating some additional well-known VRP variants.

2.1 The Standard VRP

The vehicle routing problem is a generalization of the well-known traveling salesman problem (TSP), which formalizes the problem to find the shortest tour through a given set of customers (or cities or other types of locations) visiting each customer exactly once.

The VRP introduces customer demands and a homogeneous fleet of vehicles located at a depot, each having the same fixed capacity. Here, the problem is to partition the set of customers into clusters, which are served by the same vehicle, and to determine a TSP tour within each cluster of customers which starts and ends at the depot. Within each cluster the total demand of customers must not exceed the vehicle capacity, and the objective of the VRP is to minimize the total distance traveled by the vehicles. A maximum number of disposable vehicles can be given a priori, or alternatively, a hierarchical objective is to minimize the number of vehicles as the primary objective and to minimize distance as the secondary objective. Sometimes an additional constraint demands that a tour may not exceed a given maximal length or duration.

The term "vehicle routing problem" often refers to a class of related problems rather than to a specific problem. The basic problem variant of the VRP class is the so-called classical or standard VRP. The standard VRP is defined on a graph $G = (N, A)$, where N is the set of nodes/locations and $A = \{(i, j) : i, j \in N, i \neq j\}$ is the set of arcs/routes between these locations. The set of locations contains the depot d and the customers $N_c = N \setminus \{d\}$. A cost value c_{ij} is associated with each arc $(i, j) \in A$, as well as a travel time t_{ij}. The fleet of identical vehicles V, each having a capacity Q, is located at the depot. Each customer $i \in N_c$ has a non-negative demand q_i, which has to be served by a vehicle requiring a service time s_i. A maximum allowed travel duration T, identical for all tours, is given as well. The standard VRP is also referred to as the capacitated vehicle routing problem (CVRP).

Unlike the case of exact solution approaches, explicit mathematical problem formulations are not usually required when dealing with heuristics, which incorporate models only implicitly in their implementations. Only for the purpose of clarification do we present a mathematical problem formulation of the standard VRP in the following; other VRP variants covered in the remainder of this chapter are presented verbally.

Introducing decision variables

$$
\begin{aligned}
b_{ijv} &= \begin{cases} 1 & \text{if vehicle } v \in V \text{ travels from location } i \in N \text{ to } j \in N \\ 0 & \text{otherwise} \end{cases} \\
u_{iv} &= \text{position number of location } i \in N \text{ within the tour of vehicle } v \in V \\
x_{iv} &= \begin{cases} 1 & \text{if customer } i \in N_c \text{ is served by vehicle } v \in V \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

the standard VRP can be formulated as the following integer program:

$$\text{min} \quad \sum_{i,j \in N} \sum_{v \in V} c_{ij} \cdot b_{ijv} \tag{2.1}$$

$$\text{subject to} \quad \sum_{j \in N_c} b_{djv} \leq 1 \qquad v \in V \tag{2.2}$$

$$\sum_{i \in N} b_{ikv} - \sum_{j \in N} b_{kjv} = 0 \qquad k \in N, v \in V \tag{2.3}$$

$$u_{iv} - u_{jv} + |N| \cdot b_{ijv} \leq |N| - 1 \qquad i \in N, j \in N_c, v \in V \tag{2.4}$$

$$u_{dv} = 1 \qquad v \in V \tag{2.5}$$

$$\sum_{i \in N_c} q_i \cdot x_{iv} \leq Q \qquad v \in V \tag{2.6}$$

$$\sum_{i,j \in N} t_{ij} \cdot b_{ijv} + \sum_{i \in N_c} s_i \cdot x_{iv} \leq T \qquad v \in V \tag{2.7}$$

$$\sum_{v \in V} x_{iv} = 1 \qquad i \in N_c \tag{2.8}$$

$$x_{jv} - \sum_{i \in N} b_{ijv} \leq 0 \qquad j \in N_c, v \in V \tag{2.9}$$

$$b_{ijv} \in \{0, 1\} \qquad i, j \in N, v \in V \tag{2.10}$$

$$u_{iv} \in \{1, \ldots, |N|\} \qquad i \in N, v \in V \tag{2.11}$$

$$x_{iv} \in \{0, 1\} \qquad i \in N_c, v \in V \tag{2.12}$$

Objective (2.1) minimizes the total cost. Constraint (2.2) ensures that each vehicle v departs at most once from the depot, and (2.3) ensures that each arrival of vehicle v at location k is accompanied by a departure of v from k. These two constraints together impose that, if a vehicle is used, the tour must always start and end at the depot. Subtour elimination constraint (2.4) prevents circle tours, i.e. tours that do not depart from the depot, by specifying that the position of customer j is higher than the position of location i if vehicle v travels from i to j. Since there are many potential numberings for the same tour constraint (2.5) eliminates duplicate solutions by forcing the depot to be at position 1. Constraint (2.6) states that the capacity of vehicle v must not be exceeded by the demands of customers served, and (2.7) defines the maximum travel duration of the vehicle. (2.8) ensures that each customer i is assigned to exactly one vehicle. Constraint (2.9) imposes that a vehicle v must actually visit customer j during its tour if j is assigned to the vehicle. Finally, (2.10)-(2.12) define the decision variables.

2.2 Extensions of the VRP

In the following we present a selection of rich vehicle routing problems, which are much more complex than the standard VRP and reflect practical, real-world aspects. While we go into detail with those problem variants first that are relevant in the remainder of this thesis, we give a list of other well-known problems at the end of this section. The diversity of practical problems in this area underlines the importance of flexible VRP solution methods.

2.2.1 Vehicle Routing Problem with Time Windows

The most prominent and widely studied rich VRP in the literature is the *vehicle routing problem with time windows* (VRPTW). It addresses the common situation that the service of customers is associated with time intervals. For example, customers may only be served during their business hours, consumer products such as groceries must be delivered before the opening hours of a shop, or short intervals for home deliveries are arranged with private customers to reduce waiting times and improve satisfaction.

In the VRPTW the service of a customer $i \in N_c$ has to start during a time window $[e_i, l_i]$. A vehicle may arrive at the location earlier than e_i – then there is a waiting time until the service can begin, which is usually *not* penalized – but it is not permitted to arrive later than l_i. The duration of service at the customer is denoted with s_i. Travel time values t_{ij} are often assumed to be identical to the cost values c_{ij} in the literature. In the majority of publications on the VRPTW a lexicographic objective is considered: the primary objective is to minimize the number of tours, and the secondary objective is to minimize the total tour distance.

Bräysy and Gendreau (2005a,b) presented an extensive survey on heuristic methods for the VRPTW, and an often cited, classical VRPTW publication is Solomon (1987). Time window constraints are sometimes combined with other rich vehicle routing problems; see, for example, Derigs and Vogel (2009) for the open vehicle routing problem with time windows or Cordeau et al. (2001) for the periodic vehicle routing problem with time windows.

2.2.2 Vehicle Routing Problem with Compartments

In the standard VRP all goods demanded by customers are considered to be homogeneous in the sense that they can be transported together without any issues. However, in some

industries goods are inhomogeneous, and in order to allow the transport of such goods together on the same vehicle and save transportation costs hereby, vehicles with distinct compartments for different products are useful. In some applications the compartment setup of a vehicle may be configurable, i.e. the volume of *flexible compartments* can be adjusted by separators. Other scenarios consider *fixed compartments* that cannot be configured.

Most practical applications are reported for fuel distribution. Brown and Graves (1981), for instance, deal with the case of the distribution of light petroleum products by a major US oil corporation where vehicles, consisting of trucks and trailers, have between one and six tanks for different fuel types; Cornillier et al. (2008) present a heuristic for a multi-period petrol station replenishment problem, and Jetlund and Karimi (2004) cover related problems concerning the routing and scheduling of chemical tankers with compartments. The most recent heuristics consider a special problem type that dedicates every (fixed) compartment to one product: El Fallahi et al. (2008) present a memetic algorithm and a tabu search heuristic, and Muyldermans and Pang (2010) analyze the improvement of using multiple compartments over single compartments in waste collection, applying a guided local search algorithm.

In Derigs et al. (2011a) we have introduced the *vehicle routing problem with compartments* (VRPC), which is a rather general problem formulation that covers very different applications. In fuel scenarios each compartment can carry any fuel type, but evidently different products must not be mixed within one compartment. Transportation for food retailing involves frozen and dry goods which need special equipment for fresh delivery. Typical vehicles have two compartments, one served by a refrigerator and one for dry goods.

Formally, customers in the VRPC have demands for multiple inhomogeneous products P. A customer $i \in N_c$ may place several *orders* $o \in O_i$, each referring to one single product $p_o \in P$ with a quantity q_o. Vehicles are still homogeneous, as in the standard VRP, but now each vehicle has the same set of compartments C. In addition to the total vehicle capacity Q each compartment c has an individual capacity Q_c, with $Q \leq \sum_{c \in C} Q_c$. The relation between Q and Q_c indicates whether the compartment setup is configurable or not: if $Q < \sum_{c \in C} Q_c$, the compartments are flexible, and goods loaded into one compartment affect the remaining capacity available for other compartments. If $Q = \sum_{c \in C} Q_c$, the compartments are fixed. The relation $I^{pp} \subseteq P \times P$ defines incompatibilities between products, i.e. $(p_1, p_2) \in I^{pp}$ implies that products p_1 and p_2 must not be transported to-

gether in the same compartment. Incompatibilities between products and compartments are expressed by the relation $I^{pc} \subseteq P \times C$, where $(p, c) \in I^{pc}$ indicates that product p must not be transported in compartment c.

The VRPC routes orders instead of customers, i.e. the objective is to find an assignment of orders to compartments within vehicles and to determine order tours such that all vehicle capacities and compartment capacities are respected, the incompatibility relations are not violated, and the total cost of the tours is minimized. Note that we allow a customer to place multiple orders for the same product and that the orders of a single customer may be served by multiple vehicles. However, a single order may not be split up further as in the split delivery vehicle routing problem presented in the following section.

2.2.3 Split Delivery Vehicle Routing Problem

In most vehicle routing problems each customer has to be visited exactly once. Relaxing this assumption and allowing customers to be visited by more than one vehicle may yield savings in the total distance traveled and in the number of vehicles required. This was first analyzed empirically by Dror and Trudeau (1989) who introduce the *split delivery vehicle routing problem* (SDVRP). Here, each delivery can be an arbitrary fraction of the total demand of a customer, as long as the total demand of each customer is served completely over all deliveries. Dror and Trudeau (1989) also show that, if the triangle inequality holds for the distances, an optimal solution exists where no pair of tours contain more than one customer with a split delivery in common; hence, the number of splits required to improve single-visit solutions is quite moderate. Archetti et al. (2006) demonstrate that cost savings of up to 50 percent can be realized by split deliveries.

While heuristic solution approaches for the SDVRP are presented by Archetti et al. (2008), Chen et al. (2007), and Derigs et al. (2010), presentations of real-life applications with split deliveries are still relatively rare in the literature. For example, Mullaseril et al. (1997) describe a feed distribution problem encountered on a cattle ranch in Arizona, and Sierksma and Tijssen (1998) deal with the problem to determine a helicopter flight schedule for crew exchanges on off-shore platform locations in the North Sea.

Gulczynski et al. (2010) propose a variant with minimum delivery amounts, which is motivated by the observation that it is usually undesirable for customers to be delivered too often since this means interruption and distraction from their primary activities. Also for the distributor multiple deliveries often involve additional paperwork etc. and should

thus be avoided. To overcome this issue the SDVRP-MDA defines a minimum amount for each customer that a partial delivery must not go below.

2.2.4 Periodic Vehicle Routing Problem

The standard VRP is a single-period problem, i.e. a transportation plan is constructed for one period only – a single day for example. But often customers need to be served several times during a certain period of time, for example during a week. If the days of delivery are fixed in advance for each customer, multiple independent daily VRPs can be solved including the customers that need to be served on the respective days. Yet, sometimes there is some flexibility with respect to the exact day(s) that a customer is served. In particular, demands can be periodical in the sense that a customer requires service every two days or every three days, but the commitment to specific days is left open.

Beltrami and Bodin (1974) describe and solve a routing problem for municipal waste collection where some sites need to be served three times a week (either on Monday, Wednesday, and Friday or on Tuesday, Thursday, and Saturday) and some sites on each day of the week. Other applications of the so-called *periodic vehicle routing problem* (PVRP) have, for example, been reported for grocery distribution. The PVRP extends the standard VRP to a planning horizon of t days. Each customer $i \in N_c$ requires a fixed number of visits – the service frequency f_i – and C_i defines the set of allowable combinations of visit days for the customer. For example, those customers from the waste collection application described by Beltrami and Bodin (1974) that need to be served three times a week but not on consecutive days are defined by $f_i = 3$ and $C_i = \{\{1, 3, 5\}, \{2, 4, 6\}\}$. Now, the PVRP involves the selection of a visit combination for each customer and at the same time, based upon the choices made, to solve the resulting VRP for each day of the planning period. As an additional constraint the number of tours per day is limited to m vehicles in common problem formulations from the literature.

An important algorithmic contribution for the PVRP is the tabu search algorithm of Cordeau et al. (1997), which can also be used to solve the periodic TSP and the multi-depot VRP, and a current state-of-the-art algorithm is the variation of variable neighborhood search by Hemmelmayr et al. (2009). New results of a sophisticated hybrid genetic algorithm are presented by Vidal et al. (2011).

2.2.5 Truck and Trailer Routing Problem

In many real-world road transportation scenarios trailers are used to increase the capacity of a vehicle. The vehicle fleet consists of *truck* units and *trailer* units, which can be uncoupled and recoupled again at the depot as well as during a tour. Yet, some customers may not be accessible with a long truck-trailer combination and can only be visited by the truck alone, while the trailer is left at a parking place. For example, customers may be located in city centers or in mountain areas that a large vehicle is not allowed to access or is not able to be maneuvered within safely. Gerdessen (1996) describe an application arising in the distribution of dairy products by the Dutch dairy industry. Here, many customers are located in crowded cities, and using a truck-trailer combination is much more time consuming than serving the customers with the truck only. The trailer is therefore parked outside the city. Another scenario described by Gerdessen (1996) is the distribution of compound animal feed, where farmers can be reached by narrow roads and small bridges only, not accessible by large vehicles.

Chao (2002) introduce the *truck and trailer routing problem* (TTRP), denoting a truck that is pulling a trailer, a *complete vehicle*; *vehicle customers* are reachable either by a complete vehicle or by a truck alone, and *truck customers* are only reachable by a truck alone. Three types of tours[1] are differentiated:

- A *pure truck tour* contains vehicle customers and/or truck customers served by a truck alone.
- A *pure vehicle tour* contains only vehicle customers served by a complete vehicle without uncoupling the trailer at any time during the tour.
- A *complete vehicle tour* consists of a *main tour* traveled by a complete vehicle and one or more *subtours* traveled by the truck alone. A subtour starts with uncoupling the two vehicle units after serving a customer, leaving the trailer at a parking place at or near this customer's location, while the truck continues the tour alone. The subtour ends with returning to the trailer and recoupling the units again. During the un- or recoupling it is possible to reload freight from the truck to the trailer or vice versa.

Figure 2.1 exemplifies the three types of tours in a TTRP solution:[2] a pure vehicle tour departs to the left of the central depot; a pure truck tour departs to the right. A complete

[1] Chao (2002) uses the term "route" instead of "tour".
[2] Figure adapted from Lin et al. (2009).

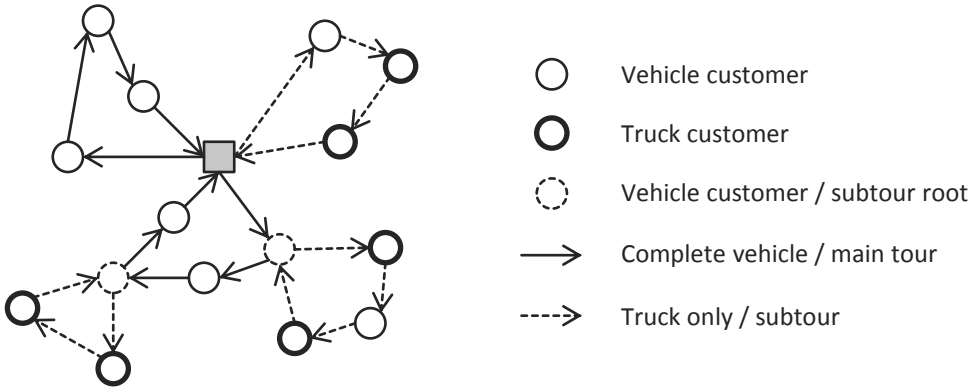


Figure 2.1: Tour types in the TTRP.

vehicle tour is below the depot, consisting of a main tour with two subtours. Customer locations used to uncouple and park the trailer are denoted as *subtour roots*. It is also allowed that a customer serves as the root of multiple subtours (not displayed in the figure). For a pure truck tour the depot is the subtour root.

The vehicle fleet consists of m_k trucks and m_l trailers ($m_k \geq m_l$). Trucks have a capacity of Q_k each, trailers have a capacity of Q_l, which implies that a complete vehicle has a total capacity of $Q_k + Q_l$. The total demand associated with the customers of a pure truck tour or a subtour may not exceed Q_k. Yet, since reloading freight between trucks and trailers during a tour is allowed, the total demand of all subtours of a complete vehicle tour may exceed the capacity of the truck nevertheless. In any case the total demand within a pure vehicle tour or a complete vehicle tour may not exceed $Q_k + Q_l$.

The most important contributions of algorithms for the TTRP are based on neighborhood search: Chao (2002) and Scheuerer (2006) present tabu search approaches, and Lin et al. (2009) propose a simulated annealing heuristic. The current state-of-the-art solution method by Villegas et al. (2011) is a hybrid metaheuristic combining multiple techniques.

In its most simple form the TTRP does neither involve different traveling costs of tours served by a complete vehicle or by a truck alone nor cost of parking a trailer. Also, the numbers of trucks and trailers are not restricted, any truck is able to pull any trailer, and a trailer can be parked at any (vehicle) customer. Drexl (2007) presents the *vehicle routing problem with trailers and transshipments* (VRPTT), which is a very complex generalization of the TTRP motivated by a real-world scenario as well as a branch-and-

cut algorithm to solve small instances. Among several other characteristics such as time windows the VRPTT introduces the following characteristics:

- Trucks and trailers can be either *collection vehicles* or *support vehicles*. Collection vehicles are used to collect the supplies of customers, while support vehicles are used as "mobile depots" that cannot visit customers.
- A trailer may be pulled by different trucks during the course of its tour, i.e. there is no fixed assignment of trucks and trailers.
- Using the equipment (presence) of a collection truck, load may be transferred from any vehicle to any other vehicle during a tour.
- Intermediate locations can be used either for parking (*parking locations*) or for load transfer (*transshipment locations*). The collected supply is delivered to *unloading stations*.

2.2.6 Other Rich VRPs

Beside the problems presented in the previous sections the VRP literature covers a multitude of additional real-world variants of the vehicle routing problem. Without claiming that our selection is complete we give an overview of other well-known rich VRPs in the following.

Schrage (1981) points out the characteristic of a tour to be open or closed: while a closed tour implies that a vehicle finally returns to its starting location, it does not return in an open tour. This slight variant of the standard VRP, named *open vehicle routing problem* (OVRP) by Sariklis and Powell (2000), reflects the use of external couriers or subcontractors instead of private vehicles. For tours conducted by subcontractors the remaining part after leaving the last customer does not have to be planned since the drivers return "home", which incurs no further cost for the company. On a higher management level a related problem may be to determine a proper mix of own and hired vehicles for which closed and open tours need to be planned, respectively. In two recent publications on the OVRP Li et al. (2007) develop a record-to-record travel algorithm and Derigs and Reuter (2009) present an attribute based hill climber heuristic.

The *vehicle routing problem with backhauls* (VRPB) considers delivery (linehaul) customers to which goods are delivered as well as pickup (backhaul) customers from which goods are brought back to the depot. Both types of customers can be served within the

same tour, but due to potential loading/unloading issues it may be necessary to serve all linehaul customers before any backhaul customer is visited. If this constraint does not apply, i.e. if linehaul and backhaul customers may be mixed within a tour, special attention has to be paid observing the vehicle capacity since the load fluctuates during the tour. *Vehicle routing problem with pickups and deliveries* (VRPPD) is another name for this variant. Ropke and Pisinger (2006b) present a unified heuristic to solve multiple problems of the VRPB class. A further generalization involves requests to pickup goods at one location and deliver them to another location. Associated pickups and deliveries must be scheduled within the same tour, and obviously a pickup location must be visited before the respective delivery location. The most prominent problem of that class is the *pickup and delivery problem with time windows* (PDPTW).

While the standard VRP only has a simple numerical capacity constraint it is necessary in certain scenarios to determine the actual arrangement of goods within a vehicle during route planning. The *capacitated vehicle routing problem with two-dimensional loading constraints* (2L-CVRP), see for example Zachariadis et al. (2009), considers customer demands as two-dimensional, rectangular, weighted items. The basic 2L-CVRP determines a feasible plan to load such items into the vehicle, and its "sequential" variant additionally ensures, depending on the visit sequence of customers within the tour, that every item delivered to a customer can be unloaded without having to reposition other items inside the vehicle. A three-dimensional version of the problem, the 3L-CVRP, is presented by Gendreau et al. (2006). The *multi-pile vehicle routing problem* (MP-VRP), see Doerner et al. (2007), considers the loading of items into piles and enforces feasible unloading sequences as well.

Further time constraints to be considered in vehicle routing and driver scheduling are imposed by the restrictive rules on driving times and rest periods from EC Regulation No. 561/2006 and by the rules on working times from Directive 2002/15/EC. These rules and their implications for route planning are explained in Kopfer et al. (2007). It is especially important to consider breaks and rest periods explicitly during route planning when deliveries are associated with time window constraints, or when tour durations exceed a usual working day.

In the *multi-depot vehicle routing problem* (MDVRP) the vehicle fleet is distributed over multiple depots, see for example Cordeau et al. (1997). Each tour has to start and end at the same depot, and customers may be served by vehicles starting from any depot. The *vehicle routing problem with multiple use of vehicles* (VRPM), see Taillard et al. (1996)

for instance, allows to schedule multiple tours for a single vehicle during the planning period. Vehicles return to the depot between two tours.

The *fleet size and mix vehicle routing problem* (FSMVRP) relaxes the assumption of a homogeneous fleet, see Golden et al. (1984). Instead, multiple types of vehicles with different capacities and acquisition costs are available, and the problem is to determine the optimal dimension and vehicle type mix of the fleet, minimizing both fixed vehicle costs and variable routing costs. Potentially, different driving speeds or traveling costs of vehicles can be considered. Related problems considering an existing fleet of limited size are commonly denoted as *heterogeneous vehicle routing problems* (HVRP), see Baldacci et al. (2008) for an overview. Nag et al. (1988) present the *site-dependent vehicle routing problem* in which certain customers can only be served by a subset of the vehicle fleet. For instance, accessibility of customers may depend on the vehicle size, or specific facilities such as cooling devices may be required for some customers.

Real-world planning scenarios are often characterized by dynamics of information, which has led to the consideration of *dynamic vehicle routing problems* (DVRP), see Psaraftis (1995). Here, the information which is relevant for routing decisions is not known completely in advance but is revealed (or updated) over time to the decision maker, when vehicles have already started their tours, or when it is not possible to revise decisions already made. Such dynamic information can, for example, be the arrivals of new customer orders at any time.

Another type of dynamics is considered in the *time-dependent vehicle routing problem* (TDVRP), see Malandraki and Daskin (1992), which assumes travel times to be dependent on the distance between locations and also the time of day. Travel times can vary to a great extent during rush hours in urban environments or due to changing weather conditions, for instance. If variations of travel times are known in advance, the TDVRP is a static problem in the sense of information revelation. If online information on changes of travel time forecasts is available, for example considering traffic congestion due to accidents, the problem is again a dynamic one.

Most rich VRPs in the literature concentrate on specific, individual aspects only, while real-world scenarios often combine multiple aspects of several such problems. For example, Derigs et al. (2011b) present a real-world VRP arising in the air cargo road feeder service business that combines the VRPM, a heterogeneous vehicle fleet, and rules on driving times and rest periods. Here, transportation tasks of a given timetable are combined into

trips, respecting the rules of EC Regulation No. 561/2006 and Directive 2002/15/EC. These trips, which start and end at the hub, are aggregated into so-called multiple-trips that are operated by the same tractor. Each trip needs to be assigned to a compatible trailer as well. The primary objective is to minimize the number of required tractors, i.e. the number of multiple-trips. Caramia and Guerriero (2010) describe a milk collection problem for an Italian dairy company collecting raw milk from farmers, which is a combined TTRP/VRPC. Trucks and trailers are used, but farms are often small and inaccessible by complete vehicles so that they can only be visited by the truck alone. At the same time vehicles have multiple compartments for different milk types that cannot be mixed.

Chapter 3

Metaheuristic Solution Methods

Although the development of exact solution methods has made significant progress over the last years, heuristic approaches are still dominant in the vehicle routing literature. Heuristics aim at producing high-quality solutions within a short time; waiving the guarantee to obtain the optimal solution of a problem they usually neither allow to measure the quality of a solution, i.e. the deviation from the unknown optimum, nor to specify a minimum quality to be obtained in advance. Exact methods provide such quality measures, but since vehicle routing problems are NP-hard combinatorial optimization problems only rather small instances can practically be solved to optimality. Laporte (2009) states that current state-of-the-art methods are able to solve standard VRP instances of up to approximately 100 customers. Consistency and robustness is also a problem since instance properties such as tightness of time windows may influence running times dramatically. Pisinger and Ropke (2007) report that a 1,000 customer VRPTW instance could be solved to optimality (by other authors), but still unsolved instances with only 50 customers exist. In some planning scenarios these limitations may not apply and exact methods can be used, but usually one has to resort to heuristic methods. Especially in real-time dispatching or within interactive decision support systems heuristic methods are preferable in general.

This chapter presents an overview of metaheuristic solution methods for the VRP, but its main purpose is to describe the techniques upon which our framework is based. Metaheuristics are usually *improvement methods*, i.e. they are started from a given solution. Generating such an initial solution is the purpose of *construction heuristics*, which we briefly review first in section 3.1. According to Laporte (2009) metaheuristic principles for the VRP can broadly be classified into local search, population search and learning mechanisms. The concept of local search, which most successful VRP heuristics incor-

porate at least partly and which is also the basis of our framework, is to define a neighborhood topology in the space of feasible solutions and to improve a given solution by iteratively moving from one neighbor solution to another. This general concept, referred to as *neighborhood search*, is introduced in section 3.2. Section 3.3 presents several popular *metaheuristic controls* to guide the search, and the terms *local search* (section 3.4) and *large neighborhood search* (section 3.5) refer to two different types of neighborhood topologies. Population search and learning mechanisms are not of specific interest in the context of our framework; we give brief attention to them in section 3.3.5.

3.1 Construction Heuristics

The purpose of a construction heuristic is to generate a feasible initial solution of acceptable quality in a rather short time for an improvement method such as local search or large neighborhood search. The two most prominent methods for the VRP are the *savings method* by Clarke and Wright (1964) and the *sweep method* by Gillett and Miller (1974) – two classical VRP heuristics which have been used a long time before the emergence of metaheuristics.

The savings method starts with a solution containing one *deadhead tour* (d, i, d) for each customer $i \in N_c$. Tours are merged sequentially: calculating savings $s_{ij} = c_{id} + c_{dj} - c_{ij}$ for any two tours ending with customer i and starting with customer j, the two tours with the highest saving are appended in each iteration, provided that the capacity is not exceeded. The procedure terminates when no tours can be appended any more. Often, the resulting tours are improved as individual TSPs in a post-optimization step.

The sweep method is applicable when polar coordinates are given for the customers or can be calculated. This is the case when customers are distributed on a plane and characterized by x- and y-coordinates in a Cartesian coordinate system, or if they are specified by latitudes and longitudes in a geographic coordinate system. The angle of 0 is assigned to one customer, and the angles of the remaining customer locations around the depot (or another specific location) are calculated from this 0-angle customer. Customers are inserted sequentially into an empty solution with increasing polar angles: each customer is appended to the last tour of the solution if this is feasible; otherwise a new tour is opened. As a variation, customers can be inserted at the cheapest position within the tour. When all customers have been inserted the resulting tours can be post-optimized as TSPs as well.

Another straightforward construction method within a large neighborhood search heuristic is the use of an insertion heuristic to insert all customers into an empty solution. See section 3.5 for details.

3.2 Neighborhood Search

Neighborhood search is a generic principle of improvement heuristics to solve hard *combinatorial optimization problems* such as VRPs. A combinatorial optimization problem can formally be represented as a triple $(E, \mathcal{F}, cost)$, where

- E is a set of ground-elements, the arcs of a graph for instance,
- $\mathcal{F} \subseteq 2^E$ is the set of feasible subsets/solutions, and
- $cost(S)$ is the value of a subset/solution $S \in \mathcal{F}$.

The optimization problem is to find the feasible solution of minimal value, i.e. to determine

$$\min \{cost(S) \mid S \in \mathcal{F}\}.$$

The idea of neighborhood search is to improve a given (feasible) solution by a sequence of modifications. Formally, it is based on the specification of a problem-specific *neighborhood* structure on $\mathcal{F}$: for every solution $S \in \mathcal{F}$ a subset $N(S) \subseteq \mathcal{F}$ specifies the *neighbors* of S. Starting from an initial solution $S_0 \in \mathcal{F}$, the process of the search is described by a sequence $S_0, S_1, S_2, \ldots$ of solutions with $S_i \in N(S_{i-1}), i = 1, 2, \ldots$. The transition from a solution S_{i-1} to S_i is also called a *move*. The result of the procedure, for which some termination criterion must be defined, is the best solution contained in that sequence.

The simple outline of neighborhood search is displayed in algorithm 3.1. Here, functions SelectNeighbor and AcceptNeighbor abstract from the so-called *metaheuristic control* that guides the search process through the solution space:

- SelectNeighbor(S, N) selects a solution S' from the neighborhood $N(S)$ of the current solution $S \in \mathcal{F}$, and
- AcceptNeighbor(S', S) decides whether solution S is replaced by solution S' or not, based on $\Delta = cost(S') - cost(S)$ and, potentially, further criteria.

Algorithm 3.1 Neighborhood search

```
1: function NeighborhoodSearch(initial solution S, neighborhood N)
2:     S_best := S
3:     repeat
4:         S' := SelectNeighbor(S, N)
5:         if cost(S') < cost(S_best) then
6:             S_best := S'
7:         end if
8:         if AcceptNeighbor(S', S) then
9:             S := S'
10:        end if
11:    until stop criterion fulfilled
12:    return S_best
13: end function
```

The general neighborhood search concept was introduced under the name *local search* (LS); it is traditionally designed to investigate a huge number of small modifications of the current solution per iteration and then to perform the most improving modification/move. For the *steepest descent* (SD) strategy the SelectNeighbor function selects the best solution S' within the neighborhood of S, with $cost(S') = \min_{S^* \in N(S)} cost(S^*)$. An alternative strategy is to select the *first improving* solution found when evaluating the neighborhood. In both cases a solution S' is only accepted by AcceptNeighbor if $cost(S') < cost(S)$; if no such solution exists in the current neighborhood, the procedure stops.

Performing only improving moves until no further improvement is possible involves the danger that the search gets "trapped in a *local optimum*" which may be significantly worse than the *global optimum* of the problem. Metaheuristic controls, which allow to perform deteriorating moves and to escape from such local optima, are presented in the next section.

3.3 Metaheuristic Controls

The term *metaheuristic* is commonly used for a variety of heuristic solution methods. Voss et al. (1999) define a metaheuristic as

> "an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may

> manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method. The family of meta-heuristics includes, but is not limited to, adaptive memory procedures, tabu search, ant systems, greedy randomized adaptive search, variable neighborhood search, evolutionary methods, genetic algorithms, scatter search, neural networks, simulated annealing, and their hybrids."

In the context of solution methods based on neighborhood search the major purpose of a metaheuristic or metaheuristic control is to guide the search and prevent termination in a (bad) local optimum. Unlike steepest descent a metaheuristic control provides a strategy to perform non-improving moves on certain occasions to be able to escape from such local optima. The two most prominent principles of metaheuristic controls are *annealing techniques* and *tabu search*:

- Annealing techniques (sections 3.3.1 and 3.3.2) refrain from the idea of selecting the best neighbor in each iteration. Instead, random moves are performed, provided that the current solution is not deteriorated "too much" by non-improving moves. The term "annealing" refers to the characteristic that non-improving moves are prohibited gradually while the search proceeds. For all annealing techniques one or more parameters have to be defined which control the speed of the annealing process.

- Tabu search methods (sections 3.3.3 and 3.3.4) in each iteration move to the best neighbor, which potentially is not improving, and in order to prevent cycling they declare certain solutions as "tabu".

For a solution method the choices of the problem-specific neighborhood structure and the domain-independent metaheuristic control are partly interrelated. Metaheuristics such as steepest descent or tabu search which evaluate the complete neighborhood of a solution require a neighborhood structure that allows the neighbors of a solution to be enumerated explicitly. This applies to the classical local search neighborhoods for VRPs described in section 3.4. Yet, a neighborhood may also be defined implicitly: in large neighborhood search, see section 3.5, a single neighbor is obtained by applying a certain algorithm to the current solution. Such neighborhoods can be combined with annealing techniques more suitably.

3.3.1 Simulated Annealing

Simulated annealing (SA), see Kirkpatrick et al. (1983), which is *the* classical and most prominent metaheuristic, uses a random number generator to decide on the acceptance of non-improving moves. A random neighbor S' of the current solution S is selected in each iteration of the search. If the move is improving, i.e. if $\Delta := cost(S') - cost(S) < 0$, it is accepted; otherwise it is performed with probability $e^{-\frac{\Delta}{T}} \in [0, 1)$, controlled by a *temperature* parameter T. By decreasing T according to a specified *cooling rate* c the probability of accepting such uphill moves gradually decreases during the search; finally, the procedure converges to a descent method. Usually, the temperature is adjusted by setting $T := T \cdot c$ every L iterations. Johnson et al. (1989) recommend setting L to a multiple of the expected neighborhood size.

3.3.2 Deterministic Annealing

Threshold accepting (TA), the *great deluge algorithm* (GDA), and *record-to-record travel* (RRT), variants of simulated annealing, are referred to as *deterministic annealing* methods because they define non-random acceptance criteria.[1] As SA, all three controls select random neighbors that are accepted if the move is improving. The extent to which non-improving moves are accepted is lowered during the search; but unlike SA, the decision on acceptance is not random.

TA, see Dueck and Scheuer (1990), accepts a neighbor if the cost increase compared to the current solution does not exceed a certain *threshold* value. This threshold is decreased gradually during the course of the search. GDA, see Dueck (1993), defines a so-called *waterlevel* which represents the maximum cost that an acceptable solution may have and which is decreased each time a solution is accepted. RRT, see Dueck (1993), uses the cost of the best solution found so far during the search – the *record* – to define acceptability: here, solutions are accepted which are not worse than the current record by a certain relative *deviation*.

3.3.3 Tabu Search

Tabu search (TS), proposed by Glover (1989, 1990), always scans entire neighborhoods in each iteration and moves to the best neighbor even if it does not lead to an improvement.

[1] Note that a heuristic based on a deterministic annealing control is still *not* a deterministic algorithm since neighbor selection remains random.

This corresponds to the "steepest descent – mildest ascent" principle and is the central difference to all annealing methods. An approach to avoid cycling is to forbid solutions which have already been visited. For this purpose solutions possessing certain attributes of recently visited solutions are temporarily declared *tabu* for a number of iterations, unless their cost is below a so-called *aspiration level*.

A tabu search strategy for a VRP can be, for example, that after moving a customer from one tour to another, moving this customer back to its former tour is forbidden for a number of iterations unless the move results in a new best solution. The tabu search idea leaves many degrees of freedom for the actual implementation. In the next section we present one specific TS-variant: the attribute based hill climber.

3.3.4 Attribute Based Hill Climber

Whittley and Smith (2004) present the *attribute based hill climber* (ABHC) heuristic as a parameter-free variant of tabu search. ABHC uses a generic concept for specifying non-tabu neighbors, which has to be specialized for every problem domain: assume a set A of *attributes* over the set $\mathcal{F}$ of feasible solutions, and let $A(S) \subset A$ denote the set of attributes that a specific solution S possesses. In a vehicle routing problem A may be chosen to represent the arcs between any two locations. In this case a solution possesses an attribute $a \in A$ if the two associated locations are visited immediately one after the other in a tour.

During the search an *attribute memory* is maintained which stores for every attribute a the objective value of the best solution S^* visited so far with $a \in A(S^*)$. Another solution is acceptable if it would be the best solution visited so far for at least one attribute that it contains, i.e. if at least one attribute is improved in the memory. The procedure stops when no acceptable neighbors exist, but it can be restarted by resetting the attribute memory, see Derigs and Kaiser (2007). After specifying the attributes ABHC is parameter-free and, except for some tie-breaking, completely deterministic.

3.3.5 Other Metaheuristic Strategies

Several other metaheuristics and enhancements of local search have been proposed in the literature over time. To conclude this section on metaheuristic solution methods we briefly state some additional well-known techniques. In chapter 8 we compare our own numerical results with those of current state-of-the-art methods from the literature; some

of these reference methods are based on metaheuristics presented in the following.

Mladenović and Hansen (1997) propose *variable neighborhood search* (VNS) that incorporates the use of multiple neighborhoods and systematic changes between them. Neighborhoods are given in a predefined order $k = 1, \ldots, k_{max}$. Starting with $k = 1$ a random neighbor x' of the initial solution x is selected from neighborhood k. This random step is also called "shaking". Then, a local search phase is started from x', terminating at a local optimum x''. If x'' is better than x, x is replaced and the search continues with shaking using neighborhood $k = 1$; otherwise a shaking step is applied to the old x with $k := k+1$. The shaking neighborhoods are usually ordered with increasing size or "distance". The local search phase neighborhoods do not necessarily correspond to the shaking neighborhoods; a single neighborhood may be used or multiple neighborhoods with systematic changes as well.

As VNS, *guided local search* (GLS), see Voudouris and Tsang (1999), incorporates sequential local search phases but modifies the objective function of a problem before starting a local search phase by adding penalty terms for certain solution features. Whenever a local optimum (according to the augmented objective function) is reached, these penalties are updated for the purpose of leaving the local optimum in the next phase and avoiding solution features which are unlikely part of good solutions.

Some approaches allow visiting infeasible solutions and penalize constraint violations in the objective function to guide the search towards repairing these violations. Penalty multipliers are adjusted dynamically depending on whether the currently examined solutions are infeasible or not; if constraints are violated, the respective multipliers are increased, and vice versa. Examples of such VRP heuristics are the well-known "taburoute" heuristic by Gendreau et al. (1994) and the solution approaches of Cordeau et al. (1997) and Hemmelmayr et al. (2009) for the PVRP.

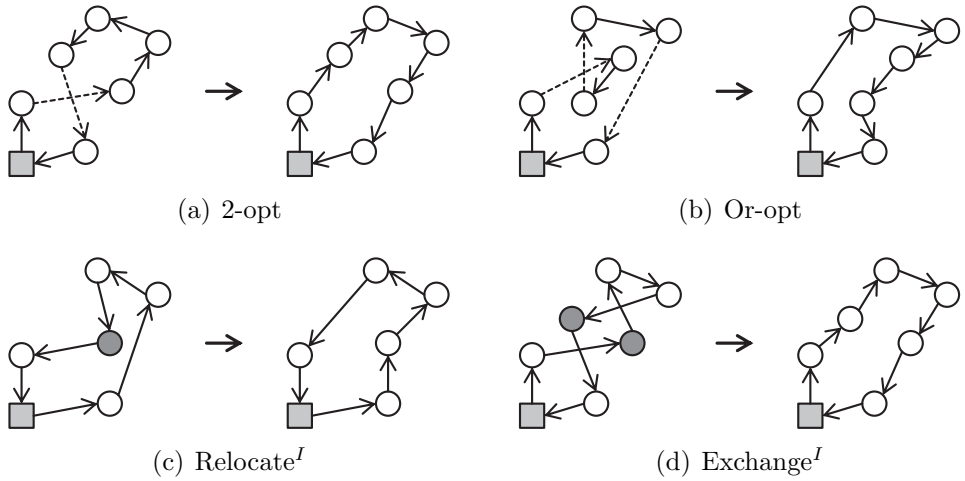
To reduce the computational complexity of tabu search Toth and Vigo (2003) introduce the *granularity* principle. It is motivated by the idea that long, costly arcs seldom belong to good solutions and can thus be removed from the network without loss of solution quality. Hereby, the number of arc exchanges to be evaluated per iteration is reduced. A granularity threshold needs to be selected to specify the minimum length of arcs to be excluded; this threshold can be adjusted dynamically when the search gets stuck. Apparently, the granularity principle can be combined with any local search based heuristic.

While the metaheuristic strategies described up to this point have the purpose to control a pure neighborhood search, the term "metaheuristic" is also used for several other algorithmic concepts. Taillard et al. (2001) propose a unified presentation of a range of methods under the name of *adaptive memory programming* (AMP). These methods include *genetic algorithms*, *scatter search*, *ant colony optimization*, and also tabu search. Genetic algorithms improve solutions by simulating the evolutionary process of reproduction by recombination and mutation of genetic representations ("genotypes"). Scatter search has not been very successful to solve VRPs; ant colony optimization, in contrast, is a biology-inspired learning method and a current trend in the VRP literature. The general AMP concept incorporates a memory which holds solutions ("populations") or characteristics of solutions encountered during the search, a mechanism to generate new solutions from this memory, and a local search procedure to improve such solutions. Metaheuristics which combine evolutionary concepts with neighborhood search are sometimes denoted as *memetic algorithms*. The AGES method by Mester and Bräysy (2005) is a hybrid algorithm of this class which has attracted a great deal of attention. In general, hybridizing multiple solution paradigms is a current trend in the field of VRP heuristics, combining the positive effects of single methods and reducing their respective shortcomings. Besides memetic algorithms, combinations of metaheuristics and mathematical programming methods are gaining attention under the term *matheuristics*, see for instance Maniezzo et al. (2010).

3.4 Local Search Neighborhoods

Traditional neighborhood search for the VRP is based on operators which apply small modifications to a solution involving exchanges of edges/arcs. Specific exchanges can be made within one tour (*intra-tour* moves) or between multiple tours (*inter-tour* moves). This section illustrates the most common moves or neighborhoods from the VRP literature; since their use is rather intuitive we refrain from giving formal definitions.

Figure 3.1 shows a selection of intra-tour moves, originating from common TSP heuristics. The most widely used intra-tour moves belong to the class of so-called *k-opt* methods, see Lin and Kernighan (1973). A k-opt move replaces a set of k edges of a tour by another set of k edges. The number of potential reconnections of tour segments rises dramatically with increasing values of k – exploring neighborhoods with $k > 4$ becomes practically infeasible due to the computational effort. For VRPs, *2-opt* is the most common move, see figure 3.1(a). The exchange of two edges corresponds to the reversal of a subsequence of customers within the tour. A restricted variant of *3-opt* is the *or-opt* operator, proposed



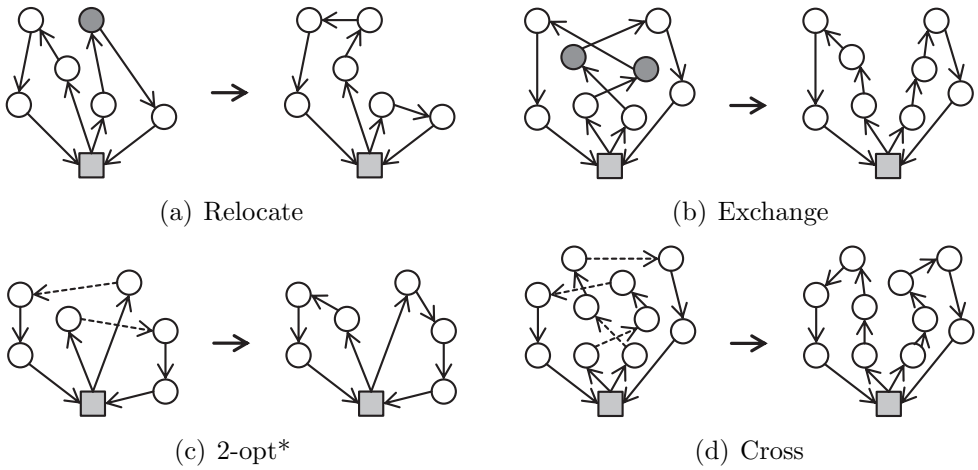
(a) 2-opt (b) Or-opt

(c) Relocate^I (d) Exchange^I

Figure 3.1: Local search intra-tour moves.

by Or (1976), which relocates a subsequence of usually up to three customers within a tour by replacing three edges as displayed in figure 3.1(b). Contrary to 2-opt the orientation of the subsequence remains unchanged.

Relocate and *exchange* are two simple operators which transfer a customer to a different position in the solution or swap two customers, respectively. Figures 3.1(c) and 3.1(d) show them being applied as intra-tour moves within a single tour. We denote these intra-tour applications as *relocate*^{I} and *exchange*^{I}. The two operators are even more common as inter-tour moves (Savelsbergh, 1992), as displayed in figures 3.2(a) and 3.2(b).



(a) Relocate (b) Exchange

(c) 2-opt* (d) Cross

Figure 3.2: Local search inter-tour moves.

Inter-tour exchanges affect multiple tours, usually two of them. Osman (1993) formally defines the so-called λ-*interchange* which selects up to λ (not necessarily consecutive)

customers from two tours, respectively, and swaps them. Relocate and exchange are covered by this definition being $(1,0)$-, $(0,1)$- and $(1,1)$-interchanges. The generalized insertion procedure (GENI) proposed by Gendreau et al. (1992) is an extension of relocate which allows a customer to be inserted between two other customers that are currently not directly connected in their tour. Assume a customer i and a sequence of customers $j, j+1, \ldots, k-1, k, k+1$ in the same or in another tour. i can be inserted between j and k resulting in a sequence $j, i, k, j+1, \ldots, k-1, k+1$.

The next operator displayed in table 3.2 is *2-opt**, the inter-tour variant of 2-opt, which is commonly attributed to Potvin and Rousseau (1995) but was already described by Savelsbergh (1992). As shown in figure 3.2(c) two edges from two different tours are removed and the segments are reconnected. In other words it swaps the rear segments of the tours.

The powerful *cross* operator described by Taillard et al. (1997) is a generalization of the exchange move that transfers sequences of customers instead of single customers only. Figure 3.2(d) gives an example move. The maximum length of a sequence is usually restricted to three due to the huge number of potential exchanges. The cross neighborhood covers relocate, exchange, 2-opt* and the inter-tour variant of or-opt as well.

The characteristic of a move to preserve the orientation of customer sequences or not is interesting when constraints on visiting sequences apply: in the VRPTW, for instance, 2-opt moves are likely to fail except when time windows are not very tight. Orientation also matters in asymmetric VRPs, for example considering detailed city street networks and different travel durations between two locations depending on the direction of the trip.

Some special cases of the presented moves change the number of tours in a solution. Relocate, 2-opt*, and cross can be used to integrate a tour into another one completely by moving customers into one direction only. Conversely, customers can be extracted to open a new tour. This case may seem to contradict the intuition of distance minimization since opening a new tour is usually associated with a high increase of distance at first. But a new tour can help during the course of the search if a problem is tightly constrained and only few feasible moves can be performed at all, adding further "space" for customers. Also consider the example of the small SDVRP instance given in figure 3.3: assuming a vehicle capacity of 15 the total demand of 30 can be served in two tours having a total distance of 7. However, by adding a third tour and joining the formerly split deliveries

we can obtain a solution with a total distance of only 6.

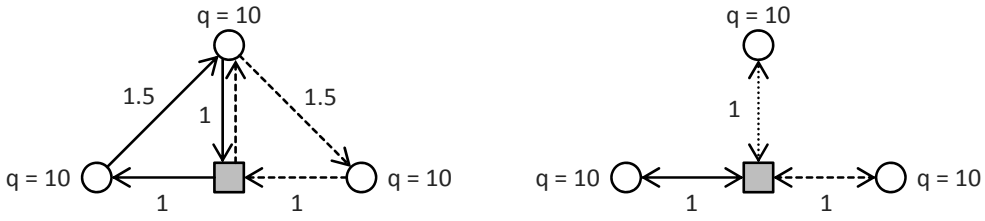


Figure 3.3: Tour distance vs. number of tours with split deliveries.

The traditional VRP neighborhoods can be evaluated rather efficiently for the standard VRP since changes of tour distance, duration, and vehicle load can generally be calculated in constant time. Only for operators such as or-opt or cross is it important to restrict the neighborhood size by setting appropriate sequence length limits. However, additional feasibility checks for certain rich VRPs may be a computational bottleneck and slow down the search. This can be especially critical when using the attribute based hill climber which evaluates a very large number of moves before making any step forward.

3.5 Large Neighborhood Search

The huge number of publications in the VRP literature underlines that the local search principle is suited very well for the solution of vehicle routing problems. Its tendency to get trapped in a bad local optimum can be tackled by proper use of metaheuristic controls. Still, relying on very small, "local" moves can be associated with a few issues, especially when dealing with tightly constrained rich VRPs:

- If many constraints are involved in a VRP, small changes to a given feasible solution often result in infeasibilities, and the number of allowed moves can actually be rather small. Consider, for instance, a VRPTW instance with very tight time windows and only little flexibility in the routing. In a typical feasible solution the tours are packed densely with few slack times that are required for shifting customers between tours.

- According to Schrimpf et al. (2000) complex problems are often "discontinuous". This describes the effect that moving from one solution to another one which is very different from the first solution but of similar quality may require a sequence of steps visiting intermediate solutions of much lower quality. For example, having

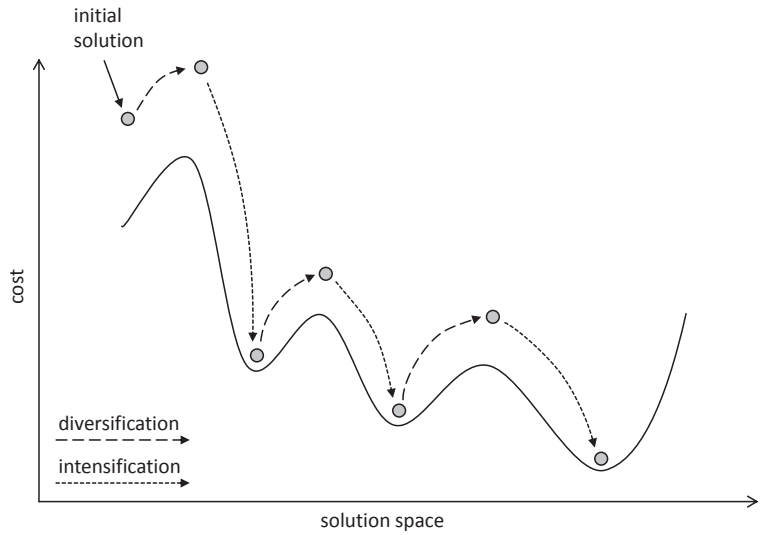


Figure 3.4: Alternation between intensification and diversification in neighborhood search.

customers distributed in clusters over the map, i.e. small distances within certain groups of customers and rather large distances between the groups, any small change to the assignment of customers to tours can (temporarily) lead to a much higher total distance.

- The strength of LS is *intensification* of the search, i.e. it is able to examine the solutions in a particular region of the solution space carefully. However, a good search combines intensification with *diversification*, which is the ability to access many different areas of the solution space. LS does not provide methods to reach distant areas other than by a long sequence of independent moves; as pointed out above this can be difficult, and also the proper use of a metaheuristic control may not be guaranteed to help. Figure 3.4 sketches in an idealized way how by alternation of diversification and intensification the solution space is explored thoroughly: when the search does not yield any more improvements in the current area of the solution space a diversification step is required to enter another region or *basin of attraction*, which is a term referring to a set of solutions from which a (steepest descent) local search process converges to the same local optimum. The diversification step (or phase) is followed by another intensification phase.

Consequently, *ruin-and-recreate* is introduced by Schrimpf et al. (2000) – a new neighborhood concept that ruins or disintegrates a large fraction of a solution and then tries to restore the solution as best as possible in each iteration of the search. The authors apply

this concept to the VRPTW (amongst others): here, the ruining step is done by removing customers from their tours by different strategies, thereby not serving them any more temporarily, and recreating is done by reinserting the customers into the solution with a cheapest insertion heuristic. In the *large neighborhood search* (LNS) heuristic proposed by Shaw (1998a,b) the ruin step is based on the random selection of customers which are "similar" with respect to their distances, and the recreate step is done with constraint programming techniques.

The ruin-and-recreate concept helps to overcome the issues of LS stated above. If a large part of a solution is destroyed, there is a lot of freedom in creating a new solution and a very high number of neighbor solutions. It becomes easier to find solutions which are a) feasible and b) not much worse than the current solution. Another common approach to overcome the problem of moving across tight constraints is allowing to visit infeasible solutions, see section 3.3.5. Schrimpf et al. (2000) argue that their approach is favorable over relaxation techniques since at any time during the search a completely feasible solution is available.

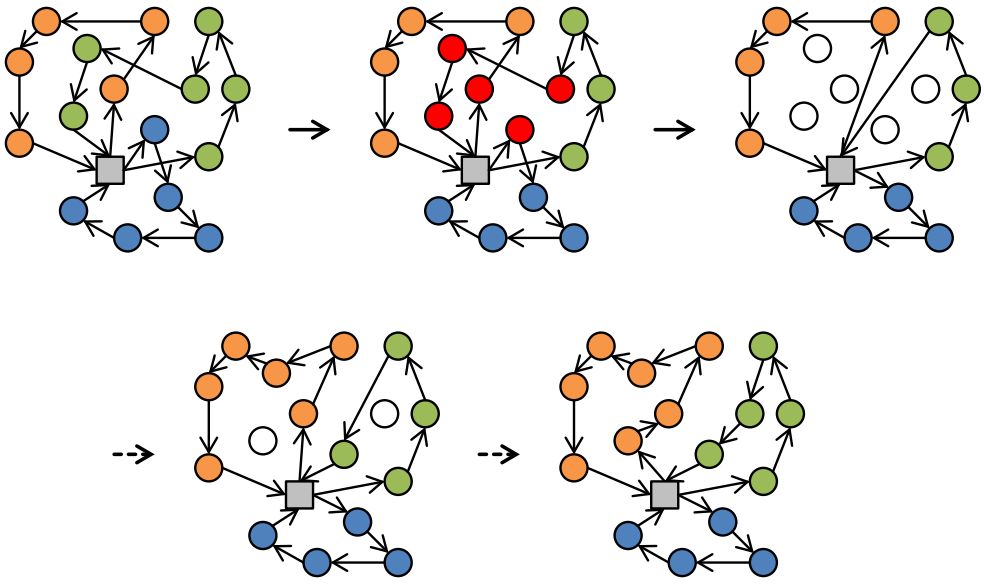


Figure 3.5: Steps of a large neighborhood search move.

Large neighborhood search was popularized later by Ropke and Pisinger (2006a) and Pisinger and Ropke (2007), who describe LNS as a generic heuristic based on domain-independent principles for destruction and repairing and then use LNS to solve the PDPTW and several VRP variants. With a problem-specific *removal heuristic* LNS removes a significant number q of customers from the solution which are then reinserted

by a problem-specific *insertion heuristic*. An incomplete solution is obtained in the intermediate step which is feasible with the only exception that not all customers are served. q, which controls the neighborhood size, is selected randomly in each iteration within certain bounds. This process is illustrated in figure 3.5. Diversification of the search is enforced by randomization and by using multiple removal and insertion heuristics with different strategies. Note that since the neighborhood is defined algorithmically it cannot be enumerated in a similar way as a local search neighborhood, and thus, LNS is combined with annealing techniques rather than with steepest descent or tabu search methods.

Now, we present an overview of common removal and insertion heuristics for VRPs; most of them are presented by Ropke and Pisinger (2006a) and Pisinger and Ropke (2007). In the following we use $U \subset N_c$ to denote the customers which are currently not served in a solution.

- *Random removal* (Ropke and Pisinger, 2006a) simply removes q randomly selected customers from the solution.

- *Worst removal* (Ropke and Pisinger, 2006a) removes customers that seem to be "misplaced" at their current positions, which is measured by the cost decrease associated with their removals, hoping that better positions in the solution can be found later. Customers are selected and removed sequentially and independently until q of them have been removed. The selection is randomized: for each removal a random number $y \in [0, 1)$ is drawn first, and $r = y^{\beta^{WR}} \cdot |N_c \backslash U|$, with a randomization parameter β^{WR}, determines the customer to be removed, which is the one that is associated with the r-th highest decrease of cost.

- *Shaw removal* (Shaw, 1998a) relies on a relatedness or similarity measure $R(i, j)$ between two customers i and j which has to be defined depending on the specific problem. This measure can, for instance, incorporate distance, demand, and time window information. The procedure starts with selecting and removing one random pivot customer and after that, $q - 1$ further customers are removed sequentially. Selection is random but biased towards customers which are similar to the customers already removed.[2] The idea of removing a set of similar customers is that such customers can be shuffled around more easily than unrelated customers.

- *Cluster removal* (Ropke and Pisinger, 2006b) removes complete clusters of customers from tours, i.e. disjoint subsets with small distances between the customers.

[2]The same randomized selection mechanism is applied as in worst removal with a randomization parameter β^{SR}.

Other than shaw removal it aims at removing large chunks of related customers from just a few tours instead of removing customers from many tours. A random tour is selected first, and then Kruskal's algorithm is used to partition the customers of the tour into two clusters. One of these clusters is chosen at random, and all of its customers are removed. The process repeats until q customers have been removed.

- *Neighbor graph removal* and *Request graph removal* (Ropke and Pisinger, 2006b) are both based on historical information collected during the search. Neighbor graph removal stores for each arc the cost of the best solution encountered so far containing that arc.[3] Based on these values customers that seem to be misplaced in the current solution are preferred for removals. Request graph removal is a variant of shaw removal which counts for any two customers the number of times they have been served within the same tour in the recently encountered best solutions, using this information to define the relatedness measure.

- *Basic greedy insertion* (Ropke and Pisinger, 2006a) follows the cheapest insertion principle. For a customer $i \in U$ and a vehicle/tour $v \in V$ let $c_{i,v}$ be the increase of cost if i is inserted into v at the cheapest feasible position or ∞ if no such position exists in v. Then, in each iteration of the insertion heuristic customer i' is inserted into tour v' at the cheapest feasible position with $c_{i',v'} = \min \{ c_{i,v} \mid i \in U, v \in V \}$.

- *Regret insertion* (Ropke and Pisinger, 2006a) addresses the tendency of the greedy heuristic to postpone the insertion of "difficult" customers which can finally be inserted only under relatively high cost increases. For $i \in U$ and $k \in \{1, ..., |V|\}$ let $v_{i,k} \in V$ be the vehicle/tour into which customer i can be inserted with the k-th lowest increase of cost, i.e. $c_{i,v_{i,k}} \leq c_{i,v_{i,k'}} \forall k < k'$. Then, for $k \geq 2$ the *regret* is defined as

$$regret_i^k := \sum_{j=2}^{k} \left(c_{i,v_{i,j}} - c_{i,v_{i,1}} \right)$$

which measures the disadvantage of not inserting i into the currently best suited tour but to a less suitable one. Large values of k yield a high degree of foresight. In each step regret-k insertion inserts customer $i \in U$ having the highest regret value $regret_i^k$.

[3] This corresponds to the information held by the ABHC memory.

Part II

Metaheuristic Framework

Chapter 4

Framework Requirements

In the introduction to this thesis we have pointed out the relevance of flexible approaches for the development of heuristic rich VRP solution methods which enable and facilitate the incorporation of real-world requirements. The rareness of such approaches has motivated the development of our own metaheuristic VRP framework. Before presenting its concepts and design in chapters 5 and 6 we review a selection of contributions in section 4.1 that point into similar directions but which are associated with certain drawbacks or have not been demonstrated to be compatible with many real-world requirements, yet. Then, in section 4.2 we discuss a set of essential attributes of a VRP framework providing valuable assistance. Finally, in section 4.3 we summarize some general characteristics of rich VRPs that a framework needs to address; this leads to the definition of a feature list of adaptation requirements.

4.1 Existing VRP Frameworks and Libraries

Although probably thousands of contributions have been published in the field of vehicle routing problems, articles addressing aspects such as the reusability, customization, or flexibility of VRP solvers are surprisingly rare despite the practical relevance. Most publications in the classical OR literature concentrate on new algorithmic ideas, new problem variants, or they present applications of existing algorithms to problems when such combinations have not been studied before. From the fact that groups of authors often focus their research on their own "pet heuristics", which they repeatedly apply in multiple contexts, one can suspect that authors indeed use self-written libraries or frameworks for that purpose. But generally no information on the implementation process of a solution method is given in a publication, and it is not unlikely either that development follows a copy-paste-and-modify fashion: starting with an existing implementation for a another

problem the code is modified for the new problem wherever necessary. Before defining our requirements for a VRP framework we review a selection of contributions which are related to the neglected reusability topic.

Voss and Woodruff (2002) report on a selection of reusable metaheuristic optimization libraries which provide generic implementations of solution methods, running on abstract solution types and neighborhood structures that the user has to implement for a specific problem. *HotFrame* is a framework containing adaptable components for different metaheuristics and an architecture for the collaboration between these generic components and problem-specific concepts. It is implemented in C++ and follows an object-oriented design consistently. *EasyLocal++* is another well-known C++ class library for local search based heuristics. Further frameworks presented by Voss and Woodruff (2002) are based on evolutionary/genetic algorithms and constraint programming. Such generic frameworks or libraries can provide a good basis for developing solution methods; yet, they leave problem-specific aspects completely to their users, who have to implement solution representations and neighborhood structures by themselves.

A library of VRP local search heuristics by Groër et al. (2010) is freely available under an open source license. Implemented in C++ it provides several methods for construction and improvement, including seven different neighborhoods and several metaheuristic controls. Many details of the search can be configured via parameter settings. The authors claim that additional constraints can be incorporated into the standard VRP heuristics by modifying the evaluation methods associated with each neighborhood. Yet, the library is not specifically designed to adapt heuristics to other VRP variants flexibly.

Derigs and Döhmer (2008) discuss the relevance of flexible modeling tools and heuristics in DSS development projects. On the one hand, real-world problems cannot usually be represented by standard models for which good algorithms and empirical computational studies are already available. On the other hand, the relevant aspects of a problem are not often formulated completely and correctly in the initial requirement phase but evolve during the development process. Hence, it is important to be able to generate a sequence of system prototypes including models and solvers, which can be criticized by the problem owner. In this context the authors demonstrate the use of their GIST-framework (*greedy indirect search technique*) for solving rich constrained combinatorial optimization problems. *Indirect search* separates problem-specific domain knowledge from general procedural problem solving knowledge and is suited for rapid prototyping and handling complex constraints flexibly. It applies metaheuristics to an auxiliary search space of simple, non-

problem-specific representations such as permutations, and a decoder maps such solutions to solutions of the original problem. The decoder incorporates decision rules to generate solutions based on greedy principles (e.g. cheapest insertion) as well as a constraint checker and covers all problem-specific aspects, which can be modified rather easily, here. Indirect search overcomes the difficulty of a direct search implementation (which is used in our framework) to enforce feasible solutions by describing feasibility-preserving moves; this task can be very complicated in the case that many complex constraints are given for a problem. Within the VRP area the GIST-framework has been applied to the PDPTW and another real-world pickup and delivery problem so far. It would be interesting to see whether the concept can be used for other complex VRP variants producing good results as well.

The lack of unifying modeling and solution approaches for VRP applications which are both efficient and general is also stated by Irnich (2008). He presents a sophisticated framework combining a giant-tour representation, resource-constrained paths, and sequential search to accelerate neighborhood enumeration and feasibility checking of moves. A single *giant-tour* represents the complete solution, containing route-start and route-end nodes to separate the individual tours. It is considered as a *resource-constrained path* which covers both individual tour constraints and inter-tour constraints. Compatibility relations between the route-start and route-end nodes cover vehicle and depot characteristics. This representation can be embedded into a local search based method, enabling the separation of constraint modeling and the actual search procedures. The focus of this interesting approach is clearly on computational efficiency rather than on user-friendliness. Irnich (2008) claims that the modeling techniques can be used for many rich VRPs, including time windows, multiple depots, pickup-and-delivery problems, compartment constraints, heterogeneous fleets, and periodic VRPs. Solutions are not presented, but experiments on the computational efficiency for a selection of problems are conducted instead.

Generalization / specialization relationships between problem variants provide rather an easy way to cover multiple VRPs with a single solver. Pisinger and Ropke (2007) design the *rich pickup and delivery problem with time windows* (RPDPTW) which is a rich model formulation similar to the PDPTW that additionally features precedence constraints between locations and inhomogeneous vehicles. An adaptive large neighborhood search heuristic developed for the RPDPTW can be used for other problems "out-of-the-box" as long as these problems can be transformed into the RPDPTW. The authors demonstrate transformations for the VRPTW, the CVRP, the MDVRP, the site-dependent VRP, and

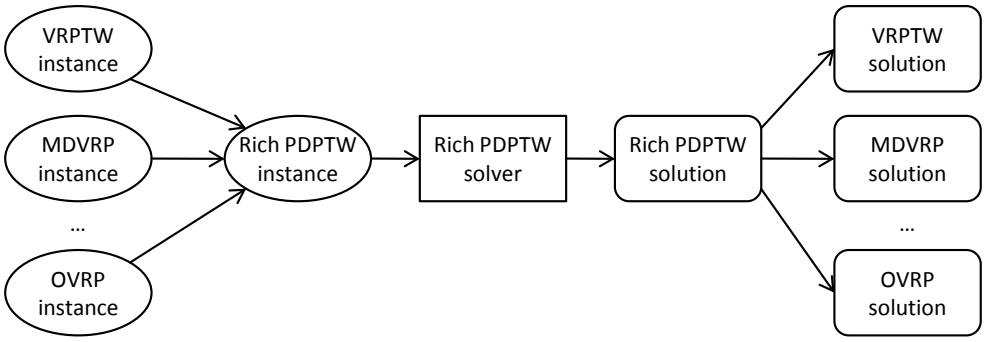


Figure 4.1: Solving multiple VRPs with a rich PDPTW formulation and solver, adapted from Ropke and Pisinger (2006b).

the OVRP and obtain results of high quality. Figure 4.1 depicts the instance level transformation process: for example, an MDVRP instance is transformed into an equivalent RPDPTW instance, solved, and finally the RPDPTW solution is interpreted as a solution for the original MDVRP instance. Similarly, Cordeau et al. (1997) present a tabu search heuristic for both the PVRP and the MDVRP, which capitalizes on the observation that the MDVRP can be formulated as a special case of the PVRP by associating depots with days. Evidently, the degree of flexibility in such an approach is limited by the generality of the underlying problem formulation. A rich formulation such as the RPDPTW may cover a variety of practical applications; when further specific requirements arise, the formulation and the solver need to be enriched accordingly, yet.

Galić et al. (2006) present the interpreted programming language *Mars* together with an interactive graphical tool to simplify the process of developing algorithms for practical VRPs. Besides typical data types, control structures, and mathematical functions this language provides vehicle routing specific data structures, e.g. for customers, vehicles, and tours, and functions to handle these data structures, e.g. to modify tours. Having these components at hand a developer shall be able to focus on the conceptional development of a specific VRP algorithm. The authors evaluate solver implementations based on Mars for the CVRP and the VRPTW. Aiming at rapid prototyping of complex real-world VRPs Carić et al. (2008) present a modeling and optimization framework which provides a library of common VRP algorithms written in another scripting language based on Mars, accompanied by a graphical user interface. An example algorithm is given for the TSP, results are presented for the VRPTW. The question remains whether the two approaches are suitable for more complicated VRPs as well.

4.2 Essential Framework Attributes

Cordeau et al. (2002) define four criteria which are widely considered as essential for good VRP heuristics: *accuracy*, *speed*, *simplicity*, and *flexibility*. We believe that these criteria are reasonable requirements to a VRP software framework in a similar way, yet with a slightly different focus. While in the literature on VRP heuristics the focus is mostly on the two quantitative attributes, accuracy and speed, we put a stronger emphasis on the two qualitative attributes, simplicity and flexibility, in the context of a framework.

Flexibility Cordeau et al. (2002) claim that "a good VRP heuristic should be flexible enough to accommodate the various side constraints encountered in a majority of real-life applications". It is, indeed, a central requirement that the framework enables the development of solution methods addressing as many real-world planning problems as possible which can be attributed to the class of vehicle routing problems. Since requirements from real life are manifold and certainly many "undiscovered" problems have not yet found their way into the literature it would certainly be overconfident to claim that a framework should support *any* thinkable VRP. A sensible guideline is that the framework should not make too many assumptions concerning the VRP variants to be solved but instead give a developer enough flexibility to realize profound adaptations to the underlying heuristic methods.

Simplicity According to Cordeau et al. (2002) simplicity refers to the characteristic to which extent the basic principle of a heuristic is easy to be understood and coded, and whether it does not contain too many (obscure) parameters which have to be configured. This requirement certainly applies to the underlying solution methods of a framework as well: the user must be able to understand how these methods basically work since without that knowledge he or she will not be able to make the necessary adjustments correctly. For this purpose it is helpful to build the framework upon methods which are well-known and accepted in the literature.

Simplicity and flexibility are, in a sense, contradictory requirements: the maximum flexibility comes with handing out the complete source code to the user, giving the possibility to modify or rewrite *every* component of a solution method. Rather overwhelming than assisting, this requires the user to have a deep understanding of the implementation details of the framework. Instead, to guide and to structure the development a framework should restrict the possibilities for customization and offer only a few punctual, yet powerful means to "intervene" into a solution method for problem-specific adaptations. These

points of intervention, properly revealed to the user, serve as a kind of checklist for the adaptation process. In any case the effort of additional implementation should be as small as possible and restricted to the very specific aspects of a rich VRP.

Accuracy and speed The quality of the solutions generated by a framework-based heuristic for a specific VRP is determined by the design of the framework itself only partly. It is reasonable to choose as foundation a solution method which has already been applied successfully to multiple VRPs, assuming that it can perform just as well on other VRP variants. Still, designing and implementing a successful problem-specific adaptation is a complex task that can require a lot of experience on the part of the user. Some VRP variants are more difficult to solve than others; while it is sufficient in some cases to simply forbid a set of moves that violate a certain constraint, new neighborhoods may need to be designed in other cases to explore all parts of the new solution space. Unfortunately, there is usually no possibility to judge the quality of heuristic VRP solutions by other means than by comparison with reference solutions from other methods; optimal solutions or bounds are seldom available, which is a problem when dealing with custom real-world VRPs. Similarly, the speed of a heuristic depends partly on the underlying framework implementation and partly on the efficient coding of adaptations by the user.

4.3 Adaptation Requirements

Balancing flexibility and simplicity, the design of a VRP framework is a compromise between exposing all details of a VRP heuristic to allow any kind of modification on the one hand, and hiding parts of its components to reduce complexity and guide the development process on the other hand. Obviously, generic elements such as metaheuristic controls can be hidden completely from the user, while the actual moves of a neighborhood search are VRP-specific and may require customization. Based on the selection of rich VRPs presented in section 2.2 a set of general characteristic aspects can be identified in which real-world VRPs may extend the standard VRP and which a framework needs to address, consequently.

Solution structure and decisions We assume that all types of VRPs basically share the same structure of a solution: a VRP solution is constituted by a set of tours, and each tour is a sequence of customers visited. This defines the scope of problems we expect to be supported by a VRP framework, implying that especially arc routing problems are not considered. The standard VRP assumption that every customer is visited exactly once is not considered as an integral property of a VRP solution. Multiple visits

of one customer are allowed, for instance, in the SDVRP and in the TTRP, even within the same tour; other scenarios are thinkable that consider visits as optional. Hence, solution structures and heuristics must support a variable number of visits of each customer.

A rich VRP may incorporate more decisions than the pure clustering of the customers and routing of the vehicles. Additional decisions may arise on different, not always clearly distinguishable levels, for instance

- on a *visit-level* the decision which amount to serve to a customer during a specific visit when deliveries may be split, or whether to decouple the vehicle units and park the trailer at a certain location or not,
- on a *tour-level* the arrangement of goods within the vehicle when specific loading constraints apply, and
- on a *solution-level* the decision which periodic delivery pattern to assign to a customer or which number of units of which vehicle type to use in fleet size and mix problems.

Consequently, data structures which represent a solution potentially need to incorporate additional information on these three levels, and new moves need to be designed or existing moves need to be adapted to examine new decision alternatives.

Constraints Usually, vehicle routing scenarios from the real world impose various constraints to be respected. Similar to additional decisions they can arise on multiple levels, for instance

- on a *sequence-level* considering the time window requirements which are related to the actual routing of customers within a tour, or serving the delivery customers before the pickup customers in backhaul scenarios,
- on a *tour-level* the compatibility of a product with a certain vehicle compartment, and
- on a *solution-level* the obligation that under split deliveries all customers are served completely considering all tours.

Additional constraints narrow the feasible solution space compared to the standard VRP, and if a solution approach is not specifically designed to temporarily visit and repair infeasible solutions, these constraints need to be checked during neighborhood evaluation to prohibit the transition from a feasible to an infeasible solution.

Objective function The objective in most VRPs is to minimize cost which is equal to the sum of all distances associated with the arcs of the solution. Yet, the objective function may have additional components such as non-delivery costs or fixed vehicle costs that need to be considered in all neighborhood evaluations. And in some VRP variants not the cost or distance is the primary objective but the number of tours in the solution.

To conclude this section, the most important possibilities for adaptations to a VRP heuristic that a framework needs to offer are

- with respect to the solution algorithms
  - incorporating additional checks and alternative cost calculations into predefined neighborhood evaluations and
  - adding new moves,
- with respect to the data structures
  - enriching the solution representation with further information on the visit-, tour- and solution-level and
  - a variable number of visits of a customer.

Chapter 5

Concepts of the Metaheuristic Framework

The purpose of our metaheuristic VRP framework is to enable an easy, rapid, and structured development of neighborhood search algorithms for the manifold variants of vehicle routing problems arising in real-world transportation scenarios. Its basic idea is to provide

- a complete, ready-to-use solver suite for the standard VRP,
- which offers several mechanisms to be adapted or extended according to the specifics of complex rich VRPs.

The solver suite provides well-known neighborhood search techniques which have already been applied successfully to many VRP variants. In fact, there is a vast amount of publications in this area, and comparing LS- and LNS-based algorithms for different VRPs it becomes evident that algorithmic designs are usually very similar. It is also our own experience that an implementation for one VRP can be reused for another VRP to some degree and that often not "too many" modifications are required to transfer a good heuristic for one problem into a good heuristic for another problem. Our decision to build a VRP framework upon adaptable standard VRP heuristics stems from this experience.

Our presentation of the framework is divided into two major parts: in this chapter we assume a conceptual view which describes the behavior of the embedded heuristics abstracting from any aspects of implementation. Briefly sketching the architecture of our framework in section 5.1, the following sections cover its components in more details: sections 5.2 and 5.3 define the heuristics of the solver suite, and section 5.4 explains how these heuristics can be adapted to other VRPs. Afterwards, chapter 6 deals with the class design and several technical aspects of our implementation.

5.1 Architecture

Figure 5.1 sketches the architecture of the framework and its role within customized algorithms for specific VRPs. In this section we give an overview of the concepts and components.

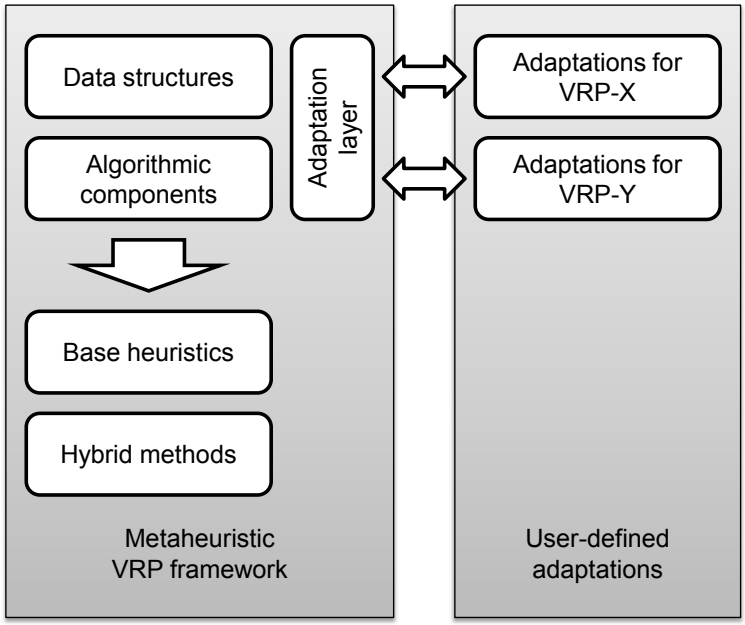


Figure 5.1: Framework architecture.

Data structures and algorithmic components The VRP framework provides standard implementations of the components of several neighborhood search approaches. The underlying *data structures*, especially those for solution representation, are covered in chapter 6. The *algorithmic components* comprise construction heuristics, metaheuristic controls, and neighborhoods – traditional local search neighborhoods as well as removal and insertion heuristics for large neighborhood search:

Construction heuristics	Metaheuristic controls	LS neighborhoods	LNS subheuristics
Savings	SD	Relocate	Random removal
Sweep	ABHC	Exchange	Worst removal
Regret insertion	RRT	2-opt*	Shaw removal
		Relocate^I	Greedy insertion
		2-opt	Regret insertion

Base heuristics The implemented algorithmic components and data structures are put together to a set of standard solution methods, the so-called *base heuristics*: ready-to-use algorithms for the standard VRP that serve as blueprints for problem-specific heuristics. Three base heuristics are implemented:

- LS-ABHC: An attribute based hill climber heuristic with traditional local search neighborhoods
- LS-RRT: A record-to-record travel heuristic with traditional local search neighborhoods
- LNS-RRT: A record-to-record travel heuristic with large neighborhood search moves

All three improvement heuristics, for which we present pseudocodes in section 5.2, can be combined with any of the construction heuristics implemented.

Hybrid methods In addition to the base heuristics that concentrate on a single neighborhood paradigm, respectively, a set of *hybrid methods* is defined to combine local search with large neighborhood search in the search process:

- HYBRID-ABHC: An attribute based hill climber heuristic with traditional local search neighborhoods and large neighborhood search moves
- HYBRID-RRT: A record-to-record travel heuristic with traditional local search neighborhoods and large neighborhood search moves

As the base heuristics these methods are implemented as ready-to-use algorithms for the standard VRP, and we present pseudocodes in section 5.3.

Adaptation layer Generally, modifying a VRP solver implementation for another VRP involves

- modifications of the solution algorithm and, as a consequence,
- modifications of the underlying data structures, in particular the solution representation.

In section 4.3 we deduced specific requirements for such modifications from analyzing a selection of rich VRPs from the literature. We denote the whole set of possibilities and means for adaptation as the *adaptation layer* of the framework. This abstract term covers diverse aspects such as the parts of data structures and algorithmic components which

are exposed to the user, an object-oriented design to create custom data structures by inheritance, schemes for user-defined functions to perform additional checks, and interfaces for the definition of new neighborhoods.

Hiding most implementation details from the user, the adaptation layer is connected with the data structures and algorithmic components only – not with the heuristics – as displayed in figure 5.1. This implies that adaptations, e.g. of a move, are propagated to all derived base heuristics and hybrid methods using this move, not requiring any additional modification of a specific heuristic. Section 5.4 explains the adaptation layer in terms of the algorithmic components provided; other aspects, rather related to the class design, are covered in chapter 6.

User-defined adaptations The code for a specific VRP written by the user is connected to the heuristics via the adaptation layer. User code and framework altogether form a set of solution algorithms for this specific VRP. Examples of adaptations are presented in detail in the last part of this thesis.

5.2 Base Heuristics

This section describes the three base heuristics LS-ABHC, LS-RRT, and LNS-RRT. Following common practice all three heuristics are designed rather straightforward. For the underlying concepts we refer to chapter 3.

5.2.1 The LS-ABHC Heuristic

Starting from a given initial solution the trajectory of an attribute based hill climber heuristic through the solution space is determined by the set of solution attributes A, which is defined over the set $\mathcal{F}$ of feasible solutions, and the set of neighborhoods $\mathcal{N}$ evaluated. In our implementation A contains all directed arcs between the locations of the problem, i.e. $A := \{(u, v) : u, v \in N\}$, and by default we use all neighborhoods $\mathcal{N} := \{\text{relocate, exchange, 2-opt*, relocate}^I\text{, 2-opt}\}$. While scanning these neighborhoods, the following moves are evaluated:

- Relocate: Every customer is moved to every other position in the solution, including the move to a new, empty tour.
- Exchange: Every customer is exchanged with every other customer.

- 2-opt*: Every subsequence of customers of every tour ending at the depot is exchanged with every such subsequence of every other tour, considering also empty subsequences, subsequences comprising the complete tour, and moves of subsequences to a new, empty tour.
- Relocate^I: Every customer is moved to every other position within its current tour.
- 2-opt: Every subsequence of customers of every tour is reversed.

Algorithm 5.1 The LS-ABHC heuristic

```
1:  function LS-ABHC(initial solution S, attributes A, neighborhoods 𝒩)
2:      amem[a] := cost(S)   ∀a ∈ A(S)
3:      amem[a] := ∞   ∀a ∉ A(S)
4:      S_best := S
5:      repeat
6:          S' := null, cost(S') = ∞                          ▷ Evaluate neighborhood
7:          for all S* ∈ N(S), N ∈ 𝒩 do
8:              if cost(S*) < cost(S') and ∃a ∈ A(S*) : cost(S*) < amem[a] then
9:                  S' := S*
10:             end if
11:         end for
12:         if S' ≠ null then                          ▷ Apply move and update memory
13:             S := S'
14:             amem[a] := min{amem[a], cost(S')}   ∀a ∈ A(S')
15:             if cost(S') < cost(S_best) then
16:                 S_best := S'
17:             end if
18:         else                                                  ▷ Reset memory
19:             amem[a] := cost(S)   ∀a ∈ A(S)
20:             amem[a] := ∞   ∀a ∉ A(S)
21:         end if
22:     until stop criterion fulfilled
23:     return S_best
24: end function
```

Algorithm 5.1 outlines the proceeding of LS-ABHC, which is completely generic since A and $\mathcal{N}$ hide all problem-specific aspects. Before starting the search from an initial solution S the attribute memory $amem$, which stores for every attribute $a \in A$ the objective value of the best solution visited so far containing this attribute, is initialized with solution S (lines 2-3). S_{best} records the best solution found during the search. In each iteration all neighborhoods are scanned completely for the best feasible and acceptable neighbor S' of the current solution S (lines 6-11). A solution S^* is acceptable if it contains at least

one attribute that has not yet occurred in a solution visited at least as good as S^*. The acceptance check is implemented efficiently as described by Whittley and Smith (2004). After moving to the best acceptable neighbor S' the attribute memory is updated (line 14). If, instead, no acceptable neighbor exists, the procedure is restarted by resetting *amem* with the current solution (lines 19-20). LS-ABHC terminates when a given time or iteration limit is reached.

5.2.2 The LS-RRT Heuristic

Record-to-record travel is designed to select neighbors randomly. Depending on the specific neighborhood there is some degree of freedom how this random selection is actually done. In the exchange neighborhood, for example, an intuitive selection scheme would be to simply select two random customers for an exchange. When generating such completely random moves, only a single move is evaluated per iteration; the search process makes a lot of iterations, but usually the vast majority of evaluated moves is discarded due to infeasibility or insufficient quality. Instead, we use the selection scheme proposed by Bent and Van Hentenryck (2004) that steers towards high quality neighbors. Here, the selection of a random customer $i \in N_c$ determines a subset of a neighborhood. Such a subset $N(S, i)$ of a neighborhood $N(S)$ contains all neighbors of S that can be reached by a move involving customer i, which implies that for the five VRP neighborhoods the following moves are evaluated:

- Relocate: Customer i is moved to every other position in the solution, including the move to a new, empty tour.
- Exchange: Customer i is exchanged with every other customer.
- 2-opt*: The sequence of customers starting with customer i and ending at the depot is exchanged with every such subsequence of every other tour, considering also empty subsequences, subsequences comprising the complete tour, and moves of subsequences to a new, empty tour.
- Relocate^I: Customer i is moved to every other position within its current tour.
- 2-opt: Every subsequence of customers starting or ending with customer i within its current tour is reversed.

The LS-RRT heuristic is displayed in algorithm 5.2. For a random neighborhood N and a random customer i the feasible solutions of the respective subneighborhood are sorted by ascending objective values (lines 4-5). An aspiration criterion is defined to select the

Algorithm 5.2 The LS-RRT heuristic

```
1: function LS-RRT(initial solution S, deviation δ, randomization β, neighborhoods
   𝒩)
2:     S_best := S
3:     repeat
4:         select neighborhood N ∈ 𝒩 and customer i randomly
5:         array N' := ⟨S* ∈ N(S, i)⟩ with k < l ⇒ cost(N'[k]) ≤ cost(N'[l])
6:         if cost(N'[0]) < cost(S_best) then
7:             S_best := N'[0]
8:             S := N'[0]
9:         else
10:            select r ∈ [0, 1) randomly
11:            S' := N'[⌊r^β · |N'|⌋]
12:            if cost(S') < cost(S_best) · (1 + δ) then
13:                S := S'
14:            end if
15:        end if
16:    until stop criterion fulfilled
17:    return S_best
18: end function
```

best move if this leads to a new best solution (line 6). If this is not the case, a random neighbor is selected: depending on the choice of a randomization parameter β the selection is steered towards good neighbors (lines 10-11). For high values of β the best neighbors are assigned a high probability. The selected neighbor is accepted if it is not worse than the best solution found so far by the allowed deviation δ (line 12). LS-RRT terminates when a given time or iteration limit is reached.

5.2.3 The LNS-RRT Heuristic

The large neighborhood search heuristic of our framework uses a set of removal heuristics $\mathcal{R} := \{$random removal, worst removal, shaw removal$\}$ and a set of insertion heuristics $\mathcal{I} := \{$greedy insertion, regret-2 insertion, regret-3 insertion, regret-4 insertion$\}$. Shaw removal is problem-specific; for the standard VRP the relatedness measure $R(i, j)$ of two customers $i, j \in N_c$ is defined as follows: let c_{max} be the maximal distance between any two customers and q_{max} the maximal difference of demand between any two customers. Then, with weight parameters $\varphi, \psi \geq 0$ the relatedness is

$$R(i, j) := \varphi \cdot \frac{c_{ij}}{c_{max}} + \psi \cdot \frac{|q_i - q_j|}{q_{max}}.$$

Algorithm 5.3 The LNS-RRT heuristic

```
1: function LNS-RRT(initial solution S, deviation δ, removal percentage r, removal
   heuristics R, insertion heuristics I)
2:     S_best := S
3:     repeat
4:         select subheuristics R ∈ R and I ∈ I randomly
5:         select q ∈ {4, ..., min{60, r · |N_c|}} randomly
6:         S' := S
7:         remove q customers from S' using R
8:         reinsert removed customers into S' using I
9:         apply 2-opt steepest descent to modified tours
10:        if cost(S') < cost(S_best) · (1 + δ) then
11:            S := S'
12:        end if
13:        if cost(S') < cost(S_best) then
14:            S_best := S'
15:        end if
16:    until stop criterion fulfilled
17:    return S_best
18: end function
```

LNS-RRT is outlined in algorithm 5.3: at the beginning of each iteration one removal heuristic and one insertion heuristic is selected randomly[1] as well as the number q of customers to be removed (lines 4 and 5). As recommended by Pisinger and Ropke (2007) the maximum value that q may assume depends on a removal percentage parameter r and an absolute upper bound preventing the overburden of the simple insertion heuristics solving large instances. The lower bound is set to 4 as proposed in Ropke and Pisinger (2006a). Then, the two selected subheuristics are applied to the current solution. After removing and reinserting the customers we perform a post-optimization step by applying 2-opt steepest descent to all modified tours (lines 6-9). As in the LS-RRT counterpart the resulting solution is accepted if it is not worse than the best solution found so far by deviation δ. LNS-RRT terminates when a given time or iteration limit is reached.

A variant of LNS-RRT implements a vehicle minimization procedure for VRPs that consider the number of tours as the primary objective, proposed by Pisinger and Ropke (2007). In this two-phase approach vehicles are minimized using LNS-RRT in the first phase, while any improvement heuristic can be used to minimize the total tour length in the second phase. At the beginning of the first phase one tour is selected randomly

[1] Ropke and Pisinger (2006a) introduce a learning mechanism to guide the selection of subheuristics under the name *adaptive* large neighborhood search; such a mechanism is not used within our framework currently but could be implemented easily.

and removed from the solution, which means that its customers are not served any more. Then, removal and insertion operations are performed as usual, until finally a solution is found with all customers being assigned to the remaining tours. Removing another tour the process continues. At a certain point the first phase is terminated, and the second phase for distance minimization starts from the last feasible solution having the minimal number of vehicles; by default, phases are switched when half of the total running time or number of iterations has elapsed.

5.3 Hybrid Methods

In section 3.5 we have pointed out that a good neighborhood search incorporates both intensification and diversification. In addition to the base heuristics our framework provides two ready-to-use algorithms which mix local search moves for the intensification part and large neighborhood search moves for the diversification part. In previous studies (Bartodziej et al., 2009; Derigs et al., 2011a) we have already experienced that combined LS/LNS methods easily improve the individual methods. The first hybrid method is based on the attribute based hill climber, the second method uses record-to-record travel as the metaheuristic control, and both methods are generic, i.e. they do not require any individual problem-specific adaptations.

Instead of combining LS and LNS in a purposeful way and addressing their specific features both HYBRID-ABHC and HYBRID-RRT are naive in the sense that they decide randomly which type of neighborhood to use in each iteration of the search. Setting a probability parameter the selection is biased towards either LS or LNS. Here, one has to keep in mind the different computational efforts per LS iteration in the context of different metaheuristic controls: while LS-ABHC puts significant effort into one iteration and the search process is characterized by a sequence of rather few but high quality moves, a lot of iterations elapse in LS-RRT, but most moves are discarded due to a lack of quality. Different numbers of iterations are required to obtain similar improvements of solution quality.

5.3.1 The HYBRID-ABHC Heuristic

Combining LNS moves with the ABHC control is inconsistent with the *steepest descent – mildest ascent* idea of tabu search since LNS does not scan complete neighborhoods to

Algorithm 5.4 The HYBRID-ABHC heuristic

```
1: function HYBRID-ABHC(initial solution S, balance p^LS)
2:     initialize memory amem as in LS-ABHC
3:     repeat
4:         select r ∈ [0, 1) randomly
5:         if r < p^LS then
6:             select best acceptable neighbor S' as in LS-ABHC
7:             if S' ≠ null then
8:                 S := S'
9:                 update amem as in LS-ABHC
10:            else
11:                reset amem as in LS-ABHC
12:            end if
13:        else
14:            generate neighbor S' as in LNS-RRT
15:            if cost(S') < cost(S) and S' acceptable according to amem then
16:                S := S'
17:                update amem as in LS-ABHC
18:            end if
19:        end if
20:        update best solution if necessary
21:    until stop criterion fulfilled
22:    return best solution found
23: end function
```

find the best solutions within but generates only one (random) neighbor per iteration. Ignoring this conceptual mismatch algorithm 5.4 sketches the HYBRID-ABHC heuristic of our framework. After initializing the attribute memory in the same way as it is done in LS-ABHC the neighborhood search is a sequence of LS and LNS moves, deciding randomly in each iteration between LS (with probability p^{LS}) and LNS (with probability $1 - p^{\mathrm{LS}}$). If a move is accepted, the memory is updated accordingly.

If LS is selected, all neighborhoods are scanned completely for the best acceptable solution, as in LS-ABHC. If such a solution exists, the search goes on to that solution; otherwise the memory is reset. If LNS is selected instead, a neighbor is generated by applying a removal heuristic and an insertion heuristic, as in LNS-RRT. This solution is accepted if a) costs decrease and b) the solution is acceptable according to the memory. Note that b) is the same criterion that applies to LS moves, i.e. a solution S' is acceptable if $\exists a \in A(S'), cost(S') < amem[a]$. Yet, unlike the acceptance check for LS moves described by Whittley and Smith (2004), which considers the sets of entering and leaving attributes beforehand, the attributes of the LNS-generated neighbor are enumerated explicitly to

check whether one attribute value has improved. Since LNS moves lead to distant solutions condition b) is fulfilled very often and hardly imposes a restriction on the quality of a move. Consequently, we introduce a) as a mandatory condition to prevent the search from accepting too many bad solutions. Still, we keep condition b) for LNS moves since demanding a) *and* b) produced slightly better results in some preliminary tests than a) alone.

5.3.2 The HYBRID-RRT Heuristic

While HYBRID-ABHC is slightly unconventional, RRT can be combined perfectly with the two neighborhood concepts. HYBRID-RRT, as shown in algorithm 5.5, determines randomly whether to perform LS or LNS in an iteration. An LS neighbor is selected from a subneighborhood in the same way as it is done in LS-RRT, and an LNS neighbor is generated as in LNS-RRT. In both cases the selected neighbor is accepted if it is not worse than the best solution found so far by deviation δ.

Algorithm 5.5 The HYBRID-RRT heuristic

```
1: function HYBRID-RRT(initial solution S, deviation δ, balance p^LS)
2:     repeat
3:         select r ∈ [0, 1) randomly
4:         if r < p^LS then
5:             select neighbor S' as in LS-RRT
6:         else
7:             generate neighbor S' as in LNS-RRT
8:         end if
9:         if cost(S') is acceptable according to deviation δ then
10:            S := S'
11:        end if
12:        update best solution if necessary
13:    until stop criterion fulfilled
14:    return best solution found
15: end function
```

5.4 Algorithm Adaptation

While the base heuristics and hybrid methods described in the previous sections are problem-independent, VRP-related aspects are covered within the algorithmic components of the framework. Adapting construction heuristics, LS neighborhoods, and LNS subheuristics to specific VRPs follows two main principles:

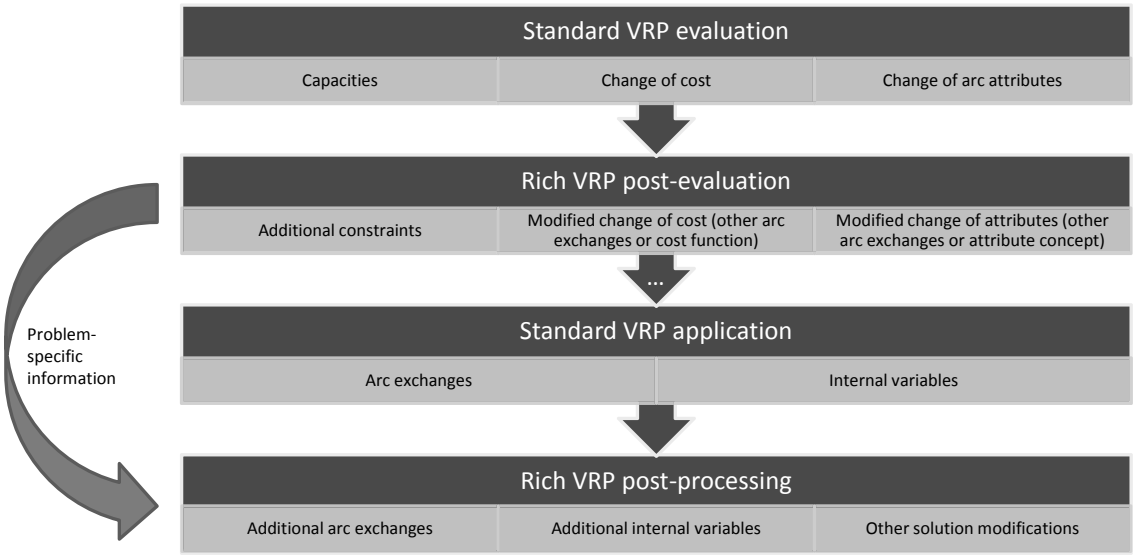


Figure 5.2: Steps of a rich VRP operation.

1. *Separation of standard VRP operations / code and specific VRP operations / code*: The standard VRP heuristics are never modified directly. Their codes are (or can be) hidden from the user and remain untouched. Instead, new specific VRP operations can be implemented separately and "injected" into the given procedures.

2. *Separation of evaluation and application*:
   During the search process a solution is never modified "blindly", i.e. without evaluating feasibility and quality of an operation beforehand. Evaluation and application of a move are independent tasks, and adaptations may be required separately on these two levels.

Figure 5.2 visualizes an adapted operation under these two principles of separation. Evaluation proceeds as follows:

1. Predefined standard VRP evaluation

   - The capacity constraints are checked first, and if the move is not a feasible standard VRP move, it is rejected.
   - If the move is feasible, the increase of solution cost Δ is calculated according to the distances associated with the arcs exchanged.
   - Within an ABHC heuristic the entering arc attributes A^e and leaving arc attributes A^l are determined simultaneously with cost calculation.

2. User-defined *post-evaluation* for a rich VRP

- The user can define additional sequence-, tour-, or solution-level checks; if the move, which is feasible for the standard VRP, now turns out to be infeasible, it is rejected.
- The properties of a move in the standard VRP case, Δ, A^e, and A^l, may have to be overridden when a different behavior is assumed. Problem-specific properties $\bar{\Delta}$, $\bar{A}^e$, and $\bar{A}^l$ need to be determined if other or additional arc exchanges are involved, for instance. Δ also needs to be overridden in the case of an alternative cost function; A^e and A^l require modifications when using an attribute concept other than directed arcs.

The application of an adapted move to the solution follows a similar principle:

1. Predefined standard VRP application

   The solution is modified according to standard VRP behavior first, which includes arc exchanges and updating of some internal variables to keep track of the current distances, loads etc.

2. User-defined *post-processing* for a rich VRP

   Post-processing steps may involve simple updates of redundant cache variables for speeding up feasibility checks, additional arc exchanges, the addition or removal of customer visits, or setting specific decision variables associated with a rich VRP. The post-processing can even start with undoing the complete standard VRP move first before performing the actual modifications.

Potentially, certain decisions are made during post-evaluation which have an influence on the post-processing later on; or certain information is gathered during the check which is required for post-processing but rather time-consuming to be generated for a second time. For such purposes the user is enabled to pass some problem-specific information Φ from the post-evaluation step to the post-processing step, as displayed in figure 5.2.

In the following sections we present the functions that can be implemented by the user to inject problem-specific code for checks and post-processing into the standard VRP heuristics. This is done for every construction heuristic (section 5.4.1), LS neighborhood (section 5.4.2), and LNS operation (section 5.4.3). The pseudocodes contain the following additional symbols: $tours(S)$ denotes the set of tours of a solution S, $customers(t)$ denotes the set of customer visits during a tour t, and $locations(t)$ contains $customers(t)$

plus the depot. Φ^t and Φ^s represent extra problem-specific information to be passed from checks to post-processing, referring to tour-level and sequence-level information, respectively.

5.4.1 Construction Heuristics

Three construction heuristics are implemented in the framework: the savings heuristic, the sweep heuristic, and an insertion-based heuristic that uses regret insertion to construct an initial solution from scratch by inserting all customers into an empty solution. For the latter heuristic we refer to the adaptation of LNS operations presented in section 5.4.3.

Savings Heuristic

The savings heuristic, see algorithm 5.6, first creates a solution containing one separate tour ($d \to i \to d$) for each customer $i \in N_c$. Then, in the process of repeatedly connecting tours function CheckMerge is used to evaluate the merging of any two tours t_1 and t_2. By default, CheckMerge checks the capacity constraint and calculates the cost increase Δ (the negative savings value). A user-defined CheckMerge function may check additional constraints, calculate a different saving, and return some problem-specific information Φ to be used later on during the merging. At the end of an iteration tours t_1^{best} and t_2^{best} associated with the highest saving are connected by the user-definable function Merge. When no feasible merge exists any more, each tour is improved with 2-opt steepest descent.

Algorithm 5.6 User-definable functions in savings heuristic

```
1: function Savings(customers N_c)
2:     S := solution with one tour for each customer i ∈ N_c
3:     repeat
4:         Δ^best := ∞, t_1^best, t_2^best, Φ^best := null
5:         for all t_1, t_2 ∈ tours(S), t_1 ≠ t_2 do
6:             (Δ, Φ) := CheckMerge(t_1, t_2)
7:             if Δ < Δ^best then
8:                 (Δ^best, t_1^best, t_2^best, Φ^best) := (Δ, t_1, t_2, Φ)
9:             end if
10:        end for
11:        Merge(t_1^best, t_2^best, Φ^best)
12:    until no feasible merge exists
13:    apply 2-opt steepest descent to tours(S)
14:    return S
15: end function
```

Sweep Heuristic

Initializing the sweep heuristic, see algorithm 5.7, all customers $c \in N_c$ are sorted by their polar angles θ_c around the depot and appended to solution S, which initially contains only one empty tour, in the resulting sequence. Function AppendCustomer, as implemented for the standard VRP, inserts a customer at the end of the most recently opened tour if the capacity of the vehicle is not exceeded; otherwise a new tour is added for that customer. The user can implement a modified AppendCustomer function to check additional constraints, for instance. When all customers have been inserted into the solution, each tour is improved with 2-opt steepest descent.

Algorithm 5.7 User-definable functions in sweep heuristic

```
1: function Sweep(customers N_c)
2:     array C' := ⟨c ∈ N_c⟩, with i < j ⇒ θ_C'[i] ≤ θ_C'[j]
3:     S := solution with one empty tour
4:     for i ∈ 0..|C'| − 1 do
5:         AppendCustomer(C'[i], S)
6:     end for
7:     apply 2-opt steepest descent to tours(S)
8:     return S
9: end function
```

5.4.2 Local Search Neighborhoods

Generally, multiple user-definable functions are available to check tour-level constraints and sequence-level constraints separately during the exploration of the built-in inter-tour neighborhoods, while intra-tour neighborhoods only provide a single checking function. In addition, a post-processing function can be defined for every neighborhood to finish a move. We can explain all relevant aspects of neighborhood adaptation using the example of the relocate neighborhood; hence we provide the details for relocate first and then state the specifics of the remaining neighborhoods rather briefly.

Relocate

Exploring the relocate neighborhood (algorithm 5.8) the CheckRelocateBetweenTours(t_1, t_2) function checks for any two tours t_1 and t_2 whether a relocation of a customer from tour t_1 to tour t_2 is generally allowed. Here, t_2 may also be a new, empty tour. If no constraints are violated on this level, the procedure iterates over all customers c of

t_1 and checks whether c would exceed the capacity of t_2 in a relocation or not. CHECKRELOCATEINTOTOUR(c, t_2) can check additional tour-level constraints afterwards and return some information Φ^t. Iterating over all positions i within tour t_2, cost increase Δ and attribute exchanges A^e and A^l of relocating c behind i are determined according to standard VRP behavior. CHECKRELOCATE(c, i, Φ^t), potentially making use of Φ^t, can finally check additional sequence-level constraints and adjust cost and attribute information on the move if necessary: the function returns modified move properties $\bar{\Delta}$, $\bar{A}^e$, and $\bar{A}^l$ and, again, some additional problem-specific information Φ^s.

Algorithm 5.8 User-definable functions in relocate

```
 1: function RELOCATE(solution S)
 2:     for all t1, t2 ∈ tours(S), t1 ≠ t2 do
 3:         CHECKRELOCATEBETWEENTOURS(t1,t2)
 4:         for all c ∈ customers(t1) do
 5:             check capacity
 6:             Φ^t := CHECKRELOCATEINTOTOUR(c,t2)
 7:             for all i ∈ locations(t2) do
 8:                 determine Δ, A^e, and A^l
 9:                 (Δ̄, Ā^e, Ā^l, Φ^s) := CHECKRELOCATE(c,i,Φ^t)
10:                 check acceptability of (Δ̄, Ā^e, Ā^l)
11:                 store move (c, i, Δ̄, Φ^t, Φ^s)
12:             end for
13:         end for
14:     end for
15: end function
```

If the neighborhood is explored within an ABHC heuristic, the acceptability of a feasible move is checked considering $\bar{\Delta}$, $\bar{A}^e$, $\bar{A}^l$, and the current state of the ABHC memory; and if a better acceptable move has already been found during the current iteration, it is discarded. Within RRT any feasible move is a candidate for being selected by the metaheuristic control, and all information, including Φ^t and Φ^s, is stored for potential application later on.

A relocate move that is actually selected by the metaheuristic control is applied to the current solution according to standard VRP behavior first, and a post-processing function is called immediately after that: RELOCATEPOSTPROCESSING$(c, i, j, \Phi^t, \Phi^s)$ finishes the relocation of a customer c from position j to position i, using information Φ^t and Φ^s generated during the checks.

Exchange

During the exploration of the exchange neighborhood (algorithm A.1 in the appendix) CheckExchangeBetweenTours(t_1, t_2) checks for any two tours t_1 and t_2 whether an exchange of customers is generally allowed. If this is the case, all exchanges of customers c_1 and c_2 between the tours are evaluated: if the capacity check passes for both tours, CheckExchange(c_1, c_2) may consider further constraints, modify Δ, A^e, and A^l and return some additional information Φ^s. After an exchange post-processing function ExchangePostprocessing(c_1, c_2, Φ^s) is called.

2-opt*

Exploring the 2-opt* neighborhood (algorithm A.2 in the appendix) CheckTwoOpt-StarBetweenTours(t_1, t_2) checks for any two tours t_1 and t_2 whether exchanges of customer sequences are generally allowed. Here, t_2 may also refer to a new, empty tour. For all combinations of locations c_1 and c_2 the procedure evaluates the exchange of sequences beginning immediately after c_1/c_2 and ending at the depot, respectively. The capacity constraints are checked first, and then CheckTwoOptStar(c_1, c_2) may check further problem-specific constraints, modify Δ, A^e, and A^l and return some additional information Φ^s. Function TwoOptStarPostprocessing(c_1, c_2, Φ^s) can be defined by the user for post-processing.

Relocate^I

The relocate^I neighborhood (algorithm A.3 in the appendix) considers any move of a customer c to another position i within the same tour. Since it is an intra-tour neighborhood tour-level checks such as capacity checks are not required. Δ, A^e, and A^l are determined for the standard VRP case first, then CheckRelocateI(c, i) may modify these properties, check problem-specific sequence-level constraints, and return some additional information Φ^s. Function RelocateIPostprocessing(c, i, j, Φ^s) is called after the relocation of customer c from position j to position i for completing the move.

2-opt

A 2-opt move removes two arcs from a tour and reconnects the remaining tour parts, which is equivalent to the reversal of a customer sequence. Hence, exploring the 2-opt neighborhood (algorithm A.4 in the appendix) involves evaluating any reversal of customer sequences from a customer c_1 to a customer c_2 within a tour t. Again, no tour-level checks are required. CheckTwoOpt(c_1, c_2) may consider problem-specific sequence-level

constraints, modify the previously generated information on cost Δ and attribute changes A^e/A^l of the move, and return some additional information Φ^s. The post-processing function TwoOptPostprocessing(c_1, c_2, Φ^s) is called at the end of the move.

5.4.3 Large Neighborhood Search Operations

Adapting an LS-based heuristic for a specific VRP requires the implementation of separate checking and post-processing functions for every neighborhood used, which can be rather time-consuming. The subheuristics of LNS are all based on just two single operations: the removal of a customer and the reinsertion of a customer. Consequently, no major adaptations are required except for these basic operations.[2]

Removal Operation

The three removal heuristics of the framework take into account whether it is feasible to remove a customer from its current tour or not, and the worst removal subheuristic also considers the associated change of cost. The (trivial) evaluation procedure of a removal is shown in algorithm 5.9: after calculating the cost increase Δ – actually a decrease of cost in the standard VRP case if the triangle inequality holds – function CheckRemoval(c) can assess feasibility concerning problem-specific constraints, alter Δ, and return some additional information Φ^s on the removal. After a removal is performed the post-processing function RemovalPostprocessing(c, Φ^s) is called.

Algorithm 5.9 User-definable functions in LNS removals

```
1: function EvaluateRemoval(customer c)
2:     determine Δ
3:     (Δ̄, Φ^s) := CheckRemoval(c)
4: end function
```

Insertion Operation

Insertion heuristics determine the best positions to insert unassigned customers; algorithm 5.10 displays the process of finding the best feasible position for a customer c in a tour t. After checking the capacity of t the function CheckInsertionIntoTour(c, t) determines whether c can be inserted into t with respect to problem-specific tour-level

[2]Individual subheuristics used within LNS may be designed for specific VRPs. Among the subheuristics implemented in our framework only shaw removal requires customization: the definition of the relatedness measure.

constraints. Some additional information Φ^t may be generated during this check. Then, for each potential insertion position i the regular cost increase Δ is calculated first, and a problem-specific check is performed by method CheckInsertion(c, i, Φ^t), which may alter Δ and return more information Φ^s. The cheapest feasible insertion position is stored and function InsertionPostprocessing(c, i, Φ^t, Φ^s) is called for post-processing after the actual insertion.

Algorithm 5.10 User-definable functions in LNS insertions

```
1: function EvaluateInsertions(customer c, tour t)
2:     check capacity
3:     Φ^t := CheckInsertionIntoTour(c,t)
4:     for all i ∈ locations(t) do
5:         determine Δ
6:         (Δ̄, Φ^s) := CheckInsertion(c,i,Φ^t)
7:         store insertion (c, i, Δ̄, Φ^t, Φ^s)
8:     end for
9: end function
```

Chapter 6

Design of the Metaheuristic Framework

The design of our framework follows the guidelines of *object-oriented programming* (OOP). According to the principles of *separation of concerns*, *information hiding*, and *encapsulation* different aspects of the algorithms such as solution representation, metaheuristic controls, neighborhood evaluation, and application of moves are implemented in separate classes in such a way that changes concerning one component of the algorithm do not force changes in other components. For example, the solution representation is encapsulated in a set of classes allowing access through public methods. Neighborhood-related components call these methods to apply moves to a solution, but they do not need to know the details of internal operations on data structure level. Similarly, encapsulating the logic of moves within separate neighborhood classes, neighborhoods can be added or modified flexibly.

We have implemented the framework in the C# programming language, based on the Microsoft .NET framework. Without a doubt any other modern object-oriented language can be used to implement the framework in a similar and appropriate way; .NET techniques, which are actually used rarely, may be substituted by alternative programming language features.

The main challenge considering the design is handling the cooperation between the two sides of a framework-based VRP heuristic: the data structures, algorithmic components, and solvers provided by the framework on the one side, and all user-defined adaptations on the other side. Now, we briefly list some techniques which are specifically used to realize the adaptation layer of the framework:

- *Inheritance* is used for the definition of problem-specific data structures. The framework provides base classes for the standard VRP, e.g. for the solution representation, which can be reused by deriving subclasses for a rich VRP. These subclasses are enriched with additional fields for specific information, or additional methods. Some methods of the base classes are marked as *virtual methods*, which means that their behavior can or must be reimplemented within a derived class by *overriding*.

- *Polymorphism* allows derived classes to be used in different contexts. Due to the strict separation of standard VRP code and specific VRP code the built-in heuristics on the framework-side do not know the new classes defined for a specific VRP. However, they can treat objects instantiated from these specific subclasses as objects of the standard VRP base classes and perform operations according to standard VRP behavior. Conversely, objects passed to the checking and post-processing functions on the user-side must be *casted* to the respective derived classes to access the additional information for a specific VRP.

- *Reflection* is a .NET technique to determine information about loaded assemblies and the types defined within, such as classes, at run time. It also involves *late binding* which allows loading a type at run time that is not known at compile time and then creating and using an instance of it. Here, reflection enables methods on the framework-side to create instances of user-defined data structures.

- *Abstract classes* and *interfaces* serve as blueprints for new algorithmic components extending a heuristic, new neighborhoods for instance. Abstract methods as well as interface methods or properties demand a certain behavior that a new, derived neighborhood class needs to fulfill.

- *Delegates* in the .NET framework are types to define the signature of a method; delegate objects are used to reference and call a method, hence they are comparable to function pointers in the C programming language. Here, they provide the templates for user-definable checking and post-processing functions. They enable methods on the framework-side to invoke such user-defined functions which are not known at compile time.

This chapter presents the most important classes of our framework implementation. “Most important” in this context refers to the classes that a user gets in touch with when adapting heuristics and embedding them into a DSS; these classes represent the adaptation layer introduced in section 5.1. At first, section 6.1 covers the classes for problem instances and solutions. Algorithmic component classes are presented in section 6.2. Finally, see section 6.3 for the single class used to encapsulate the complete solver suite.

6.1 Data Structure Classes

Representations for problem instances and their solutions are the most important data structures within the implementation of a VRP heuristic. While instance classes are usually simple and straightforward, solution classes must not be a computational bottleneck of the search process and need to allow efficient access and modifications of the tours, instead. Figure 6.1 shows a graphical overview of the classes presented in this section.

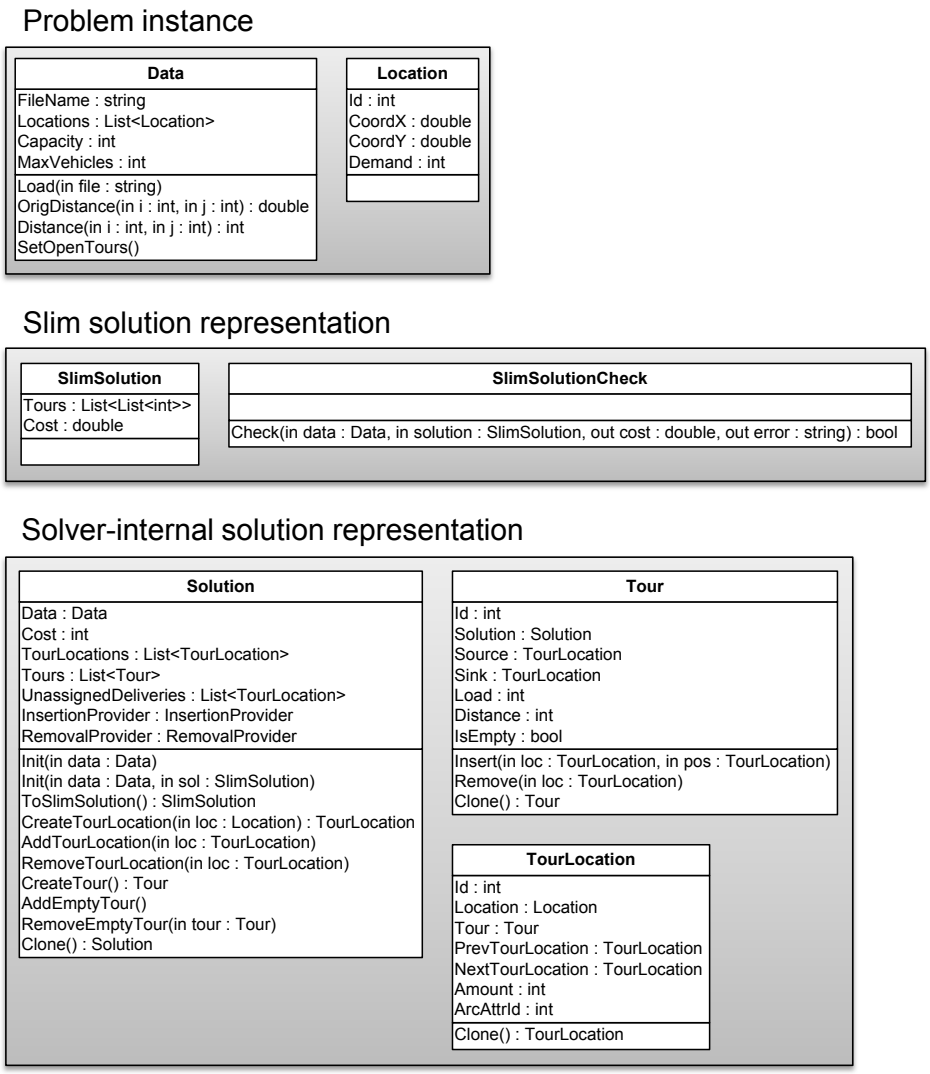


Figure 6.1: Data structure classes.

6.1.1 Problem Instance Classes

The Data class and the Location class together hold all information of a standard VRP instance. While Data represents an instance as a whole, Location covers the specific

information attributed to a depot or customer. If a specific VRP involves more problem information than the standard VRP, new classes can be derived to provide the additional data.

The *Data* Class

DATA provides a loading method to read a standard VRP instance from a text file which has to comply with a certain format commonly used in the VRP literature. In a derived DATA class this method is usually overridden with a new method reading additional information and expecting a modified file format.

The class holds the vehicle capacity, the maximum number of vehicles[1], the set of locations, and the distances between them. By convention, the first location given is always considered to represent the depot. The distance matrix is provided twice: DATA calculates the "original" floating-point Euclidian distances from the given coordinates, but for the purpose of computational efficiency it also provides rounded integer distances between the locations, which are obtained by multiplication with a factor (1000, by default) and rounding. In our implementation all distance/cost calculations are based on these integer distances.

Any VRP instance can easily be converted to its counterpart with open tours (see section 2.2.6) by setting the distances of all arcs ending at the depot to zero. For this purpose DATA provides a method that can be overridden if further problem-specific adjustments are necessary for open tours.

Note that in the following tables listing the properties and methods of classes we mark those items with a * which can or must be overridden for a specific VRP (virtual properties or methods).

Data	
Property/Method	*Description*
FileName	Instance file name.
Locations	List of all locations: depot first, followed by customers.
Capacity	Vehicle capacity.
MaxVehicles	Maximum number of vehicles.

[1]Note that the constraint for a maximum number of vehicles is ignored in our current implementation since in the most frequently used standard VRP instances this number is often set to a very high value.

Data (continued)	
Property/Method	*Description*
*Load($file$)	Read instance from $file$.
OrigDistance(i,j)	Floating-point Euclidian distance between locations i and j.
Distance(i,j)	Rounded integer distance between locations i and j.
*SetOpenTours()	Adjust distance matrix for open tours.

The *Location* Class

LOCATION provides all information about a customer or the depot – the coordinates of the location and the demand. Every location is assigned a zero-based identifying number, corresponding to its position within the list of all locations in DATA. The demand and the ID of the depot are both set to zero.

Location	
Property/Method	*Description*
ID	Zero-based ID.
CoordX	X-coordinate of location.
CoordY	Y-coordinate of location.
Demand	Customer demand.

6.1.2 Solution Classes

The framework provides two types of solution representations. One representation supports efficient operations on a solution to be used by the heuristics, while the other representation is designed for communication between the heuristics and the embedding system, e.g. a DSS. The latter representation, implemented in class SLIMSOLUTION, is a simple string-based representation; the solver-internal representation with classes SOLUTION, TOUR, and TOURLOCATION is based on doubly-linked lists and is much more complex. Figure 6.2 sketches the use of the two formats for a solver which is embedded into another system. An initial solution in the "slim" format is provided by the DSS, imported into the solver and converted to the internal format. The search is started from this solution, and when it terminates, the best solution found is finally exported into the slim format again and, potentially, interpreted and displayed to the DSS user.

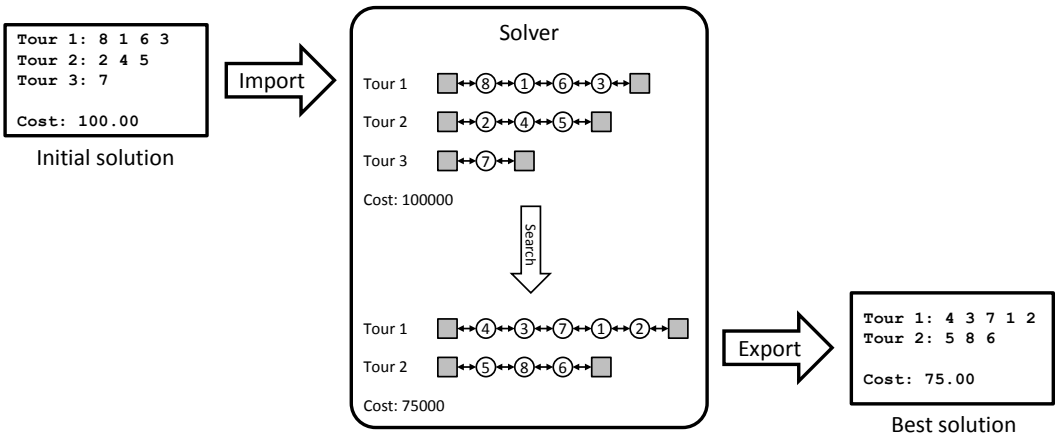


Figure 6.2: Interplay of solution representations.

All four classes can be enriched with more information by deriving new classes for a specific VRP. In the following we briefly give a few information on the slim format and then cover the solver-internal classes in detail.

The *SlimSolution* and *SlimSolutionCheck* Classes

SlimSolution holds a list of the tours, where each tour is simply represented by a list of customer IDs. In addition, it stores the solution cost value.

Especially during the development phase it is a good strategy to implement a procedure for feasibility checks and cost calculation, which is based on the slim format and is completely independent of the solver. Called at the end of the search it is useful to discover potential errors in the solver-internal checking mechanisms, and it also provides a cost value which is more accurate than the cost values calculated by the solver using rounded integer distances. The SlimSolutionCheck class provides methods to check the integrity and the feasibility of a SlimSolution for a given instance. New checks can be added by deriving a problem-specific class.

The *TourLocation* Class

TourLocation represents the visit of a location during a tour; this also includes departures from and returns to the depot. Every non-depot visit is assigned a zero-based ID which corresponds to its position within the list of visit objects maintained by the Solution class (see below). A delivery amount is associated with each visit, which equals

the customer's demand by default but which may be different in a specific VRP. For all solver-internal solution classes a cloning method is provided to create copies of solution objects on certain occasions during the search process. These methods may need to be overridden in derived classes to incorporate problem-specific information during cloning.

Beside the ID mentioned above another special identifier is defined for each visit: the ArcAttrId property is used within ABHC heuristics to associate visits with nodes and arcs (arc attributes). By default, ArcAttrId corresponds to the ID of the associated Location, so that ArcAttrIds of different TourLocations are different as well (except for depot visits). Yet, to associate different, independent visits with the same node in the graph, their ArcAttrId properties must be set to the same number. Then, if for two pairs of TourLocations (v_1, v_2) and (w_1, w_2) the ArcAttrIds of v_1/w_1 and of v_2/w_2 are identical, respectively, the connecting arcs $v_1 \rightarrow v_2$ and $w_1 \rightarrow w_2$ are considered identical as well and refer to the same solution attribute.[2]

TourLocation	
Property/Method	*Description*
ID	Zero-based ID.
Location	Associated Location of problem instance.
Tour	Tour which the visit is assigned to.
PrevTourLocation	Previous TourLocation within the tour.
NextTourLocation	Next TourLocation within the tour.
*Amount	Amount of goods delivered to the customer.
*ArcAttrId	ID to associate visits with ABHC arc attributes.
*Clone()	Create a copy of the TourLocation object.

The *Tour* Class

Tour stores a tour of a solution as a doubly-linked list of the TourLocations visited, including nodes for the depot at the beginning and at the end of the list. Every tour is assigned a zero-based ID which corresponds to its position within the list of tour objects maintained by the Solution class (see below). Properties for the current vehicle load and the total distance traveled are maintained consistently by the methods of inserting customers and removing customers.

[2] An application of the ArcAttrId property is explained in section 7.2 for our VRPC adaptation.

Tour	
Property/Method	*Description*
ID	Zero-based ID.
Solution	Solution which the tour belongs to.
Source	TourLocation node for the departure from the depot.
Sink	TourLocation node for the return to the depot.
Load	Load of the vehicle conducting the tour.
Distance	Distance of the tour.
IsEmpty	Indicates whether the tour is empty.
Insert(*loc*,*pos*)	Insert TourLocation *loc* into the tour behind node *pos*.
Remove(*loc*)	Remove TourLocation *loc* from the tour.
*Clone()	Create a copy of the Tour object.

The *Solution* Class

Solution manages the sets of tours and customer visits which are part of a solution. It provides methods to add new, empty tours (Tour objects) to the solution and, conversely, to remove empty tours from the solution again. Similarly, it provides methods to add and remove TourLocation objects: this feature is not used by standard VRP heuristics since every customer is visited exactly once and the set of visit objects always remains unchanged. Yet, for rich VRPs allowing to visit customers multiple times it can be necessary to change the number of visits dynamically during the search. *Factory methods* are used to create new Tour and TourLocation objects. If new classes are derived for a specific VRP, these factory methods need to be overridden to create objects of the derived classes, respectively.[3]

Customers may (temporarily) not be assigned to a tour, for example during the course of an LNS iteration. The Solution class provides a separate list to access the associated visit objects directly. Each solution is also associated with caches for feasible insertion and removal operations, the so-called InsertionProvider and RemovalProvider (see below). Finally, import and export methods allow the conversion between the Solution format and the SlimSolution format.

[3]The factory method pattern is an OOP design pattern to solve the problem of creating objects without knowing the exact class. Here, it allows to create objects of user-defined Tour and TourLocation classes within methods which are defined on the framework-side.

Solution	
Property/Method	*Description*
Data	Corresponding problem DATA.
Cost	Solution cost, i.e. total distance traveled.
TourLocations	List of TOURLOCATIONs, without depot nodes.
Tours	List of TOURS.
UnassignedDeliveries	List of TOURLOCATIONs currently not assigned to a tour.
InsertionProvider	INSERTIONPROVIDER cache.
RemovalProvider	REMOVALPROVIDER cache.
*Init(*data*)	Initialize an empty solution for DATA.
*Init(*data*,*sol*)	Import a SLIMSOLUTION *sol* for DATA.
*ToSlimSolution()	Export the solution to a SLIMSOLUTION.
*CreateTourLocation(*loc*)	Create a new TOURLOCATION object for LOCATION *loc* (factory method).
*AddTourLocation(*loc*)	Add a TOURLOCATION *loc* to the solution.
*RemoveTourLocation(*loc*)	Remove a TOURLOCATION *loc* from the solution.
*CreateTour()	Create a new TOUR object (factory method).
AddEmptyTour()	Add a new, empty tour to the solution (using CREATETOUR).
*RemoveEmptyTour(*tour*)	Remove an empty TOUR from the solution.
*Clone()	Create a copy of the SOLUTION object.

6.2 Relevant Algorithmic Component Classes

As stated in section 5.1 the algorithmic components of our framework include construction heuristics, metaheuristic controls, LS neighborhoods, and LNS subheuristics. This section presents the main classes which implement these components *and* which the framework user comes into contact with; figure 6.3 displays an overview of the classes.

- Construction heuristic classes are presented in section 6.2.1. Overriding certain methods of these classes the user can add checks and potentially modify the behavior of the heuristics.
- Metaheuristic controls are not addressed here; since they are generic and never need to be adapted to a specific VRP, their implementation is hidden from the user. The only aspects which are relevant in this context are related to the ABHC: in section 6.2.2 we explain how problem-specific ABHC attributes can be defined.

- We present two interfaces for the implementation of new LS neighborhoods in section 6.2.3. The built-in neighborhoods are not modified by the user directly but through the definition of checking and post-processing functions; we give examples of these functions as well.

- In section 6.2.4 we describe classes which are relevant for the implementation of LNS subheuristics.

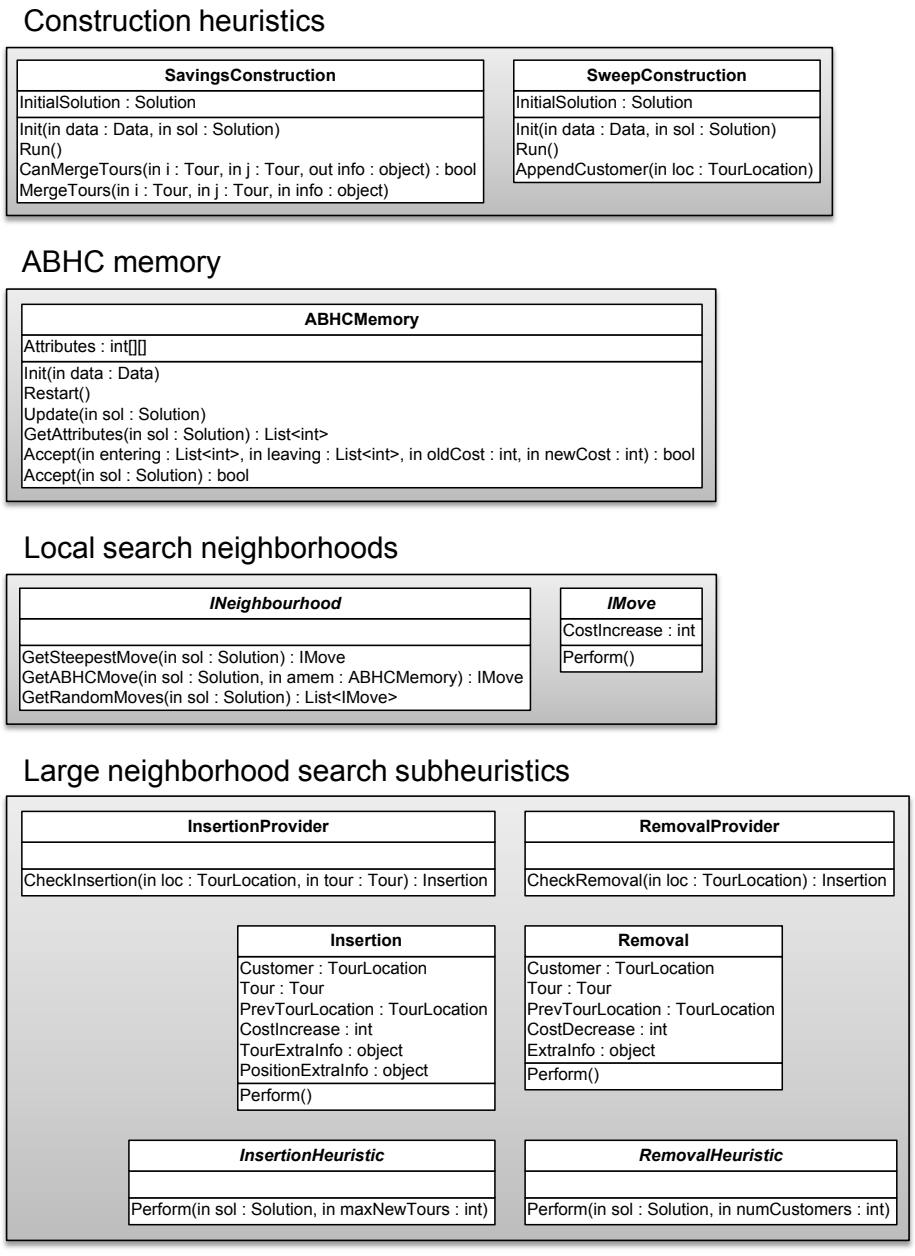


Figure 6.3: Algorithmic component classes.

6.2.1 Construction Heuristic Classes

SavingsConstruction and SweepConstruction are the two construction heuristic classes provided by the framework. Both heuristics are started from an "empty" solution not containing any tour, which needs to be passed to an initialization method, respectively. Note that the framework also allows to use the regret insertion heuristic of LNS for construction; we refer to section 6.2.4 describing the LNS classes.

The *SavingsConstruction* Class

The implementation of the savings heuristic is mainly based on a checking method and a merging method. Starting from the empty solution the heuristic first creates an individual tour for each customer, then continues merging tours: method CanMergeTours is used to determine whether two tours can be appended feasibly and to calculate the cost savings. Method MergeTours actually merges two tours. To adapt the savings heuristic these two methods can be overridden in a derived SavingsConstruction class. As described in section 5.4.1 the checking method may generate some problem-specific information which is used later on when appending the tours.

SavingsConstruction	
Property/Method	*Description*
InitialSolution	Constructed initial Solution.
Init(*data*,*sol*)	Initialize with Data and empty Solution *sol*.
Run()	Start the savings heuristic.
*CanMergeTours(i,j)	Determine whether Tours i and j can be merged, return the savings value and, potentially, some additional problem-specific information Φ.
*MergeTours(i,j,Φ)	Merge Tours i and j, potentially using additional problem-specific information Φ.

The *SweepConstruction* Class

The implementation of the sweep heuristic uses one single method to handle the addition of a customer to the tours of a temporary solution. Starting from the empty solution the sequence of customers reflecting their angles is generated first; then the solution is filled customer by customer, repeatedly calling the AppendCustomer method. By default, this method inserts a customer at the end of the last tour if the capacity is not exceeded,

or opens a new tour otherwise. It can be overridden in a derived SweepConstruction class for a specific VRP.

SweepConstruction	
Property/Method	*Description*
InitialSolution	Constructed initial Solution.
Init(*data*,*sol*)	Initialize with Data and empty Solution *sol*.
Run()	Start the sweep heuristic.
*AppendCustomer(*loc*)	Insert TourLocation *loc* at the end of the last tour or open a new tour.

6.2.2 Metaheuristic Classes

As explained above, the handling of ABHC attributes is the only aspect related to metaheuristics covered in this section.

The *ABHCMemory* Class

ABHCMemory encapsulates the attribute memory, which is an array that holds for each attribute the cost value of the best solution visited so far containing this attribute. It provides methods for the acceptance check and for updating the memory when a solution has been accepted. Internally, it maintains a list of the currently "worst" attributes, which is required for the efficient acceptance check described by Whittley and Smith (2004). Actually, two acceptance checks are provided: the regular, efficient check just mentioned is used for moves *before* being applied to the solution. Based on the entering and leaving attributes A^e/A^l and on the change of solution cost Δ involved it is used for LS moves during LS-ABHC and HYBRID-ABHC. The second check is used for moves *after* being applied to a solution, which is the case for the LNS moves of HYBRID-ABHC.

Solution attributes for the ABHC are represented by integer numbers. The attributes referring to directed arcs between the locations in N, considered for the standard VRP, are mapped to numbers $0, \ldots, |N|^2 - 1$, and an auxiliary function is provided by the framework which simply calculates the number representing the arc from location i to location j by $i \cdot |N| + j$ (with i and j being the zero-based locations IDs). If an alternative set of attributes shall be defined for a specific VRP, these new attributes have to be mapped to a sequence of numbers starting with $|N|^2$. In this case a new ABHCMemory class must

be derived which overrides the initialization method to allocate a memory array with the proper dimension. Method GetAttributes, which retrieves the list of all attributes of a solution and which is used for the second acceptance check, needs to return the new attributes as well. Finally, the new attributes must be considered within the evaluation methods of all LS neighborhoods used, see section 6.2.3.

ABHCMemory	
Property/Method	*Description*
Attributes	Array holding the attribute memory *amem*.
*Init(*data*)	Initialize attribute memory for Data.
Restart()	Reset the attribute memory.
Update(*sol*)	Update the attribute memory with Solution *sol*.
*GetAttributes(*sol*)	Return a list of all attributes of Solution *sol*.
Accept(A^e,A^l,c,c')	Determine whether a move can be accepted which involves entering attributes A^e and leaving attributes A^l, and which changes the solution cost from c to c'.
Accept(*sol*)	Determine whether Solution *sol* can be accepted.

6.2.3 Local Search Neighborhood Classes

The separation into evaluation and application pointed out in section 5.4 is reflected in the design of the neighborhood-related classes. For the traditional LS-based neighborhoods the interface INeighborhood demands the implementation of (up to) three evaluation methods which return IMove objects representing feasible moves. Via the IMove interface a metaheuristic control can query the associated change of solution cost and trigger the application of the move. For each custom neighborhood two classes need to be implemented according to these interfaces.

The *INeighborhood* Interface

INeighborhood demands three methods to explore the neighborhood of a solution, covering feasibility checks and cost calculations. Each method is designed for one specific metaheuristic control: GetSteepestMove is used during steepest descent and has the purpose to scan the complete neighborhood of the current solution and return the best feasible move. Similarly, GetABHCMove is called by ABHC heuristics to obtain the best feasible *and* acceptable move. An ABHCMemory object is passed to this method

as a parameter, so that the method can query whether a move is acceptable or not. For RRT, finally, GetRandomMoves has the purpose to select a random subset of the neighborhood and return all feasible moves within. The mechanism of selecting a specific subset of neighbors is left to the user for a custom neighborhood.

INeighborhood	
Method	*Description*
GetSteepestMove(*sol*)	Return the best feasible move applicable to solution *sol*.
GetABHCMove(*sol*,*amem*)	Return the best feasible move applicable to solution *sol* which is acceptable based on the current state of the ABHCMemory *amem*.
GetRandomMoves(*sol*)	Return a list of feasible, random moves applicable to solution *sol*.

The *IMove* Interface

An IMove object represents a feasible move and holds all the logic and information required to modify the solution – such information can be the customers, positions, or tours involved, or the problem-specific information Φ for the post-processing step in the case of the five predefined neighborhoods. The change of solution cost is provided as well, and the move can be triggered by calling a specific method.

IMove	
Property/Method	*Description*
CostIncrease	Increase of solution cost.
Perform()	Apply the move to the current solution.

Checking and Post-processing Functions

Interfaces INeighborhood and IMove are relevant for the implementation of new LS neighborhoods; to adapt the predefined neighborhoods of the framework we now explain the interaction between checking and post-processing functions by taking the example of the relocate neighborhood. The four functions, which can optionally be implemented by the user, are presented in the notation of the C# programming language.

```
delegate bool CheckRelocateBetweenTours(
    Tour tour1, Tour tour2)
```

`CheckRelocateBetweenTours` returns `true` if customers from a `tour1` can generally be relocated to a `tour2`; otherwise it returns `false`. If a derived `Tour` class is used, the two tour parameters have to be casted to this specific class to access the new properties and methods when implementing the function.

```
delegate bool CheckRelocateIntoTour(
    TourLocation customer, Tour tour,
    out object relocateIntoTourExtraInfo)
```

`CheckRelocateIntoTour` returns `true` if a `customer` can generally be relocated into a `tour`, otherwise `false`. As the tour parameter, the customer parameter has to be casted if a derived `TourLocation` class is used. If the check is successful, it may pass some information to the next check and/or to the post-processing function. For that purpose any data – simply a number or an object of a complex user-defined class – can be assigned to the `relocateIntoTourExtraInfo` output parameter.

```
delegate bool CheckRelocate(
    TourLocation customer, TourLocation insertAfterLocation,
    object relocateIntoTourExtraInfo,
    ref int costIncrease,
    List<int> enteringAttributes, List<int> leavingAttributes,
    out object relocateAfterLocationExtraInfo)
```

`CheckRelocate` returns `true` if a `customer` can be relocated behind a location referred to as `insertAfterLocation`, otherwise `false`. The information generated by the preceding checking function, `relocateIntoTourExtraInfo`, may be used by casting the parameter from `object` to the appropriate type. Before calling `CheckRelocate` the standard VRP heuristics calculate the changes of solution cost and ABHC attributes; these are passed as parameters `costIncrease`, `enteringAttributes`, and `leavingAttributes` and may be changed to problem-specific values. Here, attributes need to be coded as integer numbers as described in section 6.2.2. If the check is successful, information for post-processing can be assigned to `relocateAfterLocationExtraInfo`.

```
delegate void RelocatePostprocessing(
    TourLocation customer, TourLocation prevLocationInOldTour,
    object relocateIntoTourExtraInfo,
    object relocateAfterLocationExtraInfo)
```

Finally, `RelocatePostprocessing` is called after the `customer` has been moved to its new position. Parameter `prevLocationInOldTour` references its former position, and the information generated by the checking functions is passed as well.[4]

6.2.4 Large Neighborhood Search Classes

Two classes, InsertionProvider and RemovalProvider, evaluate insertion and removal operations for LNS and, respectively, return Insertion and Removal objects representing these operations. The user comes into contact with these four classes only when implementing new insertion or removal heuristics from scratch: abstract classes InsertionHeuristic and RemovalHeuristic, from which subheuristics can be derived, are provided for that purpose.

The checking and post-processing functions for LNS, introduced in section 5.4.3, are implemented in a similar way to the respective functions of LS, except that they do not consider attribute exchanges for ABHC heuristics. We refer to the examples given for the relocate neighborhood in the previous section.

The *InsertionProvider* and *RemovalProvider* Classes

InsertionProvider and RemovalProvider encapsulate the evaluation of insertion and removal operations, respectively. The InsertionProvider class provides a method CheckInsertion(*loc*,*tour*), which evaluates all possibilities of inserting a TourLocation *loc* into a Tour, calling problem-specific checking functions in this process, and returns an Insertion object representing the best feasible insertion. Similarly, the RemovalProvider class has a method CheckRemoval(*loc*) to evaluate the removal of TourLocation *loc* from its current tour. To avoid unnecessary recomputations both classes hold a cache for operations that are still valid even after several iterations of the search. In general, all evaluations associated with a certain tour are valid as long as the tour is not modified. A set of *invalidation methods* is defined to discard cached operations after a tour has been modified; since this is a rather technical issue we do not go into details here.

[4]Note that no information is passed from the first of the three checking function since we did not find a reasonable application. Yet, the framework could easily be extended accordingly if necessary.

The *Insertion* and *Removal* Classes

Besides the change of solution cost INSERTION and REMOVAL objects hold all information required to perform an operation, which includes the customer involved, the position that the operation take place at, and problem-specific information generated by the checking functions. A specific method triggers the operation as well as the problem-specific post-processing.

Insertion	
Property/Method	*Description*
Customer	TOURLOCATION to be inserted.
Tour	TOUR of the insertion.
PrevTourLocation	TOURLOCATION after which the customer is inserted.
CostIncrease	Increase of solution cost.
TourExtraInfo	Additional information generated by the tour-level checking function.
PositionExtraInfo	Additional information generated by the sequence-level checking function.
Perform()	Apply the insertion operation.

Removal	
Property/Method	*Description*
Customer	TOURLOCATION to be removed.
Tour	TOUR of the removal.
PrevTourLocation	Predecessor of the customer before the removal.
CostDecrease	Decrease of solution cost.
ExtraInfo	Additional information generated by the checking function.
Perform()	Apply the removal operation.

The *InsertionHeuristic* and *RemovalHeuristic* Classes

Problem-specific subheuristics are derived from abstract classes INSERTIONHEURISTIC and REMOVALHEURISTIC. In both cases only one single method needs to be implemented, which runs the subheuristic, making use of the insertion- and removal-related classes described above.

An insertion heuristic can use the UnassignedDeliveries property of a Solution to access the customers that have to be reinserted. It is allowed to add new tours to the solution if no feasible insertions exist for customers otherwise, except when the primary objective of the VRP is minimizing the number of tours: in this case the insertion heuristic may only open as many tours as have been closed before by the removal heuristic, specified by a parameter.

InsertionHeuristic	
Method	*Description*
*Perform(*sol*, m)	Reinsert all unassigned customers of Solution *sol*, allowing to open up to m new tours.

A removal heuristic selects a given number of customers and removes them from the solution. They are added to the UnassignedDeliveries list automatically.

RemovalHeuristic	
Method	*Description*
*Perform(*sol*, n)	Remove n customers from Solution *sol*.

6.3 Solver Configuration Classes

The interplay of the various components of the framework is mainly hidden from the user by the RichVRPSolver class, which allows easy access to the VRP heuristics and provides possibilities for configuration on two levels:

- For configurations related to the type of the specific VRP to be solved the RichVRPSolver class offers a range of options, especially to specify which user-defined classes and checking/post-processing functions to be used.
- Further options to select a heuristic or to set numerical parameters are bundled in a separate Parameters class.

The *Parameters* Class

The Parameters class holds a simple collection of diverse options to control the neighborhood search. An object of this class is passed to the solver

- to select the construction method and the base heuristic or hybrid method to be used,
- to choose which number of iterations to spend minimizing the number of vehicles,
- to switch on/off individual neighborhoods or subheuristics, and
- to set several numerical parameters.

The *RichVRPSolver* Class

All interaction with the VRP heuristics is done through properties and methods of the RichVRPSolver class. After creating an instance, the sequence of solver interactions is

1. configuring the solver via properties of the RichVRPSolver object and further Parameters options,
2. initializing the solver for a certain problem instance and, optionally, passing an initial solution,
3. starting the solver for a certain running time or number of iterations, and finally
4. querying the best solution found.

The last steps are straightforward and require no further explanation. To configure the solver for a rich VRP the following options are available:

- Four properties allow specifying the user-defined classes to be used, derived from Solution, SavingsConstruction, SweepConstruction, and ABHCMemory.
- Custom neighborhoods to be used in addition to the predefined ones can be specified by passing lists with INeighborhood, RemovalHeuristic, and InsertionHeuristic objects. Furthermore, it is possible to redefine the relatedness measure for shaw removal.[5]
- Problem-specific checking and post-processing functions are passed to the solver by delegates. The set of functions available is presented in section 5.4. A further problem-specific function RepairSolution can be defined to apply some post-processing to the initial solution – for some VRPs it can be difficult to even generate

[5]The class ShawRelatednessProvider, which we do not describe here, simply contains a method to calculate the similarity values for any two locations of the problem instance.

a feasible solution, and when savings or sweep fail in this respect, infeasibilities must be repaired before the improvement phase of the search is started.

- A set of five further parameters configures the built-in neighborhoods. Capacity checks can be disabled separately for inter-tour neighborhoods and LNS insertions; this is useful if for some reason the user-defined functions shall handle the checks, or to relax the capacity constraint completely. Another parameter is used to indicate whether new tours may be opened during the search. This applies to LNS insertions, relocate, and 2-opt* and should be disallowed when vehicle minimization is the primary objective of the problem.

RichVRPSolver	
Property/Method	*Description*
Data	DATA of problem instance to be solved.
Parameters	PARAMETERS for optimization run.
BestSolution	Best SOLUTION found during optimization run.
CurrentSolution	Last accepted SOLUTION of optimization run.
SolutionType	Problem-specific SOLUTION class.
SavingsConstructionType	Problem-specific SAVINGSCONSTRUCTION class.
SweepConstructionType	Problem-specific SWEEPCONSTRUCTION class.
ABHCMemoryType	Problem-specific ABHCMEMORY class.
AddLSNeighborhoods	Additional LS neighborhoods.
AddLNSRemovalHeuristics	Additional removal heuristics.
AddLNSInsertionHeuristics	Additional insertion heuristics.
ShawRelatednessProvider	Relatedness measure for shaw removal.
CheckRelocateBetweenTours	
CheckRelocateIntoTour	
CheckRelocate	
CheckExchangeBetweenTours	
CheckExchange	
CheckTwoOptStarBetweenTours	See section 5.4.
CheckTwoOptStar	
CheckRelocateI	
CheckTwoOpt	

RichVRPSolver (continued) *Property/Method*	*Description*
CheckInsertionIntoTour CheckInsertion CheckRemoval	
RelocatePostprocessing ExchangePostprocessing TwoOptStarPostprocessing RelocateIPostprocessing TwoOptPostprocessing InsertionPostprocessing RemovalPostprocessing	See section 5.4.
RepairSolution	Function repairing the initial solution.
RelocateCheckCapacities	Indicates whether capacities are checked for relocate.
ExchangeCheckCapacities	Indicates whether capacities are checked for exchange.
TwoOptStarCheckCapacities	Indicates whether capacities are checked for 2-opt*.
InsertionCheckCapacities	Indicates whether capacities are checked for LNS insertions.
AllowNewTour	Indicates whether new tours may be opened during relocate, 2-opt*, and insertion heuristics.
Init(*data*,*sol*)	Initialize the solver for problem instance DATA and, optionally, initial SOLUTION *sol*.
Run(*maxiter*,*maxsec*)	Run the solver for up to *maxiter* iterations and *maxsec* seconds.

Part III

Customizing

Chapter 7

Adaptation to Rich VRPs

The final part of the thesis demonstrates the application of our metaheuristic framework to design and implement customized heuristics by examining five different rich VRPs. Together with chapter 8, which presents detailed numerical results and shows that the heuristics based on our framework can compete with current state-of-the-art methods from the literature, this chapter validates our choices regarding framework concepts and design. In a structured way we explain specific adaptations concerning data structures, construction heuristics, and neighborhoods that can serve as a source of inspiration to the reader who is dealing with a custom problem himself. We also discuss potential drawbacks of our adaptations and make suggestions for further improvements.

Proceeding with increasing complexity section 7.1 first treats the definition of simple problem-specific checking functions, here for time windows checks using the example of the VRPTW. The VRPC, covered in section 7.2, requires additional checks as well, but here the differentiation between customers and orders adds up to the complexity. Being a very simple VRP variant conceptionally, the SDVRP demands heuristics to deal with variable numbers of customer visits. Section 7.3 demonstrates how neighborhoods can be modified for that purpose. The PVRP involves an additional type of decision besides clustering and routing, which is the selection of visit combinations. In section 7.4 we explain how visit combinations are addressed specifically in a new neighborhood but also in modifications of predefined moves. Finally, the TTRP introduces a new tour concept that requires a complex set of adaptations to construction heuristics and neighborhoods covered in detail in section 7.5. We conclude this chapter by briefly suggesting adaptations for other prominent VRPs in section 7.6 that we have not examined explicitly for this thesis.

7.1 Vehicle Routing Problem with Time Windows

The VRPTW introduces intervals for the beginning of service at customer locations. Converting a standard VRP heuristic into a VRPTW heuristic is a relatively easy task since the solution space of a VRPTW instance is included completely in the solution space of the corresponding VRP instance, so that neighborhoods and subheuristics can be reused merely by forbidding moves which violate time window constraints. The hierarchical objective of the VRPTW can be tackled by a two-phase approach based on the generic vehicle minimization procedure of LNS-RRT described in section 5.2.3, minimizing the number of tours first and total distance second.

Time window checks To speed up the evaluation of moves it is crucial to implement time window checks efficiently. Commonly, *slack times* are calculated for the customers of a tour, measuring to which degree the service can be preponed or postponed while still allowing timely arrivals at other customers. As for example described in Campbell and Savelsbergh (2004) we maintain for every customer i the earliest feasible delivery time $\tilde{e}_i$ with respect to the services of previous customers within its tour as well as the latest feasible delivery time $\tilde{l}_i$ considering services of subsequent customers. To check whether a customer j can be inserted between two customers $i-1$ and i we calculate $\tilde{e}_j := \max\{e_j, \tilde{e}_{i-1} + s_{i-1} + t_{i-1,j}\}$ and $\tilde{l}_j := \min\{l_j, \tilde{l}_i - t_{j,i} - s_j\}$, and if $\tilde{e}_j \leq \tilde{l}_j$, the insertion of customer j is feasible. This technique allows to perform time window checks efficiently in $O(1)$ for relocate moves, exchange moves, for insertions, and also in a similar way for 2-opt* moves, comparing the $\tilde{e}$ and $\tilde{l}$ values of the customers to be connected. After any modification of a tour at position i the slack times are updated by setting $\tilde{e}_k := \max\{e_k, \tilde{e}_{k-1} + s_{k-1} + t_{k-1,k}\}$ for $k \geq i$ and $\tilde{l}_k := \min\{l_k, \tilde{l}_{k+1} - t_{k,k+1} - s_k\}$ for $k \leq i-1$.

Obviously, a custom TourLocation class must be derived for the VRPTW to store the slack times. The checks are implemented within functions CheckRelocate, CheckExchange, CheckTwoOptStar, and CheckInsertion, and slack times are updated by their post-processing counterparts. Intra-tour moves are more complicated to check since they modify a tour at multiple positions: in functions CheckRelocateI and CheckTwoOpt arrival and departure times are calculated by stepping through the new sequence of customers completely to determine whether time window constraints are respected by a move.

The efficient checking technique is used within the savings and sweep construction heuristics as well. Here, we additionally modify the sweep method to allow a customer to be inserted into any position within an existing tour: customers often cannot be appended at the end of a tour in the VRPTW since the sequence of insertions derived from the customer coordinates usually does not reflect the chronological relations.

Shaw relatedness measure Finally, we add time window information to the relatedness measure $R(i, j)$ of shaw removal defined in section 5.2.3: let

$$w_{i,j} := \left| \frac{l_i - e_i}{2} - \frac{l_j - e_j}{2} \right|$$

measure the difference between the time windows of customers i and j, and let w_{max} be the maximum over all these values. Then, the relatedness is defined as

$$R(i, j) := \varphi \cdot \frac{c_{ij}}{c_{max}} + \psi \cdot \frac{|q_i - q_j|}{q_{max}} + \chi \cdot \frac{w_{ij}}{w_{max}}$$

with a weight parameter χ.

7.2 Vehicle Routing Problem with Compartments

In the VRPC customers have demands for multiple, inhomogeneous products; an *order* refers to the demand of a customer for a certain product. The orders of one customer may be served by multiple vehicles, but unlike the SDVRP each order must be served completely by a single vehicle. Vehicles are equipped with separate compartments, and loading orders into these compartments is restricted by capacities and compatibility constraints.

By interpreting orders as individual customers a VRP heuristic can be reused easily for the VRPC. An artificial *order-customer* has the same coordinates as the original customer placing the order, and its demand equals the amount of the original order; an additional flag indicates the associated product type. Neighborhoods and subheuristics, now considering orders implicitly, do not require any adjustments besides checks for compartment capacities and compatibilities. Note that the routing of orders leads to a dramatic increase of problem size when many different products are considered: for $|P|$ product types the number of orders is up to $|N_c| \cdot |P|$.

A potential drawback of this straightforward approach is that ignoring the information which orders belong to the same customer can make it more difficult to determine a good routing. Obviously, the orders of one customer are favorably served in sequence, i.e. during a single visit to the customer, yet in the solution process such sequences are only enforced by zero distances between the orders but not by problem-specific knowledge. Without further adaptation of the moves it is possible to generate tours that visit a customer multiple times. Another potential issue is that one-point moves such as relocate and exchange can only transfer single orders at once. Moving all orders of a customer together requires multiple moves of individual orders, often involving deteriorating intermediate solutions that are rejected by the metaheuristic control. Here, the design of customer-based variants of these order-based moves may be beneficial. This problem does not apply to the 2-opt* and 2-opt neighborhoods that preserve sequences of orders.

Solution representation The information which order to load into which compartment is an integral part of the solution representation: the TourLocation class is extended by a property maintaining the compartment $c_o \in C$ that an order o is currently assigned to. To speed up the checks described below the Tour class additionally holds the current compartment loads and also the number of orders of each product type currently assigned to a certain compartment. These values need to be updated by post-processing methods whenever assignments of orders to compartments change.

Local search and large neighborhood search Obviously, all intra-tour moves for the standard VRP are valid for the VRPC without modification since changes to the routing within a tour do not affect the assignments of orders to compartments. This is different in inter-tour neighborhoods:

- Evaluating relocate moves and LNS insertions a feasible compartment has to be determined for the order o considered, i.e. a compartment with sufficient remaining capacity which is compatible with o and which does not contain any orders incompatible with o. Obviously, a move or an insertion is rejected if no compartment of the respective vehicle is feasible. If multiple feasible compartments exist, we prefer selecting a compartment which already contains another order of the same product type. The additional checks are implemented within methods CheckRelocateIntoTour and CheckInsertionIntoTour, and the selected compartment is saved in Φ^t for post-processing.

- For exchange moves we perform a similar check for the two respective orders in CheckExchange.
- In the check for 2-opt*, implemented in CheckTwoOptStar, feasible compartments are determined for sequences of transferred orders by a simple procedure: we start with the first order of such a sequence, determine and hold the compartment assignment, then determine a compartment for the next order, and so on.

The same checking mechanisms are applied to the savings and sweep construction methods. Note that since we never consider the reorganizing of orders already assigned to a vehicle we might reject moves even though a feasible assignment of orders to compartments exists. In the scenarios tested this was not an issue, but with complex compatibility constraints a more sophisticated method of finding compartment assignments might be valuable.

ABHC attributes A somewhat technical issue is related to the solution attributes defined for ABHC heuristics that need to reflect arcs between real customer visits and not between orders: if a solution has no improving neighbors, the effect of considering arcs between orders (order-customers) is that many moves with a cost change of $\Delta = 0$ are performed which simply exchange orders of the same customer, not changing the actual routing at all. Instead of proceeding to a deteriorating solution the search stalls at the – virtually – same solution for a long time. To define customer-arcs instead of order-arcs the ArcAttrId properties of TourLocation objects of orders placed by the same customer have to be set to identical values, respectively.

Shaw relatedness measure Finally, we extend the relatedness measure $R(i, j)$ for shaw removal defined in section 5.2.3 by adding product type information: we define

$$p_{ij} := \begin{cases} 1 & \text{if } p_i \neq p_j \\ 0 & \text{otherwise} \end{cases}$$

indicating whether two orders i and j refer to the same product or not and an additional weight parameter ω. Then, the relatedness is measured by

$$R(i, j) := \varphi \cdot \frac{c_{ij}}{c_{max}} + \psi \cdot \frac{|q_i - q_j|}{q_{max}} + \omega \cdot p_{ij}.$$

7.3 Split Delivery Vehicle Routing Problem

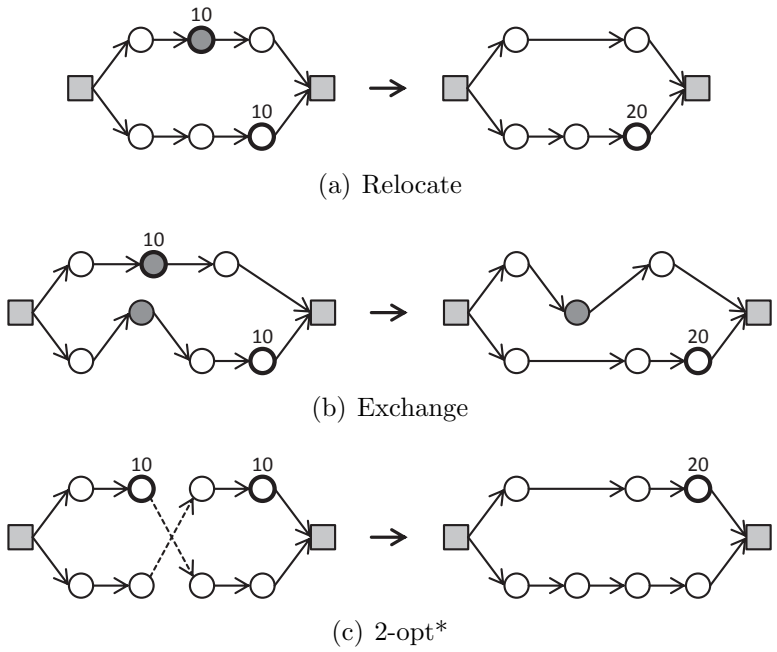
When allowing split deliveries customers may be served by multiple vehicles. Obviously, a standard VRP heuristic generates feasible SDVRP solutions under the condition that no customer demand exceeds the vehicle capacity. Yet, to capitalize on the potentials of split deliveries neighborhoods need to be designed which split up deliveries on certain occasions to serve a customer by multiple vehicles and, conversely, there must be the possibility of rejoining split deliveries again.

Solution representation In the solution representation each delivery to a customer is represented by a TourLocation object with an additional property to specify the associated amount of goods. To speed up some computations we keep track of the current deliveries of each customer by maintaining specific lists in the Solution class. These lists need to be updated accordingly after splitting or joining deliveries.

Sweep heuristic A modification of the savings method to support split deliveries did not appear promising to us, hence we only adapted the sweep method in a very straightforward way: the AppendCustomer method splits a customer's demand if it exceeds the remaining vehicle capacity of the current open tour. In this case a delivery with the maximum feasible amount is appended to the current tour, and the remaining demand is served in a second visit at the beginning of a new tour. The modified sweep heuristic generates an initial solution having the minimum feasible number of tours in which in all tours except for the last tour the capacities are exhausted completely.

Local search and large neighborhood search Standard VRP intra-tour moves do not require any special treatment since splitting and joining does not make sense, here. With respect to inter-tour moves and LNS insertions we make two kinds of modifications:

- Instead of designing new, specific neighborhoods to split up customers' deliveries we incorporate all splitting functionality into the relocate move and LNS insertions. Both operations rely on a complex splitting procedure, which is the central element of our SDVRP adaptation.
- Once deliveries have been split up, certain moves can result in tours that visit a customer multiple times. This may not be strictly prohibited, but obviously it is suboptimal. Rejoining deliveries must be considered in such situations.



(a) Relocate

(b) Exchange

(c) 2-opt*

Figure 7.1: Inter-tour moves with joins of split deliveries.

Figure 7.1 demonstrates the application of joins in inter-tour moves.[1] Evaluation methods need to check whether customers are served twice in a tour after a regular move. In such cases the reduction of total distance caused by a join is calculated, and post-processing methods finally update delivery amounts and remove obsolete deliveries.

The relocate move is modified in such a way that it can shift the amount associated with a delivery to a delivery of the same customer in another tour completely. Figure 7.1(a) shows an example of a customer being served in two tours with an amount of 10, respectively. Moving its upper tour delivery to the lower tour is simply done by increasing the delivery amount in the lower tour and removing the obsolete delivery in the upper tour, thereby reducing the number of deliveries to that customer by one. LNS insertions are modified in a similar way.

If one or even both customers involved in an exchange move are served in both tours, see figure 7.1(b), the deliveries are joined. The position of the joined delivery within a tour is the same position the customer has been served before (and not the former position of the other delivery involved in the exchange). If after a standard 2-opt* move a customer is served twice in the same tour, these deliveries are joined as displayed in figure 7.1(c).

[1]For a clearer presentation the tours of figure 7.1 are depicted as linked lists of nodes including the depot twice as a source node and a sink node. Deliveries of the same customer are indicated by bold border lines.

For easier implementation the join always takes place in the rear part of the tour, i.e. the redundant delivery is removed from the front part of the tour, shifting its associated amount to the rear part.

Ejection procedure To split up deliveries on certain occasions we have presented a complex modification of the relocate move in Derigs et al. (2010), which is motivated by the concept of embedded neighborhood structures and ejection chains described by Glover (1991). Here, a complex *compound move* is generated from a sequence of single steps. The term *ejection* in the VRP context refers to the removal of a customer from its current tour and reinsertion into another tour for the purpose of "making room" for another customer. Standard relocate moves are often infeasible in the SDVRP since tours usually have a very high load factor, and little free capacity is available to transfer deliveries from one tour to another. (Recall that the sweep heuristic generates initial solutions in which most vehicles are fully loaded.)

Our ejection procedure for relocate moves and LNS insertions incorporates the splitting and moving of deliveries and the shifting of demand between deliveries to free up sufficient capacity in the destination tour of a move: in CheckRelocateIntoTour it handles the case that the relocation of a delivery d into a tour t fails due to the lack of remaining capacity, and in CheckInsertionIntoTour it is used when an insertion turns out to be infeasible for the same reason. In algorithm 7.1 $cust_d$ denotes the customer of delivery d, and $amnt_d$ denotes the shipping amount; $rc(t)$ gives the current remaining capacity of tour t. A sequence of steps is conducted until the vehicle of tour t finally has enough capacity left to serve delivery d. In this process the goal is to keep the cost increase of transferring load to other tours as low as possible.

- First, we try to join the delivery of a customer in t with a delivery of the same customer in another tour (case 1). Given that all distances obey the triangle inequality this results in a cost decrease since the customer can be skipped during the course of tour t.

- Then, we shift as much of the amount of a delivery in t as possible to a delivery of the same customer in another tour (case 2). This does not affect costs.

- If this still does not result in sufficient free capacity, we relocate a complete delivery of tour t to another tour and insert it at the position resulting in the lowest cost increase (case 3). Here, we distinguish two subcases. Case 3a: the shortage of

Algorithm 7.1 Ejection procedure for split deliveries

```
1: function EjectDeliveries(delivery d, tour t)
2:     while rc(t) < amnt_d do
3:         if ∃ delivery e on tour t, delivery ẽ on tour t̃ ≠ t :
4:         cust_e = cust_ẽ, rc(t̃) ≥ amnt_e then                    ▷ case 1 (join)
5:             amnt_ẽ := amnt_ẽ + amnt_e
6:             remove e from t
7:         else if ∃ delivery e on tour t, delivery ẽ on tour t̃ ≠ t :
8:         cust_e = cust_ẽ, rc(t̃) > 0 then                         ▷ case 2 (shift)
9:             amnt_e := amnt_e − rc(t̃)
10:            amnt_ẽ := amnt_ẽ + rc(t̃)
11:        else if ∃ delivery e on tour t, tour t̃ ≠ t :
12:        rc(t̃) ≥ amnt_e ≥ amnt_d − rc(t) then                    ▷ case 3 (relocate)
13:            relocate e from t to cheapest position of t̃
14:        else if ∃ delivery e on tour t, tour t̃ ≠ t :
15:        rc(t̃) ≥ amnt_e then
16:            relocate e from t to cheapest position of t̃
17:        else if ∃ delivery e on tour t, tour t̃ ≠ t :
18:        amnt_e > rc(t̃) ≥ amnt_d − rc(t) then                    ▷ case 4 (split)
19:            amnt_e := amnt_e − rc(t̃)
20:            insert new delivery ẽ with cust_ẽ := cust_e and
21:                amnt_ẽ := rc(t̃) into t̃ at cheapest position
22:        else if ∃ delivery e on tour t, tour t̃ ≠ t : rc(t̃) > 0 then
23:            amnt_e := amnt_e − rc(t̃)
24:            insert new delivery ẽ with cust_ẽ := cust_e and
25:                amnt_ẽ := rc(t̃) into t̃ at cheapest position
26:        end if
27:    end while
28: end function
```

capacity can be eliminated completely. Case 3b: tour t still does not have enough free capacity after moving the delivery. Obviously, we prefer a relocation of case 3a.

- Finally, we split a delivery of tour t and insert one part into another tour at the position resulting in the lowest cost increase (case 4). We can distinguish two subcases 4a and 4b comparable to cases 3a and 3b, and again we prefer case 4a. Case 4 results in an increase of cost since the sequence of customers in t remains unchanged, but a new visit is added to another tour.

Note that since the demand of each customer in the destination tour t can be split up to an arbitrary number of deliveries on other tours it is always possible to free up sufficient capacity. The implementation of this ejection procedure within the functions provided by our framework is a little involved: during the tour-level checks CheckRelocate-

IntoTour and CheckInsertionIntoTour all ejection steps are performed on a temporary solution representation. The solution modifications are memorized in Φ^t, and the sequence-level checks CheckRelocate and CheckInsertion have to consider that some insertion positions are not valid any more when deliveries are ejected. The ejection steps are actually performed by the post-processing functions.

ABHC attributes For ABHC heuristics the ejection procedure also has to maintain the arc attribute changes involved with ejection steps. In addition, we introduce a further set of attributes $A^{\text{split}} := \{i : i \in N_c\}$ besides arc attributes which indicate whether a customer i is currently served by multiple vehicles or not. Consequently, the checking methods of all inter-tour moves monitor whether such split attributes enter or leave the solution.

7.4 Periodic Vehicle Routing Problem

The PVRP incorporates multiple types of decisions: first, a visit combination or schedule is selected for each customer to fix the day(s) of the planning period on which the customer is served. Afterwards, a vehicle routing problem is solved for each day, based on the schedule selections made before. Obviously, in a typical neighborhood search approach these two subproblems are not solved sequentially; instead, the search process makes changes to routings and schedule selections in an intertwined fashion.

Due to the fleet size constraint generating feasible solutions for the PVRP is not a trivial task. The approaches of Cordeau et al. (1997) and Hemmelmayr et al. (2009) allow violations of capacities and tour durations to be able to respect the fleet size. By penalizing such violations the search is driven towards completely feasible solutions in the improvement phase. Our PVRP heuristics do not allow infeasible solutions and hence rely on the ability of the construction heuristics to generate solutions already respecting all constraints. For that purpose an essential part of our PVRP adaptation is a repair strategy to convert solutions using too many vehicles into feasible solutions. It does not guarantee to restore feasibility in every case, but at least on our test bed (see chapter 8.2.5) it allows to obtain a feasible solution for every problem instance.

Solution representation We interpret the set of t daily VRPs as one large VRP comprising the tours of all days. Here, the sequence of tours within the solution representation

does not necessarily reflect the planning periods: the first tour of a solution may be a Thursday tour, followed by a Monday tour, and so on. f_i visits of a customer i are added to the VRP according to its service frequency. Classes TOUR and TOURLOCATION are extended by properties that indicate which tour belongs to which daily VRP and which customer visit is due on which day of the planning period. The day flags of visits need to be adjusted when a new visit combination is selected for a customer. In the SOLUTION class we maintain specific lists to access the tours of a certain day and the visits of a certain customer directly. Obviously, the solution also holds the information which visit combinations are currently selected for the customers.

Savings heuristic In our adapted savings heuristic visit combinations are selected during the course of the merging process. Initially, when all customer visits are still assigned to an individual tour, no visit combination is selected for any customer, and no tour is assigned to a specific day, yet. Then, whenever two tours are connected, visit combinations are fixed for every customer served within, and the tours are assigned to days if this has not been done already. All such assignments of days and schedules remain unchanged during construction.

A set of rules is defined in the CHECKMERGE function to specify under which conditions tours may be connected – tours and customer visits must be compatible with respect to their associated days. Concatenating two tours t_1 and t_2 the following rules apply; note that if a tour is not assigned to a specific day, this implies that it has not been merged, yet, and that it contains exactly one customer, for which a visit combination has not been selected either.

1. If neither t_1 nor t_2 is assigned to a specific day, the tours can be merged if visit combinations for the two customers contained can be selected having one day in common.

2. If t_1 is already assigned to a day and t_2 is not, the tours can be merged if a visit combination can be selected for the customer of t_2 involving that particular day.

3. If both t_1 and t_2 are assigned to a specific day, the tours can be merged only if they are assigned to the same day.

Obviously, t_1 and t_2 may not be merged if they contain visits of the same customer or if the connected tour would exceed the vehicle capacity. During a merge operation the PVRP-specific MERGE function sets the days and visit combinations accordingly. When a

visit combination is selected for a customer all tours that serve this customer are assigned to the days of the selected schedule implicitly.

Sweep heuristic Contrary to the standard VRP case our adapted sweep heuristic considers multiple open tours for appending customers – one for each day of the planning period. Starting from an empty solution no visit combinations are selected, yet. A visit combination is fixed for a customer at the moment that its visits are inserted into the solution: the InsertCustomer function calculates the total insertion cost of every visit combination of the considered customer, i.e. the total increase of distance of appending the customer's visits to the current tours of the associated days, respectively, or opening new day tours when capacities are exceeded. The schedule involving the smallest cost increase is selected for the customer, and all visits are appended to the respective day tours.

Repair heuristic Our adaptations of the savings and the sweep heuristic do not consider the fleet size constraint of the PVRP, so that generated solutions may have too many tours on one or multiple days of the planning period. Since the improvement methods of our framework do not provide mechanisms to handle infeasibilities we design a simple repair method to convert infeasible initial solutions into feasible ones. The RepairSolution function for the PVRP sequentially tries to repair every day violating the fleet size constraint by transferring customers away from an associated tour by two means:

1. All deliveries of a tour are removed and reinserted into the solution using regret insertion. The deliveries can be inserted into other tours of the same day, and customers with a service frequency of one can also be transferred to other days by switching their schedules. (See our adaptation of large neighborhood search below.) New tours may be opened on other days in this process, given that the fleet constraint is not violated further.

2. If not all deliveries of the tour can be reinserted, the repair heuristic evaluates alternative visit combinations for the remaining customers, say for a customer c. When moving the deliveries of c to other days via a schedule change the fleet size constraint may not be violated further. Such a repair step is actually a move of the switch-visit-combination neighborhood described below.

If these two steps fail transferring all customers away from the tour, all changes made to the solution are revoked and another tour is tried instead. Yet, if a tour can be saved

completely, the repair heuristic removes another tour or continues with the next infeasible day if the vehicle limit is now respected.

Local search The relocate^I and 2-opt intra-tour moves are feasible for the PVRP without modification. Inter-tour moves are, in general, only feasible if they are applied to tours of the same day. In addition, relocate and 2-opt* moves opening new tours are only feasible if they do not violate the fleet size limit. Yet, we allow relocate and exchange moves between tours of different days when customers with a visit frequency of one are involved. Such deliveries can simply be transferred to other days if their visit combinations can be switched simultaneously with the move, which is checked in functions CheckRelocateIntoTour and CheckExchange.

Switch-visit-combination neighborhood Our most important LS adaptation for the PVRP is the *switch-visit-combination* neighborhood that changes schedule assignments. Selecting a new visit combination for a customer all of its deliveries must be removed from their current tours and reinserted into tours associated with the days of the new visit combination, obeying capacities and the fleet size limit. All potential tours are scanned for the best insertion positions, and opening new tours is considered as well. Within an ABHC heuristic we select the best acceptable switch over all customers in each iteration, and in an RRT iteration we select one customer with at least two allowed schedules randomly and then evaluate all switches.

ABHC attributes We introduce an additional set of attributes $A^{\text{visit}} := \{(i, v) : i \in N_c, v \in C_i\}$, indicating whether visit combination v is selected for customer i.

Large neighborhood search In our LNS adaptation the visit combination of a customer can be changed when all of its deliveries are removed at once by a removal heuristic. Then, its schedule is considered variable and may be switched during reinsertion. If, otherwise, the schedule is still fixed, a delivery can only be inserted to the same day as before. We define the possibilities of inserting a delivery d of a customer c into a tour t in the CheckInsertionIntoTour function as follows, provided that the capacity is respected:

1. *The visit combination of c is variable*
   If c has a visit combination containing the day of t, selecting that schedule allows a feasible insertion. Otherwise inserting d into t is infeasible.

2. *The visit combination of c is fixed*
   Inserting d into t is only feasible for equal associated days.

Reinserting the first delivery of a customer fixes the schedule immediately: the post-processing step assigns all deliveries to the respective days, so that the remaining unassigned deliveries can only be reinserted to these days.

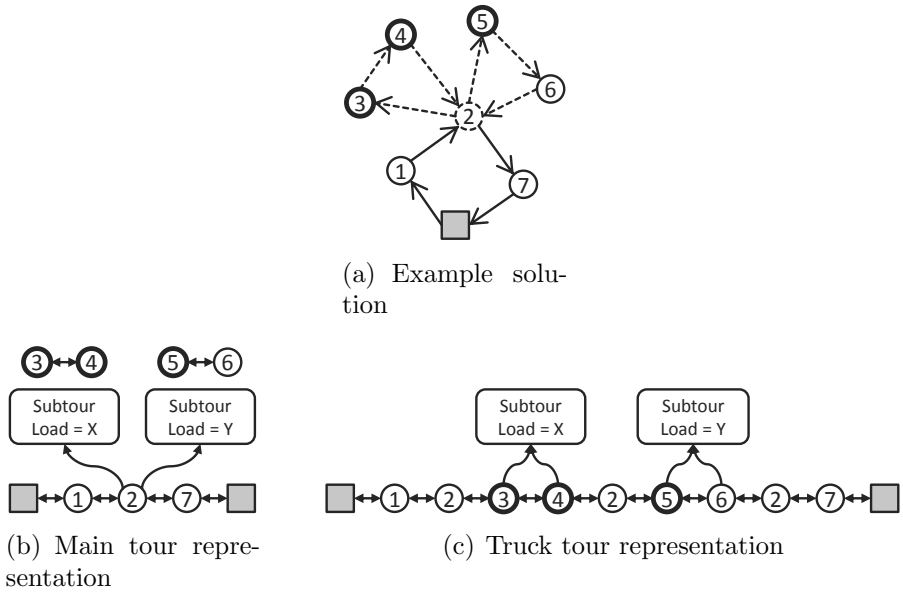
With insertion costs being calculated for individual deliveries only, visit combinations cannot be changed as systematically and purposefully as in the switch-visit-combination neighborhood of LS. The built-in insertion heuristics of the framework may select a bad visit combination for a customer when insertion costs are low for one delivery, ignoring that costs are high for the other ones. Consequently, a potential improvement of our LNS adaptation is the design of a custom insertion heuristic that considers multiple unassigned deliveries of a customer simultaneously. Specific removal heuristics could be useful as well which remove all deliveries of customers together on purpose to enable schedule switches.

7.5 Truck and Trailer Routing Problem

The subtour concept of the TTRP reflects new types of decisions besides clustering and routing, which include the decision whether to use a trailer on a tour or not and the decision whether to park a trailer at a customer location and pick it up again later on during the tour. The formulation of the TTRP is motivated by the existence of so-called truck customers that can only be served by the truck alone; in contrast, vehicle customers can also be served when the trailer is present. A tour conducted by a truck alone is called a pure truck tour. A tour conducted by a complete vehicle, i.e. a truck pulling a trailer, is called a pure vehicle tour if the trailer is not uncoupled at any time during the tour; otherwise it is denoted a complete vehicle tour, consisting of a main tour and one or several subtours. New types of constraints enforce that certain customers (truck customers) are not served during certain segments of a tour (main tours) and that the demands served within certain segments (subtours) do not exceed a certain limit. The numbers of trucks and trailers used are restricted as well. We have introduced the concepts of the TTRP in detail in section 2.2.5; in this section we present our manifold adaptations to this complex variant of the VRP.

7.5.1 Solution Representation

The modeling of subtours by means of the solution representation provided by the framework has a major impact on the behavior of move operations. Figure 7.2 displays two alternative representations for a simple example tour: a *main tour representation* and a *truck tour representation*.



(a) Example solution

(b) Main tour representation

(c) Truck tour representation

Figure 7.2: Alternative TTRP solution representations.

Main tour representation The TOUR data structure of the framework is used to cover main tour customers only, while subtours must be maintained in some user-defined data structure, see figure 7.2(b). Subtour objects, which hold the sequences of customers and current loads and which are referenced from their respective root customers, are "invisible" to the built-in heuristics, so that predefined moves are applied to main tours only. Subtours are tied to their roots in the sense that, for instance, relocating a subtour root customer relocates its attached subtours as well. Such moves preserve the integrity of solutions, yet new specific neighborhoods need to be designed for the purpose of manipulating subtours or changing the assignments of customers to main tours or subtours.

Truck tour representation The TOUR representation of the framework covers the exact sequence of locations visited by the truck, see figure 7.2(c). For the return to a root customer at the end of a subtour a *closing root node* with a delivery amount of 0 is added

to the tour. Subtours can optionally be identified by having the customers of a subtour point to a user-defined subtour object, holding the current load and potentially more information. The predefined moves and subheuristics are evaluated for complete tours: the 2-opt neighborhood, for example, includes reversals of customer sequences even within subtours. Yet, certain moves have to be forbidden or adapted since they destroy the integrity of a solution – consider the relocation of an *opening root node* to another tour without any treatment of its closing counterpart.

We have implemented a truck tour representation to avoid the necessity of designing any specific subtour neighborhoods, which has the consequence that some special cases have to be considered for the built-in moves instead. A new SUBTOUR class holds the current load of a subtour, and a derived TOURLOCATION class allows customers of the same subtour to point to the same SUBTOUR object.

7.5.2 Construction Heuristics

The TTRP formulation includes limits on the numbers of trucks and trailers used. As in our PVRP adaptation we ignore these limits within the construction heuristics and design a simple repair heuristic to restore feasibility instead. Again, producing feasible solutions is crucial since our improvement heuristics do not handle infeasibilities.

Savings heuristic Initially, we interpret the individual tours of truck customers as pure truck tours and individual tours of vehicle customers as pure vehicle tours. For combining two tours t_1 and t_2 the following rules are applied in CHECKMERGE and MERGE:

1. Two pure truck tours may be combined to a larger pure truck tour if the truck capacity is not exceeded.
2. Two complete (or pure) vehicle tours may be combined to a larger complete (or pure) vehicle tour if the complete vehicle capacity is not exceeded.
3. If t_1 is a pure truck tour and t_2 is a complete (or pure) vehicle tour, t_1 is attached to the first customer of t_2 as a subtour, provided that the complete vehicle capacity is not exceeded. Vice versa, t_2 is attached to the last customer of t_1.

Sweep heuristic The APPENDCUSTOMER function defines the following rules for appending a customer c to a tour t. In any case a new tour is opened if the capacity of the complete vehicle is exceeded.

1. *c is a truck customer:*

   (a) If t is empty, it is converted into a pure truck tour for c.

   (b) If t is a pure truck tour, c is appended to t. If the truck capacity is exceeded hereby, a new pure truck tour is opened for c.

   (c) Otherwise, if t is a complete (or pure) vehicle tour:

      i. If the last customer of t is a subtour root and the truck capacity is not exceeded, c is appended to that subtour.

      ii. Otherwise, a new subtour containing c is attached to the last customer of t.

2. *c is a vehicle customer:*

   (a) If t is empty, it is converted into a pure vehicle tour for c.

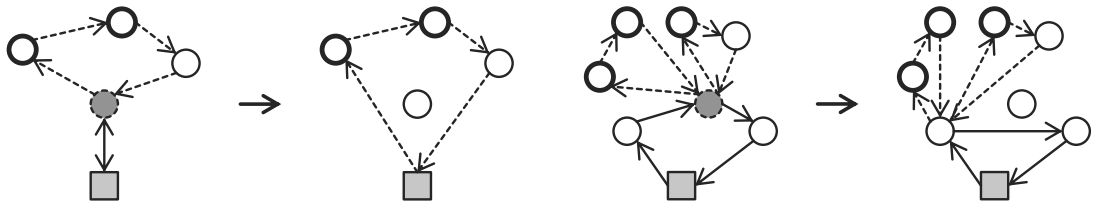
   (b) If t is a pure truck tour, t is converted into a complete vehicle tour with c being the subtour root for the remaining customers of t.

   (c) Otherwise, c is appended to the complete (or pure) vehicle tour t.

Repair heuristic If a solution generated by the savings or sweep heuristic uses more trucks and/or trailers than allowed, the repair heuristic tries to rearrange customers into a smaller number of tours. In the first step the number of trailers is reduced, if necessary, by transforming pure or complete vehicle tours into pure truck tours. We allow the number of trucks to be increased in this step, even exceeding the truck limit, to guarantee that the solution complies with the trailer limit in the end.

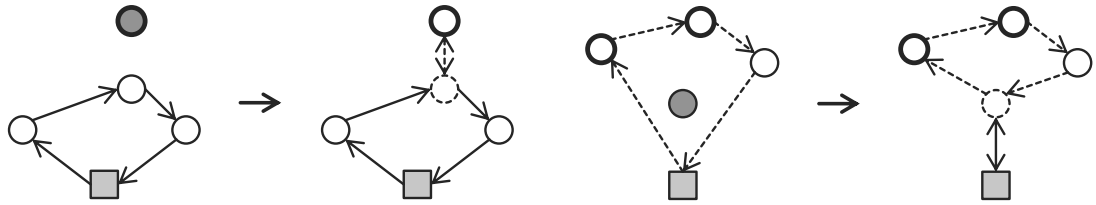
1. First, pure vehicle tours are converted into one or multiple pure truck tours, respectively. If possible, this is done first for tours for which the capacity of a single truck is sufficient. Otherwise, pure vehicle tours need to be split into multiple pure truck tours.

2. When all pure vehicle tours have been transformed and the trailer limit is still exceeded we continue with complete vehicle tours. The subtour customers of a complete vehicle tour are transferred to a new pure truck tour in the sequence of their visits in the original tour. If a single truck cannot serve all subtour customers, more tours are opened. After that, the remaining pure vehicle tour is converted as described above.



(a) Removing a subtour root which is the only customer within its main tour

(b) Removing a subtour root which is *not* the only customer within its main tour



(c) Inserting a truck customer by opening a new subtour

(d) Inserting a vehicle customer by converting a pure truck tour into a complete vehicle tour

Figure 7.3: Special removal and insertion operations in the TTRP.

Finally, when the solution is feasible with respect to the trailer limit we repeatedly try to remove tours completely and reinsert the customers into the remaining tours using regret insertion until the truck limit is respected as well. Here, we consider all types of tours but try to remove those pure truck tours first which have been generated during the previous trailer saving step.

7.5.3 Large Neighborhood Search

Our adaptations of LNS to the TTRP combine modifications of removal and insertion operations and the design of a new, specific removal heuristic. Figure 7.3 displays four special cases of operations.

Removal operations Removing a customer c from its current tour is straightforward except for the case that c is a subtour root. The following cases are defined in the CheckRemoval and RemovalPostprocessing functions:

1. *c is a subtour root:*

   (a) *c is the only customer within its main tour:*
   If c is the root of a single subtour, the tour becomes a pure truck tour with the depot as the root, see figure 7.3(a). c is not removed if multiple subtours depart from it.

(b) *c is not the only customer within its main tour:*
The departing subtour is transferred to the predecessor of c within the main tour or to the successor, depending on the change of cost. If c is the root of multiple subtours, all subtours are transferred to the same customer for simplicity, see figure 7.3(b).

2. *c is not a subtour root:*
c can be removed without further restructuring, except that a subtour is closed if c is its only customer.

Insertion operations Evaluating the insertion of a customer c into a tour t behind a customer i (or the depot) the following cases are considered in functions CheckInsertion and InsertionPostprocessing. In any case the capacities of the truck and/or the complete vehicle have to be checked, and vehicle limits must not be violated when opening a new tour or using an additional trailer.

1. *c is a truck customer:*

   (a) *t is an empty tour:*
   t becomes a pure truck tour containing c.

   (b) *The position of insertion is within a pure truck tour or a subtour:*
   c can be inserted without further restructuring.

   (c) *The position of insertion is within a main tour:*
   A new subtour is opened for c with i being the subtour root, see figure 7.3(c).

2. *c is a vehicle customer:*

   (a) *t is an empty tour:*
   t becomes a pure vehicle tour containing c.

   (b) *t is a pure truck tour:*
   c can be inserted without further restructuring if the truck capacity is not exceeded. Otherwise c is inserted as a subtour root, converting the pure truck tour into a complete vehicle tour, see figure 7.3(d).

   (c) *t is a pure or complete vehicle tour:*
   c can be inserted without further restructuring.

The implementation of some of these special cases is comparably involved and often incorporates the addition or removal of root nodes. Post-processing is demonstrated in figure

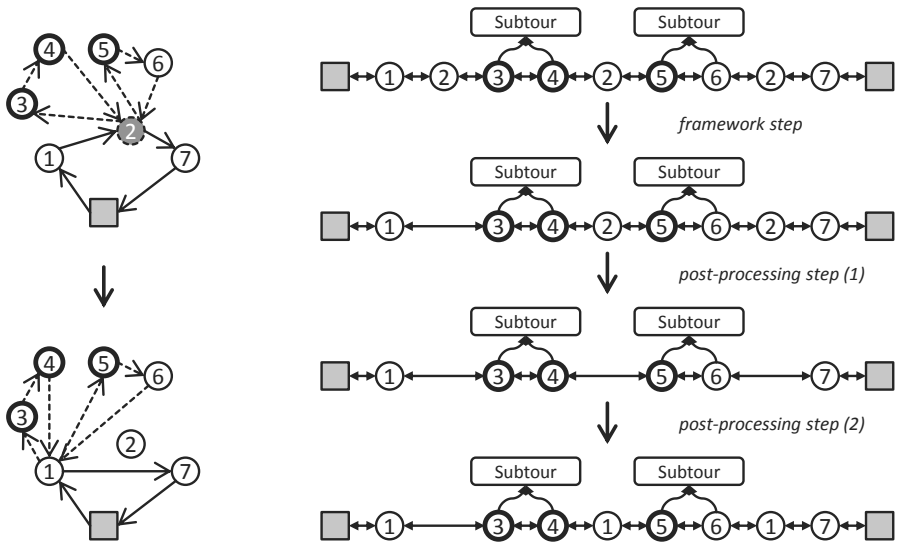


Figure 7.4: Post-processing steps for removing a subtour root.

7.4 for the case of removing a subtour root which is not the only customer of its main tour – the same case as already displayed in figure 7.3(b). Customer 2 is visited three times during the course of the tour since two subtours depart from it. The predefined operation for the standard VRP removes its first opening root node only – after that, the tour representation is invalid since the first subtour is now attached to customer 1, but the closing root nodes of the subtours still refer to customer 2. Post-processing has to be implemented in such a way that all closing root nodes are replaced with new nodes referring to customer 1 to restore integrity of the tour.

Subtour removal An additional removal heuristic enforces the rebuilding of subtours from scratch by removing a set of subtours from one area of the solution. Removing subtours completely (and not a few subtour customers only) improves the chance that new subtours are created with new root customers, potentially in other tours than before.

Algorithm 7.2 shows the procedure of *subtour removal* which removes (at least) q subtour customers from a solution S, given that so many customers are currently served in subtours. Set P maintains the subtours of the solution which have not yet been selected for removal during the course of the procedure, and set R holds the customers selected to be removed on the other hand. In this context $i \in p$ denotes a customer of a subtour $p \in P$, and $|p|$ is the number of customers within the subtour. After selecting a random subtour as a seed (lines 3-4) further subtours are selected iteratively; since complete subtours are removed $|R|$ may exceed q in the end. In each iteration the selection of a subtour is biased

towards such subtours which are near the subtours already selected. For each unselected subtour $p \in P$ the average distance (traveling cost) from its customers to the customers in R is calculated as

$$\bar{d}_p := \frac{\sum_{i \in p, j \in R} c_{ij}}{|R| \cdot |p|}$$

and based on these distance values a subtour p' is selected using the same randomized selection mechanism as in LS-RRT, controlled by a randomization parameter β^{STR} (lines 7-11).

Algorithm 7.2 Subtour removal

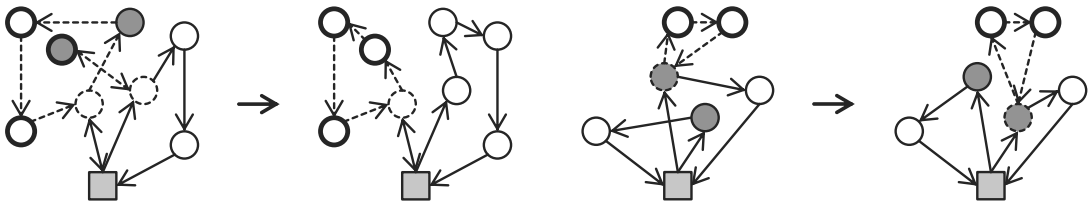
```
1:  function SubtourRemoval(solution S, q ∈ ℕ, randomization β^STR)
2:      P := subtours of S
3:      select subtour p ∈ P randomly, P := P\{p}
4:      R := customers of p
5:      while |R| < q do
6:          d̄_p := average distance between customers in R and p, ∀p ∈ P
7:          array P' := ⟨p ∈ P⟩, with k < l ⇒ d̄_P'[k] ≤ d̄_P'[l]
8:          select r ∈ [0, 1) randomly
9:          p' := P'[⌊r^β^STR · |P'|⌋]
10:         R := R ∪ customers of p'
11:         P := P\{p'}
12:     end while
13:     remove customers R from S
14: end function
```

7.5.4 Local Search

Our adaptations of LS include modifications of the five built-in neighborhoods as well as the implementation of two specific TTRP-neighborhoods.

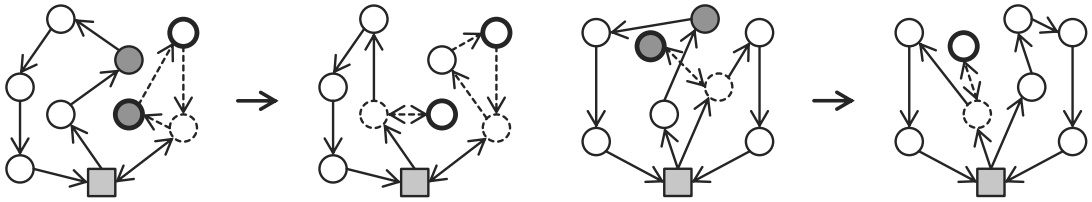
Relocate A relocate move is essentially a combined removal and reinsertion; hence our modified relocate is based on the same special operations as LNS.

Exchange Generally, exchanging customers between main tours and exchanging customers between subtours is uncritical; only exchanges between different tour types or rather exchanges involving different customer types require some considerations. In particular, we address the following special cases in CheckExchange; note that multiple of these cases can occur within one move.



(a) Replacing the single customer of a subtour with a vehicle customer

(b) Replacing a subtour root with another vehicle customer



(c) Replacing a main tour customer with a truck customer

(d) Combination of cases (a) and (c)

Figure 7.5: Special exchange moves in the TTRP.

1. *A vehicle customer c is exchanged with the single customer of a subtour:*
   The subtour is closed, and c is inserted into the main tour instead, see figure 7.5(a).

2. *A subtour root c_1 is exchanged with a customer c_2 :*

   (a) *c_2 is a vehicle customer:*
   The subtours of c_1 are reattached to c_2 after the exchange, see figure 7.5(b). This case does also apply to the exchange of two subtour roots.

   (b) *c_2 is a truck customer:*
   The exchange is not feasible since truck customers cannot serve as a subtour root.

3. *A truck customer c_1 (in a subtour) is exchanged with a customer c_2 in a main tour which is not a subtour root:*
   A new subtour is opened for c_1 with the predecessor of c_2 being the subtour root, see figure 7.5(c).

Figure 7.5(d) gives an example of multiple special cases in one move: here, a main tour (vehicle) customer is exchanged with the single (truck) customer of a subtour. In the one tour the subtour is closed, and in the other tour a new subtour is opened for the truck customer.

2-opt* Function CheckTwoOptStarBetweenTours forbids exchanges between a complete (or pure) vehicle tour and a pure truck tour, and function CheckTwoOptStar ensures that complete vehicle tours are not split up in the middle of a subtour.

Relocate^I The removal and insertion operations defined for LNS are reused for relocate^I in a similar way. In addition, function CheckRelocateI prevents that a subtour root customer is moved into its own subtour, and unlike usual intra-tour moves we need to check a capacity constraint: the truck capacity must not be exceeded when a customer is moved into a subtour.

2-opt Reversing customer sequences between two main tour customers is allowed, even when subtours are conducted in between, as well as reversing sequences within a subtour or a pure truck tour. Moves that disrupt subtours are obviously forbidden in CheckTwoOpt.

Relocate-subtour neighborhood The solution process may form a set of good subtours which could, however, depart more suitably from other customers. Obviously, a subtour can be transferred by means of a sequence of relocate moves, yet this often involves deteriorating intermediate moves. Enabling the transfer of complete subtours the *relocate-subtour* neighborhood considers

- selecting an alternative root customer for a subtour, potentially within another main tour, and at the same time
- selecting another customer within the subtour which is visited first.

The only constraint to be checked is the capacity of the complete vehicle when transferring to another main tour. In a special case of the move the root customer remains the same, but another customer of the subtour is visited first, and in another special case the subtour is attached to the depot, i.e. it is converted to a pure truck tour, or vice versa. The relocate-subtour neighborhood is an extension of the *sub-tour root-refining step* by Chao (2002), which is designed as an intra-tour move. Figure 7.6 shows the relocation of a subtour from one main tour to another with simultaneously selecting another starting customer. Within ABHC heuristics the neighborhood is evaluated by choosing for every subtour of the solution every potential root customer and every potential starting customer. In RRT we evaluate all such moves for a randomly selected subtour.

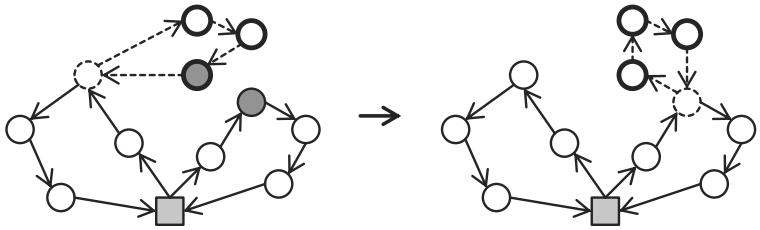
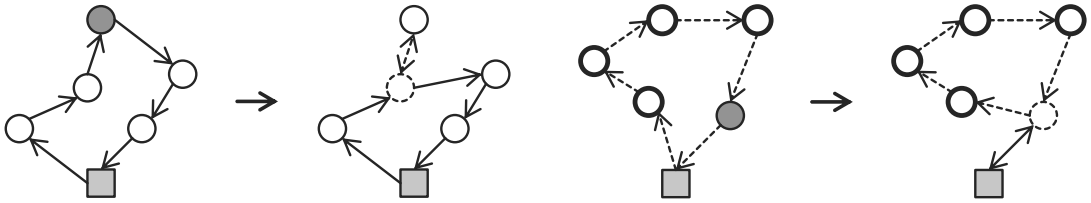


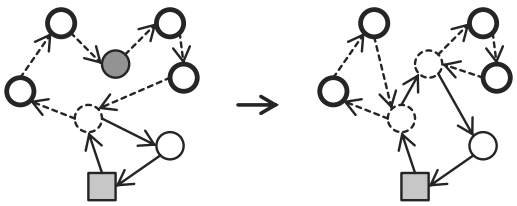
Figure 7.6: Relocate-subtour neighborhood.

Switch-vehicle-type neighborhood The *switch-vehicle-type* neighborhood, denoted as *change of service vehicle type* in Lin et al. (2009), changes the visit of a customer from a complete vehicle visit to a truck-only visit and vice versa. The following cases are distinguished for a vehicle customer c:



(a) Moving a main tour customer to a new subtour

(b) Converting a pure truck tour customer into a subtour root in a complete vehicle tour



(c) Moving a subtour customer to the main tour and splitting the subtour

Figure 7.7: Switch-vehicle-type neighborhood.

1. *Customer c is served in a main tour:*
   A new subtour is opened for c with its predecessor being the subtour root, see figure 7.7(a). This switch is not defined for the first customer of a main tour and for subtour root customers.

2. *Customer c is served in a pure truck tour:*
   The pure truck tour is converted into a complete vehicle tour with c being the subtour root for the remaining customers, see figure 7.7(b). We have implemented this switch for the first and the last customer of a tour only.

3. *Customer c is served in a subtour of a complete vehicle tour:*
   c is moved to the main tour and depending on the position of c the subtour is potentially split into two subtours hereby. In this case the part preceding c remains attached to the former root customer c', while the part succeeding c is reattached to c as the new root, see figure 7.7(c). If more subtours attached to c' are conducted subsequently, they are reattached to c as well.

Note that only the first of the three cases is defined for the opposite direction as well, and that no capacity checks are required for any of the cases. Within an ABHC heuristic we evaluate the switches of all vehicle customers, and in RRT we evaluate all switches within a randomly selected tour.

ABHC attributes For ABHC heuristics we introduce an additional set of attributes $A^{\text{subtour}} := \{(i) : i \in N_c \text{ is a vehicle customer}\}$ indicating whether a vehicle customer is served within a subtour or not.

7.6 Other Rich VRPs

The five rich VRPs treated in this chapter cover a broad range of adaptation requirements arising from practical scenarios. Now, we briefly describe basic ideas how our framework can be used to develop heuristics for a selection of other prominent VRPs listed in section 2.2.6. In fact, we believe that it is not too difficult in most cases to implement heuristics that generate acceptable feasible solutions for a problem. Still, to obtain high-quality results the design of specific moves or other algorithmic ideas may be required.

As discussed in section 4.1 generalization/specialization relationships between problem variants provide an easy way to solve new VRPs using existing solvers. For instance, the OVRP and other VRP variants with open tours can be solved by "closed-tour heuristics" through simply modifying the distance matrix of a problem instance. Setting the distances of all arcs ending at the depot to zero the search is driven towards good open tours. The SetOpenTours method of our Data class is provided exactly for this purpose. Similarly, Cordeau et al. (1997) solve the MDVRP using a PVRP heuristic by associating depots with days. The same should be possible as well with our PVRP heuristics presented in section 7.4.

Generally, if the solution spaces of instances of a specific VRP are included in the solution spaces of corresponding standard VRP instances, only additional checks have to be

implemented – the design of specific neighborhoods that address new decision variables and access new regions of the solution space is not required. This applies to the VRPTW, the VRPC, and also several other rich VRPs:

- VRPB variants impose different kinds of sequence-level constraints considering the positions of linehaul and backhaul customers within a tour. In cases that one group of customers must be served completely before any customer of the other group is visited this precedence must be preserved by every intra-tour and inter-tour move, which involves rather easy checks. If, instead, linehaul and backhaul customers may be mixed during a tour, the load fluctuations must be calculated explicitly to prevent violations of the vehicle capacity. The same kind of check applies to PDPs, but as we explained in section 1.3 this problem class requires implementing a complete set of new moves that handle associated pickups and deliveries simultaneously.

- The effort required to respect two-dimensional or three-dimensional loading constraints varies strongly with the nature of these constraints. In the non-sequential variants of the 2L-CVRP and the 3L-CVRP the loading/unloading sequence of goods is not important, so that checks need to determine a feasible loading plan on the tour-level, independent of the routing. The general proceeding can be compared to the VRPC, yet the actual problem of determining a feasible loading plan can be much more difficult and time-consuming. Here, specific (external) procedures are used to solve two- or three dimensional bin packing problems. In the sequential variants of the two VRPs mentioned these checks depend on the sequence that customers are visited. A main challenge is how to conduct checks efficiently.

- Constraints on driving times and rest periods are usually accompanied by time window constraints for visits. The following sequence-level check guarantees feasible solutions: stepping through the complete sequence(s) of customers generated by a move, breaks and rest periods are scheduled whenever forced by some rule or when sufficient waiting time is available. If the vehicle arrives too late at a location, the move is declared infeasible. (A naive, inefficient time window check for the VRPTW would be implemented in a similar way.) Yet, this comparably simple approach potentially declares many moves infeasible although tours could be conducted feasibly, in fact. The rules of EC Regulation No. 561/2006 offer several possibilities of taking breaks beforehand, extending driving times, and reducing rest durations under certain conditions, so that the real challenge in solving VRPs with driving times is the design of a checking procedure that finds a feasible schedule of breaks and rest periods whenever one exists, and which is efficient at the same time.

- Adaptations required for the FSMVRP and the HVRP depend on the specific aspects in which a vehicle fleet is heterogeneous. Tour-level checks of inter-tour moves may need to consider the individual capacities of vehicles or prevent assignments of customers to incompatible vehicles. In other scenarios the user-definable checking functions may need to recalculate the costs of certain moves. For example, if acquisition costs of vehicles are relevant, the costs of moves opening new tours (on previously unused vehicles) or closing tours must be increased/decreased by these fixed costs. If different traveling costs of vehicles are involved, the costs of inter-tour moves must be adjusted in particular.

One rich VRP requiring more specific adaptations is the VRPM that allows the scheduling of multiple tours for a single vehicle during the planning period. First of all, the solution representation has to be modified for the specific tour structure to separate the different tours assigned to a vehicle in a suitable way. There are two general types of solution approaches in the literature: the most common approach is the decomposition of the VRPM into the subproblems of generating individual tours and of assigning these tours to vehicles. The first subproblem is usually solved by common VRP methods, so that our heuristics can be reused in this case. For the second subproblem specific bin packing heuristics aggregate the tours generated, obeying constraints such as the duration of a working day. Probably, our framework requires some minor modifications for such a decomposition approach, for example providing additional user-definable functions for the coordination between the two solution steps. Alternatively, the VRPM can be solved using a neighborhood search method that considers both subproblems simultaneously. Here, the predefined moves must be modified to distinguish the different tours of a vehicle, provide the possibility of changing the times that a vehicle returns to the depot, etc. Specific moves that exchange complete tours between vehicles could be beneficial as well.

Chapter 8

Computational Results

In chapter 7 we have demonstrated the flexibility of our framework concept and design to develop rich VRP heuristics by describing its usage for five specific VRPs. In this chapter we concentrate on the quantifiable quality measures accuracy and speed, and we evaluate our implementations of the framework and of the adaptations on problem instances from the literature. Dealing with metaheuristics the outcome of the solution process depends on the proper setting of parameters to a certain degree. Consequently, evaluation starts with the determination of suitable parameter values for the different solution methods before the overall quality of the results can be assessed.

Determination of a standard parametrization As a first step we fine-tune a selection of important parameters on instances for the standard VRP. All heuristics except for LS-ABHC can be configured by numerical parameters, which may have a mild or a strong influence on the solution quality, respectively. Evaluating multiple parameter configurations we can already draw conclusions on general characteristics of the heuristics. The best parameter values are adopted as the *standard parametrization* of the framework.

Tests with standard parametrization Using the standard parametrization the rich VRP adaptations are tested on sets of problem instances which are commonly used as benchmark instances in the literature and for which other authors have published reference solutions. We compare the solutions generated by our methods against the currently best known solutions (BKS) from the literature. These tests serve as a kind of feasibility study that indicates whether good and reliable results can already be obtained consistently without putting massive efforts into parameter tuning for a specific VRP or a specific set of instances.

Tests with problem-specific parametrization Finally, parameters are fine-tuned individually for each of the five rich VRPs, and the new results are compared against those based on standard parametrization. Here, we get an insight into the robustness of the heuristics – in particular an answer to the question to which degree individual tuning is promising or even crucial. To assess whether our heuristics are competitive considering accuracy and speed we select one current state-of-the-art method published by other authors for each problem and compare our best configuration, respectively, to their results.

Our testing platform for all experiments is an Intel Xeon E5430 2.66 GHz PC with eight cores and operating system Microsoft Windows 7. All computation times listed in this chapter are obtained conducting four test runs in parallel; yet, the individual heuristics are *not* accelerated by any means of parallelization. We refrain from using all eight cores at the same time since we have found that increasing the number of parallel testing runs increases running times for a given number of iterations significantly compared to the times of sequential runs. With only four parallel processes the impact was moderate. As mentioned before we used the C# programming language based on the Microsoft .NET framework 3.5 for our implementation.

As it is commonly done in such experiments we conduct test runs with fixed numbers of iterations instead of fixed running times. Larger (or more difficult) instances generally require longer running times to be solved adequately than smaller instances; since the computational effort per iteration increases with the size of the instance a fixed number of iterations gives the larger instances more running time than the smaller instances. Note that in all tests we use both the savings method and the sweep method to generate initial solutions and select the solution with lower cost to start the improvement phase.

In section 8.1 we start determining the standard parametrization of the framework. Then, we present the final results for each problem in section 8.2 based on standard parametrization and problem-specific parametrization. Finally, section 8.3 summarizes our findings and compares the five heuristics giving attention to the overall solution quality and robustness.

8.1 Standard Parametrization

Our tests for the standard VRP are based on a set of 19 widely-used instances generated by Christofides et al. (1979) and Golden et al. (1998), which are sometimes referred to as “CMT” and “GWKC” instances according to the authors’ names. Instance sizes vary

between 50 and 483 customers, and the BKS values are published in various articles; we obtained the current values from Ropke (2011) as reference solutions. Details on the selected instances are given later in table 8.3 where we present the final results of our best configuration.

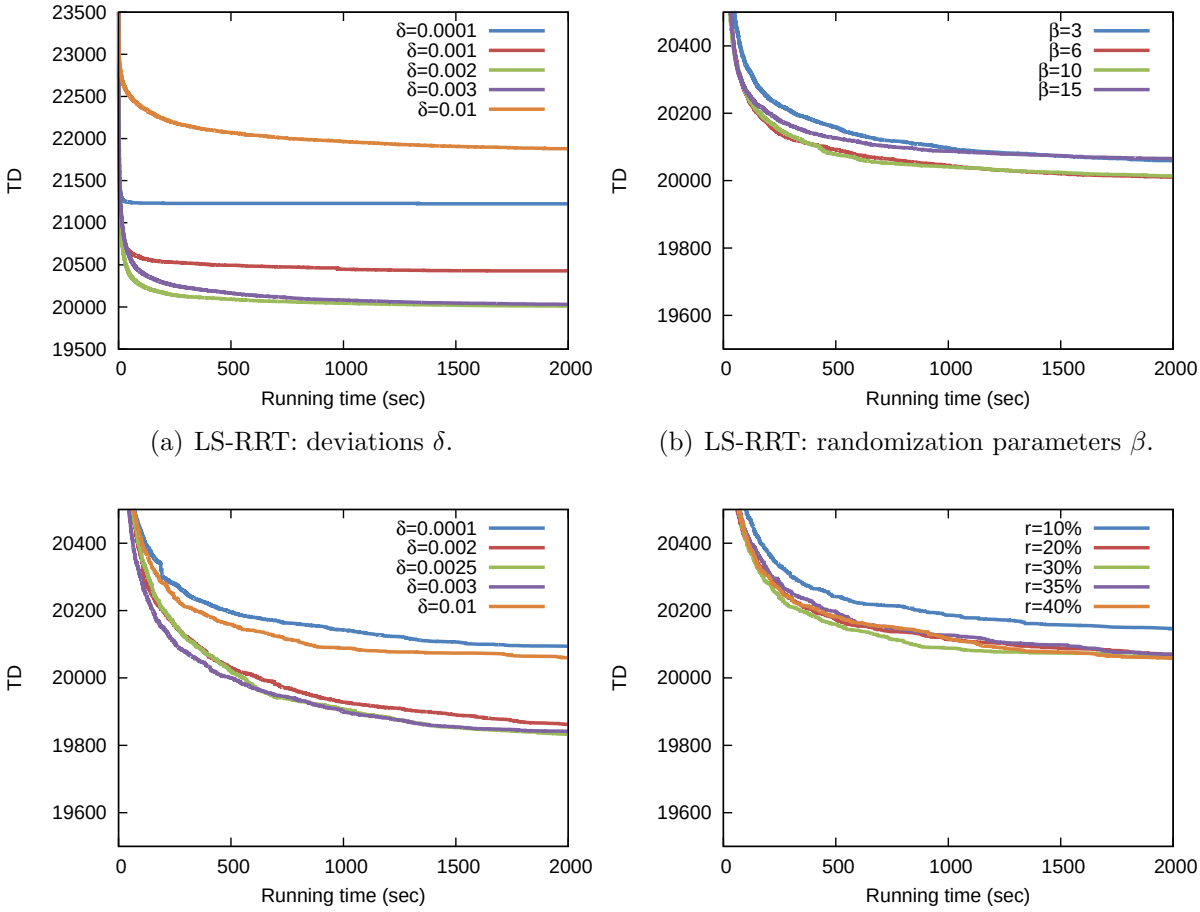
For the initial tuning we solve each instance three times under each evaluated parameter configuration, calculate the average total distance (TD)[1] obtained for each instance, and then select the best configurations according to the average deviations from the BKS. Whenever two separate parameters calibrate a heuristic, say p_1 and p_2, we evaluate different values for p_1 first, holding p_2 fixed to a sensible value, and then fix the best p_1 value while varying p_2. Now, we present our findings of parameter calibration for all heuristics, obviously except for the parameter-free LS-ABHC. During all tests some minor parameters are set to fixed values without any tuning: randomization parameters $\beta^{WR} = 3$, $\beta^{SR} = 6$, and $\beta^{STR} = 3$ as well as shaw relatedness weights $\varphi = 9$, $\psi = 2$, $\chi = 3$, and $\omega = 2$.

LS-RRT The calibration runs for LS-RRT are conducted with 10,000,000 iterations each, during which we evaluate different deviations δ and randomization parameters β. $\delta = 0.002$ yields the best results in these tests, trying more than ten different values between 0.0001 and 0.1 and holding β fixed. Fixing this setting $\beta = 6$ turns out to be best among four values between 3 and 15 afterwards. This combination is assumed as our standard parametrization for LS-RRT.

LNS-RRT 100,000 iterations per run are conducted to calibrate LNS-RRT, varying settings for the removal percentage parameter r and, again, the deviation δ. Among five different removal percentages between 10% and 40% we fix $r = 30\%$ first and then choose $\delta = 0.0025$ to complete the standard parametrization for LNS-RRT, trying values for δ from the same range as for LS-RRT.

Figure 8.1 shows the development of solution quality over time for a selection of parameter values tested for LS-RRT and LNS-RRT. The graphs display the sum of distances / costs over all instances at a given point in time, averaged over all replications. Some first conclusions on the two base heuristics can be drawn from the previous tests:

[1]Note that we use "TD" and "cost" synonymously during this evaluation.



(a) LS-RRT: deviations δ.

(b) LS-RRT: randomization parameters β.

(c) LNS-RRT: deviations δ.

(d) LNS-RRT: removal percentages r.

Figure 8.1: Parameter settings of LS-RRT and LNS-RRT.

- Regarding the progress of the curves in figure 8.1 it is a good sign that there are hardly any major overlappings among the curves of a parameter in the sense that one value yields goods results quickly but is outperformed by another setting after a longer running time. Consequently, settings do not have to be chosen depending on the running time.

- Evidently, the deviation δ of LS-RRT (note the scaling of figure 8.1(a)) is the most sensitive parameter among those tested; in particular, it is much more sensitive compared to δ in LNS-RRT. While a bad value for δ within the range tested leads to an average deviation from the BKS of 8.43% in LNS-RRT, the worst average deviation is 28.60% within the same range of δ in LS-RRT. The fact that this disastrous solution quality is yielded by $\delta = 0.1$ (not displayed in figure 8.1) can be explained by the assumption that the randomized selection scheme of LS-RRT produces much more *very* bad moves than LNS-RRT does, so that it is specifically

up to the metaheuristic control to discard a lot of these moves. The randomization parameter β also plays a role in controlling how many bad moves are proposed during the search, but its influence is rather small compared to δ.

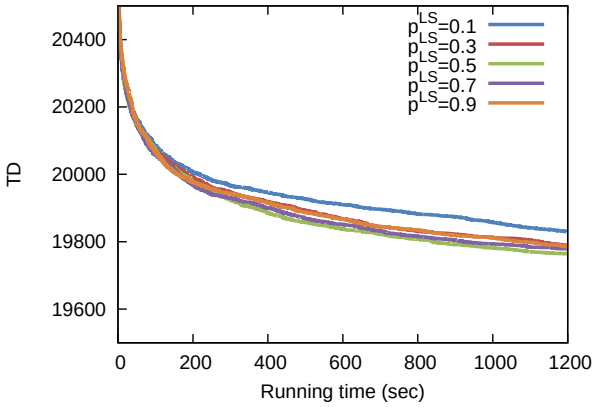
- The impact of parameter settings on the success of LNS-RRT is not that remarkable. Deviation δ has a certain influence, and removal percentage r must not be set too low, limiting the power of LNS. Anyway, the influence of r on larger problem instances is restricted by capping the number of removed customers to 100, see section 5.2.3.

The specific parameter of the two hybrid methods is p^{LS}, i.e. the proportion between LS and LNS moves during the search process. For its calibration we change the testing procedure and set a fixed running time of 1,200 seconds per run instead of a fixed number of iterations. In fact, for a fixed number of total iterations the variation of p^{LS} has a significant influence on the total computational effort since the times spent within an LS iteration and within an LNS iteration can differ dramatically. By fixing running times instead we make different parameter settings comparable fairly.

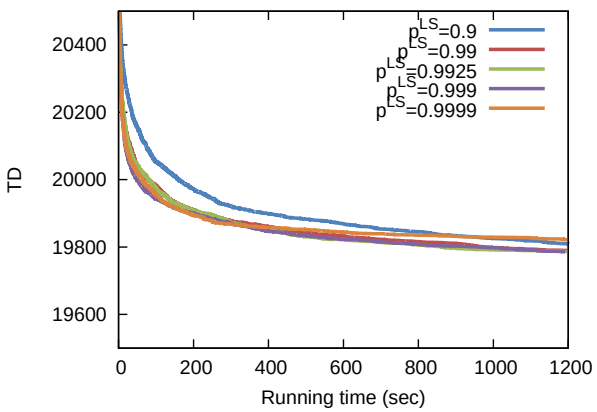
HYBRID-ABHC The percentage p^{LS} of local search moves is set to multiple values between 10% and 90% with incremental steps of 10%. It turns out that an even proportion between the neighborhood concepts produces the best results, and $p^{\text{LS}} = 0.5$ is selected as the standard parametrization.

Using this setting we try a small variation of HYBRID-ABHC: instead of applying the acceptance criterion of the ABHC to LNS moves as given in algorithm 5.4 we now accept any improving move. This, however, increases the average deviation from the BKS from 1.20% to 1.25%, indicating that the slight increase of iterations due to the omission of the time-consuming check has no positive effect.

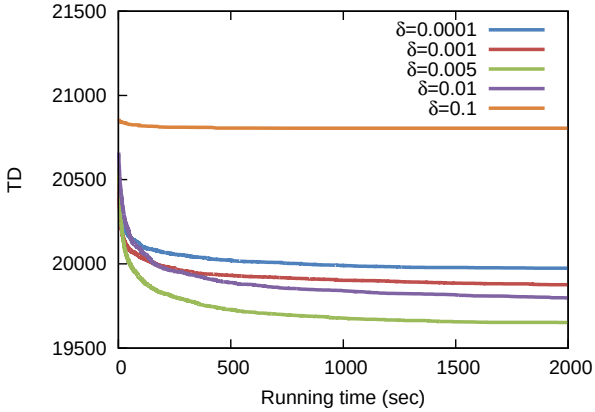
HYBRID-RRT The range of suitable p^{LS} values for HYBRID-RRT is very different from the range for HYBRID-ABHC since an LS-RRT move has a much smaller computational effort that an LNS-RRT move. In other words, many LS moves need to be evaluated and performed to yield the progress of a single LNS move, and for a "good mixture" of the neighborhood concepts p^{LS} must be set to a rather high value. Fixing deviation δ a setting of $p^{\text{LS}} = 0.9925$ turns out to be best among several values between 0.9 and 0.9999.



(a) HYBRID-ABHC: weights p^{LS}.



(b) HYBRID-RRT: weights p^{LS}.



(c) HYBRID-RRT: deviations δ.

Figure 8.2: Parameter settings of HYBRID-ABHC and HYBRID-RRT.

Afterwards, with a fixed number of iterations (15,000,000) the best deviation is $\delta = 0.005$.

We draw the following conclusions concerning HYBRID-ABHC and HYBRID-RRT, for which figure 8.2 displays the development of solution quality for some parameter values:

- The exact proportion between the types of moves is not a crucial setting: neither HYBRID-ABHC not HYBRID-RRT are very sensitive to the setting of p^{LS}. Among the wide ranges of values tested the resulting average deviations only vary between 1.20% and 1.52%, and 1.39% and 1.61%, respectively. We can state that switching from the parameter-free LS-ABHC to HYBRID-ABHC does not add much instability concerning parameter settings.

- The deviation δ of HYBRID-RRT is a much less sensitive parameter than it is for LS-RRT; its variation yields deviations from the BKS between 0.72% and 6.48%. Evidently, the incorporation of LNS moves adds stability in this respect.

8.2 Final Results

As explained at the beginning of this chapter we evaluate our standard VRP heuristics and their adaptations to rich VRPs under two parametrizations each. First, every heuristic is run on every problem using its standard parametrization determined in the previous section; then, we determine a specific parametrization for every heuristic/problem combination and compare the improved results against the standard parametrization and against state-of-the-art methods of the literature. We start listing the detailed results for the standard VRP in section 8.2.1 and then present all results for the VRPTW in section 8.2.2, for the VRPC in section 8.2.3, for the SDVRP in section 8.2.4, for the PVRP in section 8.2.5, and for the TTRP in section 8.2.6.

The tests are conducted with multiple numbers of iterations to examine how the solution quality depends on the running time and how much computational effort needs to be invested to obtain results of an appropriate quality. We define four iteration limit scenarios, see table 8.1, that lead to comparable (wall clock) running times among the heuristics in the standard VRP case on our testing platform. Due to randomization the test runs of every heuristic/problem/parametrization combination (except for LS-ABHC) are performed five times.

- *Short*: The purpose of the first scenario is to determine which solution quality can be obtained within short running times. In many practical scenarios there is not much computation time available when decisions have to be made quickly. Working with a DSS short response times to user actions are desirable for efficient and effective man/machine interaction. Here, solvers are required that can produce good solutions in a very short time.

- *Long*: In other real-world planning scenarios, e.g. dealing with strategic or tactical questions, running time is not critical. An important problem could even be solved overnight, but in this case the solutions must be of very high quality. For the long running time scenario we allow ten times the number of iterations as in the first scenario or twenty times the number if running times turn out to be comparably short on the data set used for a specific VRP (*Long**). The purpose is to assess the best solution quality a heuristic can yield, rather independent of running times and – potentially – inefficient implementations.

- *Medium*: To examine the development of solution quality over time a third scenario with medium running times is added.

	Short	Medium	Long	Long*
LS-ABHC	10,000	50,000	100,000	200,000
LS-RRT	7,500,000	37,500,000	75,000,000	150,000,000
LNS-RRT	10,000	50,000	100,000	200,000
HYBRID-ABHC	10,000	50,000	100,000	200,000
HYBRID-RRT	1,500,000	7,500,000	15,000,000	30,000,000
Running time (min)	≈ 2	≈ 10	≈ 20	≈ 40

Table 8.1: Iteration limit scenarios.

Allowing a reasonably fair comparison with results published by other authors, taking into account both solution quality and running time, we report adjusted running times to compensate different testing platforms. The use of the factors maintained by Dongarra (2011) is usually given as a recommendation for this purpose; but actually this approach, which indicates the floating-point performance (in Mflop/s) of computer systems in solving dense systems of linear equations, is not applied very often in the VRP literature. The list of systems given by Dongarra (2011) focuses on workstations and server systems, and it is often not easy to find adequate numbers for consumer PCs, which are used rather often for testing. Alternatively, a Java version of the benchmark[2] can be started easily via web browser and is accompanied by a survey of users' performance indicators[3]. We use these numbers, keeping in mind that they only allow a rough comparison of system performances for several reasons:

- The Java applet benchmark performs a *very* quick test (< 1 second on our system), which is probably not very accurate. Multiple sequential executions on the same computer yield different performance numbers.
- The performance numbers for similar computers posted by different users have a wide range, and their reliability is questionable. Probably, the operating system and the Java runtime environment have a major influence on the indicated performance.
- In general, the underlying benchmark is inappropriate to represent VRP heuristics due to its focus on solving systems of linear equations; in particular, it measures floating-point operations, while VRP heuristics, including our own implementation, often use integer numbers.

A widely-used benchmark specification for integer performance is *SPECint*, maintained by the Standard Performance Evaluation Corporation (SPEC)[4]. The current version of

[2]http://www.netlib.org/benchmark/linpackjava/
[3]http://www.netlib.org/benchmark/linpackjava/timings_list.html
[4]http://www.spec.org/

the test suite is *CPU2006*, with *CINT2006* containing 12 separate benchmarks to test the integer performance of a system. SPECint is a single-CPU benchmark, i.e. even on multi-processor or multi-core systems only a single core is used. The overall score of a system is the running time ratio compared to a certain reference machine. The SPEC maintains a list of performance measures, in particular a *base* and a *peak* value: the base scenario imposes rather strict rules on C/C++ compiler flag optimization, while the peak scenario allows full compiler optimization.

When comparing our results against the results of other authors we calculate modified running times of our own heuristics to mimic the performance on the others' testing environments. For our environment we assume a "Dongarra performance" of 533 Mflop/s, calculated as the average value over multiple benchmark executions. Whenever multiple performance numbers are given for other authors' CPUs in the survey referenced above we calculate the average as well. The CINT2006 base score of our system is 21.4.

8.2.1 Standard VRP

Table 8.2 summarizes the final results for the standard VRP on an aggregated level. For each of the five heuristics and each running time scenario we report four numbers of relative deviations from the BKS:

- The *best of five* deviations are calculated for the best solutions obtained during the five replications conducted per instance. Formally, let D be the set of instances, $\kappa = 5$ the number of replications, c_{dk} the solution cost obtained for an instance d in the k-th test run, $c_d^* := \min_{k \in \{1,\dots,\kappa\}} c_{dk}$ the best solution found for instance d, and c_d^{bks} the BKS cost for instance d. We state the average deviation over all instances

$$\frac{1}{|D|} \cdot \sum_{d \in D} \left(\frac{c_d^*}{c_d^{\text{bks}}} - 1 \right)$$

  as well as the maximum deviation among all instances

$$\max_{d \in D} \left(\frac{c_d^*}{c_d^{\text{bks}}} - 1 \right).$$

- For the *average of five* deviations we calculate the average cost obtained for an instance over the five replications instead of using the best solutions. Hence, in the formulas given above we replace c_d^* with $c_d^{\text{avg}} := \frac{1}{\kappa} \cdot \sum_{k=1}^{\kappa} c_{dk}$.

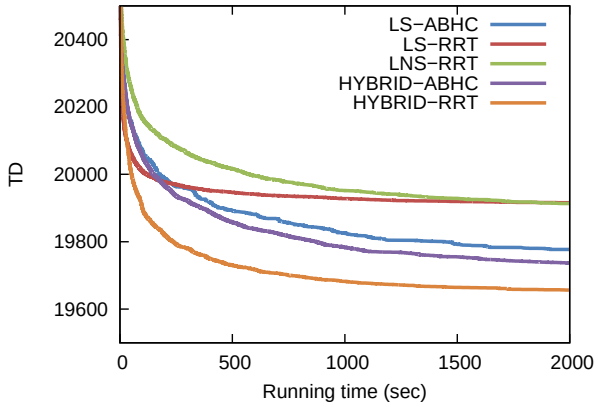


Figure 8.3: Convergence of solution quality for the standard VRP.

To improve readability we put the average deviations over all instances and all replications, respectively, in bold letters since we consider this indicator to represent the overall quality of a heuristic best. Finally, we state the average running time of a heuristic over all instances and replications under a given number of iterations. Note that the actual running time for a single instance can differ dramatically from the average since the set of instances has a wide range of problem sizes.

Figure 8.3 visualizes the development of solution quality over time: the curves show the total tour distance over all problem instances after a certain running time, averaged over all replications. From these TD convergences and from the deviation numbers of table 8.2 we learn the following about the behavior of the five heuristics solving the standard VRP:

- Evidently, the hybrid methods outperform the base heuristics. Recall that this improvement is yielded by merely combining the two neighborhood concepts in one unified search process, not adding any further intelligence in terms of specific hybridization strategies.

- Given an appropriate parameter configuration HYBRID-RRT yields quick improvements and produces the best solutions among all heuristics within any running time. HYBRID-ABHC converges more slowly and does not seem to catch up with HYBRID-RRT even with more time available.

- LS-ABHC performs only slightly worse than HYBRID-ABHC. In contrast to the two other base heuristics, which seem to get stuck after a certain time, it has the ability to explore the solution space thoroughly when longer running times are allowed.

		LS-ABHC			LS-RRT			LNS-RRT		
		10K	50K	100K	7.5M	37.5M	75M	10K	50K	100K
Best	Avg. dev. (%)	**2.55**	**1.53**	**1.28**	2.02	1.75	1.71	2.87	1.98	1.76
of five	Max. dev. (%)	4.88	3.51	3.20	4.24	3.81	3.75	6.98	4.87	4.43
Average	Avg. dev. (%)	-	-	-	**2.49**	**2.25**	**2.14**	**3.14**	**2.30**	**2.02**
of five	Max. dev. (%)	-	-	-	5.09	5.07	5.02	7.12	5.32	4.67
	Avg. time (s)	121.12	576.81	1137.86	119.02	581.77	1147.92	114.51	567.63	1131.78
		HYBRID-ABHC			HYBRID-RRT					
		10K	50K	100K	1.5M	7.5M	15M			
Best	Avg. dev. (%)	2.15	1.27	0.82	1.26	0.63	0.51			
of five	Max. dev. (%)	4.79	2.98	2.08	2.77	1.47	1.29			
Average	Avg. dev. (%)	**2.54**	**1.53**	**1.12**	**1.67**	**0.92**	**0.77**			
of five	Max. dev. (%)	5.07	3.53	2.47	3.14	1.71	1.44			
	Avg. time (s)	101.09	482.19	960.18	116.81	569.68	1134.52			

Table 8.2: Aggregated results for the standard VRP.

Finally, we compare our best heuristic, HYBRID-RRT, against the "AGES" heuristic presented by Mester and Bräysy (2007), which is widely recognized as one of the best methods for the standard VRP. Combining evolution strategies with guided local search and large neighborhood search the authors present results of two configurations, "best" and "fast", which focus on solution quality and computation time, respectively. We select "best" for this comparison, yet note that "fast" still produces very good results with incredibly short running times.

The general format of table 8.3 is also used for the remaining rich VRPs later on in this chapter. Here, the rows indicate the CMT and GWKC instances without tour length constraints on which we conduct our tests.

- For each instance we first state the name[5], the number of customers, and the BKS cost obtained from Ropke (2011).
- The following group of columns holds the costs and the computation times given by Mester and Bräysy (2007) for a single solution run as well as the deviations from the BKS.
- The average results from the 1,500,000 iterations scenario of HYBRID-RRT are displayed in the next group of columns, including column $time^*$ for scaled computation times: Mester and Bräysy (2007) obtained their results on a Pentium IV 2.8 GHz for which we assume a Dongarra (2011) factor of 238 Mflop/s and a CINT2006 base score of 9.31, leading to a factor of 2.27 to multiply our own running times with. The adjusted running times are similar to the running times of Mester and Bräysy (2007) on average, allowing a rather fair comparison between the solution methods.
- Finally, we present the *best of five* results of long running times to demonstrate the solution quality that can be obtained by our heuristic in the best case. Note that in the last column we still give average running times instead of the total computation times required for five replications.
- In the bottom rows of table 8.3 we add several aggregations, stating total tour distances and running times as well as average and maximum deviations and running times over all instances.

Evidently, our best heuristic cannot compete with the elaborate AGES method, which converges to the BKS with an average gap of 0.15% only, compared to 1.67% of our method

[5]Since different names for standard VRP instances are commonly used in the literature we specify two alternative notations.

Instance			BKS	Mester and Bräysy (2007) *Single run*			HYBRID-RRT *1.5M, Avg. of five*				HYBRID-RRT *15M, Best of five*		*Avg.*
Name		$\|N_c\|$	TD	TD	dev (%)	time (s)	TD	dev (%)	time (s)	time* (s)	TD	dev (%)	time (s)
E051-05e	CMT-1	50	524.61	524.61	0.00	0.20	524.61	0.00	10.97	24.90	524.61	0.00	106.61
E076-10e	CMT-2	75	835.26	835.26	0.00	5.50	839.72	0.53	15.52	35.22	838.60	0.40	150.76
E101-08e	CMT-3	100	826.14	826.14	0.00	1.00	828.27	0.26	31.72	71.98	827.39	0.15	312.79
E101-10c	CMT-12	100	819.56	819.56	0.00	0.20	819.56	0.00	36.11	81.94	819.56	0.00	356.50
E121-07c	CMT-11	120	1042.11	1042.11	0.00	1.10	1042.29	0.02	35.88	81.41	1042.12	0.00	355.36
E151-12c	CMT-4	150	1028.42	1028.42	0.00	10.20	1045.20	1.63	54.48	123.61	1029.79	0.13	520.88
E200-17c	CMT-5	199	1291.29	1291.29	0.00	2160.00	1323.24	2.47	72.87	165.34	1293.59	0.18	718.33
E241-22k	GWKC-17	240	707.76	707.79	0.00	30.20	713.46	0.81	104.42	236.94	709.09	0.19	1033.79
E253-27k	GWKC-13	252	857.19	859.11	0.22	400.00	872.34	1.77	103.63	235.14	863.71	0.76	1027.97
E256-14k	GWKC-09	255	580.48	583.39	0.50	360.20	595.49	2.59	104.41	236.91	582.54	0.35	1014.64
E301-28k	GWKC-18	300	995.39	998.73	0.34	150.60	1017.04	2.18	138.78	314.91	1003.26	0.79	1319.53
E321-30k	GWKC-14	320	1080.55	1081.31	0.07	48.50	1096.20	1.45	142.26	322.79	1086.72	0.57	1392.43
E324-16k	GWKC-10	323	738.73	741.56	0.38	75.00	754.99	2.20	146.36	332.09	744.18	0.74	1395.63
E361-33k	GWKC-19	360	1366.14	1366.86	0.05	23.30	1397.82	2.32	172.82	392.14	1375.62	0.69	1703.38
E397-34k	GWKC-15	396	1340.24	1345.23	0.37	27.60	1369.01	2.15	182.62	414.37	1350.22	0.74	1737.02
E400-18k	GWKC-11	399	914.75	918.45	0.40	440.80	938.76	2.62	192.20	436.11	921.81	0.77	1860.78
E421-41k	GWKC-20	420	1819.99	1820.09	0.01	230.00	1877.19	3.14	203.74	462.29	1843.42	1.29	1977.08
E481-38k	GWKC-16	480	1616.33	1622.69	0.39	800.10	1659.21	2.65	231.25	524.71	1633.48	1.06	2228.71
E484-19k	GWKC-12	483	1106.33	1107.19	0.08	647.70	1139.52	3.00	239.39	543.18	1116.21	0.89	2343.68
Total			19491.27	19519.79		5412.20	19853.91		2219.43	5035.99	19605.93		21555.87
Average					0.15	284.85		1.67	116.81	265.05		0.51	1134.52
Max					0.50	2160.00		3.14	239.39	543.18		1.29	2343.68

Table 8.3: Comparison with a state-of-the-art method for the standard VRP.

under similar (average) running times. Even with more iterations we can improve the gap only to 0.51%, which can still be appropriate in practical applications, yet. Interestingly, our running times clearly increase with the instance size, while Mester and Bräysy (2007) use a stopping criterion that triggers when improvements are made no more and that leads to greatly varying running times.

8.2.2 Vehicle Routing Problem with Time Windows

We evaluate our VRPTW adaptations on the classical and well-known data set of Solomon (1987) that contains fifty-six 100-customer instances with different characteristics of time windows (narrow / wide) and customer distribution (clustered / random / mixed). For this data set and for other large-scale instances the BKS are maintained by SINTEF (2011). The hierarchical objective of the VRPTW involves difficulties concerning the interpretation of results since the intuition that smaller numbers of tours in solutions are associated with smaller total tour distances automatically is not always correct. In fact, increasing the number of allowed tours can enable shorter routings in some cases. This effect makes parameter tuning a bit complicated: a small total distance of a solution can result from a) a good parameter setting or b) the heuristic having failed to minimize tours properly.

All of our test runs for the VRPTW combine two separate phases: the first phase is always the vehicle minimization procedure of LNS-RRT, and the best feasible solution found during that phase is used as an initial solution for one of the five regular heuristics for distance minimization. During initial tuning for the first phase we selected a deviation $\delta = 0.001$ for LNS-RRT, and it appeared helpful to shuffle around more customers per iteration than during distance minimization, so that we set a removal percentage of $r = 0.6$ instead of $r = 0.3$. Our experience is that during the second phase the number of tours can hardly be reduced further, so that we optimized parameters for this phase with respect to the distance objective only. Selecting $\delta = 0.0075$ for LS-RRT, $\delta = 0.04$ for LNS-RRT, and $p^{\mathrm{LS}} = 0.9925, \delta = 0.03$ for HYBRID-RRT we could yield measurable but not dramatic improvements (except for LS-RRT). For HYBRID-ABHC the standard parametrization is already appropriate.

In all tests we split up the computational effort evenly between the phases: for example, the short running time scenarios of LS-ABHC, LNS-RRT, and HYBRID-ABHC combine 5,000 iterations per phase. The equivalent scenario of LS-RRT spends 5,000 iterations in the first phase and 3,750,000 iterations in the second phase, and in HYBRID-RRT

	Short	Medium	Long
LS-ABHC	411.4	407.8	407.0
LS-RRT	412.0	408.6	406.8
LNS-RRT	412.4	408.6	407.0
HYBRID-ABHC	411.4	407.4	406.8
HYBRID-RRT	411.8	409.4	407.0

Table 8.4: Aggregated results for the VRPTW: number of vehicles.

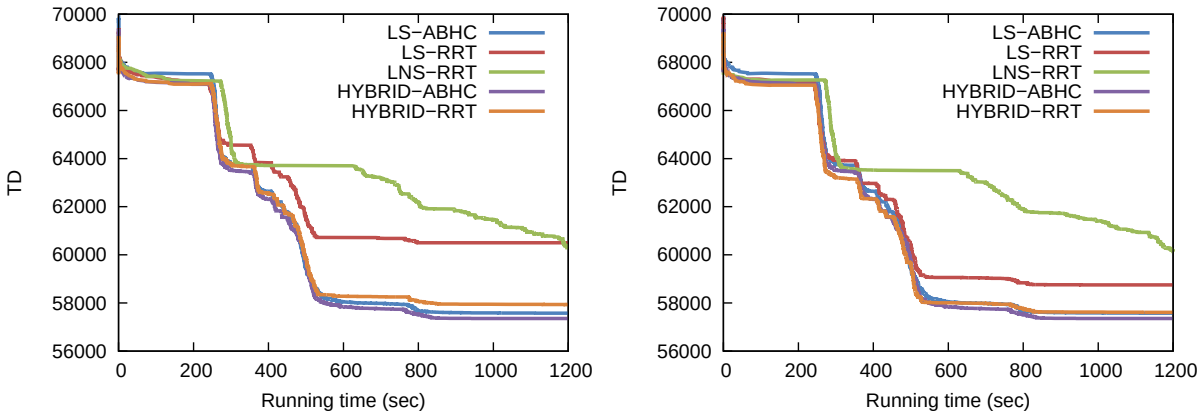
the split is 5,000 / 750,000 iterations. The checking effort for time window constraints is marginal, so that due to the short running times we double the number of iterations of the long running time scenarios as shown in table 8.1.

Table 8.4 first shows the average total number of vehicles (NV) over all instances obtained for each heuristic and running time scenario. There is no significant difference among the heuristics – the variations are rather related to the randomness of the vehicle minimization procedure. Next, table 8.5 gives an overview of the aggregated results concerning total distances of the five heuristics under standard parametrization and under VRPTW-specific parametrization. Note that, exceptionally, we present average results for LS-ABHC, which is not deterministic in this case due to the randomized first phase. Whenever improving parameter settings cannot be determined for a heuristic we indicate the missing deviations by "-*"; the average deviations that indicate the overall quality of a heuristic are printed in bold letters. Figure 8.4 displays the convergences of total tour distance. The results of LS-ABHC, HYBRID-ABHC, and HYBRID-RRT are of similar good quality; taking into account both objectives HYBRID-ABHC is the best heuristic on average. With respect to absolute tour distances LNS-RRT is rather weak, as can be seen in figure 8.4, yet the relative deviations from the BKS are much better than those of LS-RRT.

Many concepts of VRP heuristics have been applied to the VRPTW, so that there is a great variety of solution approaches in the literature. The ALNS heuristic presented by Pisinger and Ropke (2007), from which we adopted many elements for our own LNS implementation, currently is one of the best heuristics to minimize the number of vehicles in the time window case. Table 8.6 compares results under short running times: in this comparison we present our results generated by HYBRID-RRT, which are slightly worse than those of HYBRID-ABHC on average but better considering the best solutions over

			LS-ABHC			LS-RRT			LNS-RRT		
			10K	50K	200K	7.5M	37.5M	150M	10K	50K	200K
Standard parametrization	*Best of five*	Avg. dev. (%)	0.55	0.36	0.31	2.69	2.93	3.07	1.31	0.87	0.70
		Max. dev. (%)	5.98	4.37	3.71	11.87	11.58	13.52	13.62	10.01	4.74
	Average of five	Avg. dev. (%)	**1.07**	**0.77**	**0.64**	5.43	5.79	5.68	2.81	2.50	2.74
		Max. dev. (%)	6.94	6.17	6.37	17.45	19.73	17.93	17.32	13.09	15.37
Improved parametrization	*Best of five*	Avg. dev. (%)	-	-	-	1.48	1.23	1.36	1.01	0.68	0.40
		Max. dev. (%)	-	-	-	8.63	7.10	9.24	10.01	10.51	3.75
	Average of five	Avg. dev. (%)	-	-	-	**2.09**	**2.57**	**2.63**	**2.03**	**1.90**	**1.54**
		Max. dev. (%)	-	-	-	12.39	9.81	11.81	16.88	12.81	10.43
		Avg. time (s)	44.03	218.14	872.87	49.15	244.39	978.51	49.13	243.50	978.80
			HYBRID-ABHC			HYBRID-RRT					
			10K	50K	200K	1.5M	7.5M	30M			
Standard parametrization	*Best of five*	Avg. dev. (%)	0.43	0.22	0.10	0.86	0.58	0.53			
		Max. dev. (%)	5.35	2.73	2.40	7.22	4.86	4.46			
	Average of five	Avg. dev. (%)	**0.91**	**0.51**	**0.26**	1.45	1.52	1.29			
		Max. dev. (%)	5.49	4.49	3.88	10.75	7.34	5.62			
Improved parametrization	*Best of five*	Avg. dev. (%)	-*	-*	-*	0.31	0.30	0.35			
		Max. dev. (%)	-*	-*	-*	2.74	2.48	3.59			
	Average of five	Avg. dev. (%)	-*	-*	-*	**0.71**	**0.50**	**0.71**			
		Max. dev. (%)	-*	-*	-*	3.75	4.76	5.50			
		Avg. time (s)	35.79	178.00	710.00	35.51	175.17	704.92			

Table 8.5: Aggregated results for the VRPTW: total distance.



(a) Standard parametrization

(b) Improved parametrization

Figure 8.4: Convergence of solution quality for the VRPTW: total distance.

all replications.[6] Our running times are multiplied with 2.22 to allow a fair comparison with their times obtained on a Pentium IV 3.0 GHz for which we assume a Dongarra (2011) factor of 242 Mflop/s and a CINT2006 base score of 9.55.

Evidently, a particular strength of the ALNS heuristic is the ability to generate solutions with a small number of tours quickly. Although having implemented basically the same vehicle minimization procedure our solutions have considerably more tours (411.8 compared to 407.5 on average), yet with slightly smaller distances overall. A possible explanation is that we omitted some of the features, most notably the adaptive weights adjustment and some removal heuristics. Nevertheless, the results produced with longer running times indicate that our heuristics can generate very high quality solutions as well: the best solutions within five replications of HYBRID-RRT have the same numbers of vehicles as the BKS, and the average relative gap of total distance is only 0.35 percent.

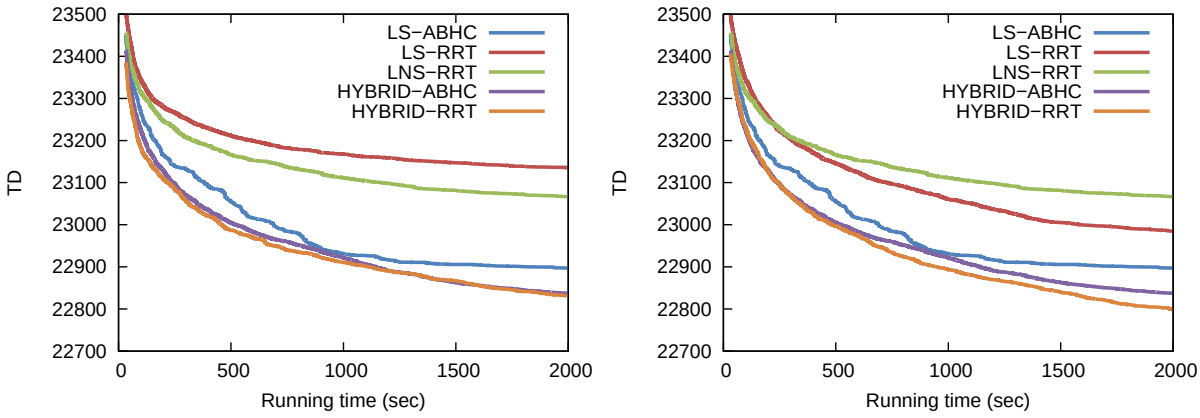
8.2.3 Vehicle Routing Problem with Compartments

The VRPC formulation introduced in Derigs et al. (2011a) is more complex than most other compartment-related VRP variants in the literature. A large set of testing instances has been designed specifically for this variant, yet due to the lack of reference solutions by other authors we conduct our tests on artificial instances generated by El Fallahi et al. (2008) and Muyldermans and Pang (2010), who transform several well-known standard VRP instances. These instances (23 instances without tour length constraints) incorporate

[6]The detailed results of the ALNS heuristic of Pisinger and Ropke (2007) are obtained from Pisinger and Ropke (2005).

Instance		BKS		Pisinger and Ropke (2007) *25K, Avg. of ten*					HYBRID-RRT *1.5M, Avg. of five*						HYBRID-RRT *30M, Best of five*				*Avg.*
Name	$\|N_c\|$	NV	TD	NV	dev	TD	dev (%)	time (s)	NV	dev	TD	dev (%)	time (s)	time* (s)	NV	dev	TD	dev (%)	time (s)
c101	100	10	828.94	10.0	0.0	828.94	0.00	29	10.0	0.0	828.94	0.00	23.99	53.30	10	0	828.94	0.00	477.24
c102	100	10	828.94	10.0	0.0	828.94	0.00	59	10.0	0.0	828.94	0.00	24.37	54.14	10	0	828.94	0.00	483.69
c103	100	10	828.06	10.0	0.0	828.06	0.00	65	10.0	0.0	828.06	0.00	25.53	56.72	10	0	828.06	0.00	511.57
c104	100	10	824.78	10.0	0.0	824.78	0.00	69	10.0	0.0	824.78	0.00	26.86	59.67	10	0	824.78	0.00	534.43
c105	100	10	828.94	10.0	0.0	828.94	0.00	31	10.0	0.0	828.94	0.00	24.59	54.62	10	0	828.94	0.00	489.79
c106	100	10	828.94	10.0	0.0	828.94	0.00	32	10.0	0.0	828.94	0.00	25.16	55.90	10	0	828.94	0.00	496.89
c107	100	10	828.94	10.0	0.0	828.94	0.00	32	10.0	0.0	828.94	0.00	26.10	57.98	10	0	828.94	0.00	515.28
c108	100	10	828.94	10.0	0.0	828.94	0.00	61	10.0	0.0	828.94	0.00	26.07	57.91	10	0	828.94	0.00	517.65
c109	100	10	828.94	10.0	0.0	828.94	0.00	64	10.0	0.0	828.94	0.00	26.89	59.74	10	0	828.94	0.00	531.50
c201	100	3	591.56	3.0	0.0	591.56	0.00	78	3.0	0.0	591.56	0.00	37.72	83.80	3	0	591.56	0.00	738.96
c202	100	3	591.56	3.0	0.0	591.56	0.00	88	3.0	0.0	591.56	0.00	37.66	83.66	3	0	591.56	0.00	746.34
c203	100	3	591.17	3.0	0.0	591.17	0.00	96	3.0	0.0	591.17	0.00	38.36	85.23	3	0	591.17	0.00	764.70
c204	100	3	590.60	3.0	0.0	590.60	0.00	102	3.0	0.0	590.60	0.00	41.68	92.60	3	0	590.60	0.00	830.73
c205	100	3	588.88	3.0	0.0	588.88	0.00	81	3.0	0.0	588.88	0.00	41.06	91.23	3	0	588.88	0.00	816.91
c206	100	3	588.49	3.0	0.0	588.49	0.00	83	3.0	0.0	588.49	0.00	41.65	92.54	3	0	588.49	0.00	832.75
c207	100	3	588.29	3.0	0.0	588.29	0.00	84	3.0	0.0	588.29	0.00	43.53	96.70	3	0	588.29	0.00	860.12
c208	100	3	588.32	3.0	0.0	588.32	0.00	85	3.0	0.0	588.32	0.00	42.88	95.27	3	0	588.32	0.00	854.51
r101	100	19	1645.79	19.0	0.0	1650.86	0.31	55	19.0	0.0	1671.14	1.54	29.71	66.00	19	0	1658.91	0.80	583.55
r102	100	17	1486.12	17.0	0.0	1486.89	0.05	62	17.0	0.0	1509.30	1.56	30.80	68.44	17	0	1496.41	0.69	605.42
r103	100	13	1292.68	13.0	0.0	1294.89	0.17	64	13.0	0.0	1314.88	1.72	28.63	63.61	13	0	1298.66	0.46	557.50
r104	100	9	1007.24	9.8	0.8	987.85	-1.93	61	10.0	1.0	995.25	-1.19	26.64	59.19	9	0	1007.31	0.01	512.19
r105	100	14	1377.11	14.0	0.0	1378.77	0.12	56	14.0	0.0	1387.98	0.79	27.53	61.16	14	0	1377.11	0.00	534.88
r106	100	12	1251.98	12.0	0.0	1258.40	0.51	61	12.0	0.0	1271.58	1.57	27.48	61.06	12	0	1259.20	0.58	533.66
r107	100	10	1104.66	10.0	0.0	1118.18	1.22	52	10.0	0.0	1125.68	1.90	26.05	57.88	10	0	1109.92	0.48	509.19
r108	100	9	960.88	9.0	0.0	969.37	0.88	40	9.2	0.2	970.19	0.97	25.81	57.35	9	0	964.81	0.41	497.80
r109	100	11	1194.73	11.1	0.1	1213.09	1.54	47	12.0	1.0	1164.65	-2.52	26.99	59.96	11	0	1198.72	0.33	516.55
r110	100	10	1118.59	10.0	0.0	1149.56	2.77	41	11.0	1.0	1089.48	-2.60	26.68	59.27	10	0	1119.02	0.04	509.63
r111	100	10	1096.72	10.0	0.0	1112.14	1.41	46	10.0	0.0	1123.16	2.41	26.11	58.00	10	0	1097.23	0.05	512.20
r112	100	9	982.14	9.5	0.5	983.16	0.10	58	9.8	0.8	974.92	-0.74	26.31	58.45	9	0	991.52	0.96	509.64
r201	100	4	1252.37	4.0	0.0	1253.23	0.07	133	4.0	0.0	1252.89	0.04	35.50	78.86	4	0	1252.37	0.00	705.81
r202	100	3	1191.70	3.0	0.0	1229.81	3.20	96	3.0	0.0	1199.13	0.62	40.99	91.06	3	0	1195.99	0.36	852.80
r203	100	3	939.50	3.0	0.0	944.64	0.55	164	3.0	0.0	954.90	1.64	45.44	100.94	3	0	943.52	0.43	898.97
r204	100	2	825.52	2.0	0.0	841.48	1.93	182	2.2	0.2	836.14	1.29	65.21	144.87	2	0	832.06	0.79	1366.17
r205	100	3	994.42	3.0	0.0	1018.90	2.46	97	3.0	0.0	1013.62	1.93	46.81	104.00	3	0	994.43	0.00	918.72
r206	100	3	906.14	3.0	0.0	923.91	1.96	192	3.0	0.0	934.50	3.13	47.85	106.30	3	0	906.97	0.09	944.61
r207	100	2	890.61	2.0	0.0	928.28	4.23	180	2.2	0.2	901.30	1.20	58.75	130.52	2	0	914.18	2.65	1265.17
r208	100	2	726.75	2.0	0.0	736.12	1.29	185	2.0	0.0	731.36	0.64	71.70	159.29	2	0	726.82	0.01	1412.71
r209	100	3	909.16	3.0	0.0	926.72	1.93	101	3.0	0.0	921.11	1.31	47.83	106.25	3	0	909.31	0.02	924.12
r210	100	3	939.34	3.0	0.0	955.02	1.67	112	3.0	0.0	959.10	2.10	46.42	103.14	3	0	948.65	0.99	915.77
r211	100	2	885.71	2.3	0.3	889.99	0.48	216	2.2	0.2	895.33	1.09	54.02	120.02	2	0	903.19	1.97	1144.70
rc101	100	14	1696.94	14.2	0.2	1688.35	-0.51	53	14.4	0.4	1679.40	-1.03	27.10	60.21	14	0	1697.43	0.03	525.75
rc102	100	12	1554.75	12.1	0.1	1547.04	-0.50	56	12.4	0.4	1543.40	-0.73	26.12	58.02	12	0	1557.22	0.16	503.93
rc103	100	11	1261.67	11.0	0.0	1270.78	0.72	58	11.0	0.0	1285.11	1.86	25.42	56.46	11	0	1265.85	0.33	496.94
rc104	100	10	1135.48	10.0	0.0	1135.80	0.03	60	10.0	0.0	1155.67	1.78	24.67	54.80	10	0	1138.13	0.23	483.25
rc105	100	13	1629.44	13.0	0.0	1640.18	0.66	54	13.2	0.2	1627.10	-0.14	26.30	58.42	13	0	1631.80	0.14	513.38
rc106	100	11	1424.73	11.5	0.5	1413.07	-0.82	49	12.0	1.0	1393.16	-2.22	25.41	56.46	11	0	1432.12	0.52	492.21
rc107	100	11	1230.48	11.0	0.0	1232.48	0.16	56	11.0	0.0	1245.77	1.24	24.46	54.34	11	0	1230.54	0.01	478.29
rc108	100	10	1139.82	10.0	0.0	1167.55	2.43	41	10.0	0.0	1160.37	1.80	24.22	53.81	10	0	1139.82	0.00	469.94
rc201	100	4	1406.91	4.0	0.0	1417.80	0.77	83	4.0	0.0	1414.13	0.51	35.90	79.75	4	0	1406.94	0.00	698.97
rc202	100	3	1365.65	3.0	0.0	1405.16	2.89	96	3.2	0.2	1356.13	-0.70	39.64	88.06	3	0	1414.71	3.59	828.03
rc203	100	3	1049.62	3.0	0.0	1075.51	2.47	100	3.0	0.0	1079.88	2.88	44.97	99.91	3	0	1058.33	0.83	911.49
rc204	100	3	798.41	3.0	0.0	818.00	2.45	228	3.0	0.0	823.42	3.13	48.28	107.25	3	0	798.61	0.02	921.40
rc205	100	4	1297.19	4.0	0.0	1318.01	1.61	134	4.0	0.0	1310.30	1.01	36.01	80.00	4	0	1297.65	0.04	712.87
rc206	100	3	1146.32	3.0	0.0	1155.91	0.84	87	3.0	0.0	1183.88	3.28	44.20	98.21	3	0	1146.32	0.00	858.62
rc207	100	3	1061.14	3.0	0.0	1095.29	3.22	96	3.0	0.0	1091.60	2.87	44.39	98.62	3	0	1079.19	1.70	892.50
rc208	100	3	828.14	3.0	0.0	834.83	0.81	109	3.0	0.0	859.21	3.75	48.13	106.93	3	0	828.71	0.07	953.27
Total		405	57180.84	407.5		57641.31		4800	411.8		57580.37		1988.55	4417.89	405		57403.94		39475.64
Average					0.0		0.79	85.71		0.1		0.71	35.51	78.89		0		0.35	704.92
Max					0.8		4.23	228		1.0		3.75	71.70	159.29		0		3.59	1412.71

Table 8.6: Comparison with a state-of-the-art method for the VRPTW.



(a) Standard parametrization

(b) Improved parametrization

Figure 8.5: Convergence of solution quality for the VRPC.

two products, two fixed-size compartments per vehicle, and each product is compatible with one compartment exclusively. In the transformation the original demand of each customer is split into two orders of equal amounts, one order of each product type, and vehicles have the same total capacities as in the original instances with two compartments of half the total capacity each. A solution for the original VRP is always feasible for the associated VRPC and, conversely, the optimal VRPC solutions correspond to their optimal VRP counterparts. For the following analysis we take the BKS values for standard VRP instances from Ropke (2011) or, if not listed, from Toth and Vigo (2002).[7]

Initial tests yield an improved VRPC-specific parametrization of $\delta = 0.003$ for LS-RRT and $p^{\text{LS}} = 0.9975, \delta = 0.007$ for HYBRID-RRT; for LNS-RRT and HYBRID-ABHC the standard settings perform best. Table 8.7 compares the aggregated results for the five heuristics under standard parametrization and improved parametrization; figure 8.5 displays the development of solution quality for the two parametrizations. Note that in table 8.7 it appears that the optimized parametrization of HYBRID-RRT is worse than its standard parametrization; however, the higher value of p^{LS} (0.9975 instead of 0.995) leads to shorter total running times for the same numbers of iterations, and the curves of figure 8.5 confirm that the heuristic performs slightly better with the new settings.

Evidently, running times increase significantly compared to the standard VRP, which is due to double instance sizes (each customer is translated to two individual orders) and only partly an effect of the additional checking effort for compartment constraints. The

[7]For instances E076-07u and E076-08s we could not find appropriate solution values based on floating-point distances, so that we have to resort to the best solutions obtained with exact solution methods, which usually assume rounded or truncated distances.

			LS-ABHC			LS-RRT			LNS-RRT		
			10K	50K	100K	7.5M	37.5M	75M	10K	50K	100K
Standard parametrization	*Best of five*	Avg. dev. (%)	**2.90**	**2.01**	**1.71**	3.67	3.28	3.34	3.43	2.74	2.59
		Max. dev. (%)	6.58	5.76	3.92	6.89	7.49	7.49	7.84	6.76	6.66
	Average of five	Avg. dev. (%)	-	-	-	4.08	3.66	3.63	**3.93**	**3.24**	**3.06**
		Max. dev. (%)	-	-	-	7.31	7.49	7.49	8.43	6.97	6.84
Improved parametrization	*Best of five*	Avg. dev. (%)	-	-	-	3.19	2.45	2.33	-*	-*	-*
		Max. dev. (%)	-	-	-	5.98	5.71	5.61	-*	-*	-*
	Average of five	Avg. dev. (%)	-	-	-	**3.62**	**2.99**	**2.76**	-*	-*	-*
		Max. dev. (%)	-	-	-	6.99	5.84	5.80	-*	-*	-*
		Avg. time (s)	544.79	2544.23	5132.54	377.85	1770.17	3560.82	176.38	850.20	1691.09
			HYBRID-ABHC			HYBRID-RRT					
			10K	50K	100K	1.5M	7.5M	15M			
Standard parametrization	*Best of five*	Avg. dev. (%)	2.45	1.66	1.27	2.41	1.61	1.35			
		Max. dev. (%)	5.94	4.22	3.40	5.62	4.53	3.90			
	Average of five	Avg. dev. (%)	**2.86**	**2.02**	**1.61**	2.85	2.01	1.77			
		Max. dev. (%)	6.36	5.01	3.83	6.08	4.64	4.18			
Improved parametrization	*Best of five*	Avg. dev. (%)	-*	-*	-*	2.70	1.79	1.51			
		Max. dev. (%)	-*	-*	-*	7.88	5.09	4.63			
	Average of five	Avg. dev. (%)	-*	-*	-*	**3.21**	**2.25**	**1.96**			
		Max. dev. (%)	-*	-*	-*	8.40	5.50	5.36			
		Avg. time (s)	370.48	1735.45	3472.92	163.60	764.44	1504.96			

Table 8.7: Aggregated results for the VRPC.

increase is most apparent for LS-ABHC in which the neighborhood size depends quadratically on the number of orders; in contrast, the cap on the number of orders to be removed in LNS (at most 100 orders) limits the additional computational effort, so that running times increase mildly.

The adapted LS-ABHC, HYBRID-ABHC, and HYBRID-RRT heuristics clearly produce the best results for the VRPC with deviations from the BKS between 1% and 2% in the running time scenarios tested. Probably, further improvements can still be obtained with longer running times. The convergences of solution quality of the two hybrid methods are almost identical under standard settings; HYBRID-RRT is slightly favorable after tuning. LS-RRT and LNS-RRT alone perform rather weak; some improvement of LS-RRT can be obtained by tuning.

In table 8.8 we compare our HYBRID-RRT heuristic against the guided local search heuristic by Muyldermans and Pang (2010), which currently is one of the best VRPC heuristics in the literature.[8][9] To compare our running times fairly with those given by the authors for a Pentium M740 1.73 GHz we multiply our times with 2.47; this multiplier results from assuming a Dongarra (2011) factor of 221 Mflop/s and a CINT2006 base score of 8.45 for their machine. Actually, the CINT2006 score of the M740 CPU is not available to us, thus we simply scale the score of the Pentium M750 1.86 GHz CPU according to the clock rate.

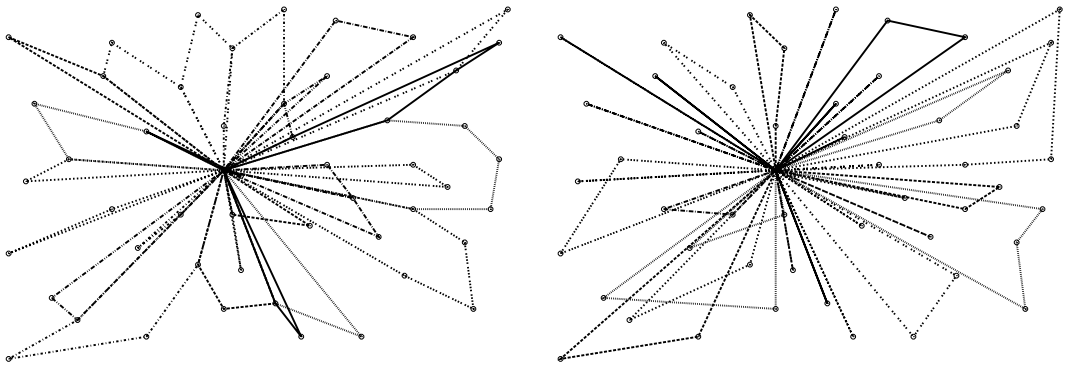
Muyldermans and Pang (2010) conduct experiments with multiple numbers of iterations; table 8.8 shows their results obtained with 1,200,000 iterations. Then, in the following columns we state our average HYBRID-RRT results with 1,500,000 iterations. Evidently, our heuristic does not yield an equal quality in this comparison: a deviation of 3.59% on average compared to 2.86% with only slightly smaller (scaled) running times. But as already seen above better results are possible with more time available: table 8.8 also lists the best solutions obtained from five replications of 15,000,000 iterations each, resulting in an average deviation of 1.70%. We put on record that HYBRID-RRT can produce good results but is not very fast.

[8] In Derigs et al. (2011a) we present a high-quality heuristic for the VRPC that generates even better solutions than the ones given by Muyldermans and Pang (2010), yet since it uses the same concepts as our framework-based heuristics a comparison of results would not yield as much insight as the comparison with the method of Muyldermans and Pang (2010).

[9] For the comparison we leave out four instances from our calibration data set for which Muyldermans and Pang (2010) do not present results: E072-04f, E076-07u, E076-08s, and E135-07f.

Instance		BKS	Muyldermans and Pang (2010) *1.2M*			HYBRID-RRT *1.5M, Avg. of five*				HYBRID-RRT *15M, Best of five*		*Avg.*
Name	$\|N_c\|$	TD	TD	dev (%)	time (s)	TD	dev (%)	time (s)	time* (s)	TD	dev (%)	time (s)
E051-05e	50	524.61	524.61	0.00	538.20	524.61	0.00	23.43	64.28	524.61	0.00	232.06
E076-10e	75	835.26	837.40	0.26	516.60	845.24	1.19	30.55	83.82	838.60	0.40	299.99
E101-08e	100	826.14	829.84	0.45	564.60	829.22	0.37	79.85	219.08	827.39	0.15	774.97
E101-10c	100	819.56	819.56	0.00	558.60	819.56	0.00	90.88	249.33	819.56	0.00	896.48
E121-07c	120	1042.11	1048.67	0.63	499.80	1043.06	0.09	66.75	183.12	1042.12	0.00	654.99
E151-12c	150	1028.42	1040.18	1.14	606.60	1050.18	2.12	101.52	278.52	1039.96	1.12	1064.24
E200-17c	199	1291.29	1313.96	1.76	612.00	1339.92	3.77	115.40	316.61	1308.84	1.36	913.50
E241-22k	240	707.76	719.71	1.69	658.20	729.28	3.04	141.83	389.10	715.30	1.07	1373.77
E253-27k	252	857.19	885.03	3.25	605.40	883.77	3.10	147.80	405.49	866.39	1.07	1290.44
E256-14k	255	580.48	605.65	4.34	783.60	608.57	4.84	154.12	422.82	594.24	2.37	1467.24
E301-28k	300	995.39	1036.22	4.10	702.00	1046.30	5.11	214.96	589.75	1028.04	3.28	2037.35
E321-30k	320	1080.55	1125.75	4.18	619.20	1114.12	3.11	198.92	545.72	1096.43	1.47	1810.79
E324-16k	323	738.73	770.98	4.37	848.40	779.47	5.52	221.37	607.34	752.17	1.82	2039.12
E361-33k	360	1366.14	1404.58	2.81	738.00	1440.56	5.45	245.06	672.31	1406.12	2.93	2338.00
E397-34k	396	1340.24	1405.56	4.87	646.80	1397.79	4.29	263.30	722.36	1366.07	1.93	2280.46
E400-18k	399	914.75	965.56	5.55	940.20	972.11	6.27	302.99	831.25	936.80	2.41	2741.42
E421-41k	420	1819.99	1897.76	4.27	808.80	1927.73	5.92	320.12	878.23	1904.32	4.63	2952.73
E481-38k	480	1616.33	1691.45	4.65	681.00	1707.92	5.67	346.30	950.07	1659.14	2.65	3056.45
E484-19k	483	1106.33	1173.37	6.06	1087.80	1199.23	8.40	398.07	1092.09	1147.22	3.70	3533.36
Total		19491.27	20095.84		13015.80	20258.65		3463.22	9501.28	19873.34		31757.36
Average				2.86	685.04		3.59	182.27	500.07		1.70	1671.44
Max				6.06	1087.80		8.40	398.07	1092.09		4.63	3533.36

Table 8.8: Comparison with a state-of-the-art method for the VRPC.



(a) SDVRP solution (b) VRP solution

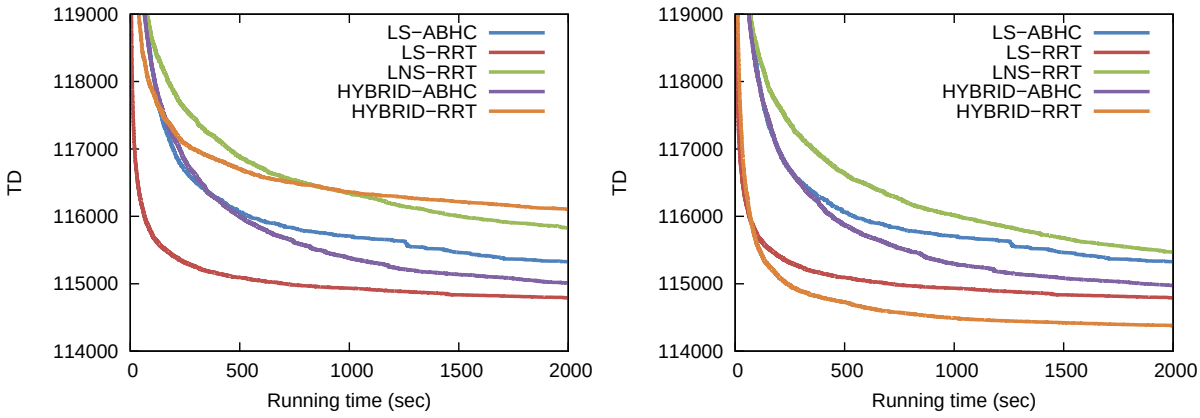
Figure 8.6: Solutions with and without split deliveries.

8.2.4 Split Delivery Vehicle Routing Problem

Our test bed for the SDVRP is a data set of 36 instances by Archetti et al. (2008). These instances are derived from a set of basic instances, ranging between 50 and 199 customers, by varying the proportions between customer demands and vehicle capacities: each customer's demand is chosen from an interval $[\alpha \cdot Q, \gamma \cdot Q]$ with lower- and upper-bound parameters α and γ.[10] Complexity rises with increasing values for α and γ since the potential of cost savings by splits increases when only a very small number of customers can be served together completely by one vehicle. Figure 8.6 illustrates the split/non-split cases for instance "p01" with $\alpha = 10\%$ and $\gamma = 90\%$. In the SDVRP solution displayed in figure 8.6(a) (26 tours, distance 1488.58) several customers are served by two vehicles, while the VRP solution in figure 8.6(b) (31 tours, distance 1678.51) has several tours containing a single customer only and some long arcs connecting the remaining customers.

The only construction heuristic we use in our SDVRP tests is the sweep method, which always generates a set of tours with minimum cardinality. In fact, the initial solutions generated by the savings heuristic tend to have smaller distances than the sweep solutions, but the relatively high number of tours appears to be unfavorable during the subsequent improvement phase and takes too long to be reduced. Only for problem instance "p11_00" this strategy turns out to be inappropriate: the savings heuristic generates a solution which is already very close to the BKS; the sweep solution, however, is much worse, and the gap to the BKS often is still greater than 10% in the end. For the purpose of consistency we adhere to the sweep method nevertheless.

[10] The original data set of Archetti et al. (2008) actually comprises 42 instances, yet we omitted all instances based on instance "p10" since they are identical to the "p05" instances.



(a) Standard parametrization

(b) Improved parametrization

Figure 8.7: Convergence of solution quality for the SDVRP.

We cut down iterations to prevent running times from exploding, dividing the numbers of allowed iterations by 10 and also limiting running time to three hours per instance. Having determined an SDVRP-specific parametrization of $\delta = 0.0008$ for LNS-RRT, $p^{\text{LS}} = 0.6$ for HYBRID-ABHC, and $p^{\text{LS}} = 0.999, \delta = 0.0015$ for HYBRID-RRT we compare the aggregated results for the five heuristics in table 8.9. For LS-RRT we could not find a better δ than the standard value, and in general the gain by parameter tuning is rather small. Only for HYBRID-RRT the improvement over the standard settings is significant, which becomes evident comparing the quality convergences in figure 8.7(a) and figure 8.7(b).

The complex ejection procedure presented in section 7.3 has its greatest impact on running times within LS-ABHC, where it is used excessively during every iteration of the search to evaluate relocate moves. Its influence is milder within LS-RRT and, especially, LNS-RRT. All five heuristics are able to generate solutions with an overall deviation from the BKS of 1% or better. HYBRID-ABHC generates the best solutions among the configurations tested, yet requires much longer running times than HYBRID-RRT that yields very good solutions quickly. Figure 8.7(b) demonstrates the good convergence of LS-RRT and HYBRID-RRT, which indicates that our adaptation of the relocate operator is very powerful but must be used more carefully than within an ABHC heuristic. LNS-RRT alone appears to perform not as well, yet it becomes clear again that LNS moves improve LS-only approaches significantly.[11]

[11]Note that LNS-RRT is inferior considering absolute distances as displayed in figure 8.7 but performs quite well according to the deviations stated in table 8.9. This discrepancy is explained by the observation that LNS moves handle the critical instance "p11_00" better than LS moves do, avoiding extreme deviations occurring for this instance.

			LS-ABHC			LS-RRT			LNS-RRT		
			1K	5K	10K	750K	3.75M	7.5M	1K	5K	10K
Standard parametrization	*Best of five*	Avg. dev. (%)	**2.26**	**1.41**	**1.05**	1.09	0.87	0.95	1.96	0.90	0.61
		Max. dev. (%)	13.82	13.76	12.66	13.80	13.75	13.88	4.15	2.96	2.96
	Average of five	Avg. dev. (%)	-	-	-	**1.67**	**1.37**	**1.57**	2.70	1.29	1.02
		Max. dev. (%)	-	-	-	13.98	13.85	14.07	8.80	3.33	3.39
Improved parametrization	*Best of five*	Avg. dev. (%)	-	-	-	-*	-*	-*	1.80	0.76	0.44
		Max. dev. (%)	-	-	-	-*	-*	-*	3.69	2.58	2.78
	Average of five	Avg. dev. (%)	-	-	-	-*	-*	-*	**2.57**	**1.32**	**0.88**
		Max. dev. (%)	-	-	-	-*	-*	-*	10.57	4.49	3.06
		Avg. time (s)	874.64	3176.39	4705.17	469.58	2243.40	3880.63	285.29	1255.60	2140.98
			HYBRID-ABHC			HYBRID-RRT					
			1K	5K	10K	150K	750K	1.5M			
Standard parametrization	*Best of five*	Avg. dev. (%)	1.34	0.47	0.33	1.77	1.19	1.06			
		Max. dev. (%)	10.80	2.40	2.35	4.93	4.02	3.79			
	Average of five	Avg. dev. (%)	1.93	1.01	0.64	2.36	1.60	1.38			
		Max. dev. (%)	13.10	4.33	2.57	6.65	4.20	4.11			
Improved parametrization	*Best of five*	Avg. dev. (%)	1.54	0.45	0.22	1.22	0.53	0.30			
		Max. dev. (%)	13.49	2.42	2.28	13.71	3.11	2.42			
	Average of five	Avg. dev. (%)	**2.04**	**1.01**	**0.60**	**1.80**	**1.17**	**0.98**			
		Max. dev. (%)	13.85	6.26	2.52	14.13	7.77	9.73			
		Avg. time (s)	677.69	2644.24	3968.64	144.70	684.33	1349.56			

Table 8.9: Aggregated results for the SDVRP.

In a previous study (Derigs et al., 2010) we already presented results of several LS-based heuristics for the SDVRP, including an ABHC heuristic that performed best. According to a recent survey by Archetti and Speranza (2011) this method is still among the state-of-the-art methods for the SDVRP. A few other approaches have been proposed in the meanwhile, yet there is some shortage of comparable numerical results since heuristics were often tested on similarly generated but actually different sets of instances. In this evaluation we compare our HYBRID-RRT heuristic against the matheuristic of Archetti et al. (2008), which is a good SDVRP method, though beaten by our former ABHC method, and for which we are confident to dispose of exactly the same data set for testing.[12]

Table 8.10 presents a detailed comparison against the results of Archetti et al. (2008). The instance names indicate the basic instance and, behind the underscore, parameters α and γ in percent. The best solutions known so far are given in Derigs et al. (2010). Unfortunately, Archetti et al. (2008) do not state their testing environment, so that we are not able to scale our own computation times for a fair comparison. A second complicating aspect is that the only solution values given in their publication stem from an unknown number of tests of multiple configurations. Behind these reference solutions we present the average results of HYBRID-RRT with short running times first, and then the best results of five replications with longer running times. Evidently, HYBRID-RRT produces better solutions within short times, generates several new best solutions for the benchmark instances, and can be considered as a state-of-the-art method for the SDVRP consequently.

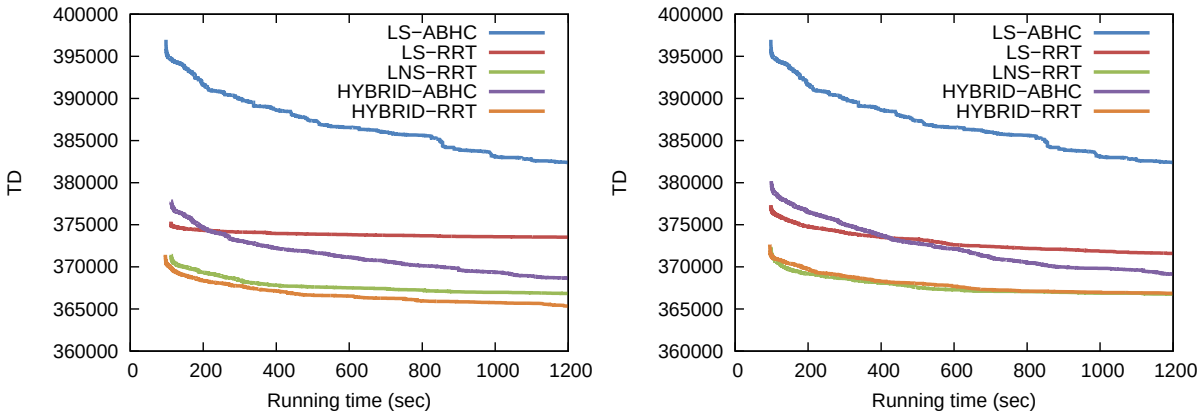
8.2.5 Periodic Vehicle Routing Problem

Our heuristics for the PVRP are evaluated on a widely-used set of 32 instances contributed by multiple authors. Most instances contain between 50 and 200 customers with different service frequencies; the largest instance in the set has 417 customers. The planning horizon usually spans over four, five, or six days. More detailed instance characteristics are presented in Hemmelmayr et al. (2009). Recent publications consider a new set of ten testing instances in addition, yet due to tour duration constraints we cannot solve these instances.

[12] We do not compare HYBRID-RRT with our previously developed ABHC heuristic for the same reason we did not consider our previously developed VRPC heuristics in section 8.2.3: since the underlying concepts are similar a comparison of results does not yield very much insight.

Instance		BKS	Archetti et al. (2008) *Best of multiple tests*			HYBRID-RRT *150K, Avg. of five*			HYBRID-RRT *1.5M, Best of five*		*Avg.*
Name	$\|N_c\|$	TD	TD	dev (%)	time (s)	TD	dev (%)	time (s)	TD	dev (%)	time (s)
p01	50	524.61	527.68	0.58	97.00	524.67	0.01	7.20	524.61	0.00	68.41
p01_1030	50	758.20	758.20	0.00	256.00	792.88	4.57	9.73	776.56	2.42	92.24
p01_1050	50	1007.51	1021.02	1.34	866.00	1030.22	2.25	15.17	1018.84	1.12	147.37
p01_1090	50	1488.58	1497.28	0.58	2939.00	1508.53	1.34	31.02	1496.45	0.53	312.81
p01_3070	50	1487.81	1502.00	0.95	1684.00	1507.16	1.30	28.92	1491.52	0.25	267.31
p01_7090	50	2160.66	2166.80	0.28	834.00	2184.72	1.11	60.85	2171.13	0.48	605.64
p02	75	823.89	853.61	3.61	52.00	842.70	2.28	14.31	831.98	0.98	139.65
p02_1030	75	1116.75	1122.91	0.55	161.00	1142.47	2.30	17.10	1124.99	0.74	166.31
p02_1050	75	1504.74	1548.54	2.91	646.00	1532.99	1.88	29.23	1510.06	0.35	300.76
p02_1090	75	2318.53	2337.81	0.83	361.00	2335.49	0.73	64.49	2319.16	0.03	682.56
p02_3070	75	2228.69	2263.12	1.54	2551.00	2262.80	1.53	61.66	2243.60	0.67	572.75
p02_7090	75	3234.64	3250.39	0.49	1872.00	3258.85	0.75	123.29	3239.12	0.14	1226.76
p03	100	826.14	840.12	1.69	51.00	850.09	2.90	15.42	827.39	0.15	144.54
p03_1030	100	1472.53	1505.46	2.24	159.00	1507.01	2.34	26.17	1483.54	0.75	247.75
p03_1050	100	2018.94	2024.58	0.28	201.00	2042.97	1.19	51.51	2010.08	-0.44	527.25
p03_1090	100	3116.61	3136.29	0.63	620.00	3129.28	0.41	119.52	3116.48	0.00	1176.72
p03_3070	100	3002.64	3055.51	1.76	1605.00	3037.77	1.17	105.20	3006.16	0.12	1093.73
p03_7090	100	4411.32	4452.56	0.93	2433.00	4427.36	0.36	237.71	4409.87	-0.03	2151.97
p04	150	1028.42	1055.08	2.59	298.00	1074.46	4.48	31.91	1050.29	2.13	307.81
p04_1030	150	2037.00	2093.28	2.76	1152.00	2068.06	1.52	65.20	2048.62	0.57	620.90
p04_1050	150	2901.62	2977.00	2.60	517.00	2902.57	0.03	120.49	2884.43	-0.59	1052.32
p04_1090	150	4581.32	4659.90	1.72	592.00	4624.09	0.93	280.14	4583.06	0.04	2544.07
p04_3070	150	4374.56	4465.47	2.08	251.00	4411.25	0.84	239.44	4367.97	-0.15	2384.57
p04_7090	150	6462.78	6462.78	0.00	2460.00	6501.58	0.60	558.45	6442.79	-0.31	4916.92
p05	199	1296.66	1338.36	3.22	297.00	1330.74	2.63	56.55	1308.87	0.94	535.65
p05_1030	199	2528.82	2582.62	2.13	567.00	2549.28	0.81	103.48	2513.76	-0.60	877.09
p05_1050	199	3548.31	3594.00	1.29	1138.00	3576.07	0.78	160.04	3536.12	-0.34	1319.62
p05_1090	199	5669.26	5710.21	0.72	806.00	5651.45	-0.31	424.39	5588.19	-1.43	4053.89
p05_3070	199	5487.55	5549.77	1.13	1702.00	5526.45	0.71	390.95	5464.72	-0.42	3527.04
p05_7090	199	8297.71	8355.45	0.70	656.00	8388.10	1.09	969.83	8284.08	-0.16	9015.23
p11	120	1042.12	1056.96	1.42	262.00	1189.32	14.13	24.56	1048.77	0.64	238.19
p11_1030	120	2907.39	3017.92	3.80	585.00	2952.67	1.56	40.33	2925.05	0.61	378.51
p11_1050	120	4261.74	4476.38	5.04	365.00	4299.97	0.90	67.39	4239.41	-0.52	653.46
p11_1090	120	6881.04	7117.24	3.43	4882.00	7014.91	1.95	169.52	6919.07	0.55	1630.55
p11_3070	120	6658.52	7126.84	7.03	7147.00	6791.03	1.99	143.70	6717.61	0.89	1399.01
p11_7090	120	10233.37	10429.75	1.92	3948.00	10402.17	1.65	344.44	10287.47	0.53	3204.89
Total		113700.96	115932.88		45013.00	115172.15		5209.32	113811.85		48584.23
Average				1.80	1250.36		1.80	144.70		0.30	1349.56
Max				7.03	7147.00		14.13	969.83		2.42	9015.23

Table 8.10: Comparison with a state-of-the-art method for the SDVRP.



(a) Standard parametrization

(b) Improved parametrization

Figure 8.8: Convergence of solution quality for the PVRP.

During initial tests we fixed a PVRP-specific parametrization of $\delta = 0.007$ for LS-RRT, $\delta = 0.008$ for LNS-RRT, $p^{\text{LS}} = 0.6$ for HYBRID-ABHC, and $p^{\text{LS}} = 0.9925, \delta = 0.01$ for HYBRID-RRT, and based on these settings table 8.11 compares the aggregated results for the five heuristics. Since running times are comparably short on the data set used we double the number of iterations in the third scenario of each heuristic. Figure 8.8 displays the convergences of total distances.[13] Our best heuristics produce solutions with deviations from the BKS between one and two percent on average. Evidently, LS-ABHC proceeds very slowly since it conducts an expensive evaluation of moves for all days in each iteration but finally applies only a move for a single day. In contrast, our adaptation of LNS performs rather well: LNS-RRT is not much worse than HYBRID-RRT regarding the deviations from the BKS, and as can be seen in figure 8.8 both heuristics are on the same level in terms of absolute tour distances. HYBRID-ABHC is a bit slow but can finally catch up with HYBRID-RRT within the running time tested and produces the best results in the end (in terms of deviation). The standard parametrization can be improved by parameter tuning only little, except for LS-RRT, which cannot compete with the best heuristics, however.

The current (published) state-of-the-art method for the PVRP is the VNS based heuristic of Hemmelmayr et al. (2009). Recently, Vidal et al. (2011) presented a technical report on a sophisticated hybrid genetic algorithm for the PVRP (and the multi-depot VRP) combining evolutionary search, local search, and population diversity management schemes along with many new best solutions. Their average results over ten runs are presented

[13] Note that the graphs of figure 8.8 start at about 100 seconds. This is due to the expensive construction of an initial solution for 417-customer instance "v-p13": for technical reasons the graphs are plotted from that point in time when solutions have been generated for all instances in the set.

			LS-ABHC			LS-RRT			LNS-RRT		
			10K	50K	200K	7.5M	37.5M	150M	10K	50K	200K
Standard parametrization	*Best of five*	Avg. dev. (%)	**4.69**	**3.19**	**2.24**	3.66	3.47	3.30	2.07	1.65	1.18
		Max. dev. (%)	15.33	13.14	10.13	9.31	9.31	7.33	5.83	5.99	5.83
	Average of five	Avg. dev. (%)	-	-	-	4.52	4.46	4.21	2.77	2.28	1.83
		Max. dev. (%)	-	-	-	10.63	10.85	10.93	6.86	6.97	5.83
Improved parametrization	*Best of five*	Avg. dev. (%)	-	-	-	2.42	1.96	1.77	1.96	1.42	1.22
		Max. dev. (%)	-	-	-	13.66	10.58	8.31	6.45	5.83	5.83
	Average of five	Avg. dev. (%)	-	-	-	**3.04**	**2.56**	**2.35**	**2.54**	**1.95**	**1.55**
		Max. dev. (%)	-	-	-	14.24	10.93	8.48	6.62	5.98	5.83
		Avg. time (s)	67.12	298.93	1112.47	77.84	358.07	1382.35	36.27	156.44	604.24
			HYBRID-ABHC			HYBRID-RRT					
			10K	50K	200K	1.5M	7.5M	30M			
Standard parametrization	*Best of five*	Avg. dev. (%)	2.48	1.67	0.82	1.62	1.10	0.99			
		Max. dev. (%)	8.96	6.32	4.44	7.11	5.83	5.83			
	Average of five	Avg. dev. (%)	3.20	2.25	1.30	2.19	1.62	1.37			
		Max. dev. (%)	9.79	7.31	4.86	8.04	5.84	5.83			
Improved parametrization	*Best of five*	Avg. dev. (%)	2.59	1.60	0.75	1.49	1.15	1.01			
		Max. dev. (%)	8.06	5.83	4.41	5.88	5.50	5.87			
	Average of five	Avg. dev. (%)	**3.26**	**2.12**	**1.21**	**1.97**	**1.46**	**1.27**			
		Max. dev. (%)	9.34	7.57	4.56	5.98	5.81	5.89			
		Avg. time (s)	57.04	254.84	960.02	60.25	273.03	1068.71			

Table 8.11: Aggregated results for the PVRP.

in table 8.12 and compared with HYBRID-RRT under short running times and with HYBRID-ABHC under long running times. The authors conduct their tests on an AMD Opteron 2.4 GHz but scale computations times to mimic a Pentium IV 3.0 GHz, so that we multiply our own running times with 2.22, which results from assuming a Dongarra (2011) factor of 242 Mflop/s and a CINT2006 base score of 9.55 for their "virtual" machine. Evidently, our heuristics cannot compete with the elaborate method of Vidal et al. (2011) that generates solutions with an average deviation from the BKS of 0.31 percent with average running times of around four minutes. HYBRID-RRT yields a deviation of 1.97 percent on average with slightly shorter running times, but even our best results during longer runs obtained with HYBRID-ABHC stagnate at 0.75 percent. Note that the heuristic of Hemmelmayr et al. (2009) produces solutions with an average deviation of 1.60 percent within 147.66 seconds on a PC with 3.2 GHz, which is a quality comparable to the results of our framework-based heuristics, so that we can state that our heuristics are still among the best methods for the PVRP.

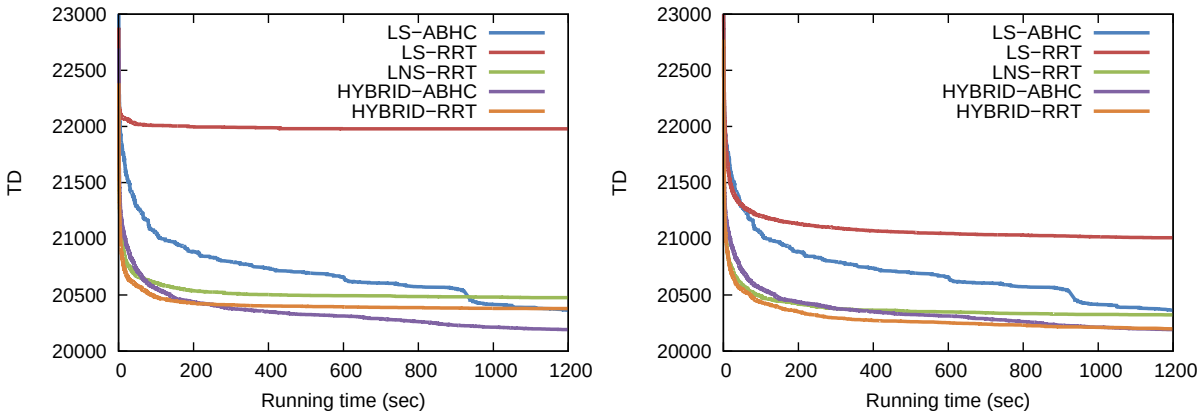
8.2.6 Truck and Trailer Routing Problem

We evaluate our heuristics for the TTRP on a set of 21 instances generated by Chao (2002). These instances are derived from seven CMT problems for the standard VRP having between 50 and 199 customers by specifying 25%, 50%, and 75% of the customers as truck customers, respectively. In table 8.14 we use instance names indicating the basic CMT problem and the truck customer percentage, separated by an underscore. Villegas et al. (2011) present the BKS for these instances; a few solutions are contributed by the authors themselves, while most of the rest were generated for the first time by the SA heuristic of Lin et al. (2009). Note that for a fair comparison it is important to consider the fleet size constraint of the TTRP; allowing to use indefinite numbers of trucks and trailers we could generate solutions with tour distances around 1.2 percent shorter than the solutions presented in the following.

After initial tuning we use a TTRP-specific parametrization of $\delta = 0.012$ for LS-RRT, $\delta = 0.011$ for LNS-RRT, and $p^{\mathrm{LS}} = 0.9925, \delta = 0.025$ for HYBRID-RRT; we could not improve the settings of HYBRID-ABHC. Table 8.13 compares the aggregated results for the five heuristics under standard parametrization and improved parametrization. The instance sizes of the data set are rather small, but nevertheless the short running times indicate that the increase of computational effort of our complex neighborhood adaptations is moderate; especially for LNS moves the effort per iteration is small. Since solution times are so short we double the number of iterations in the third scenario of each heuris-

Instance		BKS	Vidal et al. (2011) *Avg. of ten*			HYBRID-RRT *1.5M, Avg. of five*				HYBRID-ABHC *200K, Best of five*		*Avg.*
Name	$\|N_c\|$	TD	TD	dev (%)	time (s)	TD	dev (%)	time (s)	time* (s)	TD	dev (%)	time (s)
v-p01	50	524.61	524.61	0.00	13.20	524.61	0.00	11.07	24.59	524.61	0.00	99.82
v-p02	50	1322.87	1322.87	0.00	26.40	1341.97	1.44	19.76	43.89	1324.10	0.09	285.65
v-p03	50	524.61	524.61	0.00	10.80	524.61	0.00	11.58	25.74	524.61	0.00	113.39
v-p04	75	835.26	836.59	0.16	63.00	841.31	0.72	15.95	35.43	835.26	0.00	158.47
v-p05	75	2024.96	2033.72	0.43	136.20	2060.61	1.76	32.19	71.52	2039.89	0.74	457.53
v-p06	75	835.26	842.48	0.86	53.40	843.26	0.96	18.90	41.99	835.26	0.00	214.96
v-p07	100	826.14	827.02	0.11	52.80	827.75	0.19	31.58	70.16	826.14	0.00	451.86
v-p08	100	2022.47	2022.85	0.02	152.40	2066.16	2.16	54.45	120.96	2038.56	0.80	963.25
v-p09	100	826.14	826.94	0.10	60.60	828.25	0.26	35.19	78.19	826.14	0.00	513.48
v-p10	100	1593.43	1605.22	0.74	108.00	1649.03	3.49	46.76	103.88	1604.52	0.70	770.11
v-p11	139	770.89	775.84	0.64	276.00	801.16	3.93	68.20	151.52	780.14	1.20	1000.43
v-p12	163	1186.47	1195.29	0.74	320.40	1257.41	5.98	79.18	175.90	1238.75	4.41	1167.37
v-p13	417	3492.89	3599.86	3.06	2400.00	3660.95	4.81	333.36	740.61	3506.45	0.39	3801.04
v-p14	20	954.81	954.81	0.00	4.80	954.81	0.00	9.31	20.69	954.81	0.00	73.42
v-p15	38	1862.63	1862.63	0.00	10.20	1862.63	0.00	16.80	37.33	1862.63	0.00	222.04
v-p16	56	2875.24	2875.24	0.00	19.20	2875.24	0.00	26.56	59.00	2875.24	0.00	464.46
v-p17	40	1597.75	1597.75	0.00	16.20	1636.12	2.40	12.31	27.36	1611.07	0.83	138.56
v-p18	76	3131.09	3131.09	0.00	53.40	3215.44	2.69	41.89	93.06	3151.93	0.67	611.76
v-p19	112	4834.34	4834.50	0.00	135.60	4846.49	0.25	73.13	162.48	4846.49	0.25	1740.35
v-p20	184	8367.40	8367.40	0.00	240.60	8367.40	0.00	171.00	379.90	8367.40	0.00	4423.02
v-p21	60	2170.61	2170.61	0.00	54.00	2182.78	0.56	22.49	49.97	2184.33	0.63	297.58
v-p22	114	4193.95	4194.23	0.01	256.20	4320.96	3.03	67.09	149.05	4269.03	1.79	1242.44
v-p23	168	6420.71	6434.10	0.21	257.40	6801.11	5.92	150.91	335.28	6620.50	3.11	3373.05
v-p24	51	3687.46	3687.46	0.00	19.20	3734.50	1.28	18.08	40.18	3687.46	0.00	238.62
v-p25	51	3777.15	3777.15	0.00	35.40	3781.57	0.12	19.78	43.95	3777.15	0.00	285.99
v-p26	51	3795.32	3795.32	0.00	19.80	3795.95	0.02	17.70	39.32	3795.32	0.00	314.38
v-p27	102	21833.87	21885.70	0.24	211.20	22716.99	4.04	53.10	117.98	21963.83	0.60	758.55
v-p28	102	22242.51	22272.60	0.14	280.20	22671.10	1.93	54.45	120.97	22354.26	0.50	786.84
v-p29	102	22543.75	22564.05	0.09	231.60	23201.20	2.92	54.12	120.24	22593.09	0.22	780.16
v-p30	153	73875.19	74534.38	0.89	599.40	77393.14	4.76	118.94	264.24	76453.06	3.49	1643.22
v-p31	153	76001.57	76686.65	0.90	600.00	79023.84	3.98	121.49	269.90	77425.50	1.87	1740.34
v-p32	153	77598.00	78168.82	0.74	600.00	80371.31	3.57	120.68	268.12	78958.70	1.75	1588.67
Total		358549.35	360732.39		7317.60	370979.66		1928.02	4283.40	364656.25		30720.79
Average				0.31	228.68		1.97	60.25	133.86		0.75	960.02
Max				3.06	2400.00		5.98	333.36	740.61		4.41	4423.02

Table 8.12: Comparison with a state-of-the-art method for the PVRP.



(a) Standard parametrization

(b) Improved parametrization

Figure 8.9: Convergence of solution quality for the TTRP.

tic. Figure 8.9 shows the convergences of solution quality over time.

Evidently, all heuristics except for LS-RRT can produce solutions within a gap of one percent or less above the BKS. LNS-RRT performs remarkably well and again, the hybrid methods are the best, most notably HYBRID-ABHC in its standard configuration. HYBRID-RRT can compete with HYBRID-ABHC under improved settings during shorter running times. In general, the influence of parameter calibration is rather moderate for all heuristics except for the inferior LS-RRT.

We compare HYBRID-ABHC with the current state-of-the-art TTRP method which is the recently published hybrid metaheuristic of Villegas et al. (2011) that combines different concepts such as the *greedy randomized adaptive search procedure* (GRASP), VNS, and *path relinking*. The authors obtain their best results using a configuration *GRASP/VNS with evolutionary path relinking*, and we state their average cost values from ten replications in table 8.14. The next group of columns lists the average results of HYBRID-ABHC with 100,000 iterations (not listed in table 8.13); evidently, HYBRID-ABHC can yield the same solution quality within slightly shorter running times on average. Note that Villegas et al. (2011) run their tests on the same CPU as we do, so that computation times can be compared directly. Finally, the best results of five replications of 200,000 iterations each turn out to be very close to the BKS and include four new best solutions. We conclude that our adapted heuristics, especially HYBRID-ABHC, are competitive concerning both solution time and quality or even slightly better than the current state-of-the-art methods for the TTRP.

			LS-ABHC			LS-RRT			LNS-RRT		
			10K	50K	200K	7.5M	37.5M	150M	10K	50K	200K
Standard parametrization	*Best of five*	Avg. dev. (%)	**4.84**	**2.63**	**1.06**	9.16	8.85	8.96	1.93	1.41	1.36
		Max. dev. (%)	12.38	12.02	3.37	21.06	21.06	21.06	4.78	3.47	3.76
	Average of five	Avg. dev. (%)	-	-	-	10.50	10.13	10.19	3.10	2.43	2.14
		Max. dev. (%)	-	-	-	24.42	22.17	21.38	6.57	5.40	4.23
Improved parametrization	*Best of five*	Avg. dev. (%)	-	-	-	5.05	4.28	4.25	1.74	1.10	0.79
		Max. dev. (%)	-	-	-	14.40	14.06	13.47	5.55	4.33	2.99
	Average of five	Avg. dev. (%)	-	-	-	**6.19**	**5.46**	**5.26**	**2.85**	**1.84**	**1.33**
		Max. dev. (%)	-	-	-	15.93	15.93	15.07	7.93	4.82	3.55
		Avg. time (s)	101.03	497.29	2045.80	85.30	415.67	1622.29	32.94	161.43	643.40
			HYBRID-ABHC			HYBRID-RRT					
			10K	50K	200K	1.5M	7.5M	30M			
Standard parametrization	*Best of five*	Avg. dev. (%)	1.72	0.71	0.14	1.68	1.14	1.12			
		Max. dev. (%)	5.01	2.81	1.13	3.73	2.94	2.81			
	Average of five	Avg. dev. (%)	**2.76**	**1.20**	**0.49**	2.41	1.84	1.68			
		Max. dev. (%)	8.15	3.59	1.67	4.85	4.24	3.57			
Improved parametrization	*Best of five*	Avg. dev. (%)	-*	-*	-*	1.23	0.83	0.52			
		Max. dev. (%)	-*	-*	-*	4.12	3.48	1.66			
	Average of five	Avg. dev. (%)	-*	-*	-*	**1.96**	**1.16**	**0.77**			
		Max. dev. (%)	-*	-*	-*	4.53	3.96	3.14			
		Avg. time (s)	68.06	332.64	1339.21	75.89	368.67	1453.84			

Table 8.13: Aggregated results for the TTRP.

Instance		BKS	Villegas et al. (2011) *Avg. of ten*			HYBRID-ABHC *100K, Avg. of five*			HYBRID-ABHC *200K, Best of five*		*Avg.*
Name	$\|N_c\|$	TD	TD	dev (%)	time (s)	TD	dev (%)	time (s)	TD	dev (%)	time (s)
CMT1_25	50	564.68	565.99	0.23	70.20	564.81	0.02	103.87	564.68	0.00	206.21
CMT1_50	50	611.53	614.23	0.44	77.40	612.44	0.15	114.79	611.53	0.00	229.68
CMT1_75	50	618.04	618.04	0.00	63.00	618.04	0.00	114.42	618.04	0.00	227.64
CMT2_25	75	798.53	803.51	0.62	161.40	804.49	0.75	199.76	799.34	0.10	399.70
CMT2_50	75	839.62	841.63	0.24	169.20	839.62	0.00	218.12	839.62	0.00	433.33
CMT2_75	75	930.64	961.47	3.31	173.40	945.36	1.58	224.12	940.69	1.08	444.44
CMT3_25	100	830.48	830.48	0.00	363.00	830.48	0.00	460.24	830.48	0.00	930.20
CMT3_50	100	872.56	876.21	0.42	417.60	876.45	0.45	490.70	871.98	-0.07	999.06
CMT3_75	100	912.02	918.45	0.71	502.80	926.14	1.55	473.17	922.36	1.13	944.93
CMT4_25	150	1039.07	1050.11	1.06	1130.40	1046.58	0.72	851.15	1040.46	0.13	1833.16
CMT4_50	150	1093.37	1100.95	0.69	1272.00	1102.07	0.80	1042.95	1093.89	0.05	2171.29
CMT4_75	150	1152.32	1158.88	0.57	1546.80	1170.32	1.56	1082.40	1154.82	0.22	2171.94
CMT5_25	199	1287.18	1305.83	1.45	2636.40	1319.07	2.48	1681.80	1289.20	0.16	3274.64
CMT5_50	199	1339.36	1354.04	1.10	2734.20	1352.50	0.98	1707.98	1341.25	0.14	3427.71
CMT5_75	199	1420.72	1437.52	1.18	3589.80	1452.25	2.22	1834.77	1423.55	0.20	3728.35
CMT11_25	120	1002.49	1003.07	0.06	883.80	1026.59	2.40	582.68	1001.48	-0.10	1158.57
CMT11_50	120	1026.20	1042.61	1.60	790.20	1031.81	0.55	629.41	1026.20	0.00	1252.41
CMT11_75	120	1098.15	1118.63	1.86	761.40	1098.46	0.03	667.54	1098.15	0.00	1323.42
CMT12_25	100	813.30	819.81	0.80	312.60	819.43	0.75	417.63	812.69	-0.08	836.99
CMT12_50	100	848.93	860.12	1.32	337.20	848.20	-0.09	493.06	848.12	-0.10	997.35
CMT12_75	100	909.06	909.06	0.00	378.60	909.25	0.02	567.25	909.06	0.00	1132.30
Total		20008.25	20190.64		18371.40	20194.36		13957.79	20037.59		28123.31
Average				0.84	874.83		0.81	664.66		0.14	1339.21
Max				3.31	3589.80		2.48	1834.77		1.13	3728.35

Table 8.14: Comparison with a state-of-the-art method for the TTRP.

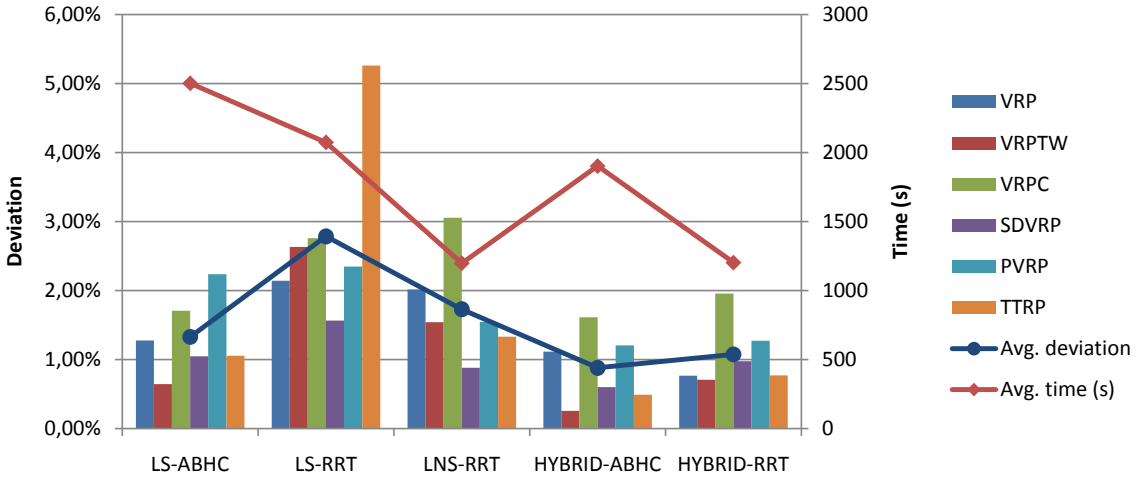


Figure 8.10: Overall solution qualities and computation times.

8.3 Summary

Completing our numerical evaluations we finally compare the five heuristics with respect to their overall solution quality, and we also address aspects of stability and robustness, which are important quality measures in practical application as well. First, figure 8.10 gives an overview of the average deviations from the BKS of each VRP obtained during the longest running time scenarios under problem-specific parametrization, respectively, and in addition it plots the average deviations and running times over all problems in order to rank solution quality in relation to computational effort. This presentation confirms our findings from the previous sections: heuristics combining LS and LNS perform significantly better than methods applying only one of the two neighborhood paradigms. Overall, we consider HYBRID-RRT as the winner of this comparison with respect to both quality and time. Slightly better solutions are obtained with HYBRID-ABHC, yet requiring significantly more computational effort. The observation that the ABHC control enables a thorough exploration of the solution space when long running times are available applies to LS-ABHC as well. LNS-RRT is a rather fast heuristic, and the solutions generated are not too bad either; in contrast, the use of LS-RRT is not recommendable.

The robustness of a heuristic refers to multiple criteria and levels; most notably, we can assess whether a heuristic yields stable and reliable results over

- multiple problem variants,
- multiple parameter settings,

- multiple instances of a problem variant, and
- multiple runs on the same problem instance.

Figure 8.10 visualizes that for a given heuristic the variability of solution quality over different problem variants is moderate in the sense that there are only few cases in which a heuristic performs exceptionally good or bad on one problem compared to its average performance. As a consequence, we expect (in particular) our hybrid methods to perform also well when adapted to new problems.

With respect to parameter settings we have already experienced in section 8.1 during initial tuning that the deviation parameter δ of RRT heuristics must be tuned carefully. Section 8.2 confirmed the impression that LS-RRT and also HYBRID-RRT are not very robust, yielding exceptionally bad results for certain settings. The incorporation of LNS moves generally adds stability to the search, yet the exact proportion between LS and LNS moves is not particularly important.

In the same context figure 8.11 analyzes the impact of parameter tuning on the success of the individual heuristics (except for the parameter-free LS-ABHC). For each heuristic and each VRP we present the decrease of the average deviation from the respective BKS obtained by tuning in percentage points. Generally, one must be careful concluding from unsuccessful tuning on one particular problem to the fact that a heuristic is exceptionally stable – the standard parametrization of the framework could "accidentally" be very suitable for that problem. Yet, the overall picture confirms that LS-RRT is very sensitive to parameter settings, while HYBRID-ABHC is a very robust heuristic that hardly requires tuning of its main parameter. Figure 8.11 also displays the increase or decrease of running time due to tuning in percent: most notably, the average running time of HYBRID-RRT is shorter by around 15 percent compared to standard settings, which reflects the modified balances of LS and LNS moves. The impact of tuning appears rather moderate for this heuristic at first, but assuming that solution quality still improves with longer running times the impact is much higher, in fact, as we have already experienced in the previous section.

Developing VRP heuristics one sometimes makes the experience that a heuristic produces good results overall but fails (or performs significantly worse) on a small, particular set of problem instances. Instances may have very special (and rare) characteristics that hinder the solution process in a certain way, and such issues often cannot be identified easily. A similar problem is related to bad or unsuitable initial solutions: a construction heuristic

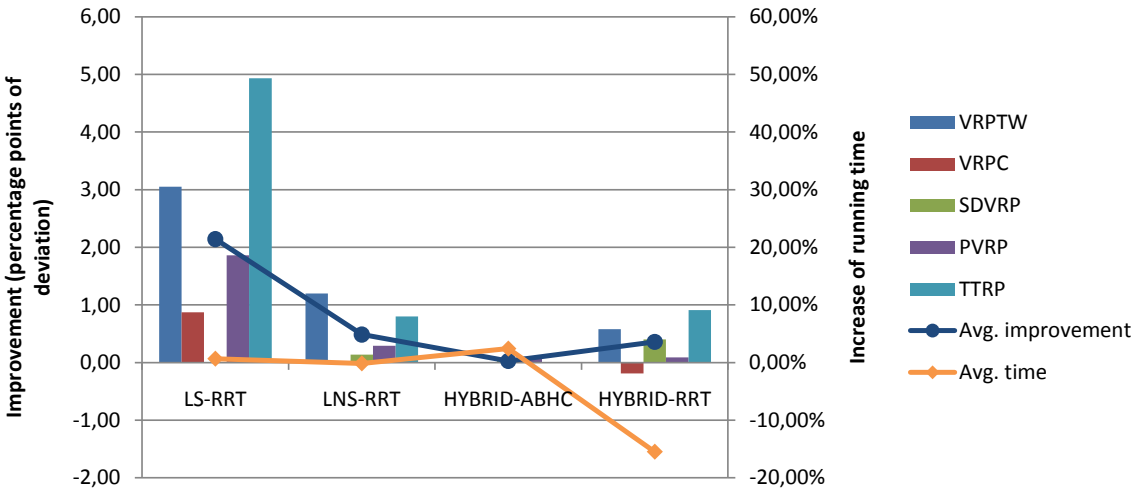


Figure 8.11: Impact of parameter tuning.

predetermines a solution structure, e.g. a specific clustering, that an improvement heuristic may have difficulties to overcome. Here, a heuristic is robust if its outcome does not depend too much on the initial solution of the search. In this context we take a look at the worst results, in terms of deviation from the BKS, that our five heuristics produced among the instances of the six VRP data sets, respectively: for every heuristic and every problem figure 8.12 compares the average deviations (in strong colors) and the maximum deviations[14] (in pale colors). In addition, we plot the averages over all VRP variants, respectively. Clearly, LS-RRT has the most difficulties producing reliable results for all instances of a data set, yielding deviations of more than ten percent for three of the six VRPs in the worst case. HYBRID-ABHC, in contrast, is very robust, yielding deviations of less than five percent in the worst case. The observation that the ABHC concept and the power of LNS moves contribute to robustness is underlined by the fact that HYBRID-RRT does *not* turn out to be significantly more robust than LS-ABHC and LNS-RRT are.

Finally, we examine how the solution quality of a heuristic varies among multiple runs on a single problem instance. In practical applications there is often time for a single solution run only, so that major fluctuations in quality are not desirable. For this purpose we measure the variability of results for a given problem instance by the coefficient of variation (CV) of tour distances obtained over all replications conducted. Figure 8.13 presents the average CVs over all instances tested for a problem class. We consider the interpretation of distance variability as a little problematical in the case of the VRPTW,

[14]The deviations are calculated for the average cost over five replications of each instance, as defined in section 8.2.1.

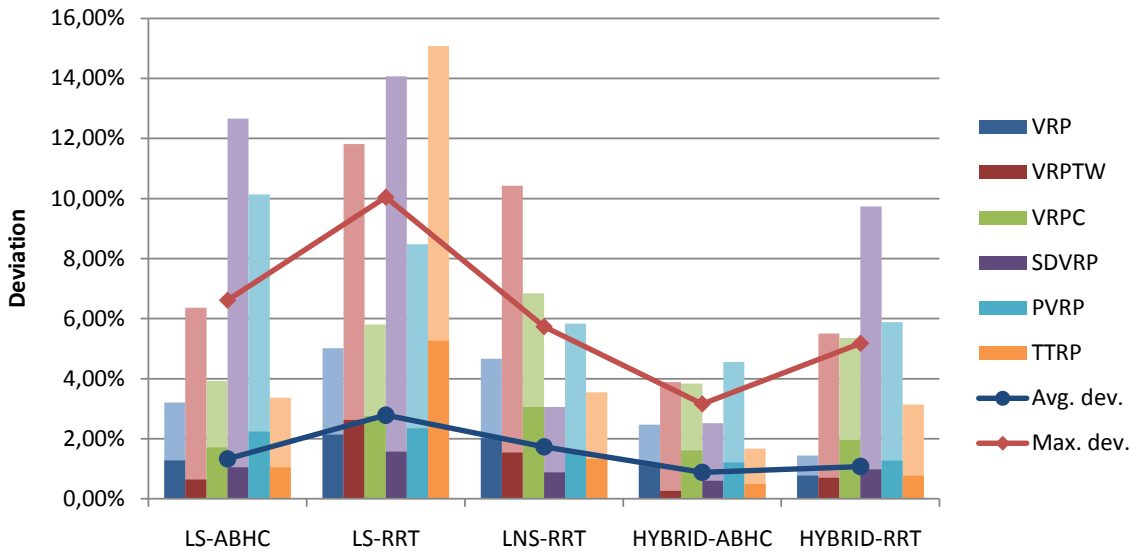


Figure 8.12: Average and maximum deviations within data sets.

since the distance is influenced by the success of the (randomized) vehicle minimization procedure, as discussed in section 8.2.2. Hence, figure 8.13 shows the average CVs over all problems denoted by "Avg. CV", but also over all problems except for the VRPTW denoted by "Avg. CV*". Ignoring the VRPTW, variability does not appear to differ dramatically among the heuristics in general, yet LNS moves seem to have a positive influence on stability, and the hybrid methods are most stable again.

As a general recommendation on the procedure of developing a solution method based on our framework to be embedded into a specific DSS we propose the following steps:

1. For most problems the quickest way to create a runnable heuristic for DSS prototyping is adapting the algorithmic components required for LNS-RRT first, i.e. a construction heuristic and the two basic, yet powerful operations for removal and reinsertion. Producing acceptable results with moderate running times and without too much parameter tuning the LNS-RRT heuristic is a good foundation for the first experiments with a new VRP variant.

2. Then, to obtain high-quality results the implementation of LS moves is mandatory. Often, code written for LNS operations can be reused to a certain degree. Since HYBRID-ABHC hardly requires parameter tuning it is the hybrid method of choice in this development phase.

3. Finally, when all adaptations are completed and when also real-world data is available it is worth a try to switch to HYBRID-RRT and tune parameters on a set of

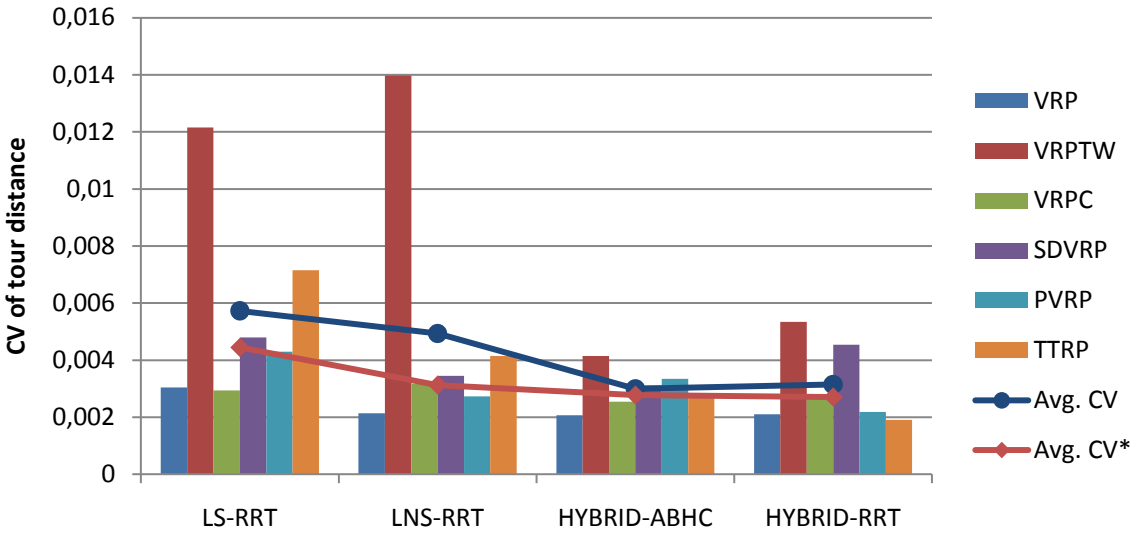


Figure 8.13: Variability of tour distance over multiple runs.

representative problem instances to obtain a high-quality heuristic tailored to the customer's typical planning scenario.

Chapter 9

Conclusion

Small profit margins and cost pressure in the road transportation business require an efficient planning of operations on the part of trucking companies. Complicating aspects such as legal restrictions, specific customer requirements, or the necessity of responding quickly to unforeseen events disqualify manual planning entirely, and even off-the-shelf route planning software can be inappropriate for setting up both feasible and cost-effective transportation plans in a certain business context. Vendors of sophisticated, customized decision support systems that provide support on different planning levels are confronted with the task of tailoring embedded vehicle routing algorithms to the specific demands and business rules of their clients. For such purposes we proposed the concept and design of a flexible metaheuristic framework to facilitate and accelerate the development of solution methods for a wide range of rich vehicle routing problems arising in diverse industries. After a brief summary of this thesis we discuss our work critically in section 9.1 and point to a few potential starting points for further research in section 9.2.

Part I provided an introduction into the field of VRPs and metaheuristic solution methods. Our framework is based on solution methods for the standard VRP and consequently, we started chapter 2 with defining this basic variant of the VRP class. Then, we presented an overview of real-world extensions to the VRP – the class of rich VRPs. Partly, this overview served as an illustration of the relevance of flexible and adaptable solution approaches, but it also served as the foundation of our requirements analysis in the second part of the thesis. In chapter 3 we introduced common metaheuristics from the literature and explained those local search and large neighborhood search techniques in detail upon which our framework is based.

Part II was dedicated to the framework itself. Reviewing existing publications on VRP frameworks in chapter 4 we first documented that there is a lack of flexible approaches that explicitly address the solution of wide ranges of VRPs. We formulated requirements for a good framework, including the four attributes flexibility, simplicity, accuracy, and speed, as well as characteristics of rich VRPs that it should be able to take account of. Aiming to meet these requirements we presented the architecture and main concepts of our framework in chapter 5, mainly on a pseudocode level: we defined the five implemented standard VRP heuristics and the mechanisms available for adapting these heuristics to rich VRPs. The central aspect is the separation of standard VRP operations and specific VRP operations by user-definable checking and post-processing functions. Chapter 6 first covered some technical aspects referring to our choice of the Microsoft .NET framework as the foundation of our implementation; afterwards we summarized our framework design and especially presented its most relevant classes.

Part III finally dealt with the evaluation of our framework, mainly regarding flexibility, accuracy, and speed, by developing customized metaheuristics for five rich VRPs. In chapter 7 we explained in detail which modifications of data structures and neighborhoods are required to adapt the built-in heuristics to the VRPTW, the VRPC, the SDVRP, the PVRP, and the TTRP. We have chosen this broad range of VRPs with diverse characteristics to demonstrate that our framework can be used flexibly for many scenarios. Chapter 8 presented computational results for the heuristics developed; first, we determined a standard parametrization for the framework, then we tuned every heuristic individually for each problem. Comparing our own results with state-of-the-art solvers of the literature we verified the good solution quality that can be obtained from heuristics based on our framework approach. Finally, we summarized our results and made some additional remarks on aspects of robustness and stability.

9.1 Critical Review

In section 4.2 we interpreted the four attributes flexibility, simplicity, accuracy, and speed defined by Cordeau et al. (2002) for VRP heuristics as important attributes of a good VRP framework, putting a stronger emphasis on flexibility and simplicity. We believe that the framework developed for this thesis complies with all four attributes.

Flexibility Our framework provides a good basis for the development of metaheuristics to solve VRPs with diverse characteristics, as we demonstrated in chapter 7. Giving the

user freedom to extend the solution representation and to incorporate custom operations into several stages of the solution process, for instance implementing checks for solution-, tour-, or sequence-level constraints, the framework is flexible enough to account for a broad range of real-world requirements. Yet, it is conceivable that for some problems not covered in this thesis a few minor changes or extensions to the framework could be required.

Simplicity Simplicity is a very vague criterion and difficult to measure. The framework has been designed upon well-known neighborhood search techniques which are among the most common methods to solve VRPs. This is helpful in the first place since a lot of knowledge and experience is available in the literature on how to apply these techniques to rich VRPs. The actual effort required for a specific adaptation greatly depends on the problem under consideration. Since the intellectual complexity can hardly be quantified we resort to the amount of code as an auxiliary measure. These are the numbers of lines of code of our VRP-specific implementations:

VRP variant	lines of code
Standard VRP (framework)	*2,671*
VRPTW	332
VRPC	481
SDVRP	714
PVRP	714
TTRP	1,556

Evidently, adaptation is rather easy when only new constraints have to be checked and certain moves have to be forbidden, which is the case for the VRPTW and also the VRPC. With complicating aspects such as new decision variables or a new tour structure the effort can increase dramatically. Note that we put a lot of effort into our TTRP heuristics, which is evident from the huge amount of lines of code; yet, this does not necessarily mean that results of similar quality could not have been obtained with less effort. Also consider that for all problems we applied both local search and large neighborhood search. However, to obtain a runnable heuristic producing acceptable solution quality it is sufficient to adapt a single base heuristic as a start. For instance, to create an LS-based heuristic for the PVRP it is almost sufficient to add a neighborhood for switching visit combinations, as described in section 7.4. In contrast, time window checks required for the VRPTW can be incorporated very quickly into the insertion mechanisms of the LNS approach.

Accuracy and speed In chapter 8 our customized heuristics turned out to be very competitive against state-of-the-art methods presented in the literature. On some prob-

lems, most notably the SDVRP and the TTRP, our hybrid methods performed better than the reference methods with respect to both solution quality and computational effort, and we could generate some new best solutions for widely-used problem instances as well. On other problems they were beaten by highly sophisticated and quick algorithms but still produced good results. We observed that combining LS and LNS techniques easily improves solution quality and stability of the search. In particular, HYBRID-ABHC is a heuristic generating high-quality solutions that is very robust and that can be applied out-of-the-box without much parameter tuning. HYBRID-RRT, if calibrated well, can often produce even better results with less computational effort.

Even though the use of the framework worked out well for the five rich VRPs considered, there is no guarantee that high-quality heuristics can be derived for every other VRP. The adaptation to a complicated problem can demand a lot of experience and creativity from the user – as seen in the case of the TTRP the effort required can be very high. Still, we conclude that our framework provides a feasible and pragmatic approach to deal with real-world requirements in solving VRPs. Small disadvantages concerning solution quality and computation times in comparison with – sometimes very elaborate and specialized – reference methods are compensated by the increase of flexibility and productivity in practical development processes. In general, we consider a solid heuristic tailored to the customer's planning scenario and observing all relevant problem-specific aspects more valuable than a heuristic performing better on an inappropriate problem formulation by a few percent.

9.2 Future Research

Evidently, work on the subject can be continued almost indefinitely by evaluating even more VRP variants. Since the present framework cannot be used for road transportation scenarios corresponding to pickup and delivery problems or arc routing problems it could be interesting as well to translate its concepts into new frameworks targeting these two problem classes.

Despite the good results obtained for the five selected rich VRPs the performance of the built-in heuristics for the standard VRP indicates that there is still some room for improvement. One starting point for additional research could be the coordination of LS and LNS within the hybrid methods. Those methods are successful despite mixing the two neighborhood concepts in a random, naive way only. Combining LS and LNS more intelligently it could be possible to take advantage of their respective strengths –

intensification and diversification – even better. A source of inspiration might be variable neighborhood search, which is a technique that combines different types of moves in a purposeful way.

It could be a promising approach to enrich the neighborhood search methods of the framework with evolutionary or population-based concepts which have attracted a great deal of attention over the last few years. The effort of converting conventional VRP heuristics into memetic algorithms appears to be moderate since the additional operations related with the management of populations can probably be realized rather generically on the framework-level. Finally, complexity-reducing techniques for the solution of large-scale VRPs are useful to improve running times. The granularity principle introduced by Toth and Vigo (2003), for instance, which dynamically hides costly arcs from the network and thereby controls the neighborhood size, could easily be incorporated into the framework and used generically for all VRPs.

Appendix A

Pseudocodes

In section 5.4.2 we explained how the predefined LS neighborhoods can be adapted by user-definable checking and post-processing functions, and we presented a pseudocode showing the exploration of the relocate neighborhood that covers all relevant aspects of the remaining neighborhoods as well. For the sake of completeness we present the respective codes of the

- exchange neighborhood in algorithm A.1, the
- 2-opt* neighborhood in algorithm A.2, the
- relocate^I neighborhood in algorithm A.3, and the
- 2-opt neighborhood in algorithm A.4.

Algorithm A.1 User-definable functions in exchange

```
1: function EXCHANGE(solution S)
2:     for all t_1, t_2 ∈ tours(S), t_1 ≠ t_2 do
3:         CHECKEXCHANGEBETWEENTOURS(t_1,t_2)
4:         for all c_1 ∈ customers(t_1), c_2 ∈ customers(t_2) do
5:             check capacities
6:             determine Δ, A^e, and A^l
7:             (Δ̄, Ā^e, Ā^l, Φ^s) := CHECKEXCHANGE(c_1,c_2)
8:             check acceptability of (Δ̄, Ā^e, Ā^l)
9:             store move (c_1, c_2, Δ̄, Φ^s)
10:        end for
11:    end for
12: end function
```

Algorithm A.2 User-definable functions in 2-opt*

```
1: function TwoOptStar(solution S)
2:     for all t_1, t_2 ∈ tours(S), t_1 ≠ t_2 do
3:         CheckTwoOptStarBetweenTours(t_1,t_2)
4:         for all c_1 ∈ locations(t_1), c_2 ∈ locations(t_2) do
5:             check capacities
6:             determine Δ, A^e, and A^l
7:             (Δ̄, Ā^e, Ā^l, Φ^s) := CheckTwoOptStar(c_1,c_2)
8:             check acceptability of (Δ̄, Ā^e, Ā^l)
9:             store move (c_1, c_2, Δ̄, Φ^s)
10:        end for
11:    end for
12: end function
```

Algorithm A.3 User-definable functions in relocate^I

```
1: function Relocate^I(solution S)
2:     for all t ∈ tours(S), c ∈ customers(t) do
3:         for all i ∈ locations(t), i ≠ c, i ≠ predecessor of c do
4:             determine Δ, A^e, and A^l
5:             (Δ̄, Ā^e, Ā^l, Φ^s) := CheckRelocateI(c,i)
6:             check acceptability of (Δ̄, Ā^e, Ā^l)
7:             store move (c, i, Δ̄, Φ^s)
8:         end for
9:     end for
10: end function
```

Algorithm A.4 User-definable functions in 2-opt

```
1: function TwoOpt(solution S)
2:     for all t ∈ tours(S) do
3:         for all c_1, c_2 ∈ customers(t), c_1 ≠ c_2 do
4:             determine Δ, A^e, and A^l
5:             (Δ̄, Ā^e, Ā^l, Φ^s) := CheckTwoOpt(c_1,c_2)
6:             check acceptability of (Δ̄, Ā^e, Ā^l)
7:             store move (c_1, c_2, Δ̄, Φ^s)
8:         end for
9:     end for
10: end function
```

References

Archetti, C., Savelsbergh, M., and Speranza, M. G. (2006). Worst-case analysis for split delivery vehicle routing problems. *Transportation Science*, 40(2):226–234.

Archetti, C. and Speranza, M. G. (2011). Vehicle routing problems with split deliveries. *International Transactions in Operational Research*. doi: 10.1111/j.1475-3995.2011.00811.x.

Archetti, C., Speranza, M. G., and Savelsbergh, M. (2008). An optimization-based heuristic for the split delivery vehicle routing problem. *Transportation Science*, 42(1):22–31.

Baldacci, R., Battarra, M., and Vigo, D. (2008). Routing a heterogeneous fleet of vehicles. In Golden, B., Raghavan, S., and Wasil, E., editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 3–27. Springer, USA.

Bartodziej, P., Derigs, U., Malcherek, D., and Vogel, U. (2009). Models and algorithms for solving combined vehicle and crew scheduling problems with rest constraints: an application to road feeder service planning in air cargo transportation. *OR Spectrum*, 31(2):405–429.

Beltrami, E. J. and Bodin, L. D. (1974). Networks and vehicle routing for municipal waste collection. *Networks*, 4(1):65–94.

Bent, R. and Van Hentenryck, P. (2004). A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530.

Bräysy, O. and Gendreau, M. (2005a). Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118.

Bräysy, O. and Gendreau, M. (2005b). Vehicle routing problem with time windows, Part II: Metaheuristics. *Transportation Science*, 39(1):119–139.

Brown, G. G. and Graves, G. W. (1981). Real-time dispatch of petroleum tank trucks. *Management Science*, 27(1):19–32.

Campbell, A. M. and Savelsbergh, M. (2004). Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378.

Caramia, M. and Guerriero, F. (2010). A milk collection problem with incompatibility constraints. *Interfaces*, 40(2):130–143.

Carić, T., Galić, A., Fosin, J., Gold, H., and Reinholz, A. (2008). A modelling and optimization framework for real-world vehicle routing problems. In Carić, T. and Gold, H., editors, *Vehicle Routing Problem*, pages 15–34. InTech, Vienna.

Chao, I. M. (2002). A tabu search method for the truck and trailer routing problem. *Computers and Operations Research*, 29(1):33–51.

Chen, S., Golden, B., and Wasil, E. (2007). The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results. *Networks*, 49(4):318–329.

Christofides, N., Mingozzi, A., and Toth, P. (1979). The vehicle routing problem. In Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester.

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

Cordeau, J.-F., Gendreau, M., and Laporte, G. (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–120.

Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y., and Semet, F. (2002). A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, 53(5):512–522.

Cordeau, J.-F., Laporte, G., and Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52(8):928–936.

Cornillier, F., Boctor, F. F., Laporte, G., and Renaud, J. (2008). A heuristic for the multi-period petrol station replenishment problem. *European Journal of Operational Research*, 191(2):295–305.

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

Derigs, U. and Döhmer, T. (2008). Indirect search for the vehicle routing problem with pickup and delivery and time windows. *OR Spectrum*, 30:149–165.

Derigs, U., Gottlieb, J., Kalkoff, J., Piesche, M., Rothlauf, F., and Vogel, U. (2011a). Vehicle routing with compartments: applications, modelling and heuristics. *OR Spectrum*, 33:885–914.

Derigs, U. and Kaiser, R. (2007). Applying the attribute based hill climber heuristic to the vehicle routing problem. *European Journal of Operational Research*, 177(2):719–732.

Derigs, U., Kurowsky, R., and Vogel, U. (2011b). Solving a real-world vehicle routing problem with multiple use of tractors and trailers and EU-regulations for drivers arising in air cargo road feeder services. *European Journal of Operational Research*, 213(1):309–319.

Derigs, U., Li, B., and Vogel, U. (2010). Local search-based metaheuristics for the split delivery vehicle routing problem. *Journal of the Operational Research Society*, 61:1356–1364.

Derigs, U. and Reuter, K. (2009). A simple and efficient tabu search heuristic for solving the open vehicle routing problem. *Journal of the Operational Research Society*, 60(12):1658–1669.

Derigs, U. and Vogel, U. (2009). A computational study on neighborhood search heuristics for the open vehicle routing problem with time windows. In *Proceedings of the 8th Metaheuristic International Conference (MIC 2009)*, pages 109–122.

Doerner, K. F., Fuellerer, G., Hartl, R. F., Gronalt, M., and Iori, M. (2007). Metaheuristics for the vehicle routing problem with loading constraints. *Networks*, 49(4):294–307.

Dongarra, J. (2011). Performance of various computers using standard linear equations software. CS-89-85. Technical report, University of Tennessee, Knoxville TN, USA.

Drexl, M. (2007). On some generalized routing problems. PhD thesis, RWTH Aachen, Germany.

Drexl, M. (2011). Marktstudie Tourenplanungssoftware. *OR News*, 41:6–9.

Dror, M. and Trudeau, P. (1989). Savings by split delivery routing. *Transportation Science*, 23(2):141–145.

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

Dueck, G. and Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1):161–175.

DVZ (2011a). Präzisionswerkzeug für die Filialbelieferung. *DVZ Deutsche Logistik-Zeitung*, BLEB (08.02.2011).

DVZ (2011b). Umstellung läuft auf vollen Touren. *DVZ Deutsche Logistik-Zeitung*, 81 (07.07.2011).

El Fallahi, A., Prins, C., and Wolfler Calvo, R. (2008). A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem. *Computers and Operations Research*, 35(5):1725–1741.

European Commission, editor (2010). *EU energy and transport in figures 2010*. Publications Office of the European Union, Luxembourg.

Eurostat (2011a). Eurostat statistics database. `http://ec.europa.eu/eurostat` (last accessed July 14, 2011).

Eurostat (2011b). Freight transport statistics. `http://epp.eurostat.ec.europa.eu/statistics_explained/index.php/Freight_transport_statistics` (last accessed July 14, 2011).

Galić, A., Carić, T., and Gold, H. (2006). Mars – A programming language for solving vehicle routing problems. In *International Conference on City Logistics.; Recent advances in city logistics*, pages 47–58.

Gendreau, M., Hertz, A., and Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094.

Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290.

Gendreau, M., Iori, M., Laporte, G., and Martello, S. (2006). A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350.

Gerdessen, J. C. (1996). Vehicle routing problem with trailers. *European Journal of Operational Research*, 93(1):135–147.

Gillett, B. E. and Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349.

Glover, F. (1989). Tabu search – Part I. *ORSA Journal on Computing*, 1(3):190–206.

Glover, F. (1990). Tabu search – Part II. *ORSA Journal on Computing*, 2(1):4–32.

Glover, F. (1991). Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Working paper, College of Business and Administration, University of Colorado, Boulder, CO.

Golden, B., Assad, A., Levy, L., and Gheysens, F. (1984). The fleet size and mix vehicle routing problem. *Computers and Operations Research*, 11(1):49–66.

Golden, B., Raghavan, S., and Wasil, E., editors (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, Berlin.

Golden, B., Wasil, E. A., Kelly, J. P., and Chao, I.-M. (1998). Metaheuristics in vehicle routing. In Crainic, T. G. and Laporte, G., editors, *Fleet Management and Logistics*, pages 33–56. Kluwer Academic Publishers, Boston.

Groër, C., Golden, B., and Wasil, E. (2010). A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2:79–101.

Gulczynski, D., Golden, B., and Wasil, E. (2010). The split delivery vehicle routing problem with minimum delivery amounts. *Transportation Research Part E: Logistics and Transportation Review*, 46(5):612–626.

Hemmelmayr, V. C., Doerner, K. F., and Hartl, R. F. (2009). A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, 195(3):791–802.

Hillier, F. S. and Lieberman, G. J. (1980). *Introduction to Operations Research*. Holden-Day, Inc., third edition.

Irnich, S. (2008). A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. *Journal on Computing*, 20(2):270–287.

Jetlund, A. S. and Karimi, I. A. (2004). Improving the logistics of multi-compartment chemical tankers. *Computers and Chemical Engineering*, 28:1267–1283.

Johnson, D. S., Aragon, C. R., Mcgeoch, L. A., and Schevon, C. (1989). Optimization by simulated annealing: An experimental evaluation; Part I, graph partitioning. *Operations Research*, 37(6):865–892.

Keen, P. G. W. and Scott Morton, M. S., editors (1978). *Decision Support Systems: An Organizational Perspective*. Addison-Wesley, Inc., Reading, MA.

Kirkpatrick, S., Gelatt, Jr., C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Kopfer, H., Meyer, C. M., and Wagenknecht, A. (2007). Die EU-Sozialvorschriften und ihr Einfluss auf die Tourenplanung. *Logistik Management*, 9(2):32–47.

Laporte, G. (2009). Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416.

Laporte, G. and Osman, I. H. (1995). Routing problems: A bibliography. *Annals of Operations Research*, 61:227–262.

Li, F., Golden, B., and Wasil, E. (2007). The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers and Operations Research*, 34(10):2918–2930.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516.

Lin, S. W., Yu, V. F., and Chou, S. Y. (2009). Solving the truck and trailer routing problem based on a simulated annealing heuristic. *Computers and Operations Research*, 36(5):1683–1692.

Malandraki, C. and Daskin, M. S. (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, 26(3):185–200.

Maniezzo, V., Stützle, T., and Voß, S., editors (2010). *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*. Springer, USA.

Mester, D. and Bräysy, O. (2005). Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers and Operations Research*, 32(6):1593–1614.

Mester, D. and Bräysy, O. (2007). Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers and Operations Research*, 34(10):2964–2975.

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

Mullaseril, P. A., Dror, M., and Leung, J. (1997). Split-delivery routeing heuristics in livestock feed distribution. *Journal of the Operational Research Society*, 48(2):107–116.

Muyldermans, L. and Pang, G. (2010). On the benefits of co-collection: Experiments with a multi-compartment vehicle routing algorithm. *European Journal of Operational Research*, 206(1):93–103.

Nag, B., Golden, B., and Assad, A. (1988). Vehicle routing with site dependencies. In Golden, B. and Assad, A., editors, *Vehicle Routing: Methods and Studies*, pages 149–159. North-Holland, Amsterdam.

Or, I. (1976). Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(4):421–451.

Partyka, J. and Hall, R. (2010). On the road to connectivity. *OR/MS Today*, 37(1):42–49.

Pisinger, D. and Ropke, S. (2005). A general heuristic for vehicle routing problems. Technical Report 05/01, DIKU, University of Copenhagen.

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

Potvin, J.-Y. and Rousseau, J.-M. (1995). An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446.

Psaraftis, H. N. (1995). Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61:143–164.

Ropke, S. (2011). Heuristic solutions for the CVRP. `http://www.diku.dk/~sropke/` (last accessed January 19, 2011).

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

Ropke, S. and Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775.

Sariklis, D. and Powell, S. (2000). A heuristic method for the open vehicle routing problem. *Journal of the Operational Research Society*, 51(5):564–573.

Savelsbergh, M. (1992). The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4:146–154.

Scheuerer, S. (2006). A tabu search heuristic for the truck and trailer routing problem. *Computers and Operations Research*, 33:894–909.

Schrage, L. (1981). Formulation and structure of more complex/realistic routing and scheduling problems. *Networks*, 11(2):229–232.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., and Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159:139–171.

Scott Morton, M. S., editor (1971). *Management Decision Systems: Computer Based Support for Decision Making*. Division of Research, Graduate School of Business Administration, Harvard University, Boston.

Shaw, P. (1998a). A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, APES group.

Shaw, P. (1998b). Using constraint programming and local search methods to solve vehicle routing problems. Proceedings CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming).

Sierksma, G. and Tijssen, G. A. (1998). Routing helicopters for crew exchanges on offshore locations. *Annals of Operations Research*, 76:261–286.

SINTEF (2011). Transportation optimization portal of SINTEF applied mathematics. `http://www.sintef.no/projectweb/top/` (last accessed August 12, 2011).

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–265.

Sprague, R. H. and Carlson, E. D., editors (1982). *Building Effective Decision Support Systems*. Prentice-Hall, Englewood Cliffs, NJ.

Taillard, E. D., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

Taillard, E. D., Gambardella, L. M., Gendreau, M., and Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1–16.

Taillard, E. D., Laporte, G., and Gendreau, M. (1996). Vehicle routeing with multiple use of vehicles. *Journal of the Operational Research Society*, 47(8):1065–1070.

Toth, P. and Vigo, D. (2002). *The Vehicle Routing Problem*. SIAM.

Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *Journal on Computing*, 15:333–346.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. (2011). A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. Technical Report, CIRRELT-2011-05, University of Montreal.

Villegas, J. G., Prins, C., Prodhon, C., Medaglia, A. L., and Velasco, N. (2011). A GRASP with evolutionary path relinking for the truck and trailer routing problem. *Computers and Operations Research*, 38(9):1319–1334.

Voss, S., Osman, I. H., and Roucairol, C., editors (1999). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Norwell, MA, USA.

Voss, S. and Woodruff, D., editors (2002). *Optimization Software Class Libraries*. Kluwer Academic Publishers, Boston, MA, USA.

Voudouris, C. and Tsang, E. (1999). Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499.

Whittley, I. M. and Smith, G. D. (2004). The attribute based hill climber. *Journal of Mathematical Modelling and Algorithms*, 3(2):167–178.

Zachariadis, E. E., Tarantilis, C. D., and Kiranoudis, C. T. (2009). A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 195(3):729–743.