A Unified Modeling and Solution Framework for Vehicle Routing and Local Search-based Metaheuristics

Stefan Irnich [a,*]

[a] *Deutsche Post Endowed Chair of Optimization of Distribution Networks, RWTH Aachen University, Templergraben 64, D-52056 Aachen, Germany.*

Abstract

This paper presents a new unified modeling and *heuristic* solution framework for vehicle-routing problems (VRPs) with complex side constraints. The work is focused on strong modeling capabilities as well as efficient solution procedures to be used in all kinds of metaheuristics. From the modeling point of view, the framework covers a variety of standard VRP types with classical constraints, such as capacity, distance, route length, time window, pairing and precedence constraints, but also non-standard "rich" VRPs. From the methodological point of view, local search (LS) is the key *solver engine* to be used in heuristic solution procedures. First and foremost, the framework introduces two generic techniques for the efficient exploration of edge and node exchange neighborhoods. On the one hand, new preprocessing methods allow $\mathcal{O}(n^k)$ neighborhoods to be searched in time complexity $\mathcal{O}(n^k)$, i.e., without an additional effort for feasibility testing. On the other hand, Irnich et al. (2006) have introduced *sequential search* as a generic method for accelerating LS in the average case. The computational tests on different types of VRPs indicate that the proposed methods are highly efficient. Sequential search procedures outperform the currently most efficient search methods—which are based on lexicographic search—on large-scale instances and for nearly all types of neighborhoods by factors of between 10 and 1 000.

Key words: local search, vehicle routing, rich VRPs, resource-constrained paths

1. Introduction

The diversity of models and solution approaches in vehicle routing is enormous (see, e.g., Golden and Assad, 1988; Toth and Vigo, 2002a; Laporte, 1992, 1997). This can be estimated, for instance, by the fact that, in 2006 alone, a few hundred scientific papers were published. Many of these publications meet the challenge of extending known models and methods to cope with new or extended types of vehicle-routing problems (VRPs). Under the name *rich models*, researchers summarize "non-idealized models that represent

* Corresponding author.
Email address: sirnich@or.rwth-aachen.de (Stefan Irnich).

the application at hand in an adequate way by including all important optimization criteria, constraints, and preferences" (Hasle et al., 2006). Plenty of contributions to "rich VRPs" exist in the form of specialized algorithms that incorporate different types of extensions into existing problems (see, e.g, Janssens et al., 2006, and several articles in this special issue). However, many publications are mainly case studies, and it is not clear whether their results are transferable to other cases. What is missing are *unifying* modeling and solution approaches that are general (=generic) and flexible enough to be used in a broad range of applications.

In many other publications, the focus is on enhancing the efficiency of existing methods or on devising alternative approaches that solve larger instances, compute solutions faster, or provide solutions of better quality. In this context, much progress has been made with regard to the design and analysis of metaheuristics, i.e., problem-independent top-level general strategies which guide other heuristics to search for high-quality feasible solutions (Ribeiro and Hansen, 2002; Resende and de Sousa, 2004). The principles of well-performing metaheuristics are now much better understood and metaheuristic implementations become reusable using software libraries (Voß and Woodruff, 2002). However, what is missing are powerful lower-level VRP algorithms that are *efficient* and, at the same time, *general*.

Research on unifying approaches for VRPs has been undertaken in different directions: Formal schemes like those of Desrochers et al. (1990) are helpful to structure and classify different types of VRPs. Integrated models, as presented by Desrosiers et al. (1995) and Desaulniers et al. (1998), provide comprehensive mixed-integer programming formulations. They can be used to devise powerful decomposition approaches, such as column generation and Lagrangean relaxation integrated into branch-and-bound schemes (Desaulniers et al., 2005). These methods are primarily intended to be used as exact solution procedures, even if they can be redesigned into approximative algorithms (Desaulniers et al., 2002).

In practice, VRPs can almost never be solved with exact methods, since instances are too large and response times of decision support systems have to be short. Thus, heuristics and metaheuristics have to be applied. Since the majority of metaheuristics in vehicle routing use local-search (LS) components, the efficiency and effectiveness of LS is crucial.

One way to cope with non-standard side constraints and options in VRPs is to use LS in combination with constraint programming, as suggested by Shaw (1998); Kilby et al. (2000). Constraint programming-based methods appear attractive, since new side constraints can easily be added to existing solvers by stating additional rules (typically formulated in a high-level constraint programming language). The problem of identifying feasible improving solutions is solved through a general-purpose search engine. A known drawback of constraint programming-based (VRP) solution methods is however that the additional flexibility in modeling is bought by the expense of loosing efficiency, in particular, compared to traditional LS methods. It is worth mentioning that the *large neighborhood search* (LNS) principle, which has been used in the context of constraint programming, is very successful in finding least-cost solutions. However, LNS neighborhoods can also be searched directly (Schrimpf et al., 2000; Røpke and Pisinger, 2006).

Research on efficient LS methods for VRPs and traditional (k-edge exchange) neighborhoods has been undertaken by Kindervater and Savelsbergh (1997). It seems that these

techniques are *not* widely used, probably, since they seem to be intricate. In addition, they were not explicitly presented in a way that allows a direct adaptation to different LS operators and to new types of side constraints (cf. Shaw, 1998, p. 6).

This paper presents a new unified modeling and *heuristic* solution framework for VRPs with complex side constraints. The work is focused on strong modeling capabilities and, first and foremost, on efficient solution procedures. The contribution is threefold: First, the aim of the framework is to help model different real-world VRPs in a generic way, so that a broad class of standard problem types and also rich VRPs can be handled. The modeling capabilities cover all standard types of VRPs, such as the capacitated and distance-constrained VRP (CVRP, DCVRP), the VRP with multiple depots (MDVRP), time windows (VRPTW), simultaneous delivery and pickup (VRPSDP), backhauling (VRPB), pickup-and-delivery problems (PDP), the periodic VRP (PVRP), fleet mix problems (FMP), VRPs with site dependencies, vehicle and request (in)compatibilities, multiple-start option, limited waiting times and times on duty as well as mixtures and extensions of these (Section 4 provides a more detailed overview of types of VRPs that can and cannot be modeled and solved with the framework). The framework is mainly based on the giant-tour representation (Christofides and Eilon, 1969) and the concept of resource-constrained paths (Desaulniers et al., 1998; Irnich and Desaulniers, 2005). It provides a flexible and generic but well-defined representation of feasible and infeasible route plans.

Second, the framework is intended to support efficient solution procedures that are based on LS. The importance of LS lies in the fact that it is *the* key component for finding improving solutions within nearly all metaheuristics for VRPs. Because of its generic representation, the unified framework helps to separate the modeling phase of a specific problem at hand from the development of efficient solution methods that use LS as a major building block. The key idea of any LS-based procedure is to iteratively build a neighbor solution first and check its feasibility and gain afterwards. If implemented in a straightforward way, this feasibility check causes an extra effort bounded by the length of a longest tour. This length is in general only bounded by $\mathcal{O}(n)$ for instances of size n, where n is the number of nodes in the problem. Techniques that avoid the additional factor in the worst-case for cost computations and feasibility checks are already known, but they are intrinsically tied to the lexicographic tree search paradigm (see (Kindervater and Savelsbergh, 1997) and Section 3.3.1). Here, we present *new* techniques for searching neighborhoods of size $\mathcal{O}(n^k)$ in $\mathcal{O}(n^k)$ time. We give sufficient conditions on the update of resources that guarantee $\mathcal{O}(1)$ feasibility tests. The new techniques are more generic and compatible with any kind of neighborhood exploration strategy and, thus, enable *accelerated* search methods. Examples of neighborhoods to which the methods apply are the k-opt and k-opt* neighborhoods, the relocation and Or-opt neighborhoods, different node and string swap/exchange neighborhoods, and others (recent surveys on VRP neighborhoods and search techniques are (Bräysy and Gendreau, 2005a; Funke et al., 2005a)).

Third, the goal of all efficient LS procedures is to find a best or first improving neighbor solution as fast as possible, i.e., not only from a worst-case but from an average-case point of view. An analysis of the structure of the classical exchange procedures in the routing context yields that any neighbor solution of a (giant) tour can be generated by removing ℓ edges and replacing them by ℓ others (even if it is a node exchange procedure).

The choice of these edges is typically made by taking $k \leq \ell$ independent decisions. Hence, the associated local search procedure can be considered a tree search method where the search tree has depth k. The two main criteria for a reduction of the search space, i.e., for terminating the search or "pruning the search tree", are *cost* and *feasibility* considerations. It has been discussed in (Funke et al., 2005a; Irnich et al., 2006) that one can distinguish between two efficient approaches. *Sequential search* is based on the idea of cost-based reductions, i.e., one tries to prove at an early stage $i < k$ that no improvement can be found which includes the nodes or edges of the stages $1, \ldots, i$. *Lexicographic search* is driven by feasibility reductions, i.e., one tries to prove at an early stage $i < k$ that no feasible exchange exists which includes the nodes or edges of the stages $1, \ldots, i$. This paper presents concepts for applying sequential search procedures to the generic modeling framework in order to further reduce the effort of evaluating a neighborhood of size $\mathcal{O}(n^k)$. The goal is to perform less than $\mathcal{O}(n^k)$ operations in the average case. The acceleration methods can be applied in the context of best improvement as well as first improvement pivoting strategies. Computational results indicate the superiority of sequential search-based approaches for a variety of VRPs with side constraints over straightforward and also lexicographic implementations (Kindervater and Savelsbergh, 1997). Note that lexicographic search approaches already ensure the $\mathcal{O}(n^k)$ worst-case time bound for neighborhoods of size $\mathcal{O}(n^k)$.

Finally, we would like to stress that the paper does not present a specific metaheuristic. The presented research is a contribution to the foundations of efficient search techniques. These efficient search techniques can be seen as basic building blocks that can easily be integrated into different metaheuristics (see Section 6).

The paper is structured as follows: Section 2 presents the unified framework from a modeling point of view, introducing concepts for representing VRP solutions generically. Section 3 points out the major tasks that have to be performed in an efficient LS procedure. These tasks include efficient cost computations and feasibility testings as well as setting up well-suited search strategies that match with these computational tasks. Section 4 presents real-world constraints fitting into the framework and also discusses limitations of the approach. The computational tests of Section 5 show the effectiveness of the new solution framework. Final conclusions are given in Section 6.

2. Modeling Framework

The proposed unified modeling and solution framework for vehicle routing and LS-based metaheuristics can be seen as a counterpart to the framework of Desaulniers et al. (1998). Both frameworks follow the idea that resource-constrained paths capture which routes or schedules are feasible. While the unified framework of Desaulniers et al. (1998) is intended to be used with an exact column-generation or Lagrangean-relaxation method, the framework presented here focuses on heuristic procedures based on enumerative LS algorithms. Moreover, in (Desaulniers et al., 1998) only the feasibility of individual routes and schedules is encoded in the definition of resource-feasible paths. Constraints that couple together different routes form the constraints of the master program, see (Lübbecke and Desrosiers, 2005). Here, the feasibility of individual routes as well as several types of *inter-tour constraints* is defined by resource-constrained paths. The building blocks of the representation are the *routing graph*, the *giant-tour* representation, a *compatibility relation* between route-start and route-end nodes, and the consideration of the entire

giant route as a single *resource-feasible path*. The following subsections explain the above building blocks in more detail.

2.1. Routing Graph

In order to describe neighborhoods and solution procedures formally, a concise representation of VRP solutions, i.e., route plans, is needed. This representation has to be flexible to model a wide range of rich VRPs and has to cover typical node-exchange and edge-exchange neighborhoods, but must still allow efficient algorithmic procedures to explore neighborhoods. The basis for such a representation is a directed *routing graph* $G = (V, A)$. Any solution of the rich VRP is represented by a single cycle in G, the so-called *giant tour*. For those VRPs for which transportation tasks are *uniquely* represented by nodes, solutions coincide with Hamiltonian cycles of the routing graph.

The more general case is that alternative service or delivery options exist, e.g., in (Cardeneo, 2005) goods have to be delivered to alternative delivery points. In general, a set of tasks $\mathcal{Q}$ has to be covered. Subsets $\mathcal{Q}_v$ and $\mathcal{Q}_e$ of tasks (possibly empty) are associated with each node $v \in V$ and arc $e \in A$ of the routing graph (see also Irnich and Desaulniers, 2005, p. 40) and (Irnich and Villeneuve, 2006, §7.3). Feasible VRP solutions are cycles $(v_0, e_1, v_1, e_2, v_2, \ldots, e_{p-1}, v_{p-1}, e_p, v_0)$ (not necessarily Hamiltonian) where $\bigcup_{i=1}^{p}(\mathcal{Q}_{v_{i-1}} \cup \mathcal{Q}_{e_i})$ is a partitioning or covering of the tasks $\mathcal{Q}$. In classical node-routing applications, all customers/requests require a *single* visit and, hence, different tasks are associated with the customer/request nodes. If there is a delivery option, e.g., to deliver something (=task q) to location v_1 between 8:00 and 11:00 or to deliver it to location v_2 between 10:00 and 18:00, one can model this option with a network containing nodes v_1 and v_2 (with different time windows) that have the same associated task $\mathcal{Q}_{v_1} = \mathcal{Q}_{v_2} = \{q\}$. Moreover, more that one task might be performed when visiting a particular location v, i.e., $\mathcal{Q}_v$ can contain more than one element. In all these cases, tasks are associated with nodes and there are no tasks on arcs. Conversely, in arc-routing applications, the tasks are associated with arcs.

We call any cycle *task-feasible* if it implies a partitioning or covering of the tasks. For the entire paper we assume that testing whether $(i, j) \in A$ (for any $i, j \in V$) and the determination of tasks associated with nodes and arcs is possible in $\mathcal{O}(1)$ time.

Solutions of VRP involving more than a single vehicle can be represented as a collection of routes. Hence, the node set $V = R \cup O \cup D$ of the routing graph consists of *request nodes* R and *route-start* O and *route-end nodes* D. A *route* is a path $(v_0, v_1, \ldots, v_p)$ in G, starting with a route-start node $v_0 = o \in O$, continuing with request nodes $v_1, \ldots, v_{p-1} \in R$, and ending with a route-end node $v_p = d \in D$. The interpretation of the request nodes depends on the problem at hand. In the case of the VRP, request nodes correspond to customers that have to be visited. For the PDP, a request node is either a pickup or a delivery. In more complex routing applications, a request may even consist of more than a pair of nodes.

2.2. Compatibility Relation between Route-Start and Route-End Nodes

The aim of route-start and route-end nodes is to introduce vehicle and depot characteristics into the problem. First and foremost, these nodes represent spatial points where vehicles start and end their trips. In order to ensure that route-start and route-end nodes

are compatible, we define a relation $\sim$ on $O \times D$. Again, the compatibility of pairs (o, d) of route-start and route-end nodes depends on the problem at hand: For single-depot problems with a homogeneous fleet, all $o \in O$ and $d \in D$ are compatible, since all nodes represent the same physical location independent of the vehicle. In multi-depot problems, the sets O and D are partitioned according to the n_D depots or garages, e.g., $O = O^1 \cup \cdots \cup O^{n_D}$, $D = D^1 \cup \cdots \cup D^{n_D}$. Pairs $o \in O^k$, $d \in D^l$ are compatible if and only if $k = l$. Sets $O^k \times D^k$, consisting of a single pair, can be used to model VRPs with individual vehicles departing from and going to different locations. In general, we assume that O and D have the same cardinality, $|O| = |D|$. The easiest way to implicitly encode the compatibility relation into the routing graph is to define an arc $(o, d) \in A$ if and only if $o \sim d$ holds.

2.3. Giant Route and Giant Tour

A solution to a VRP is called a *route plan*. A route plan can be written as $x = (p^1, p^2, \ldots, p^H)$ with an H-tuple of *disjoint* routes in G. Note that this definition implies that every route-start and route-end node occurs in exactly one route. We will denote the (maximum) number of nodes in a route plan by $n = |V|$.

The *giant route* is the path $(p^1, p^2, \ldots, p^H)$ in which each route-end node d^i is connected to the next route-start node o^{i+1} (for $i = 1, 2, \ldots, H-1$). Similarly, the *giant tour* is the cycle in which, additionally, d^H is connected to o^1. In the following, $P(p^1, p^2, \ldots, p^H)$ denotes the giant route and $C(p^1, p^2, \ldots, p^H)$ the giant tour. The giant-tour representation of a route plan is a generalization of the MTSP representation of the VRP (Christofides and Eilon, 1969) to more general VRPs. It has the advantage of allowing single and multiple route problems to be handled in a very similar way. Figure 1 depicts such a representation for the case of four routes, departing from two depots.

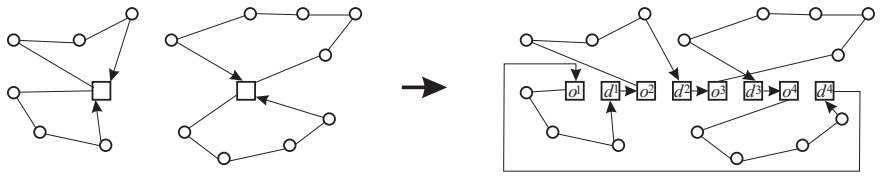

Fig. 1. Giant–Tour Representation

2.4. Resource-Constrained Paths

Resource-Constrained Paths (RCPs) and associated shortest-path problems have been very successfully used in the context of column generation methods, not only applicable to VRP but also to vehicle- and crew-*scheduling* problems, see (Desaulniers et al., 1998; Irnich and Desaulniers, 2005). The success of RCPs is based on the fact that the resource concept constitutes a very flexible tool for modeling complex cost structures for routes and schedules as well as a wide variety of rules that define their feasibility. In the context of VRPs, column generation and branch-and-price-and-cut give rise to exact solution procedures that are restricted to small and medium-sized instances of up to about 100 nodes, see e.g. (Fukasawa et al., 2004; Desaulniers et al., 2006; Jepsen et al., 2006). Here, we propose to transfer the concept of RCPs from exact to heuristic solution methods. The goal is to provide *LS components* for metaheuristics, which are flexible and at the same time powerful, so that they can be applied to large-scale rich VRP instances in order to produce high quality solutions.

Resource-constrained paths (RCP) are defined over a so-called routing (di)graph $G = (V, A)$. For the sake of convenience, we assume that G is simple, so that a path can be written as $P = (v_0, v_1, \ldots, v_p)$ with the understanding that $(v_{\ell-1}, v_\ell) \in A$ holds for all $\ell \in \{1, \ldots, p\}$. Resource constraints can be formulated by means of *(minimal) resource consumptions* and *resource intervals*, e.g., the travel times t_{ij} along arcs $(i, j) \in A$ and time windows $[a_i, b_i]$ at nodes $i \in V$ for the time resource. Let R be the number of resources (such as time, load, cost etc.). A vector $T = (T^1, \ldots, T^R)^\top \in \mathbb{R}^R$ is called a *resource vector* and its components *resource variables*. For two resource vectors a and b the interval $[a, b]$ is defined as the set $\{T \in \mathbb{R}^R : a \leq T \leq b\}$ (componentwise).

Resource intervals, also called *resource windows*, are associated with nodes $i \in V$ and are denoted by $[a_i, b_i]$ with $a_i, b_i \in \mathbb{R}^R$, $a_i \leq b_i$. (In the following, a_i^r refers to a resource vector of node i and its component for the resource r.) The changes in the resource consumptions associated with an arc $(i, j) \in A$ are given by a vector $f_{ij} = (f_{ij}^r)_{r=1}^R$ of so-called *resource extension functions* (REFs). An REF for resource r, i.e., $f_{ij}^r : \mathbb{R}^R \to \mathbb{R}$, depends on a resource vector $T_i \in \mathbb{R}^R$. The vector T_i corresponds to the resource consumption accumulated along a path from a given start node s to a node i, i.e., up to the tail node i of arc (i, j). Hence, the result $f_{ij}(T_i) \in \mathbb{R}^R$ can be interpreted as a resource consumption accumulated along the path $(s, \ldots, i, j)$. *Classical* REFs are of the form $f_{ij}^r(T_i) = \max\{a_j^r, T_i^r + t_{ij}^r\}$, where t_{ij}^r are constants associated with the arc (i, j) and a_j^r the lower bound of the resources r at node j. Classical REFs are separable by resources, i.e., no interdependencies exists between different resources. More general definitions of REFs provide powerful instruments for modeling practically relevant constraints over resources that are interdependent (see Irnich and Desaulniers (2005), Irnich (2006), and Section 4). A path $P = (v_0, v_1, \ldots, v_p)$ is *resource-feasible* if resource vectors $T_i \in [a_{v_i}, b_{v_i}]$ exist for all positions $i = 0, 1, \ldots, p$ such that $f_{v_i, v_{i+1}}(T_i) \leq T_{i+1}$ holds for all $i = 0, \ldots, p-1$. We denote by $\mathcal{F}$ the set of all resource-feasible paths.

Concluding, a route plan $(p^1, p^2, \ldots, p^H)$ is feasible if and only if all of the following four conditions hold: (1) $p^1, p^2, \ldots, p^H$ are node-disjoint routes, (2) $C(p^1, p^2, \ldots, p^H)$ is a task-feasible cycle in the routing graph G, (3) all route-start and route-end nodes of routes $p^i = (o^i, \ldots, d^i)$ are compatible, i.e., $o^i \sim d^i$ for all $i \in \{1, \ldots, H\}$, and (4) $P(p^1, p^2, \ldots, p^H)$ is a resource-feasible path. The novelty in this definition is that the entire giant route $P(p^1, p^2, \ldots, p^H)$ is considered as *one* RCP. This implies that particular REFs are needed to connect consecutive routes in the giant tour. Whenever a route-end node $d^k \in D$ is connected to a route-start $o^{k+1} \in O$, all intra-tour resources r have to be reset. This fits in nicely with the definition of classical REFs, since a reset function is given by the REF $f_{d^k, o^{k+1}}^r(T) = \max\{a_{o^{k+1}}^r, T^r - M\}$ (with an appropriate large number M). Note that inter-tour resources r (such as cost) should not be reset but kept, i.e., $f_{d^k, o^{k+1}}^r(T) = \max\{-M, T^r\} = T^r$.

3. Efficient Local Search

Local search is the most frequently used heuristic technique for solving combinatorial optimization problems. It provides the basis for modern metaheuristics, such as Tabu Search, GRASP, and variable neighborhood search (VNS), see (Hoos and Stützle, 2005). Most of the effort spent within an *enumerative* LS algorithm is used for scanning the neighborhood (for a classification of LS algorithms the reader is referred to (Funke et al., 2005a)). It is, therefore, desirable to use efficient algorithms within LS to speed up the

procedure that performs this scan. In this section, we first clarify the relationship between neighborhoods, moves, the order in which the search tree is explored, and—in detail—algorithms that compute costs and test the feasibility of neighbor solutions.

3.1. Local Search, Neighborhoods, and Moves

An instance (X, c) of a *combinatorial optimization problem* can be stated as $\min_{x \in X} c(x)$, where X is the set of feasible solutions and c the cost function. The heart of an LS procedure is the definition of a *neighborhood* $\mathcal{N}$, which is a mapping $\mathcal{N} : X \rightarrow 2^X$. Each element $x' \in \mathcal{N}(x)$ is called *neighbor of* x. Neighbors x' with cost $c(x') < c(x)$ are *improving neighbors*. LS starts with an initial feasible solution $x^0 \in X$. In each iteration t it replaces the current solution x^t by an improving neighbor $x^{t+1} \in \mathcal{N}(x^t)$, if such an improving neighbor exists. The LS procedure terminates with a *local optimum*, i.e., a solution x^t for which the neighborhood $\mathcal{N}(x^t)$ contains no improving solution.

Algorithm 1 Generic Local Search

1: **Input:** A feasible solution $x^0 \in X$.
2: LET $t = 0$.
3: REPEAT
4: SEARCH for an improving neighbor x' in the neighborhood $\mathcal{N}(x^t)$ of the current solution x^t.
5: IF there exists an improving neighbor solution $x' \in \mathcal{N}(x^t)$ THEN
6: LET $x^{t+1} = x'$ and $t = t + 1$.
7: UNTIL no more improvements can be found.
8: **Output:** A local optimum x^t.

For further details of local search, we refer the reader to the books by Rayward-Smith et al. (1996), Aarts and Lenstra (1997), and Hoos and Stützle (2005). The naming of specific VRP moves and neighborhoods used in the following is also taken from the survey (Funke et al., 2005a).

Note that there are several options for choosing improving neighbor solutions in Step 4. If the search method is enumerative (i.e., neighbor solutions $x' \in \mathcal{N}(x^t)$ and their costs $c(x')$ are evaluated one by one), taking *the first improving solution* or taking a *best improving solution* are two extreme strategies known as *first improvement* and *best improvement*. Another well-known strategy, referred to as *d-best improvement*, terminates the search when d improving neighbor solutions have been found and returns a best of them. From the worst-case point of view, all search strategies are equivalent, since showing that x^t is a local optimal solution requires the entire neighborhood $\mathcal{N}(x^t)$ to be scanned. However, from an average case point of view, these strategies might significantly differ in their efficiency (we expect from best improvement that it will perform less iterations with larger steps that take longer compared to first improvement). It is, in general, not clear which strategy works better, but the problem, the neighborhoods, and the characteristics of the instances can have an impact. Note that all of these pivoting strategies may determine different paths through the search space and end up in different local optima.

Typically, neighborhoods and neighbor solutions are neither constructed by the function $\mathcal{N} : X \rightarrow 2^X$ nor given by subsets $\mathcal{N}(x) \subset X$. Instead, they are defined implicitly by a set of *moves* M. A move $m \in M$ transforms a solution into a neighbor solution. Some of the moves $m \in M$ might transform a feasible solution x into an object $m(x)$, which has a structure similar to a feasible solution, but does not necessarily satisfy all constraints that define feasible solutions. In the following, we will refer to such an object as a *solution*. Examples in the case of VRPs are the removal of a customer node and its insertion

into another position or the swapping of two customers between two tours. These moves might violate a constraint. Let $Z \supseteq X$ be the set of all *solutions*. In general, we denote by M the set of moves, where a move $m \in M$ maps from Z to Z, i.e., $m : Z \to Z$. For a given $x \in Z$, the *extended neighborhood* $\hat{\mathcal{N}}$ contains all neighbors of x, either feasible or infeasible, i.e., $\hat{\mathcal{N}}(x) \supseteq \mathcal{N}(x)$. Every move $m \in M$ with $m(x) \in X$ is called a *feasible move* w.r.t. x. Concluding, finding a feasible move consists of two parts: the manipulation of a current solution and the test of feasibility.

3.2. Major Tasks in a Local Search Procedure for Vehicle Routing

The focus of this paper is on the efficient implementation of Step 4 of Algorithm 1. The major tasks that have to be performed are the implicit or explicit construction of neighbor solutions $x' \in \hat{\mathcal{N}}(x)$, for each of them the computation of the cost $c(x')$ or *gain* $g(x') = c(x) - c(x')$ compared to the current solution x, and the test of whether the newly constructed neighbor is feasible or not (separating candidates $x' \in X$ from those in $Z \setminus X$).

The problem of checking the feasibility of a neighbor solution is best explained by an example: A swap move chooses two nodes w_i and w_j of the giant route and exchanges them. Hence, the four arcs $(w_{i-1}, w_i), (w_i, w_{i+1}), (w_{j-1}, w_j), (w_j, w_{j+1})$ are deleted and the four arcs $(w_{i-1}, w_j), (w_j, w_{i+1}), (w_{j-1}, w_i), (w_i, w_{j+1})$ are added to the current solution x, see Figure 2. The extended swap neighborhood $\hat{\mathcal{N}}(x)$ of giant route x consists of all other gi-

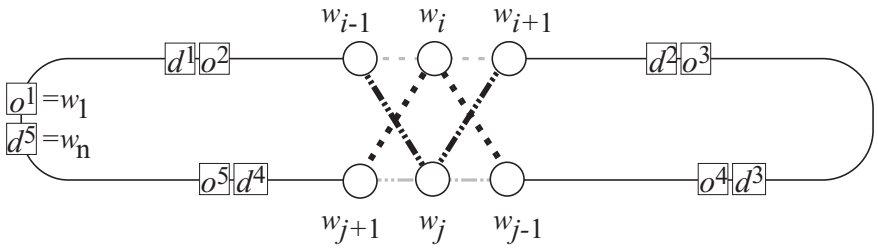

Fig. 2. Principle of a Swap Move, Giant Tour with 5 Routes

ant routes that can be generated by choosing different nodes w_i and w_j, so that the swap neighborhood is of size $\mathcal{O}(n^2)$. A newly constructed neighbor solution $x' \in \hat{\mathcal{N}}(x)$ can be rejected if it is non-improving or infeasible. Moreover, we see that a neighbor solution x' is uniquely determined after making $k = 2$ independent decisions (the decisions about the two nodes/positions to swap).

In general, all enumerative search procedures for $\mathcal{O}(n^k)$ neighborhoods work on a search tree with (at least) k-levels. They differ in two aspects:

(1) In the *order*, in which objects, i.e., nodes or arcs defining the move, are determined (nodes w_i and w_j for the swap move). Enumeration rules for nodes can consider nodes by increasing (decreasing) index, position in the giant tour, or ordered by an auxiliary attribute (e.g., lower or upper limit of an associated resource interval). Alternative enumeration rules choose nodes one after another—sequentially—such that distance, cost, or any other resource consumption of an associated arc is increasing. Lexicographic and sequential search approaches described in (Irnich et al., 2006) differ exactly with respect to these orderings. Different orderings allow tailored

(2) *Criteria for pruning the search tree.* If some branch of the search tree does not contain any feasible or improving neighbor solution, it can be pruned. It means that we do not have to build and evaluate the corresponding solutions x' but can take a shortcut. This is the

key idea for accelerating enumerative search approaches for $\mathcal{O}(n^k)$-sized neighborhoods to be searched in less that n^k operations. In contrast to heuristic techniques like those used in (Toth and Vigo, 2003), we can be sure to find a best (improving) neighbor solution.

Note that gain-based criteria try to show that there is no improving (or less strictly, no acceptable) solution relative to the quality of the current solution x and, possibly, relative to another improving neighbor solution $x'' \in \mathcal{N}(x)$ already computed. Computing the gain of a move is trivial as long as it can be expressed as the difference of the costs of all arcs changed replacing x by x', e.g., $g(x, x') = c_{w_{i-1},w_i} + c_{w_i,w_{i+1}} + c_{w_{j-1},w_j} + c_{w_j,w_{j+1}} - c_{w_{i-1},w_j} - c_{w_j,w_{i+1}} - c_{w_{j-1},w_i} - c_{w_i,w_{j+1}}$ for the swapping of w_i and w_j. However, this is not the case if cost depends on other resources, such as load-dependent transport tariffs, wages for drivers depending on the time on duty etc. Then, the preprocessing and search techniques presented in Section 3.3 still allow constant time cost computations provided that REFs are generalizable to segments. Sequential search techniques, however, are not directly applicable then (cf. Section 3.4).

Feasibility-based arguments try to identify branches of the search tree that do not contain any feasible solution at all. Both types of arguments need tailored search strategies in the sense that the sequence in which decisions are taken must allow the argument that all remaining solutions of the branch under consideration are either more costly or "less" feasible. It is, therefore, hardly possible to directly mix both approaches. Considering feasibility of a swap, note first that one or two routes are affected, depending on whether w_i and w_j are in the same route or different routes. Testing a constructed route in a straightforward way means looping over the nodes of the route in order to compute minimum resource consumptions which are then checked against upper bounds. This is at least possible if all REFs f_{ij} are non-decreasing, see (Irnich, 2006). The loop over the nodes of a single route causes an effort of $\mathcal{O}(n)$ if the length of a tour is not limited by a fixed number, independent of n. Even if there is a maximum length of a tour, the presence of inter-tour constraints can require that resource consumptions have to be propagated along the entire giant route.

3.3. Feasibility Checks and Cost Computations in Constant Time

If cost is one of the resources (this is no restriction, but the standard case in Irnich and Desaulniers (2005)), feasibility checking and cost computation can be seen as identical algorithmic procedures. Computing the cost of a giant tour $C(p^1, \ldots, p^H)$ is equivalent with finding a least cost resource vector at the destination node of $P(p^1, \ldots, p^H)$. Improving solutions w.r.t. x are exactly those giant routes that respect an upper bound $c(x) - \varepsilon$ for the cost resource (with $\varepsilon > 0$ small). In the following (if not stated otherwise), we speak of "constant time feasibility tests" for both cost computations and for checking the remaining resource variables.

Before we introduce our new approach, an alternative method proposed by Kindervater and Savelsbergh (1997) is explained along with its capabilities and limitations.

3.3.1. Global Variables Approach of Kindervater & Savelsbergh

According to Kindervater and Savelsbergh (1997), "the basic idea is to use a specific search strategy in combination with a set of global variables such that testing the feasibility of a single exchange and maintaining the set [of] global variables requires no more than

constant time". The specific search strategy they use is *lexicographic search*. Note that the traditional node and edge exchange procedures are characterized by the fact that a given tour (or two or several affected tours) are split into paths (from now on called *segments*). These segments are permuted, some may be inverted, and finally concatenated together again to form a new tour. Lexicographic search is characterized by the fact that, in the *innermost loop* of the search algorithm, from one iteration to the next, an inner segment grows by exactly one node. In this way, global variables for a segment $(w_i, w_{i+1}, \ldots, w_{j-1}, w_j)$ are computed by either concatenating $(w_i, w_{i+1}, \ldots, w_{j-1})$ with (w_{j-1}, w_j) or (w_i, w_{i+1}) with $(w_{i+1}, \ldots, w_{j-1}, w_j)$. Contrary, in an initialization phase and in outer loops of the search algorithm, global variables for starting and ending segments, i.e., $(w_1, w_2, \ldots, w_{i-1})$ and $(w_{j+1}, \ldots, w_n)$, are computed and stored. Together, these global variables of the segments allow constant time feasibility checks. For instance, time window constraints require the computation of a total travel time, earliest departure time, and a latest arrival time. This is based on certain forward and backward computations along segments. Kindervater and Savelsbergh (1997) clarify these procedures for 2-opt and Or-opt moves in connection with time windows and precedence constraints as well as for problems with simultaneous deliveries and pickups.

Their approach is intrinsically tied to the lexicographic order in which moves are considered, because a constant time update of the global variables from one iteration to the next requires that only a fixed number of nodes (typically one node) is added to a segment. In the case of a swap move (see Figure 2), an outer loop considers nodes w_i (at position i in the giant tour) in any order, e.g., in the order in which they appear in the tour. Contrary, the inner loop must choose the second customer nodes w_j, one by one, at positions $i+2, i+3, \ldots, n-1$. The constant time computation of global variables is possible for the segments $P_2 = (w_i)$, $P_3 = (w_{i+1}, \ldots, w_{j-1}, w_{j-1})$, and $P_4 = (w_j)$, since these global variables are either computed from scratch (for segments of length 1) or from global variables of the previous segment $P_3' = (w_{i+1}, \ldots, w_{j-2})$. The initial phase has to provide global variables for all segments $P_1 = (w_1, \ldots, w_{i-1})$ for $i = 1, 2, \ldots, n-3$ and $P_5 = (w_{j+1}, \ldots, w_n)$ for $j = 3, \ldots, n-1$.

Kindervater and Savelsbergh (1997, p. 350) point out that their global variables approach, combined with lexicographic search, can be used for multiple constraints and all k-edge exchange neighborhoods. However, a unifying theory explaining which types of constraints can and which cannot be dealt with is missing. For instance, resource constraints with resources that depend on each other (such as load-dependent travel times etc.) are not considered. On the other hand, resource extension functions, as introduced by Desaulniers et al. (1998), provide a well defined, flexible, and generic formalism for the description of side constraints relevant for rich VRPs.

3.3.2. Segment REFs

The following subsection explains how REFs can be inverted and generalized to segments, so that extensions of the ideas of Kindervater and Savelsbergh can be used (1) for more general VRPs defined by non-standard REFs, (2) in the context of giant tours, i.e., when segments can also contain nodes from more than just a single tour, and (3) within different search strategies allowing more flexibility than the lexicographic search approach.

The key idea is to separate the search strategy from the computation of global variables (or any similar information, e.g., given by segment REFs). Note that all the classical moves

can be considered as k-edge exchanges, even if their intention is to exchange nodes. The swap move, for instance, is a specialized 4-opt move (except for the case where w_i and w_j are adjacent yielding a 2-opt move; cf. *legitimacy conditions*, explained in (Glover, 1996) and (Irnich et al., 2006)). Therefore, moves decompose the giant route into a small fixed number of segments. The swap move depicted in Figure 2 implies the segmentation $P_1 = (o^1, \ldots, w_{i-1})$, $P_2 = (w_i)$, $P_3 = (w_{i+1}, \ldots, w_{j-1})$, $P_4 = (w_j)$, and $P_5 = (w_{j+1}, \ldots, d^5)$. The paths $P_1, \ldots, P_5$ depend on the giant tour $(w_1, w_2, \ldots, w_n, w_1)$ currently under consideration (the *incumbent* giant tour) and the choice of the nodes w_i and w_j (or, equivalently, their positions i and j). These five segments are permuted and constitute the new giant route $P = P(P_1, P_4, P_3, P_2, P_5)$ (cf. notation introduced in Section 2.3). The move is feasible if and only if P is resource-feasible and $C(P)$ is a task-feasible cycle in which route-start and route-end nodes are compatible. Testing the last two conditions, i.e., task-feasibility and that all route-start and route-end nodes are compatible, is straightforward and possible in $\mathcal{O}(1)$. The following analysis, therefore, focuses on resource-feasibility.

Our goal is now to determine *attributes* for each of the possible segments such that one can decide in $\mathcal{O}(R)$ time whether the concatenation of two segments also forms a feasible or infeasible segment. Furthermore, we want to compute the attributes of the concatenated segment in $\mathcal{O}(R)$, so that, in summary, testing the feasibility of P can be performed in constant time $\mathcal{O}(R)$, too. Irnich (2006) provides the theoretical background for accomplishing this task. The attributes which have to be computed are the defining coefficients of the *segment REFs* as well as *inverse segment REFs* for some of the segments underlying the incumbent giant tour. For the sake of clarity, we start by pointing out the basic assumptions to hold for the rest of the paper:

(a1) All REFs have a finite representation and allow function evaluations in $\mathcal{O}(R)$ time. This is true for several types of non-decreasing REFs presented in Section 4.

(a2) All inverse REFs exist. The inverse of a non-decreasing REF $f_{ij} : \mathbb{R}^R \to [a_j, \infty)$ is a function $f_{ij}^{inv} : \mathbb{R}^R \to (-\infty, b_i]$. It has to be non-decreasing and its defining property is

$$f_{ij}(T) \leq T' \iff T \leq f_{ij}^{inv}(T') \quad \text{for all } T \in (-\infty, b_i] \text{ and all } T' \in [a_i, \infty).$$

(a3) All *inverse* REFs have a finite representation and allow function evaluations in $\mathcal{O}(R)$ time.

(a4) All REFs and inverse REFs can be generalized to segments. Segment REFs also allow function evaluations in $\mathcal{O}(R)$ time.

(a5) The concatenation of any two segments has a REF that can be computed in $\mathcal{O}(R)$ time from the REFs of the two segments.

Obviously, if the number R of resources is fixed, i.e., independent of the size n of the giant tour, all the above mentioned operations can be performed in constant time $\mathcal{O}(1)$.

We refrain from giving a formal presentation of all the details concerning REFs and required properties, derivations, and proofs concerning finite representation, inversion, generalization to segments, function evaluation and concatenation. These details can be

found in (Irnich, 2006). However, some remarks for explaining and interpreting the newly introduced segment REFs and inverse REFs seem appropriate: We consider an arbitrary path P. The segment REF $f_P : \mathbb{R}^R \to \mathbb{R}^R$ gives for each initial minimum resource consumption T at the start node the minimum resource consumption at the final node of P. Note first, that this is exactly the idea of arc REFs, i.e., for $P = (i, j)$ the value $f_{ij}(T)$ is the minimum resource consumption at j given the resource consumption T at node i. Note further, that the term "the minimum resource consumption" is only well-defined if the REFs are nondecreasing. While ordinary REFs for arcs and segments propagate minimum resource consumptions forwards, inverse REFs propagate *upper bounds* for resource consumptions backwards. The inverse REF $f_{ij}^{inv} : \mathbb{R}^R \to \mathbb{R}^R$ takes any upper bound T' for the resource consumption at node j and computes the value $f_{ij}^{inv}(T')$ which is an upper bound for the resource consumption on node i. Similarly, for the inverse segment REF $f_P^{inv} : \mathbb{R}^R \to \mathbb{R}^R$, the resource vector $f_P^{inv}(T')$ is the upper bound for the resource consumption at the start node of P under the condition that one propagates resources along P and that T' is an upper bound for the resource consumption at the final node. The importance of segment REFs and their inverses is due to the following result:

Proposition 1 *(Irnich (2006), Theorem 3) Given resource-feasible paths $P_1, P_2, \ldots, P_q \in \mathcal{F}$, where the ith path P_i starts with a node w_{i-1} and ends with a node w_i, such that the end-node of P_i coincides with the start-node of P_{i+1} for all $i \in \{1, \ldots, q-1\}$. Their concatenation $P_1 + P_2 + \cdots + P_q$ is resource-feasible if and only if all inequalities*

$$\begin{aligned} f_{P_1}(a_{w_0}) &\le f_{P_2}^{inv}(b_{w_2}) \\ f_{P_1} \circ f_{P_2}(a_{w_0}) &\le f_{P_3}^{inv}(b_{w_3}) \\ &\vdots \\ f_{P_1} \circ f_{P_2} \circ \cdots \circ f_{P_{q-1}}(a_{w_0}) &\le f_{P_q}^{inv}(b_{w_q}) \end{aligned} \tag{1}$$

hold. (Note: $f \circ g(x)$ is defined as $f(g(x))$.)

A direct consequence of Proposition 1 is that the problem of efficiently testing the feasibility and computing gains is—at least partially—solved. A prerequisite is, however, that segment REFs must be available.

Theorem 2 *Let x be a feasible giant tour and let all segment REFs as well as inverse segment REFs w.r.t. x be already computed for all possible segments. Then, any neighbor solution $x' = m(x)$ of a ℓ-edge exchange move m can be tested for feasibility in $\mathcal{O}(\ell R)$ time.*

Since for all node-exchange and edge-exchange neighborhoods that are explored with tree search methods (cf. Funke et al., 2005a) the number ℓ of segments is constant (and small), Theorem 2 implies $\mathcal{O}(R)$ time feasibility checks.

3.3.3. Preprocessing

What remains to be done is to find efficient procedures to provide REFs and inverse REFs for all or (at least) a suitable subset of segments. Computing segment REFs and upper bounds for a given giant tour can be undertaken with a straightforward procedure requiring $\mathcal{O}(Rn^2)$ time and space. The reason is that there are $2n^2$ segments and inverted segments spanned between the n^2 pairs of nodes (note that moves might invert

some of the segments, so that inverted segments have to be considered, too). Segment REFs f_P for a segment $P = (v_i, \ldots, v_{j-1}, v_j)$ are generated from the segment REF $f_{P'}$ of the segment $P' = (v_i, \ldots, v_{j-1})$ and the REF f_{v_{j-1},v_j}. Similarly, for the segment $Q = (v_i, v_{i+1}, \ldots, v_j)$, $j > i$ the inverse segment REF f_Q^{inv} is computed from the inverse REF $f_{v_i,v_{i+1}}^{inv}$ and the inverse segment REF $f_{Q'}^{inv}$ of $Q' = (v_{i+1}, \ldots, v_j)$. With the general assumptions (a1)-(a5) on REF operations, each step requires $\mathcal{O}(R)$ time leading to the desired result.

Proposition 3 *Segment REFs and inverse segment REFs for all $2n^2$ segments and inverted segments of a giant tour of length n can be computed by a straightforward procedure in $\mathcal{O}(Rn^2)$ time and space.*

From a worst case point of view, a quadratic preprocessing is satisfactory if neighborhoods $\mathcal{N}$ of size $\mathcal{O}(n^k)$ with $k \geq 2$ are inspected. However, we would like to accelerate the average case and corresponding search strategies that scan less than $\mathcal{O}(n^2)$ neighbors. Moreover, it has been shown by Funke (2003) that restricting the length of some segments can lead to interesting neighborhoods that can be searched quickly. For instance, restricting the length of inverted segments to a fixed value K for 2-opt moves yields a $\mathcal{O}(Kn)$-sized neighborhood. Using first-improvement pivoting strategies in LS also requires accelerated methods for the preprocessing phase. Our aim is, therefore, to reduce the number of segments that have to be considered in feasibility testing procedures.

A solution to this problem is the definition of *seed points* dividing the nodes of the giant tour uniformly into *sections*. A 1-level hierarchy with parameter $\beta \leq 1$ uses equidistant *sections* of length n^β, so that n/n^β sections result, see Figure 3. The idea of a *hierarchy of*

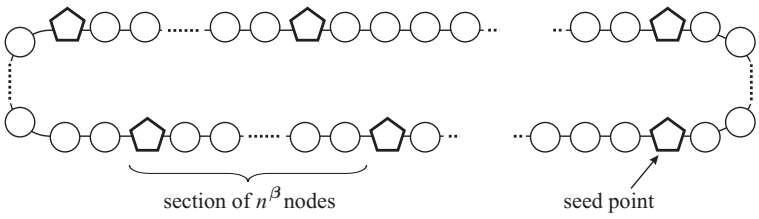

Fig. 3. 1-Level Hierarchy

REFs is that, instead of computing all $2n^2$ REFs for all segments, only segments *within a section* (i.e., between two consecutive seed points) and between *all pairs of seed points* need to be considered. In order to compute a REF ranging from position i to position j, one has to consider three cases: (1) If positions i and j fall into the same section, the REFs are already available. (2) If i and j are in two different but consecutive sections surrounding the unique seed point s, the REF between i and j can be computed as the concatenation of the REF from i to s and the REF from s to j. (3) Otherwise, there are at least two seed points between i and j with s_1 the first seed point following i, and s_2 the last seed point preceding j. The REF from i to j can be computed as the concatenation of three REFs, i.e., from i to s_1, from s_1 to s_2, and from s_2 to j. In all three cases, the segment REF from i to j is available in, at the utmost, $\mathcal{O}(3R) = \mathcal{O}(R)$ steps. The number of segment REFs to compute is

$$\mathcal{O}\left(2\frac{n}{n^\beta}n^{2\beta} + 2\left(\frac{n}{n^\beta}\right)^2\right) = \mathcal{O}\left(n^{\max\{1+\beta, 2-2\beta\}}\right),$$

where factor 2 is for forward or inverted segments, the first term is the computation of all REF inside sections, and the second term for the REFs between seed points. The effort is minimal for $\beta^* = 1/3$ resulting in $\mathcal{O}(n^{4/3})$ computations.

Proposition 4 *Segment REFs and inverse segment REFs for a 1-level hierarchy of seed points for a giant tour of length n can be computed in $\mathcal{O}(Rn^{4/3})$ time and space.*

Generalizations to hierarchies with two and more levels can be found in the Online Supplement.

3.3.4. Generic Search Procedure

The following pseudo-code formalizes a generic search procedure for searching an $\mathcal{O}(n^k)$ neighborhood $\mathcal{N}(x)$ of a current feasible solution x to determine a best neighbor solution x' with a gain $g(x,x') > G_{min}$. The parameter G_{min} is chosen as $G_{min} = 0$ for classical local search, $G_{min} = \infty$ if any non-improving neighbor could be accepted, and $G_{min} > 0$ for more selective procedures that consider only substantial improvements. Independent of G_{min}, the procedure guarantees a worst-case running time of $\mathcal{O}(n^{\max\{k,h\}})$ and needs $\mathcal{O}(n^h)$ space, with $h \in \{2, \frac{4}{3}, \frac{8}{7}\}$ depending on the type of hierarchy used.

Algorithm 2 Generic Local Search (=Step 4 of Algorithm 1)

1: **Input:** A feasible solution (giant route) $x = (w_0, \ldots, w_n) \in X$;
$G_{min} \in \mathbb{R}$ minimum gain.
(Phase 1 – Preprocessing)
2: LET $\mathcal{H}$ be the ℓ-level hierarchy of segment REFs f_P, f_P^{inv} describing the current solution x.
3: STORE positions i_w, and positions n_i, l_i of last/first route-start and route-end nodes (see remarks below).
(Phase 2 – Tree Search)
4: LET $G^* := G_{min}$.
5: LOOP decision d_1
6: LOOP decision d_2
⋮
7: ⋮
8: LOOP decision d_k
9: **(Implicit construction of move and neighbor solution)**
10: LET $m := m_d$ be the move implied by decisions $(d_1, d_2, \ldots, d_k)$.
11: LET $x' = (P_1, P_2, \ldots, P_q)$ the permutation of the segments of $(w_0, \ldots, w_n)$ implied by m.
(Feasible and Improving?)
12: LET $\mathcal{P} := (P_1^1, \ldots, P_1^{\ell_1}, P_2^1, \ldots, P_2^{\ell_2}, \ldots\ldots, P_q^1, \ldots, P_q^{\ell_q})$ be the segmentation
13: implied by $(P_1, P_2, \ldots, P_q)$ and $\mathcal{H}$.
14: LET feasible:=Formula (1) is fulfilled for $\mathcal{P}$
15: AND $C(x')$ task-feasible
16: AND $P(x')$ feasible w.r.t. route-start and route-end nodes.
17: LET $G := g(x,x') := f_{P_1^1}^{cost} \circ \cdots \circ f_{P_1^{\ell_1}} \circ f_{P_2^1} \circ \cdots \circ f_{P_2^{\ell_2}} \cdots \circ \cdots \circ f_{P_q^1} \circ \cdots \circ f_{P_q^{\ell_q}}(a_{w_0})$.
18: IF (feasible and $G > G^*$) THEN
19: **(Update of best neighbor solution found)**
LET $G^* := G$.
20: LET $d^* := (d_1, \ldots, d_k)$.
21: **Output:** Gain G^* and for $G^* > G_{min}$ optimal decisions d^* and best neighbor $x' = m_{d^*}(x)$.

Remarks:

(1) The preprocessing phase has to build REFs f_P and inverse REFs f_P^{inv} for some segments P and some inverted segments P (if the neighborhood also inverts one or several segments). The description of the preceding Section 3.3.3 and the corresponding extensions presented in the Online Supplement make clear which segments have to be computed. The results guarantee a worst-case effort of $\mathcal{O}(Rn^2)$, $\mathcal{O}(Rn^{4/3})$, and $\mathcal{O}(Rn^{8/7})$ for the feasibility test in Step 14 if one uses no hierarchy, a 1-level, or a 2-level hierarchy respectively. Note that this first preprocessing phase is identical for any type of neighborhood. By contrast, the second phase, the actual (tree) search, must be tailored to the neighborhood.

(2) With an $\mathcal{O}(n)$ preprocessing (Step 3), we store for each position $i \in \{1, \ldots, n\}$ of the giant tour the node w_i and, conversely, for each node w of the giant tour its position i_w.

(3) The loops in Steps 5-8 exactly determine the order in which moves and neighbor solutions are constructed. Section 3.2 has already explained that this order is crucial to the development of rules for (exactly) pruning the search tree.

(4) Some parts of the construction of the neighbor solution of steps 10 and 11 might already be performed in some of the outer loops in Steps 5-7. This can be useful for seeing that the resulting moves are infeasible neighbor solutions, so that the search can be terminated, i.e., only a part of the search tree has to be scanned.

Additionally, if outer loops can estimate the gain of the moves that are under construction, a pruning of the search based on gain consideration becomes possible. The next section on sequential search will explain a specialized criterion that often also takes the symmetry into account.

(5) The tasks exchanged by a move (if any) are typically determined by the nodes and edges that are removed and added. Hence, Step 14 can be performed in $\mathcal{O}(1)$ if appropriate data-structures are used.

(6) In Step 12, the segmentation $\mathcal{P}$ results from x' and the hierarchy $\mathcal{H}$. For instance, let $n = 1\,000$ and $\mathcal{H}$ be the 1-level hierarchy introduced in Section 3.3.3. Then, the $n^{2/3} + 1 = 101$ seed points are located at positions $0, 10, 20, 30, \ldots, 1\,000$. Let m be the swap move that exchanges the nodes at the positions 17 and 322. Then, $k = 5$ and $x' = (P_1, P_2, P_3, P_4, P_5)$ with P_1 the segment from position 0 to 16, P_2 the segment consisting of the node located at position 322, P_3 the segment from 18 to 321, P_4 the single-node segment at position 17, and P_5 the segment from position 323 to 1 000. Now, the hierarchy $\mathcal{H}$ implies a split of P_1 into P_1^1 from position 0 to 10, and P_1^2 from 10 to 16. P_3 is split into three segments P_3^1, P_3^2, P_3^3 from position 18 to 20, 20 to 320, and 320 to 321, respectively. Finally, P_5 is split into P_5^1 from position 323 to 330 and P_5^2 from 330 to 1 000, while P_2 and P_4 are not split. Hence, $\mathcal{P}$ consists of $2+1+3+1+2 = 9 \leq 3 \cdot 5 = \mathcal{O}(k)$ segments.

(7) In order to check feasibility w.r.t. route-start and route-end nodes in Step 16, one has to a priori record, for each position i of the giant tour, the next position n_i of a route-start node and the last position l_i of a route-end node. Along $\mathcal{P}$, consider pairs (P, P') of (consecutive) segments in $\mathcal{P}$. Let the first segment P contain the route-start node o_P as the last route-start node. If P' does not contain a route-end node (i.e., its last position j is smaller than its next route-start position n_i for the start position i) replace P' by its successor segment in $\mathcal{P}$. Repeat, until P' contains some route-start node and let $d_{P'}$ be the first route-start node in P'. Now that one knows route-start node o_P and $d_{P'}$ are linked (by request nodes or directly), one can check their compatibility. To iterate, replace P by P' and choose P' as the successor segment.

3.4. Sequential Search

Sequential search is a technique that allows neighborhoods within local-search algorithms to be scanned in a highly efficient way. It was discovered independently in the 1970s by Christofides and Eilon (1972) and Lin and Kernighan (1973) in algorithms for the traveling-salesman problem (TSP) and the graph-partitioning problem (Kernighan and

Lin, 1970). Apparently, the idea has since been forgotten and has not been tested for any type of constrained problem. Irnich et al. (2006) have introduced sequential search as a general method for accelerating LS procedures. It is based on the idea of decomposing moves into so-called partial moves, so that partial moves are cost-independent and imply partial gains whose sum is the overall gain of the move. Lin and Kernighan (1973) proved that if the sum of a sequence of numbers (gains) is positive, then there exists a cyclic permutation of these numbers such that every partial sum is positive. This can be generalized to restrict a k-decision search procedure to consider only those branches where the sum of the gains of the first $p \leq k$ partial moves has to be greater than pG^*/k, where G^* is a lower bound the overall gain. Details and pseudo-code of the application to several node and edge-exchange neighborhoods for CVRP can be found in (Irnich et al., 2006).

Note that sequential search is directly applicable only to those routing problems where the REFs are separable w.r.t. the cost resource, i.e., where the cost is given by the sum of the costs of all arcs in the giant tour. For more complicated cost functions that are not separable (see Section 3.2), the gain criterion might remain applicable if upper bounds for the resulting gain can be deduced from removed arcs and lower bounds of the resulting loss can be determined for added arcs. As far as we know, these ideas have not been tested thus far.

We present the main idea of sequential search for the case of a swap move, depicted in Figure 2. We decompose the swap move into two parts: The first part is the removal of the arcs (w_{i-1}, w_i), (w_i, w_{i+1}) and the addition of (w_{j-1}, w_i), (w_i, w_{j+1}). The second part consists of removing (w_{j-1}, w_j), (w_j, w_{j+1}) and adding (w_{i-1}, w_j), (w_j, w_{i+1}). For the entire move to be improving, the sum of the costs of the added arcs has to be smaller than the sum of the costs of the deleted arcs. Hence, either the first or the second part has to be improving. In the first case, starting the search at node w_i, the cost of the removed arcs is given by $B := c_{w_{i-1},w_i} + c_{w_i,w_{i+1}}$. It follows that either $c_{w_{j-1},w_i} < B/2$ or $c_{w_{j-1},w_i} < B/2$ must hold. By scanning the in-arcs $(w, w_i) \in A$ and out-arcs $(w_i, w) \in A$ of node $w_i \in V$ by increasing length, the search can be terminated whenever an arc longer than $B/2$ is found. Because of the symmetry, identical arguments cover the second case for starting the search with node j.

A prerequisite of this bounding procedure is that all in-arcs and out-arcs of a given node w_i are explored in an order, where they are sorted by increasing cost. Since in-arcs and out-arcs of w_i are fully determined by the other endpoint w of the arc, one can retrieve the required information from so-called *neighbor lists* $N^+(w_i)$ and $N^-(w_i)$. $N^+(w_i)$ is the list of head nodes of out-arcs (w_i, w) of w_i sorted by increasing cost. Analogous to this, $N^-(w_i)$ is the sorted list of tail nodes of in-arcs (w, w_i).

(Irnich et al., 2006) contains more detailed explanations of the theoretical background, such as the *gain criterion* and its application to routing and non-routing problems. The sequential search algorithm for the swap neighborhood can be formulated as follows.

Algorithm 3 Sequential Search for Swap (Phase 2, Tree Search)

1: **Input:** A feasible solution (giant route) $x = (w_1, \ldots, w_n) \in X$;
$G_{min} \in \mathbb{R}$ minimum gain.
It is assumed that Phase 1 (=preprocessing) is already performed.
2: LET $G^* := G_{min}$.
3: **(Outer Loop)**
4: LOOP $i \in \{1, \ldots, n\}$
5: LET $B := (c_{w_{i-1},w_i} + c_{w_i,w_{i+1}})/2 - G^*/2$.

6: **(Inner Loop, Case 1: Arc $(w_i, w_{j+1}) \in A$ must be short)**
7: LOOP $w_i \in N^+(w_i)$ AS LONG AS $c_{w_i,w} < B$
8: LET $j := i(w) - 1$.
9: IF $i > j$ THEN LET $t := i$, $i := j$, $j := t$
10: **(Implicit construction of move and neighbor solution)**
11: LET $P_1 := (w_1, \ldots, w_{i-1})$, $P_2 := (w_j)$, $P_3 := (w_{i+1}, \ldots, w_{j-1})$, $P_4 := (w_i)$, $P_5 := (w_{j+1}, \ldots, w_n)$.
12: LET $x' := (P_1, P_2, P_3, P_4, P_5)$.
13: LET $G := c_{w_{i-1},w_i} + c_{w_i,w_{i+1}} + c_{w_{j-1},w_j} + c_{w_j,w_{j+1}} - c_{w_{i-1},w_j} - c_{w_j,w_{i+1}} - c_{w_{j-1},w_i} - c_{w_i,w_{j+1}}$.
14: LET feasible:=Formula (1) is fulfilled for $(P_1, P_2, P_3, P_4, P_5)$
15: AND $C(x')$ task-feasible
16: AND $P(x')$ feasible w.r.t. route-start and route-end nodes.
17: IF ($G > G^*$ and feasible and $j \neq i + 1$) THEN
18: **(Update of best neighbor solution found)**
19: LET G*:=G.
20: LET $(i^*, j^*) := (i, j)$.
21: **(Inner Loop, Case 2: Arc $(w_{j-1}, w_i) \in A$ must be short)**
22: LOOP $w_i \in N^-(w_i)$ AS LONG AS $c_{w,w_i} < B$
23: LET $j := i(w) + 1$.
⋮
24: ⋮
25: /* Steps 9-20 */
⋮
26: ⋮
27: **Output:** Gain G^* and for $G^* > G_{min}$ optimal decisions (i^*, j^*) and best neighbor $x' = m^{swap}_{i^*,j^*}(x)$.

The most important part of the above algorithm is the computation of the bound B in Step 5 used to limit the iterations of the inner loops that have to be performed. This bound limits the length of the out-arc $(w_i, w) \in A, w \in N^+(w_i)$ in Step 7 or the in-arc $(w, w_i) \in A, w \in N^-(w_i)$ in Step 22 for any improving move. The sorting of the neighbor lists allows the termination of the inner loop whenever an arc not smaller than B comes up. Complete neighbor lists require $\mathcal{O}(n^2)$ space (for dense routing graphs) which can be computationally prohibitive when VRP instances with several thousands of nodes and millions of arcs are considered. Note that the neighbor list computation has to be performed only once in an initial preprocessing. Its time complexity is $\mathcal{O}(n^2 \log n)$ but, anyway, this time complexity is always dominated by the total running time of LS in practice. In order to reduce the required space, one can replace full neighbor lists by reduced neighbor lists, also called *candidate lists* (Glover, 1996), that contain only a subset of arcs (hopefully, the relevant ones!). A standard approach is to build candidate lists N^+_K, N^-_K that contain a fixed number K of request nodes while *all* route-start and route-end-nodes (depot nodes) are inserted into the candidate lists by default. Clearly, when using proper candidate lists, there is a tradeoff between the accuracy of the search and the computational burden. Irnich et al. (2006) have compared this tradeoff for the standard CVRP.

It should be pointed out that all *infeasible arcs*, i.e., arcs that cannot be part of any feasible giant tour, can be omitted from the neighbor lists. Using specialized probing techniques, as in (Desrochers et al., 1992; Ascheuer, 1995), one might substantially reduce the number of possible arcs. The combination of both the static and a priori determination of relevant arcs and the dynamic pruning of the search tree based on partial gains, is—as far as we know—the first approach to effectively combine feasibility-based and gain-based reductions. This technique is not limited to the swap neighborhood but can be applied to all enumerate search procedures for edge and node-exchange VRP neighborhoods. For a systematic explanation of move decomposition and, especially, of the gain criterion in sequential search procedures for different VRP neighborhoods, we refer the reader to (Funke et al., 2005a,b; Irnich et al., 2006).

4. Modeling Issues

This section summarizes which types of VRPs can be handled with the unified framework. Before we discuss particular types of constraints, we briefly repeat the basic assumptions:

(1) All feasible solutions of the given VRP can be modeled as giant tours. A giant tour is defined over a routing graph $G = (V, A)$. The length of a giant-tour is bounded by $n = \mathcal{O}(|V|)$, see Section 2.1.

(2) It must be possible to formulate the VRP as a discrete tasks-partitioning or task-covering problem, where tasks are associated with nodes and arcs or the routing graph, see also Section 2.1.

(3) The compatibility relation between route-start and route-end nodes must be given, see Section 2.2.

(4) All intra-tour and inter-tour constraints have to be modeled as resource constraints on paths, see Section 2.3. The resulting REFs must fulfill the assumptions (a1)-(a5) of Section 3.3.2. These assumption are in depth discussed and exemplified in (Irnich, 2006). Inter-tour constraints are the subject of Section 4.8* and (Hempsch and Irnich, 2007).

(5) All moves $m \in M$ of the neighborhood $\mathcal{N}$ under consideration decomposes a giant-route into ℓ segments. Any neighbor solution result from the permutation, (partial) inversion, and concatenation of the segments, see (Irnich et al., 2006).

The complexity of the segment REF representation, evaluation, and concatenation determines the effort for the preprocessing and the feasibility check in the tree search. If all these operations can be performed in $\mathcal{O}(R)$ time, then any $\mathcal{O}(n^k)$ neighborhood can be fully explored in $\mathcal{O}(\ell R n^k)$ time and $\mathcal{O}(R n^{4/3})$ space. These worst-case results are fully independent from the search tree exploration strategy. If REF manipulations require more than $\mathcal{O}(R)$ time, additional factors result in the above worst-case complexities (e.g., for multiple time windows, see below).

For the sequential search strategy, the only additional assumption needed is that the gain of a move is directly associated with the exchanged arcs. Thus, for any move m transforming x into x' (i.e., $x' \in \mathcal{N}(x)$), the gain $G = g(x, x')$ is given by the cost difference of the deleted and added arcs. In the case of more complex cost functions, e.g., if the overall cost of a tour depends on several resource consumptions (traveled distance, time on duty, ton-kilometers etc.), the gain criterion and the resulting sequential search principle are not applicable. However, if a lower bound for the cost of a neighbor solution can be estimated on the basis of exchanged arcs, the gain criterion remains applicable and gain-based tree search methods can be used to accelerate the tree search in the average case.

Table 3 provides a detailed overview of the modeling and solution capabilities of the unified framework: The modeling of capacity, distance, and time window constraints by REFs is straightforward. Section 4.1 show how to model capacity constraints in the context of combined collection and distribution. Different ways of modeling precedence constraints are presented in Section 4.2, and the consideration of lower and bound on the number of vehicles is discussed in Section 4.3. Several other examples of resources and their proper representation by REFs and resource intervals can be found in (Irnich and Desaulniers, 2005; Irnich, 2006; Hempsch and Irnich, 2007). Additional material can be

found in the Online Supplement. Topics discussed there are VRPs with compatibility constraints (Section 4.4*), interdependent resources (Section 4.5*), heterogeneous fleet VRPs (Section 4.6*), periodic VRPs (Section 4.7*), and inter-tour resources and constraints (Section 4.8*).

The column *Number of Resources* explains how many resources are needed to model the particular constraint. For instance, the constraint of not exceeding the vehicle capacity requires only one resource (which is reset to 0 on arcs that connect a route-end with a route-start node of the giant tour). A parenthesis *(dep.)* indicates dependent resources. The next to columns *Compatible with Lex. and Seq. Search* shows whether or not the constraints are compatible with the lexicographic or sequential search paradigm. Finally, column *Complexity of Feas. Check* states the time complexity of feasibility checking. The non-trivial complexity results (when REFs cannot be represented or evaluated in $\mathcal{O}(R)$ time) are taken from (Irnich, 2006).

4.1. VRPs with Collection and Distribution

Several types of VRPs exist where delivery and pickup (distribution to and collection from customers) are performed on the same tour. In backhauling applications (VRPB, e.g., Toth and Vigo (2002b); Røpke and Pisinger (2006)) all linehaul customers must be serviced before the backhaul customers of the same tour. The modeling framework can capture this constraint easily by a routing graph with one node for each customer by not allowing arcs that connect backhaul with linehaul customers.

When the visit of a customer implies that delivery and pickup at this location are performed simultaneously (VRPSDP, e.g., Min (1989); Halse (1992); Dell'Amico et al. (2006)), two dependent resources (pickup quantity and maximum load on partial path) are coupled by a non-classical REF. This technique with two dependent resources has been used by several authors, cf. (Desaulniers et al., 1998). Irnich (2006) shows that these REFs can be used in the context of efficient local search as explained in Section 3.

A mixture of VRPB and VRPSDP occurs if one allows the model to decide whether delivery and pickup at each specific customer are to be performed simultaneously or not (cf. Gribkovskaia et al. (2006)). The results are, e.g., so-called *lasso tours* where some customers are first supplied only, then a round trip along customers with simultaneous delivery and pickup is performed, and finally pickups at the first customers are performed (visited in reverse order). The saving in such an approach lies in a better utilization of the vehicle capacity, since performing deliveries at the beginning yields additional space for the collection in the second combined delivery and pickup phase. The paper by Gribkovskaia et al. (2006) shows that such a mixed approach has the potential for notable cost savings. The unified framework can handle the option of separate or simultaneous deliveries and pickups in the following way: Each customer is modeled by two nodes, one for the delivery and one for the pickup, with an additional pairing constraint guaranteeing that both nodes are served on the same tour (if required). Since the modeling of pairing constraints is very similar to the techniques applied for the PDP, we refer the reader to the next paragraph.

4.2. Precedence Constraints

For any two nodes $u, v \in V$, the relation $u \to v$ states that node u must precede node v in any feasible (giant tour) solution. In pickup and delivery applications, requests (i^+, i^-) impose unique pairs of precedences $i^+ \to i^-$. In order to cover these and alternative applications, we allow precedences given by a relation $\to$ on $V \times V$. For notational convenience, we define $P \to P'$ if and only if for two segments P, P' nodes $u \in P$ and $v \in P'$ exist with $u \to v$. It is assumed that the sets of predecessors and successors, i.e., $pred(v) = \{u : u \to v\}$ and $succ(u) = \{v : u \to v\}$ are of size $\mathcal{O}(1)$, such that the relocation of single nodes can always be checked for feasibility w.r.t. precedences in constant time.

The efficient handling of precedence constraints dates back to papers by Psaraftis (1983) and Savelsbergh (1990) and is also differently discussed by Kindervater and Savelsbergh (1997). Their idea is, again, that any move permutes and possibly inverts the segments $(P_1, \ldots, P_k)$ of the current (giant) tour according to $(P_{\pi(1)}^{\sigma(1)}, \ldots, P_{\pi(k)}^{\sigma(k)})$ with π a permutation of $\{1, 2, \ldots, k\}$ and $\sigma(i) \in \{-1, 1\}$ (indicating inversion by -1). Hence, feasibility tests require constant time procedures to check

(A) whether an inverted segment P_i^{-1} is feasible w.r.t. precedences and

(B) whether or not $P_i \to P_j$ holds for two segments with $j > i$ and $\pi(j) < \pi(i)$.

For the task (A) and a given giant tour $(v_1, \ldots, v_n)$ let $first_u := \min\{p : u \to v_p\}$ be the first position of a destination of a precedence starting at node u. Moreover, for each position $p \in \{1, 2, \ldots, n\}$ let $firstdest_p := \min\{first_{v_\ell} : \ell \geq p\}$ be the position of the first destination of a precedence pair beyond position p. The computation of $first_u$ for all nodes u and of $firstdest_p$ for all positions p can be undertaken in $\mathcal{O}(n)$ steps. Since the inversion of a segment $P = (v_p, v_{p+1}, \ldots, v_\ell)$ is feasible w.r.t. precedences if and only if $\ell < firstdest_p$, the result is a constant time feasibility test for all moves that only invert segments. The 2-opt move is the most prominent example. Since the relocation of a fixed number of nodes requires an $\mathcal{O}(1)$ feasibility test only, all classical moves of quadratic neighborhoods can be checked in $\mathcal{O}(1)$, too. These neighborhoods include node relocation, node swap, Or-opt (with or without inversion of the short segment), and string-exchange moves. Similar straightforward procedure can be applied to the 2-opt* neighborhood.

However, larger neighborhoods, such as 3-opt and 3-opt* neighborhoods, can be applied to the giant tour and require efficient procedure to perform task (B). Here, the methods of Kindervater and Savelsbergh are applicable only if lexicographic search is used. In order to handle more powerful neighborhoods inspected by sequential search, we describe another technique for the PDP which uses one binary resource for each pickup/delivery pair $i = (i^+, i^-)$. The corresponding resource has a resource interval $[0, 1]$ at all nodes except the pickup node i^+ and end-tour nodes where the interval is $[0, 0]$. Entering into node i^+ increases the resource by one unit, entering i^- decreases the resource by one unit. All other REFs do not change the resource value. It is easy to see that these simple rules guarantee that no tour contains a delivery without a corresponding pickup node at an earlier position. Since the number of resources coincides with the number of requests, constant time feasibility checks are no longer guaranteed. However, an encoding with *binary resources* leads to a compact representation, since 32 or 64 resource can be encoded in one integer resource on a computer with 32 or 64 bit arithmetic.

4.3. Limiting the Number of Vehicles

The giant-tour representation implies that the number of routes is $N := |O| = |D|$. If arcs $(o, d) \in O \times D$ with costs $c_{od} = 0$ are present in the routing graph, routes $p = (o, d)$ can be part of the giant tour and, therefore, the possibility of using less than N proper tours is taken into account. Moreover, the constraint of using between $N_{\min}$ and N proper tours can be modeled by partitioning O and D into $O = O_1 \cup O_2$ and $D = D_1 \cup D_2$ with $|O_1| = |D_1| = N - N_{\min}$ and $|O_2| = |D_2| = N_{\min}$. Nodes pairs from $O_1 \times D_2$ and $O_2 \times D_1$ are incompatible. Let $c_{od} = 0$ for $(o, d) \in O_1 \times D_1$ and $c_{od} = M$ for $(o, d) \in O_2 \times D_2$ and let M be a sufficiently large number. If existent, a cost-minimal route plan with between N_{min} and N proper tours can be found in the routing graph as a feasible Hamiltonian cycle which does not use arcs $(o, d) \in O_2 \times D_2$.

An additional complication arises if a construction heuristic provides a route plan with more than N routes. This solution cannot be represented directly as a Hamiltonian cycle in G. The following technique solves the task of finding a feasible initial solution by means of additional *dummy* route-start and route-end nodes together with a single additional resource for counting the lengths of tours. Dummy route-start and route-end nodes $(\hat{o}, \hat{d}) \in \hat{O} \times \hat{D}$ are introduced in order to hold a single request node i that is (currently) not assigned to a feasible route. More precisely, a dummy route is either of the form $(\hat{o}, i, \hat{d})$ or $(\hat{o}, \hat{d})$ (i.e., occupied or empty). In order to stipulate the movement of a request node from a dummy $(\hat{o}, i, \hat{d})$ to a feasible route, costs are defined as $c_{\hat{o},i} = c_{i,\hat{d}} = M$, and $c_{\hat{o},\hat{d}} = 0$. Furthermore, the upper bound on the length of a route is set to 2 at all dummy route-end nodes but unbounded at all other nodes. The bound of 2 guarantees that no nodes are shifted from a feasible into a dummy tour that is already occupied. Using a sufficiently large number of dummy nodes, one can transform any start solution with more than N routes into a formally feasible solution with only N regular routes but several dummy routes.

The same technique can be used in different contexts. First, if the objective is to minimize the number of routes, one can resolve one route (which contains only a few nodes) and put these into dummy routes. Applying different LS operators, e.g., relocation and swap in combination with edge exchanges, one systematically tries to reduce the number of unassigned nodes from dummy routes. Second, the implementation of large neighborhood search (LNS) operators, as suggested by Shaw (1998); Schrimpf et al. (2000); Røpke and Pisinger (2006), is straightforward. Tailored removal operators determine a subset of nodes which are removed from their current positions of the giant tour. These nodes are relocated into empty dummy routes. Different insertion strategies (the order in which removed nodes are inserted into feasible tours again) can be controlled by putting different values M onto the arcs $(\hat{o}, i)$ and $(i, \hat{d})$. Third, VRPs in which tasks can be covered by alternative nodes (see Section 2.1) need mechanisms to select one or several nodes from given subsets to be serviced. Unselected nodes can be kept in dummy routes while algorithmic procedures in the feasibility test have to ensure that moves do only produce solutions where a task is covered the right number of times.

5. Computational Results

The previous sections were mainly focused on modeling and the theoretic aspects of efficient LS algorithms for rich VRPs. In contrast, this section is intended to present

empirical results that show the effectiveness of the preprocessing and the sequential search procedures in practice.

5.1. Preliminaries

Before analyzing the proposed new techniques based on benchmark problems and instances from the literature, we have to explain and clarify the following aspects: Which neighborhoods $\mathcal{N}$ and sequential search procedures are used? How are different neighborhoods combined to form a well-structured metaheuristic? How are sequential search and lexicographic search procedures compared, in particular, how is the speedup measured? Finally, at least two significantly different implementation concepts exist that constitute two extreme points w.r.t the tradeoff between fast runtime and economical use of memory.

5.1.1. Neighborhoods, Moves, and Sequential Search Procedures

We have implemented lexicographic and sequential search procedures for the neighborhoods listed in Table 1: For a detailed description of the neighborhoods and for pointers

Neighborhood	Size $\lvert\hat{\mathcal{N}}(x)\rvert$	Priority
swap, 2-opt, (special) 2-opt*, node relocation	$\mathcal{O}(n^2)$	1
string exchange, Or-opt with and w/o inversion	$\mathcal{O}(n^2)$	2
acb-opt, request relocation	$\mathcal{O}(n^3)$	3

Table 1
Neighborhoods, Sizes, and Priorities in VND

to the (original) literature, we refer the reader to the surveys (Funke et al., 2005a; Bräysy and Gendreau, 2005a) while the corresponding sequential search procedures with pseudocode are explained in (Irnich et al., 2006; Bellscheidt, 2005). In order to be self-contained, we briefly recall basic properties of these neighborhoods.

Figure 4(a) depicts the principle of a swap move which was already used for explanation in the preceding sections. Figure 4(b) shows a (special) 2-opt* move. Its interpretation is that two routes are cut into two pieces and the resulting end-pieces are exchanged. A 2-opt move takes a segment of the giant tour and inverts it as depicted in Figure 4(c). Sequential search is applicable directly only to cost-symmetric instances and we have restricted the generic search procedure to invert only segments which do not contain route-start nodes and route-end nodes. In these cases, the 2-opt neighborhood is an intra-tour neighborhood although our implementation does not make use of this fact. However, Funke et al. (2005a) have suggested inversion principles for segments that also contain route-start nodes and route-end nodes. The Or-opt neighborhood relocates a string to another position in the giant tour, and the string-exchange neighborhood swaps to strings. Both types of moves are depicted in Figure 4(e) and (g), respectively. Because of the giant-tour representation, they are at the same time intra-tour and inter-tour neighborhoods. We have chosen to limit the length of the swapped/relocated strings to a length of $\ell \leq 3$. A variant of the Or-opt move, here called *inverted* Or-opt, relocates a string and inverts it, see Figure 4(f). A special case of the Or-opt move is the relocation move that relocates a single node, i.e., a string of length 1. It is depicted in Figure 4(d).

The only cubic neighborhoods considered here are the acb-neighborhood and request-

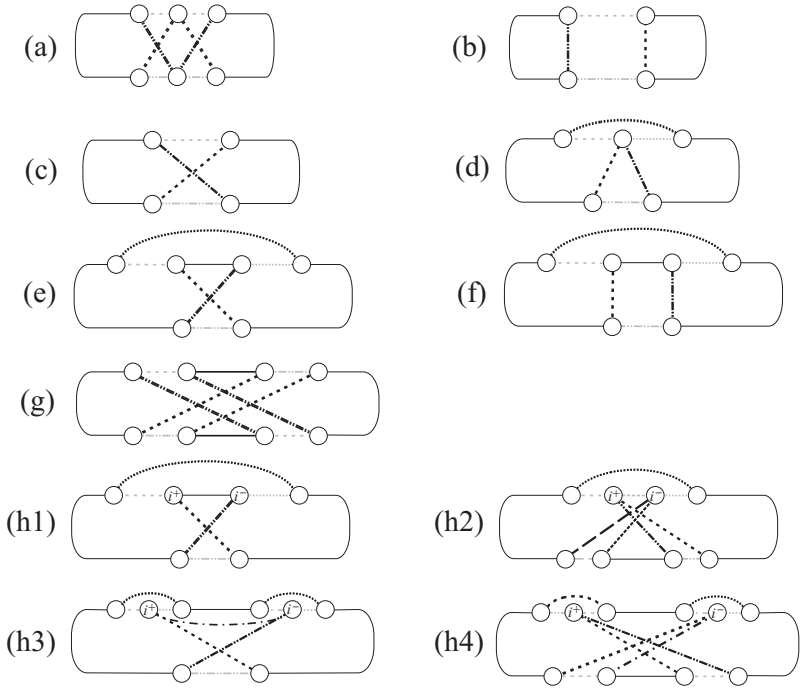

Fig. 4. Moves and their Decomposition, (a) Swap, (b) (Special) 2-opt*, (c) 2-opt, (d) Node Relocation, (e) Or-opt, (f) Or-opt with String Inversion, (g) String-Exchange, (h) Request Relocation with 4 Subcases

relocation neighborhood. An acb-move cuts the giant tour into three segments a,b,c and rearranges them to a,c,b (a classification of k-opt* neighborhoods and moves based on this notation was introduced by Funke et al. (2005b)). This neighborhood constitutes a proper extension of the Or-opt neighborhood because a string of *unlimited* length is relocated. If all three strings contain route-start and route-end nodes, the acb-move permits the cutting of three routes into two pieces and the re-connecting of three end-pieces with the three start-pieces.

5.1.2. General Setup for Local Search

Our comparisons of sequential search and lexicographic search procedures are always performed using the following setup that combines VND (Hansen and Mladenović, 2001, 2002) with LNS (Shaw, 1998; Røpke and Pisinger, 2006) strategies to escape local optima. An initial solution is computed by a problem-specific start heuristic. Starting from this solution, a local optimum w.r.t. all neighborhoods is computed. In order to apply computationally costly operators not too often, we have associated priorities (see Table 1) to all neighborhoods. Neighborhoods with priority 1 are searched exhaustively first. More precisely, we alternate between the swap, 2-opt, 2-opt*, and relocation neighborhoods on the first search level. Here, sequential and lexicographic search procedures are both applied to the same current solution x. If an improving solution $x' \in \mathcal{N}(x)$ is found, the corresponding move is performed and a new search step continues with the next neighborhood of level 1. Since we are using a best-improvement pivoting strategy for both sequential and lexicographic search, the corresponding two procedures return improving solutions with identical gain (note however, that, due to degeneracy, we cannot assure that identical solutions are computed; the improving solution found by sequential search is taken for the next search step). If none of the search procedures finds a move with positive gain, the search is continued with neighborhoods of priority 2 following the same cyclic alternating strategy as for level 1. The only difference is that, when improving solutions are found by a neighborhood of priority $p > 1$, then faster neighborhoods of priority level 1 are tested

again. This strategy is a minor modification of Hansen and Mladenovic's VND metaheuristic which makes the search more balanced for equally-sized neighborhoods with (empirically) identical search effort.

For small-sized instances, VND with prioritized neighborhoods can result in only a few calls of search procedures of priority 3. Therefore, three iterations of LNS with a random removal of 20 nodes (implemented as suggested in Section 4.3) and a simple cheapest-insertion procedure are used to perturb the current solution such that one can iteratively apply the above VND procedure. Hence, the VND procedure is called for four (in general) different start solutions. This setup guarantees that a mix of solutions with poor as well as already good quality are presented to the LS procedures.

5.1.3. Relative Speedup of Sequential Search vs. Lexicographical Search

The main part of the computational study compares the running times of lexicographic search and sequential search procedures for the neighborhoods given above. Recall that both approaches guarantee constant time feasibility checks. We will not compare our approach with a trivial implementation using straightforward node-by-node feasibility tests, since these techniques are obviously inferior.

A fair comparison of the running times by means of a *relative speedup factor* is rather delicate to compute for the following reasons. First, the preprocessing for the sequential search procedure (cf. Section 3.3.3) has to be executed only if the giant tour has changed, i.e., a preceding search (of the same or another neighborhood) has found an improving solution that has now become the incumbent solution. Hence, there is no intrinsic connection between the current search procedure and the preprocessing. Second, the ratio between successful and unsuccessful searches strongly depends on the general setup in which LS is performed, i.e., the start solutions, the mix of neighborhoods and the priorities for mixing them in VND/VNS. Third, the most frequently called procedures in the search algorithm are the test of whether or not an arc exists, and the computation of the arc costs. The following section will distinguish between two implementation principles that also have an impact on the speedup factors.

The most optimistic acceleration factor does not consider the additional effort of the necessary preprocessing for sequential search at all. Let $t_{\mathcal{N}}^{lex}$ and $t_{\mathcal{N}}^{seq}$ be the running times of lexicographic and sequential search procedure (without time for preprocessing) for a neighborhood $\mathcal{N}$. Then $f_{\mathcal{N}}^{max} = t_{\mathcal{N}}^{lex}/t_{\mathcal{N}}^{seq}$ is the *maximum speedup* or *maximum acceleration factor*. Note that running times might significantly vary depending on the current giant tour x and whether a good bound $B = B(G^*)$ (see Step 5 of Algorithm 3) is available early in the sequential search procedure. Therefore, only average values for $t_{\mathcal{N}}^{seq}$ and $t_{\mathcal{N}}^{lex}$ are considered here. A very pessimistic and conservative factor is based on the assumption that every sequential search procedure is preceded by a preprocessing. Defining t^{pre} as the (average) time of the preprocessing procedure (Steps 2–3 of Algorithm 1), the factor $f_{\mathcal{N}}^{min} = t_{\mathcal{N}}^{lex}/(t^{pre} + t_{\mathcal{N}}^{seq})$ denotes the *minimum speedup* or *minimum acceleration factor*. This factor applies to pure local search procedures in which only a single neighborhood N is incorporated such that the number of preprocessing and search steps coincide. From our point of view, the most fair definition of the speedup factor takes into account that only a fraction of search steps is preceded by a preprocessing. Let $r^{pre} \in (0, 1]$ be the (instance and setup specific) ratio of the number of improvement steps performed to the overall number of search procedure calls. We define $f_{\mathcal{N}} = t_{\mathcal{N}}^{lex}/(r^{pre}t^{pre} + t_{\mathcal{N}}^{seq})$ as the

speedup or *acceleration factor*. Note that $f_{\mathcal{N}}^{min} \leq f_{\mathcal{N}} < f_{\mathcal{N}}^{max}$ holds, but that all values still depend on the initial solution, the choice of neighborhoods, the VND/LNS strategy as well as on several implementation issues.

5.1.4. Implementation Issues

An instance with $\mathcal{O}(n)$ request nodes and a giant tour of length n can have up to $n(n-1)$ arcs in the routing graph and, therefore, a quadratic number of REFs. In the case of large-scale instances (with more than about 2 500 nodes), the representation of the routing graph and the associated REFs becomes an issue. We propose two alternative techniques for implementing the unified framework.

The first option is to a priori compute all arcs and associated REFs and to then store them in a matrix. This matrix needs to have $n \times n$ entries, with entry ij undefined if the arc (i, j) is infeasible. For classical REFs of the form $f_{ij}(T) = \max\{a_{ij}, T + t_{ij}\}$ with inverse REF $f_{ij}^{inv}(T) = \min\{b_{ij}, T - t_{ij}\}$, it is natural to store the defining coefficients $a_{ij}, b_{ij}, t_{ij} \in \mathbb{R}^R$ together at entry ij of the matrix in order to have a direct constant time access to the REFs. Since the memory requirement for the REF matrix is already quadratic, one can combine this representation of the routing graph with full neighbor lists $N^+(v)$, $N^-(v)$ for all nodes $v \in V$ as explained in Section 3.4. The computational results will show that this straightforward representation is the fastest but obviously consumes a lot of memory.

The second option is to use (heuristically) reduced candidate lists $N_K^+(v), N_K^-(v)$ for the in-arcs and out-arcs together with a procedure that computes REFs *on-the-fly*. For any pair (i, j) of nodes, a first procedure checks whether (i, j) is a feasible arc of the routing graph (V, A). In the case where (i, j) is feasible, a second procedure returns the REFs f_{ij} and f_{ij}^{inv} (as an object containing a_{ij}, b_{ij}, t_{ij} or implicitly, e.g., by computing $f_{ij}(T)$ for T given). If the entire VRP instance can be represented in $\mathcal{O}(n)$ memory, e.g., when times and costs are computed using coordinates and distances in the 2-dimensional Euclidean plane, the on-the-fly computation reduces the memory requirement for the framework. Since the techniques of Section 3.3.3 enable us to store segment REFs in $\mathcal{O}(n^{4/3})$ space, the overall memory requirement typically results from storing neighbor or candidate lists. As a result, we are able to handle VRPs with more than 10 000 nodes at the cost of not being fully accurate (since candidate lists must heuristically exclude some parts of the neighborhood to be scanned). Moreover, computing (complicated) REFs on-the-fly takes more time than a direct access to REFs stored in main memory and, therefore, this second option is, in general, slower. However, the computational results of the next sections indicate that, for on-the-fly REF computations, the speedup of sequential search over lexicographical search increases. At the same time, speedup factors increase when on-the-fly computation is performed. The reason for this is that the computational overhead in sequential search procedures (caused by the handling of neighbor lists, computing partial gains etc.) becomes less important.

The unified framework was coded in C++, different resource concepts and types of REFs were integrated as *template parameters*. The algorithms were compiled in release mode (using MS-Visual C++ .NET 2003 version 7.1), and all runs were performed on a standard PC (Intel x86 family 15 model 2, 2.4 GHz, 1GB main memory, on MS-Win 2000). Times were recorded using the `time.h` library. In order to be more precise, especially for times smaller than 10ms, we performed multiple identical runs of the same procedure. We made

sure that running times of multiple runs exceeded 100ms such that the average run time is a rather accurate estimate for a single run.

5.2. Vehicle Routing Problems with Time Windows

The VRPTW is certainly the most studied variant of VRPs and can be considered *the prototype* of "rich" VRPs, since time window constraints already require sophisticated techniques for constant time feasibility tests. Early work on VRPTW dates back to the 1960s and, since then, hundreds of scientific articles have addressed modeling as well as methodological aspects of developing exact and heuristic solution algorithms. For an overview, we refer the reader to the comprehensive surveys (Cordeau et al., 2002; Kallehauge et al., 2005; Bräysy and Gendreau, 2005a,b).

The Solomon (1987) and Homberger (see Homberger and Gehring, 1999) VRPTW instances have been used as benchmark problems in numerous empirical studies. While Solomon's instances have a fixed number of 100 customers, the Homberger instances range from 200 to 1,000 customers. We therefore use the latter because we are mainly interested in analyzing the behavior of the search procedures w.r.t. the number of tasks and the (average) number of nodes in a route. Initial solutions were computed using Solomon's I1-heuristic (Solomon, 1987) and REFs were a priori computed according to the first implementation concept sketched in the previous section.

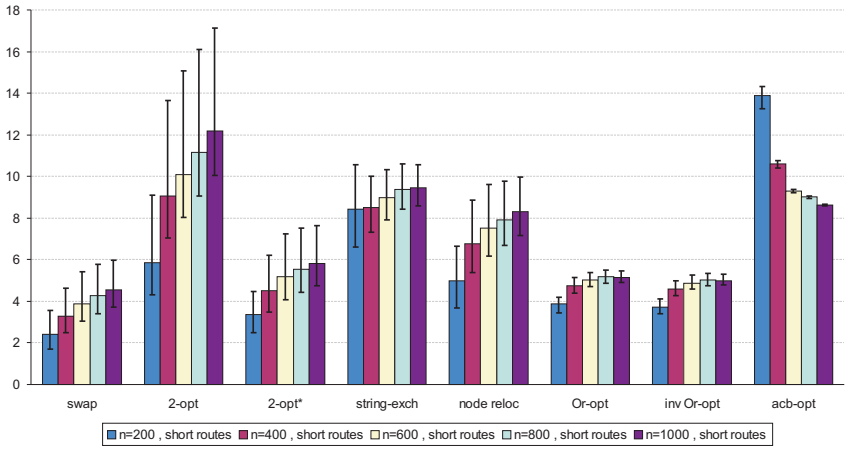

Fig. 5. Speedup of Sequential Search vs. Lexicographic Search for Homberger VRPTW Instances with Short Routes

The main results for the Homberger instances are depicted in the Figures 5 and 6 for instances with short (C1, R1, RC1) and long (C2, R2, RC2) routes respectively. Each column shows the speedup factors for 30 VRPTW instances, reflecting different problem characteristics (10 clustered, 10 randomly distributed, 10 mixed). The speedup factor $f_{\mathcal{N}}$ is depicted as a bar, while $f_{\mathcal{N}}^{min}$ and $f_{\mathcal{N}}^{max}$ are shown as error indicators. Both diagrams indicate that there is always a speedup when a lexicographic search approach is replaced by a sequential search procedure. In the first group, capacities and time windows are chosen in such a way that the average number of customers in a route is about 10. Here, the acceleration factors vary from 2.4 to 4.5 for swap, from 5.9 to 12.2 for 2-opt, from 3.4 to 5.8 for 2-opt*, from 8.4 to 9.5 for string exchange, from 5.0 to 8.3 for node relocation, from 3.9 to 5.2 for Or-opt, from 3.7 to 5.0 for Or-opt with segment inversion, from 8.6 to

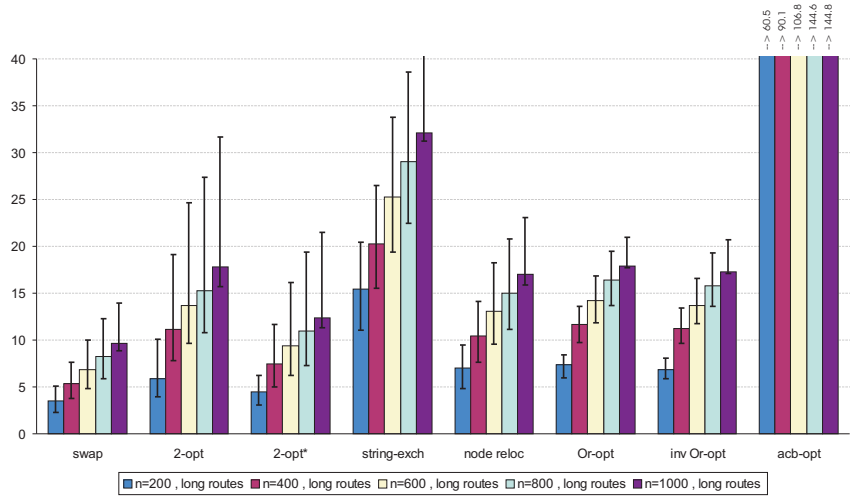

Fig. 6. Speedup of Sequential Search vs. Lexicographic Search for Homberger VRPTW Instances with Long Routes

13.9 for acb-opt. For all neighborhoods, except for acb-opt, there is a positive correlation between the size of the instance and the speedup. This contrasts with the results for the CVRP in (Irnich et al., 2006) where a clear negative correlation was only observed for the string-exchange neighborhood. It remains unclear to us which characteristics of instances or properties of neighborhoods imply such negative correlations.

The second group, depicted in Figure 6, contains instances with long routes, i.e., between 20 and 40 customers per route. Similar to the results reported for the CVRP in (Irnich et al., 2006), the speedup grows when problems are less constrained. Here, the acceleration factors vary from 3.5 to 9.7 for swap, from 5.9 to 17.8 for 2-opt, from 4.5 to 12.3 for 2-opt*, from 15.4 to 32.1 for string exchange, from 7.0 to 17.0 for node relocation, from 7.3 to 17.9 for Or-opt, from 6.9 to 17.3 for Or-opt with segment inversion, from 60.5 to 144.8 for acb-opt. These are substantial speedups! The superiority of sequential search over lexicographic search for less constrained instances can be explained as follows: In more constrained problems (especially with tight time windows), optimal feasible routes can differ significantly from cost-minimal TSP tours (and geometric intuition). Hence, a larger fraction of moves seems improving (when looking at costs/gains only) but is in fact infeasible. Consequently, gain-based arguments to terminate the search apply less often.

The absolute running times of the sequential search procedures applied to the Homberger instances are shown in Table 2. Each entry t/d shows the absolute average running time t (ten groups, each with 30 instances) and standard deviation d for the preprocessing and the actual sequential (tree) search. The preprocessing times (computation of the segment REFs, see Section 3.3.3) are growing with the size of the instances. The standard deviation is small, and the absolute values and deviations for instances with short and long routes are similar. It seems that the values primarily depend on the size of the instances, since the time complexity of the preprocessing does not depend on the number of arcs/REFs of the instance. By contrast, the running times of the tree searches very much depend on characteristics of the instances: First, the running times of the instances with short tours are significantly larger than those for the instances with long routes. We think that the reason for this difference is again that for more constrained problems gain-

Instances	Prepro-cessing (Phase 1)	Swap	2-opt	2-opt*	string-exch	node reloc	Or-opt	inv Or-opt	acb-opt
					(Phase 2)				
200.short	0.6/0.02	1.0/0.4	0.9/0.1	1.4/0.5	2.8/1.7	1.3/0.5	5/2	5/2	31/23
400.short	1.9/0.1	3.6/1.6	2.9/0.3	4.9/2.6	12.1/7.9	5.0/2.1	21/10	21/10	264/221
600.short	3.7/0.2	7.5/3.6	6.2/0.6	10.9/7.1	27.8/19.1	11.5/5.3	49/26	50/26	1014/945
800.short	5.3/0.3	12.1/6.3	10.1/1.4	19.4/13.9	49.2/35.5	20.4/10.1	88/51	90/53	2435/2363
1000.short	7.2/0.4	19.3/10.8	15.5/2.6	32.8/24.9	81.8/62.9	34.6/18.8	151/94	156/97	5528/5690
200.long	0.5/0.02	0.7/0.3	0.5/0.1	0.8/0.2	1.3/0.6	0.9/0.3	2/1	2/1	5/3
400.long	1.7/0.1	2.5/0.8	1.8/0.2	2/0.8	4.4/2.2	3.0/0.9	7/3	7/3	29/32
600.long	3.2/0.2	4.4/1.6	3.2/0.4	3.6/1.9	8.3/5.0	5.7/2.1	14/7	14/7	90/134
800.long	4.9/0.2	6.6/2.7	4.9/0.6	5.5/3.6	13.3/8.8	8.9/3.6	21/12	21/12	182/293
1000.long	6.7/0.3	9.9/4.2	7.0/1.0	8.6/6.4	20.5/14.6	14.0/6.8	34/22	34/22	385/676

Table 2
Avg. Running Times of a Preprocessing (Phase 1) and Sequential Search (Phase 2), Values in Milliseconds [ms], Absolute Value and Standard Deviation

based arguments for terminating the tree search are less effective. Second, one can see that the standard deviations of the running times are enormous. The explanation for this is that instances within the same group are still very much different: The time window constraints imply routing graphs that have arc sets of completely different sizes. Consequently, instances have neighbor lists of different magnitude, which directly imposes heavily varying running times. Finally, it is worth mentioning that due to the techniques presented in Section 3.3.3 (1-level hierarchy of REFs and $\mathcal{O}(n^{4/3})$ time complexity for its update), the time required for preprocessing is always smaller that the time for the sequential tree search.

Very large-scale VRPTW instances are—as far as we know—not available. Hence, we created a small test set of 10 instances ranging from $n = 1\,000$ to $10\,000$ customer nodes. The instances allow an average number of about 45 customers per route. Results for these instances are visualized in Figure 7. Note that we have considered only quadratic neighborhoods because the running times of the lexicographic search procedure for the acb-opt neighborhood of size $\mathcal{O}(n^3)$ were unacceptably long (more than 5 minutes for the largest instance and a single call of the search procedure). In contrast to the first tests, we have used on-the-fly computations of REFs and candidate list $N_K^+(v)$ and $N_K^-(v)$. K is chosen such that each candidate list contains the $1\,000$ closest customer nodes and all possible route-start and route-end nodes, i.e., $K \leq 1\,000 + |O|$. The most important insight for the large-scale problem instances is that the speedup grows even further. This is partly caused by the fact that we use on-the-fly computation of costs and REFs and also because average route lengths increase. We have also computed acceleration factors with the a priori computed REFs for the instances with $n = 1\,000$ and $n = 2\,000$ and compared them with those results obtained for the on-the-fly implementation: The on-the-fly computation gives a contribution to the speedups by an average factor of between 1.5 and 1.7 (but varying more strongly for different move types).

5.3. Capacitated VRPs and VRPs with Globally Constrained Resources

Simple versions of VRPs, such as the CVRP or the distance-constrained VRP, have pure additive REFs along the routes and globally fixed upper bounds (a maximum load or travelled distance). Therefore, they do not need the $\mathcal{O}(n^{4/3})$ preprocessing as presented in Section 3.3.3. Instead, a linear time and space preprocessing already allows constant time feasibility tests, see (Irnich et al., 2006). Very similar methods can be used for the

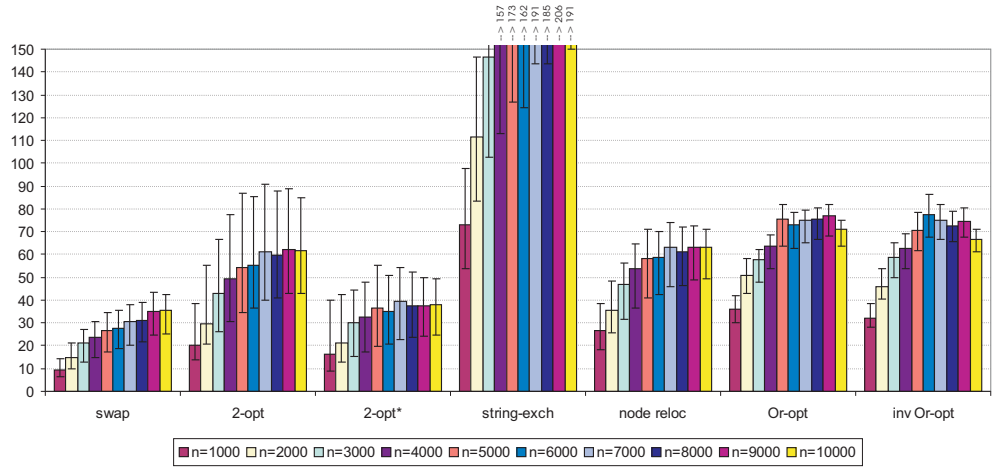

Fig. 7. Speedup of Sequential Search vs. Lexicographic Search for Large-Scale VRPTW Instances with between $n = 1\,000$ and $10\,000$

"pure" multi-depot VRP with capacity and distance constraints. Thus, these types of VRPs are beyond the scope of this article and we refer the reader to (Irnich et al., 2006) for results on the CVRP.

5.4. Multi-Depot VRPs with Time Windows

The Solomon (1987) benchmark set for VRPTW can be easily extended to generate multi-depot VRPTW (MDVRPTW) instances. Since locations for depots and customers are given as pairs (x, y) in the Euclidean plane, these locations can be copied and shifted in space. Given that the locations of a VRPTW instance are in a rectangle of size $\Delta_x \times \Delta_y$, we have chosen to shift these locations by multiples of $0.9 \cdot \Delta_x$ horizontally and by multiples of $0.9 \cdot \Delta_y$ vertically. For example, in order to create a $20 = 5 \times 4$ depot instance, we generate 19 copies and shift them by $(0.9m_x\Delta_x, 0.9m_y\Delta_y)$ for $(m_x, m_y) \in \{0,1,2,3,4\} \times \{0,1,2,3\}, (m_x, m_y) \neq (0,0)$. Initial solutions for the separate VRPTW instances belonging to one depot are created with the VND approach of Section 5.2 (we do not report the corresponding running times and speedup factors for those runs, since VRPTW instances are small). Multiple copies of these separate VRPTW solutions are taken as initial solutions for the MDVRPTWs. Because of the overlap created by the test generator (factor 0.9), the subsequent VND and LNS procedures have the potential to create improving solutions. These improvements primarily result from exchanges between tours of different depots. In turn, modified partial solutions belonging to a single depot might be improved by node and edge exchange, too. The result is a mix of intra-depot and inter-depot exchanges, which are all uniformly handled by the giant-tour representation.

The Figure 8 depicts results for MDVRPTW instances with n between 100 and 2 450. All instances were created from 12 selected 50 customer VRPTW instances (c103, c109, r103, r112, rc101, rc106, c205, c208, r204, r208, rc202, rc207). The criterion for selecting these instances was to yield a mix of clustered and unclustered instances, instances with tight and wide time windows, and with short and long routes. We used both implementation concepts, on-the-fly computation of REFs for instances with $n \geq 1\,250$ and full REF representation for smaller instances. Neighbor and candidate lists $N_K^+(i)$ and $N_K^-(i)$ were restricted to contain a maximum of 1 000 request nodes but all route-start and route-end

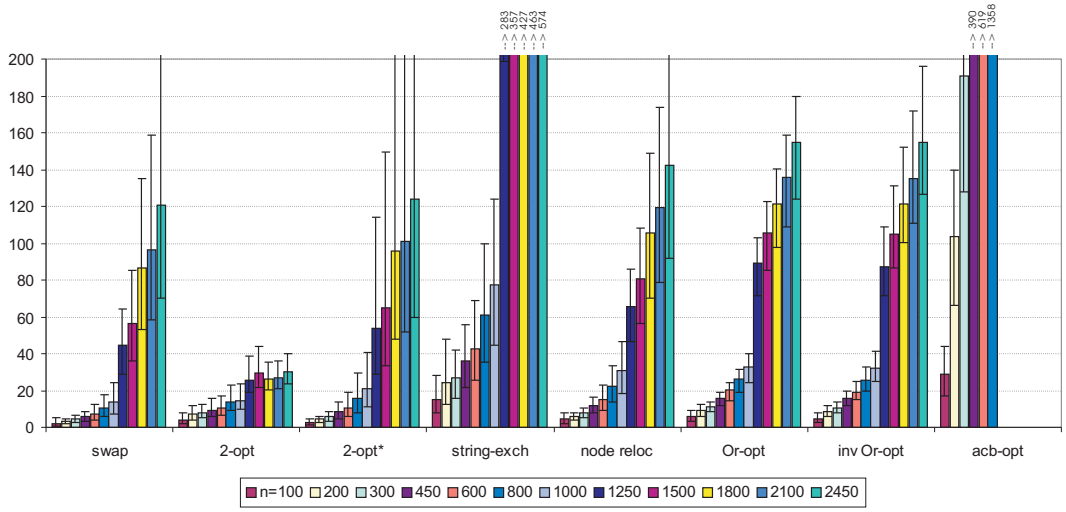

Fig. 8. Speedup of Sequential Search vs. Lexicographic Search for Multi-Depot VRPTW Instances

nodes.

The smallest speedup was found—as was to be expected—for the smallest instances with $n = 100$ and for the swap move. Here the factors $f_{swap} = 2.3$, $f_{swap}^{max} = 2.7$, and $f_{swap}^{min} = 1.3$ mean that there is still an acceleration. For medium-sized instances with $n = 800$ all speedup factors are already above 10.0. The largest speedups were again observed for the cubic acb-neighborhood with values $f_{acb} = 29$ for $n = 100$ and $f_{acb} \approx 1\,350$ for $n = 800$. Again, because of the high running times of the lexicographic search implementation, we skipped the comparison for the acb-neighborhood for $n > 800$. The conducted experiments also gave remarkable speedups for the string-exchange neighborhood with factors of between $f_{str-exch} = 15.4$ and $f_{str-exch} \approx 575$. This is, again, similar to the results for CVRP and VRPTW.

The results depicted in Figure 8 also indicate that the on-the-fly implementation concept (for $n \geq 1\,250$) benefits more from the sequential search approach than the full representation (for $n \leq 1\,000$) does. The factor caused by the on-the-fly computation is approximately factor 2.0.

Additional Results

Additional results for pickup-and-delivery problems and periodic VRPs can be found in the Online Supplement in the Sections 5.6* and 5.7*.

6. Conclusions

The paper has presented a new modeling framework and corresponding efficient LS methods for VRPs with classical and also non-standard side constraints. One of the most important advantages of the framework is that it is generic and, therefore, allows various types of VRPs to be handled in a similar and concise way. The giant-tour representation is intuitive and enables a unified view on moves, which can either be intra-tour moves or moves between different tours of the same or different depots, periods, vehicle types etc. The unified framework also has advantages from a software development point of view;

once the search procedures of the framework are implemented, additional constraints can easily be integrated, since feasibility is generically encoded by the routing-graph, start-route node and end-route node compatibilities and—most important—resource-constrained paths. Consequently, the framework separates the modeling (with instance-specific data and constraint formulation) from the actual search methods. The addition or change of standard constraints becomes "simply" a question of gathering input data and declaring constraints; it has no implications for the search procedures.

Besides the powerful modeling capabilities of this framework, its main contribution is the incorporation of highly-efficient LS techniques. They allow constant time feasibility tests as well as exact search-tree pruning based on sequential search (Irnich et al., 2006). The extensive computational tests clearly show that sequential search procedures outperform the lexicographic search methods. On large-scale instances and for nearly all types of neighborhoods, the speedup factors are between 10 and 1 000. We observed that the potential of large speedups grows with the size of neighborhoods. Hence, sequential search procedures might become the only efficient technique for implicitly scanning even larger neighborhoods than those traditionally applied to VRPs thus far. We expect that neighborhoods of size $\mathcal{O}(n^k)$ for $k \geq 3$ will be used more often.

One key property of sequential search algorithms for rich VRPs is the separation of the LS procedure into two phases, namely, a preprocessing phase to compute $\mathcal{O}(n^{4/3})$ segment REFs and the actual enumerative search phase. This separation also allows alternative heuristic and exact search-tree pruning techniques for the second phase including, e.g., granular edge selection procedures, as proposed by Toth and Vigo (2003), and methods to terminate the search on the basis of feasibility arguments.

Obviously, the proposed LS techniques can be easily integrated into different metaheuristics, which are substantial for producing high-quality solutions. It was beyond the scope of this paper to also analyze and compare different metaheuristics based on the unified framework. However, different meta-strategies can benefit from the new techniques in the following way: First, methods, such as multi-start and iterated local search, VND, GRASP, directly use LS procedures, see (Hoos and Stützle, 2005). Second, metaheuristics, such as tabu search (Glover and Laguna, 1997), also scan neighborhoods, but ask for best *non-tabu* neighbors. It is straightforward to integrate tabu-constraints into the framework. They will cause no additional worst-case effort for testing neighbor solutions as long as "simple" tabu-criteria and tabu-lists of limited length are used. All of the above metaheuristics will therefore directly benefit from accelerations of LS. Third, some metaheuristics sample from neighborhoods (such as simulated annealing, threshold accepting, and related strategies). For these metaheuristics, our methods do not apply directly. However, extensions of these sampling methods, such as the large-step Markov chain metaheuristic of Martin et al. (1992), aim at finding better quality solutions in each major iteration. Only local optimal solutions are presented to the acceptance algorithm and, hence, efficient LS procedures can speed up the metaheuristic. Other metaheuristics, such as genetic algorithms, evolutionary strategies, or ant systems do not even use neighborhood-based search procedures, at least not in their "pure" versions. However, hybrid versions of these mostly use LS post-processing improvement procedures, which is often the decisive device for designing a highly-effective metaheuristic. Numerous examples are given in the survey of Bräysy and Gendreau (2005b).

For the future, one challenge will be to model new real-world constraints or options and

to integrate them by means of REFs that possess all the properties required for the unified framework. A first formal analysis of conditions for REFs to be invertible and extendable to segments has been given in (Irnich, 2006). Nevertheless, numerous real-world applications, not only those sketched in Section 4, need to be examined in depth. Tailored neighborhoods for special routing applications not considered here (e.g., arc-routing, routing with choice of requests) need to be analyzed and suitable search procedures have to be implemented. A better understanding of the interplay between different start heuristics, neighborhoods, improvement and diversification phases of metaheuristics, considered in various scientific and real-world applications will certainly offer an interesting field for more theoretical and empirical research. Finally, we hope that the unified framework will help researchers and practitioners get a more unified view on modeling and efficient search methods for VRPs.

References

Aarts, E., J.K. Lenstra. 1997. *Local Search in Combinatorial Optimization*. Wiley, Chichester.

Ascheuer, Norbert. 1995. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. Ph.D. thesis, Technische Universität Berlin.

Bellscheidt, L. 2005. Implementierung und Analyse von sequentiellen Suchverfahren für Pickup- und Delivery Probleme. Magister thesis, Deutsche Post Endowed Chair of Optimization of Distribution Networks, RWTH Aachen University. (in German).

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

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

Cardeneo, A. 2005. Modellierung und Optimierung des B2C-Tourenplanungsproblems mit alternativen Lieferorten und -zeiten. Dissertation, Fakultät für Maschinenbau, Universität Karlsruhe (TH), Karlsruhe, Germany. (in German).

Christofides, N., S. Eilon. 1969. An algorithm for the vehicle-dispatching problem. *Operational Research Quarterly* **20** 309–318.

Christofides, N., S. Eilon. 1972. Algorithms for large-scale travelling salesman problems. *Operational Research Quarterly* **23** 511–518.

Cordeau, J.-F., G. Desaulniers, J. Desrosiers, M.M. Solomon, F. Soumis. 2002. VRP with time windows. Toth and Vigo (2002a), chap. 7, 155–194.

Dell'Amico, M., G. Righini, M. Salani. 2006. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transportation Science* **40** 235–247.

Desaulniers, G., J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis, D. Villeneuve. 1998. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. T.G. Crainic, G. Laporte, eds., *Fleet Management and Logistics*, chap. 3. Kluwer Academic Publisher, Boston, Dordrecht, London, 57–93.

Desaulniers, G., J. Desrosiers, M.M. Solomon. 2002. Accelerating strategies in column generation for vehicle routing and crew scheduling problems. Ribeiro and Hansen (2002), chap. 14, 309–324.

Desaulniers, G., J. Desrosiers, M.M. Solomon, eds. 2005. *Column Generation*. Springer.

Desaulniers, G., F. Lessard, A. Hadjar. 2006. Tabu search, generalized k-path inequalities, and partial elementarity for the vehicle routing problem with time windows. Les Cahiers

du GERAD G-2006-45, GERAD, École des Hautes Études Commerciales, Montréal, Canada.

Desrochers, M., J. Desrosiers, M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* **40** 342–354.

Desrochers, M., J.K. Lenstra, M.W.P. Savelsbergh. 1990. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research* **46** 322–332.

Desrosiers, J., Y. Dumas, M.M. Solomon, F. Soumis. 1995. Time constrained routing and scheduling. M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser, eds., *Network Routing*, *Handbooks in Operations Research and Management Science*, vol. 8, chap. 2. Elsevier, Amsterdam, 35–139.

Fukasawa, Ricardo, J. Lysgaard, Marcelo Reis Marcus Poggi de Aragão, Eduardo Uchoa, R.F. Werneck. 2004. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Tech. Rep. Vol.3, No.8, Departamento de Engenharia de Produção, Universidade Federal Fluminense R. Passo da Pátria, Niterói, Brasil.

Funke, B. 2003. Effiziente lokale suche für vehicle routing und scheduling probleme mit ressourcenbeschränkungen. Ph.D. thesis, Fakultät für Wirtschaftswissenschaften, RWTH Aachen, Templergraben 64, 52062 Aachen.

Funke, B., T. Grünert, S. Irnich. 2005a. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics* **11** 267–306.

Funke, B., T. Grünert, S. Irnich. 2005b. A note on single alternating cycle neighborhoods for the TSP. *Journal of Heuristics* **11** 135–146.

Glover, F. 1996. Ejection chains, reference structures and alternating path structures for traveling salesman problems. *Discrete Applied Mathematics* **65** 223–253.

Glover, F., M. Laguna. 1997. *Tabu Search*. Kluwer, Dortrecht.

Golden, B.L., A.A. Assad, eds. 1988. *Vehicle Routing: Methods and Studies*. Elsevier Science, Amsterdam.

Gribkovskaia, I., Ø. Halskau sr, G. Laporte, M. Vlček. 2006. General solutions to the single vehicle routing problem with pickups and deliveries. Technical report, Molde University College, Molde, Norway.

Halse, K. 1992. Modelling and solving complex vehicle routing problems. PhD dissertation No. 60, IMSOR, Technical University of Denmark, Lyngby, Denmark.

Hansen, P., N. Mladenović. 2001. Variable neighborhood search: Principles and applications. *European Journal of Operational Research* **130** 449–467.

Hansen, P., N. Mladenović. 2002. Developments of variable neighborhood search. Ribeiro and Hansen (2002), chap. 19, 415–439.

Hasle, Geir, Arne Løkketangen, Silvano Martello. 2006. Rich models in discrete optimization: Formulation and resolution (ECCO XVI). *European Journal of Operational Research* **175** 1752–1753.

Hempsch, C., S. Irnich. 2007. Vehicle-routing problems with inter-tour resource constraints. Technical Report 2007-01, Deutsche Post Endowed Chair of Optimization of Distribution Networks, RWTH Aachen University, Aachen, Germany. Available at `www.dpor.rwth-aachen.de`.

Homberger, J., H. Gehring. 1999. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *Information Systems and Operations Research* **37** 297–318.

Hoos, H.H., T. Stützle. 2005. *Stochastic Local Search Foundations and Applications*.

Morgan Kaufmann Publishers, Elsevier, San Francisco, CA.

Irnich, S. 2006. Resource extension functions: Properties, inversion, and generalization to segments. Technical Report 2006-01, Deutsche Post Endowed Chair of Optimization of Distribution Networks, RWTH Aachen University, Aachen, Germany. Available at `www.dpor.rwth-aachen.de`.

Irnich, S., G. Desaulniers. 2005. Shortest path problems with resource constraints. Desaulniers et al. (2005), chap. 2, 33–65.

Irnich, S., B. Funke, T. Grünert. 2006. Sequential search and its application to vehicle-routing problems. *Computers & Operations Research* **33** 2405–2429.

Irnich, S., D. Villeneuve. 2006. The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing* **18** 391–406.

Janssens, G.K., R. Hartl, G. Hasle. 2006. Special issue on rich vehicle routing problems. *Central European Journal of Operations Research* **14** 103–104.

Jepsen, M., S. Spoorendonk, B. Petersen, David Pisinger. 2006. A non-robust branch-and-cut-and-price algorithm for the vehicle routing problem with time windows. DIKU Technical Report no. 06/03, Dept. of Computer Science, University of Copenhagen, Copenhagen, Denmark.

Kallehauge, B., J. Larsen, O.B.G Madsen, M.M. Solomon. 2005. Vehicle routing problem with time windows. Desaulniers et al. (2005), chap. 3, 67–98.

Kernighan, B.W., S. Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.* **49** 291–307.

Kilby, P., P. Prosser, P. Shaw. 2000. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints* **5** 389–414.

Kindervater, G.A.P., M.W.P. Savelsbergh. 1997. Vehicle routing: Handling edge exchanges. Aarts and Lenstra (1997), chap. 10, 337–360.

Laporte, G. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* **59** 345–358.

Laporte, G. 1997. Vehicle routing. M. Dell'Amico, F. Maffioli, S. Martello, eds., *Annotated Bibliographics in Combinatorial Optimization*. Wiley, Chichester, 223–240.

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

Lübbecke, M., J. Desrosiers. 2005. Selected topics in column generation. *Operations Research* **53** 1007–1023.

Martin, O., S.W. Otto, E.W. Felten. 1992. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters* **11** 219–224.

Min, H. 1989. The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research* **23** 377–386.

Psaraftis, H.N. 1983. k-Interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Operational Research* **13** 391–402.

Rayward-Smith, V.J., I.H. Osman, C.R. Reeves, G.D. Smith. 1996. *Modern Heuristic Search Methods*. Wiley, Cichester.

Resende, M.G.C., J.P. de Sousa, eds. 2004. *Metaheuristics Computer Decision-Making*. Kluwer Academic Publishers, Boston, Dordrecht, London.

Ribeiro, C.C., P. Hansen, eds. 2002. *Essays and Surveys in Metaheuristics*. Operations Research/Computer Science Interfaces Series, Kluwer, Boston.

Røpke, S., D. Pisinger. 2006. A unified heuristic for a large class of vehicle routing

problems with backhauls. *European Journal of Operational Research* **171** 750–775.

Savelsbergh, M.W.P. 1990. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research* **47** 75–85.

Schrimpf, G., J. Schneider, H. Stamm-Wilbrandt, G. Dueck. 2000. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics* **159** 139–171.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. Tech. rep., Department of Computer Science, University of Strathclyde, Glasgow.

Solomon, M.M. 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research* **35** 254–265.

Toth, P., D. Vigo, eds. 2002a. *The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications*. Society for Industrial and Applied Mathematics, Philadelphia.

Toth, P., D. Vigo. 2002b. VRP with backhauls. Toth and Vigo (2002a), chap. 8, 195–224.

Toth, P., D. Vigo. 2003. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing* **15** 333–346.

Voß, S., D. Woodruff. 2002. *Optimization Software Class Libraries*. Kluwer Academic, Boston.

Type of Constraint/Option	See Ref./Section	Number of Resources	Compatible with Lex. Search	Compatible with Seq. Search	Complexity of Feas. Check
Capacity constraints; CVRP	[REF, §3.2]	1	✓	✓	$\mathcal{O}(1)$
Distance constraints; DCVRP	[REF, §3.2]	1	✓	✓	$\mathcal{O}(1)$
Collection and delivery backhauls; VRPB	4.1, [REF, §2.4.3]	1 (with reset)	✓	✓	$\mathcal{O}(1)$
mixed backhauls; VRPMB	[REF, §2.4.3]	2 (dep.)	✓	✓	$\mathcal{O}(2)$
VRPSDP	4.1, [REF, §2.4.3]	2 (dep.)	✓	✓	$\mathcal{O}(2)$
VRP with lasso tours	[REF, §2.4.3]	2 (dep.)	✓	✓	$\mathcal{O}(2)$
Time window constraints					
single TW	[REF,§2.3]	1	✓	✓	$\mathcal{O}(1)$
TW; no waiting	[REF,§2.3]	2	✓	✓	$\mathcal{O}(2)$
multiple TW	[REF,§2.4.5]	1	✓	✓	$\mathcal{O}(L \cdot T)$
soft TW					
with linear penalty/positive slope (=with linear waiting costs)	[REF, §2.4.2,§4.4]	2	✓	no	$\mathcal{O}(1)$
with general soft TWs	[REF, §2.4.2,§4.4]	2	no	no	$\geq \mathcal{O}(n)$
limited waiting times	[REF,§2.4.4]	3 (2 dep.)	✓	✓	$\mathcal{O}(3)$
limited times on duty	[REF,§2.4.4]	3 (2 dep.)	✓	✓	$\mathcal{O}(3)$
Precedence and pairing					
PDP	4.2	(altern. model)	✓	✓	$\mathcal{O}(1)$ for some neighborhoods
or: PDP	4.2	P	✓	✓	$\mathcal{O}(P)$ for arbitrary neighborhoods and general precedences
only (anti-)pairing	[SPPRC,§3]	P	✓	✓	$\mathcal{O}(P)$
only precedence	[SPPRC,§3]	P	✓	✓	$\mathcal{O}(P)$
Multiple depots					
MDVRP	2.2	–	✓	✓	$\mathcal{O}(1)$, check of $\sim$-relation
tours with individual start and end	2.2	–	✓	✓	$\mathcal{O}(1)$, check of $\sim$-relation
Multiple use of vehicles	[REF,§2.3]	–	✓	✓	$\mathcal{O}(1)$
Multiple compartments	–	C	✓	✓	$\mathcal{O}(C)$

(continued on next page)

(continued from previous page)

Type of Constraint/Option	See Ref./Section	Number of Resources	Compatible with Lex. Search	Compatible with Seq. Search	Complexity of Feas. Check
Heterogeneous fleet					
different capacities	4.6*	1	✓	✓	$\mathcal{O}(1)$
site dependencies	4.4*	$\min\{G, H\}$	✓	✓	$\mathcal{O}(\min\{G, H\})$
different travel times	4.6*	$2 + H$	✓	✓	$\mathcal{O}(H)$
different fixed costs	4.6*	1	✓	✓	$\mathcal{O}(1)$
different costs for vehicles	4.6*	$2 + H$	✓	no	$\mathcal{O}(H)$
Periodic; PVRP	4.7*	-	✓	✓	$\mathcal{O}(1)$
Load incompatibilities	4.4*	G	✓	✓	$\mathcal{O}(C)$
Inter-tour constraints					
assign limited fleet to depots	[Inter§2.4.2]	1	✓	✓	$\mathcal{O}(1)$
ramp assignment	[Inter,§2.4.2]	$1 + D \cdot T$	✓	✓	$\mathcal{O}(D \cdot T)$
staggered arrival/sorting	[Inter]	$2 + D \cdot T$	✓	✓	$\mathcal{O}(D \cdot T)$
limit no. tours with certain property	[Inter,§2.4.2]	2	✓	✓	$\mathcal{O}(2)$
Additional costs					
time-dependent travel times	[REF,§2.4.6,§4.3.3]	1	no	no	?
time-dependent travel costs	–	2	no	no	?
linear waiting costs	[REF,§2.4.2,§4.4]	2	✓	no	$\mathcal{O}(1)$
load-dependent costs					
with polynomial cost fnct.	[REF,§2.4.1,§4.3.1]	2	✓	no	$\mathcal{O}(1)$
general/piecew. linear cost fnct.	[REF,§2.4.1,§4.3.1]	2	no	no	?
REFs with decreasing components					
VRP with synchronization	[REF,§3.1,§3.2]	T	no	no	?
VRP comb. with inventory mgmt.	[REF,§3.1,§3.2]	$I \cdot T$	no	no	?
VRP with split delivery	[REF,§3.1,§3.2]	1	no	no	$\mathcal{O}(1)$

[SPPRC] = (Irnich and Desaulniers, 2005), [REFs] = (Irnich, 2006), [Inter] = (Hempsch and Irnich, 2007)
(dep.)=dependent, (indep.)=independent, C=no. compartments, D=no. depots, G=no. customer groups, H=no. vehicle types, I=no. inventories, L=max. length of a tour, P=no. precedences/pairing constraints, T=num time windows/slices

Table 3: Overview: Types of VRP, Compatibility with the Lexicographic and Sequential Search Approach of the Unified Model, and Complexity of Feasibility Checks