Journal of Heuristics, 11: 267–306, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Local Search for Vehicle Routing and Scheduling Problems: Review and Conceptual Integration

BIRGER FUNKE
TORE GRÜNERT
GTS Systems and Consulting GmbH, Herzogenrath, Germany
email: funke@gts-systems.de
email: gruenert@gts-systems.de

STEFAN IRNICH
Deutsche Post Lehrstuhl für Optimierung von Distributionsnetzwerken, RWTH Aachen University, Aachen, Germany
email: sirnich@or.rwth-aachen.de

Submitted in April 2004 and accepted by Stefan Voss in May 2005 after 2 revisions

Abstract

Local search and local search-based metaheuristics are currently the only available methods for obtaining good solutions to large vehicle routing and scheduling problems. In this paper we provide a review of both classical and modern local search neighborhoods for this class of problems. The intention of this paper is not only to give an overview but to classify and analyze the structure of different neighborhoods. The analysis is based on a formal representation of VRSP solutions given by a unifying giant-tour model. We describe neighborhoods implicitly by a set of transformations called moves and show how moves can be *decomposed* further into partial moves. The search method has to *compose* these partial moves into a complete move in an efficient way. The goal is to find a local best neighbor and to reach a local optimum as quickly as possible. This can be achieved by search methods, which do not scan all neighbor solutions explicitly. Our analysis shows how the properties of the partial moves and the constraints of the VRSP influences the choice of an appropriate search technique.

Key Words: local search, search techniques, vehicle routing and scheduling

The paper gives an overview of *local search* for *vehicle routing and scheduling problems* (VRSPs). This will be achieved by a classification and conceptual integration of different approaches from the literature. We will focus on two major building blocks of local search: the definition of the *neighborhood* and the exploration of the neighborhood using a suitable *search algorithm*. The close relationship between neighborhoods and search methods is exploited by some of the most successful algorithms for VRSPs, such as the famous Lin-Kernighan algorithm for the traveling salesman problem (TSP). Despite the success of these methods, only a small fraction of papers in this area exploit the relationship between neighborhood definition and suitable search algorithms. This can, to a certain degree, be attributed to the difficulty of handling the complex constraints and/or the non-additivity of the objective function in constrained VRSPs as opposed to the TSP. However, algorithms for more complex VRSPs can also benefit from a thorough analysis of the problem, the neighborhood, and the search methods. As pointed out in Cordeau et al. (2002b), one of the major challenges of metaheuristic design is to make them run faster, especially for larger instances. A proper design of the neighborhood and search methods is one way to achieve this goal.

The intention of the paper is to provide a framework of concepts helpful to analyze the structure of different (VRSP) neighborhoods and to build efficient search algorithms. We describe neighborhoods implicitly by a set of transformations called *moves* and show how moves can be *decomposed* further into partial moves. The search method has to *compose* these partial moves into a complete move in an efficient way. The goal is to find a local best neighbor and to reach a local optimum as quickly as possible. This can be achieved by search methods, which do not scan all neighbor solutions explicitly. Our analysis shows how the properties of the partial moves and the constraints of the VRSP influences the choice of an appropriate search technique.

Vehicle routing and scheduling problems require that a fleet of vehicles serves a number of requests in order to minimize costs. We consider the node-routing version of these problems, where requests occur at nodes of a network, and routes can be characterized by sequences of nodes that are visited consecutively by a vehicle. When both the order of the nodes in the routes *and* the determination of arrival and/or departure times have to be considered, one obtains a combined routing and scheduling problem (Bodin et al., 1983). Prominent examples of routing problems are the TSP (Lawler et al., 1985; Gutin and Punnen, 2002), the vehicle routing problem (VRP) (Toth and Vigo, 2002b), and the pickup-and-delivery problem (PDP) (Savelsbergh and Sol, 1985; Desaulniers et al., 2002). When single or multiple time windows occur at the nodes, one obtains the TSP, VRP or PDP with time windows, abbreviated as TSPTW, VRPTW and PDPTW, respectively (e.g., Desrosiers et al.; 1995, 1998; Cordeau et al., 2002a; Bräysy and Gendreau, 2005a, 2005b). The scope of this paper is not limited to these problems but also encompasses the more general class of routing problems that can be modeled with resource variables (Desaulniers et al., 1998; Funke, 2003; Irnich and Desaulniers, 2004).

Solving these types of problems optimally is still a challenge in the case of larger instances. With the exception of the TSP, where instances with several thousand nodes can be solved to optimality on a regular basis (Gutin and Punnen, 2002), instances of the other problems with more than about fifty nodes can be intractably hard to solve optimally. Therefore, heuristics and metaheuristics are used for solving larger instances of these problems.

In this paper, we address both classical and newly developed methods for local search in the context of VRSPs. These approaches are classified according to the structure and size of the neighborhood and the search methods that are employed to search the neighborhood. Roughly speaking, the quality of the solutions increases with the size of the neighborhood. However, large neighborhoods may be too big to search exhaustively. This is why methods for intelligently *pruning* the neighborhood search are so important. Intelligent pruning allows local search methods to scan potentially larger neighborhoods without exceeding the requirement for limited computational time.

The paper is organized as follows: In Section 1, we shortly introduce local search and the important concept of moves and move decompositions. A formalism for the representation of these concepts in the context of vehicle routing and scheduling is provided and the decomposition of the objective function is discussed. In Section 2, we classify the different neighborhoods that have been suggested in the literature and discuss their similarities and differences. If possible, an estimation of the neighborhood size and possible move decompositions are given. Section 3 introduces different search techniques. We provide generic descriptions of the approaches and give examples of their possible applications. We cover both direct search techniques, which are used within classical approaches of local search as well as indirect search approaches that have been introduced in the context of very large-scale neighborhood search. We finally give some conclusions and outline some promising paths for future research.

1. Local search

Nearly all heuristics and metaheuristics for VRSPs rely heavily on the definition of neighborhood solutions and local search (LS). This section introduces LS applied to VRSPs. We present the basic notation and concepts of local search algorithms for combinatorial optimization problems, clarify the definition of moves and their decomposition into partial moves, discuss possibilities to represent VRSP solutions in the context of LS, and formalize the concept of gains of (partial) moves for different VRSP cost functions.

Let (X, c) be a combinatorial optimization problem of the form $\min_{x \in X} c(x)$, where X is a finite but large set of *feasible solutions* and $c(x)$ the cost of x. For instance, X is the set of tours in the TSP and $c(x)$ is the cost of the tour $x \in X$. The basis of all local search methods is the use of a set of *elementary moves* that transform a given solution $x \in X$ into a different, so-called, *neighbor solution* x'. The set of all solutions that can be reached from the current solution using the set of moves is called the *neighborhood* of the current solution w.r.t. the move set, i.e., $\mathcal{N}(x) \subset X$. An $x' \in \mathcal{N}(x)$ with the property $c(x') < c(x)$ is an *improving neighbor solution*. Feasible solutions $x \in X$, which do not have any improving neighbor solutions, i.e., $c(x') \geq c(x)$ for all $x' \in \mathcal{N}(x)$, are called *local optima* w.r.t. $\mathcal{N}$. In every iteration of a local search method, some (and in the worst case all) neighbors $x' \in \mathcal{N}(x)$ of the current solution x are evaluated in order to find at least one improving neighbor solution. If it exists, a move is made to the improving neighbor solution, which then becomes the current solution. Otherwise, a local optimum w.r.t. the current neighborhood $\mathcal{N}$ is found and the local search stops.

1.1. The local search algorithm

Denoting the current iteration counter by t and the current solution by $x^t \in X$, one obtains the following generic description of a local search method:

Algorithm 1: Generic Local Search

```
1: Initialize the algorithm with a feasible solution $x^0 \in X$ and set the iteration counter
   $t := 0$.
2: REPEAT
3:   Search for an improving neighbor $x'$ in the neighborhood $\mathcal{N}(x^t)$ of the current solution
$x^t$.
```

```
4 :   IF there exists an improving neighbor solution $x' \in \mathcal{N}(x^t)$,
4 :      THEN set $x^{t+1} := x'$ and $t := t + 1$.
5 : UNTIL no more improvements are found.
```

Several remarks regarding the design of Algorithm 1 should be made. First, the speed of the neighborhood evaluation depends on the computational effort to determine the cost $c(x')$ of a neighbor solution x' and checking its feasibility. If this can be achieved quickly, e.g., in constant time for each neighbor $x' \in \mathcal{N}(x)$, larger neighborhoods can be searched (e.g., Aarts and Lenstra, 1997, p. 128ff; Kernighan and Lin, 1970).

Second, search step 3 gives the flexibility of terminating the search whenever *one improving neighbor* from the set of all improving neighbor solutions has been found. If the search method is enumerative (i.e., all neighbor solutions $x' \in \mathcal{N}(x)$ and their cost $c(x')$ are evaluated one after another), taking *the first improving solution* or taking a *best improving solution* are two extreme strategies known as *first search* and *best search*. Another well known strategy, called *d-best search*, consists of terminating the search when d improving neighbor solutions are found. Then, the best solution from this set is taken as the next solution. All these strategies try to take advantage of the inherent trade-off between searching more thoroughly within a single neighborhood or searching more quickly in several neighborhoods with smaller incremental improvements. From the worst-case point of view, all search strategies are equivalent, since showing that the last x^t is a local optimal solution requires the entire neighborhood $\mathcal{N}(x^t)$ to be scanned.

Third, we would like to emphasize that finding a *best* improving solution in a given neighborhood is itself an optimization problem. This optimization problem can be solved by explicit enumeration techniques *or* by suitable optimization algorithms. These include techniques, such as dynamic programming and branch-and-bound or network optimization algorithms for shortest paths or cycles, matchings, etc. In all cases, heuristics can be employed to speed up the search.

Finally, the neighborhood $\mathcal{N}$ can be dynamic, i.e., using different neigborhoods $\mathcal{N}^t$, depending on the iteration t and the search history. Dynamic neighorhoods are a core concept of metaheuristics, but are beyond the scope of this paper. We refer the interested reader to the books of Aarts and Lenstra (1997); Gambardella, Taillard, and Agazzi (1999); Voß et al. (1999).

1.2. Moves and their decomposition

For a precise definition of the term *move*, it is helpful to consider an enclosing superset of *solutions* $Z \supseteq X$. The idea of a solution $y \in Z$ is that some of the moves $m \in M$ might transform a feasible solution x into an object $y = m(x)$, which has a structure similar to a feasible solution, but does not necessarily satisfy all constraints that define feasible solutions. For instance, shifting a node from one position to another position in a TSPTW tour transforms one tour x into another tour x', but might violate several time window constraints. Another example is the swap of two customers between two VRP tours, which might violate a capacity constraint. In general, we denote by M the *set of moves* where a move $m \in M$ is a (possibly partial) map from Z to itself, i.e., $m : Z \rightarrow Z$. Since a move might not be applicable to all solutions $x \in Z$, $m(x)$ is not always well-defined. From the above discussion, it is clear that a move m does *not* necessarily map feasible solutions $x \in X$ into feasible solutions. For a given $x \in Z$, the *extended neighborhood* $\hat{\mathcal{N}}(x) = \{m(x) : m \in M\}$ contains all neighbors of x, either feasible or infeasible. Clearly, the neighborhood $\mathcal{N}(x) \subset X$ is given by $\mathcal{N}(x) = \hat{\mathcal{N}}(x) \cap X$. Every move, $m \in M$, with $m(x) \in X$ is called a *feasible move* w.r.t. x.

The number of neighbor solutions of a given solution x is called *the size of the neighborhood*. When all moves $m \in M$ generate different neighbor solutions $m(x) \in \hat{\mathcal{N}}(x)$, the size of $\hat{\mathcal{N}}(x)$ can easily determined by counting the elements of M. Since the number $|\mathcal{N}(x)|$ of *feasible* elements in the neighborhood is (in general) depending on x, we will mostly examine sizes of extended neighborhoods $\hat{\mathcal{N}}(x)$.

In order to analyze different moves, we decompose them into smaller parts, the so-called *partial moves*. A given decomposition $m = p_l \circ \ldots \circ p_2 \circ p_1$ of a move m into $l \geq 2$ partial moves $p_1, p_2, \ldots, p_l$ means that an $x \in Z$ is first transformed into $p_1(x)$, second $p_1(x)$ is transformed into $p_2(p_1(x))$, and so on. Of course, we have to consider the structures that occur after having applied one or several partial moves. In general, the ith partial move transforms elements of an intermediate structure Y_{i-1} to elements of another intermediate structure Y_i, while for the first and last structure $Y_0 = Y_l = Z$ holds. As a result, $m : Z \to Z$ decomposes into

$$m : \quad Z = Y_0 \overset{p_1}{\to} Y_1 \overset{p_2}{\to} Y_2 \overset{p_3}{\to} \cdots \overset{p_{l-1}}{\to} Y_{l-1} \overset{p_l}{\to} Y_l = Z.$$

We neither claim that the decomposition into partial moves is self-evident nor that it is unique. Nevertheless, there are some interesting cases that we would like to study. In the case that the intermediate structures are taken from a single set Y differing from Z (i.e., $Y = Y_1 = Y_2 = \cdots = Y_{l-1}$ and $Y \neq Z$), we call p_1 an *opening partial move* and p_l a *closing partial move*, while all other *intermediate partial moves* map Y into itself. Such intermediate structures are called *reference structures*. Whenever the intermediate structures have an identical form, it is possible to vary the number of intermediate partial moves, which results in chains of intermediate partial moves of variable length. As we will see later, the famous Lin-Kernighan neighborhood as well as the ejection-chain neighborhoods are constructed in this way. A second important case is when Z coincides with all intermediate structures. We will refer to *composite moves* whenever $X = Y = Z$. This implies that a composite move consists of a chain of partial moves such that every partial move maps from one reference structure to an identical reference structure.

1.3. Representation of VRSP solutions

In order to describe the neighborhoods formally, a concise representation of VRSP solutions is needed. The basis for such a description is a directed routing graph $G = (V, A)$. The node set V consists of request nodes $R \subset V$ and possibly route-start $O \subset V$ and route-end nodes $D \subset V$. The interpretation of the request nodes depends on the problem at hand. In the case of the VRP, every request node corresponds to a customer that has to be visited exactly once. In the PDP, a request node is either a pickup or a delivery. In complex routing applications, a request may even consist of more than a pair of pickup and delivery nodes. In the context of vehicle scheduling (e.g., the multi-depot vehicle scheduling problem), one request node corresponds to an operation starting at a given location and ending at another location. This operation has to be performed without overlapping other operations at a predefined time or within a given time window.

Solutions of VRSP, which involve more than a single vehicle can be represented as a collection of routes. Such a route is a path in G, starting with a route-start node $o \in O$ and ending with a *compatible* route-end node $d \in D$, visiting a sequence of request nodes in between. 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 of vehicles, all $o \in O$ and $d \in D$ are compatible. In multi-depot problems the sets O and D are partitioned according to the n_D depots/garages, e.g., $O = O^1 \dot\cup \cdots \dot\cup O^{n_D}$, $D = D^1 \dot\cup \cdots \dot\cup D^{n_D}$, and $o \in O^k$, $d \in D^l$ are compatible if and only if $k = l$. In general we assume that O and D have the same cardinality, $H = |O| = |D|$, and that compatible route-start and route-end nodes are defined by a relation$\sim$, i.e., a subset of $O \times D$.

A solution to a VRSP is called a *route plan.* A route plan $x = (r^1, r^2, \dots, r^H)$ is an H-tuple of paths in G where each node $v \in V$ is covered exactly once, each path r^i starts with a route-start node $o^i \in O$ and ends with a compatible route-end node $d^i \in D$. Note that this definition implies that every route-start and route-end node occurs in exactly one route. If o^i and d^i are connected directly, then the corresponding vehicle travels directly from its origin to its destination, e.g., from and to its depot. In some applications the possibility of not using a vehicle is modeled exactly in this way. We will denote the number of nodes in a route plan by $n = |V|$ and call its elements r^i *routes*. The nodes covered by route r^i are denoted by $V(r^i)$.

Note that the order of the routes in such a representation is arbitrary, since any permutation of the routes represents the same solution. Alternatively, one might think of solutions to VRSPs as sets of routes (instead of tuples). We do not consider this option mainly for two reasons: First, in a software implementation, one has to chose an ordering. Second, ordering the routes gives rise to two "natural" representations. The *giant route* is the path $(r^1, r^2, \dots, r^H)$ in which each route-end node d^i is connected to the next route-start node o^{i+1} (for $i = 1, 2, \dots, H - 1$). Similarly, the *giant tour* is the cycle $(r^1, r^2, \dots, r^H)$ in which, additionally, d^H is connected to o^1. 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 VRSPs. 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 with four routes, departing from two depots.

1.4. Constraints

Up to now nothing has been said about the *constraints* of VRSPs and modeling *feasibility*. Most practically relevant constraints can be modeled by so-called *resource variables* (Desaulniers et al., 1998; Funke, 2003; Irnich and Desaulniers, 2004). These resource variables may be part of intra-route or inter-route constraints. Intra-route constraints pertain

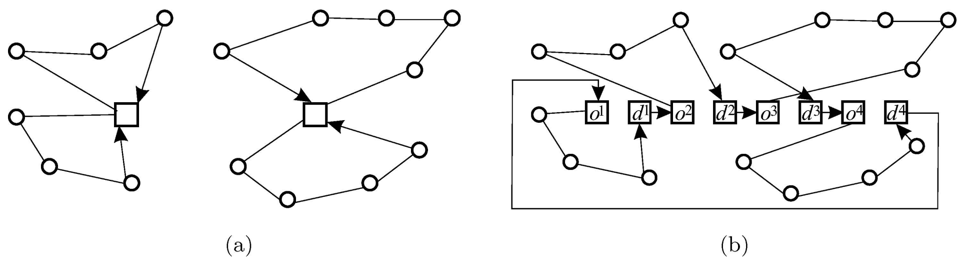

Figure 1. Giant-tour representation.

to the resource variables of a single route whereas inter-route constraints are defined for several, interdependent routes. Well-known examples of *intra-route constraints* are vehicle capacities, tour length restrictions, precedences, and time window constraints. Examples of *inter-route constraints* are a limited number of ‘long’ tours, sorting capacity constraints in parcel or letter delivery systems, and a restricted number of docking stations at depots. Consider, for example, a depot where vehicles arrive over time. If the number of vehicles, which can be served in a certain interval is bounded, then the feasibility of a route plan depends on the arrival time of all vehicles, which visit the depot. We know of no systematic heuristic approaches in the literature for handling inter-route constraints, although they can be very important in some applications. However, since the literature on this is scarce, we will focus on intra-route constraints in the following.

A formal description of constrained resources can be easily explained by the example of time windows. The *beginning of service times* T_i, $i \in V$, are given for every node i, and the vehicle schedule can be described entirely by giving these times at all nodes. If the node o^i is the route-start node, then T_{o^i} is the *departure time*. In most scheduling problems, single or multiple time windows $[a_i^\ell, b_i^\ell]$, $\ell = 1, \dots, L(i)$ are given for all nodes $i \in V$ of the network. Every interval $[a_i^\ell, b_i^\ell]$ defines feasible beginning of service times for node i. If a vehicle arrives within such a time window, the schedule is feasible. If it arrives before the first or between two time windows, it has to wait until the beginning of (the second) time window. If it arrives later than the end of the last time window $L(i)$, the schedule is infeasible. When a vehicle moves from node i to node j, the resource variable *time* increases by at least $t_{ij} + s_i$, which is the travel time along arc $(i, j) \in A$ plus the service time at node i. Thus, choosing arc (i, j) within a route plan implies $T_j - T_i \geq t_{ij} + s_i$.

There are more general resource variable concepts, e.g., for modeling time or load dependent travel times, multiple capacities, etc. Resource extension functions (Desaulniers et al., 1998; Irnich and Desaulniers, 2004) are useful for this type of model extensions, but their description and discussion is beyond the focus of this paper.

In the following it will be sufficient to reduce the question of feasibility of a route plan to an *oracle* $\hat{o}$, i.e., a function $\hat{o} : Z \to \{yes, no\}$. The neighborhood $\mathcal{N}(x)$ of a route plan x consists of all feasible route plans and can be computed by $\mathcal{N}(x) = \{m(x) : m \in M, \hat{o}(m(x)) = yes\}$. Nevertheless, we have to keep in mind that the effort of calling the oracle has a strong impact on the efficiency of the overall search algorithm. In successful LS implementations, the oracle and the search strategy are carefully adapted to each other.

1.5. Costs and gains of (partial) moves

In this subsection we study how the objective function of the VRSP *and* the decomposition of a move into partial moves *both* influence the ability to allocate costs or gains to partial moves. Such an allocation is attractive within search algorithms, since one may be able to show that the move cannot lead to an improvement before all partial moves have been applied.

Recall that $c(x)$ is the cost of a solution $x \in Z$. We denote the *gain* of move $m \in M$ applied to solution $x \in Z$ by $g(m, x) := c(x) - c(m(x))$. For VRSPs costs are related to the edges chosen in the solution x. We assume that the overall cost is the sum of the costs $c_{ij}(x)$ associated with the edges of the giant-tour representation, i.e.,

$$c(x) = \sum_{(i,j)\in\text{giant tour}(x)} c_{ij}(x).$$

The following three cases apply to nearly all practically relevant applications:

1. *Edge-dependent* costs $c_{ij}(x) = c_{ij}$ for each $(i, j) \in A$. This is the simplest case where costs depend only on the edge under consideration. In this case gains can be computed fast and immediately when edges are added or removed. Edge-dependent costs cover the situation of standard TSP, VRP and PDP.
2. *Vehicle-dependent* costs $c_{ij}(x) = c_{ij}^{od}$, which depend on the route-start node o and route-end node d of the route that contains the edge (i, j). The respective gains of (partial) moves can only be determined when the assignment of request nodes to routes (i.e., to (o, d)-combinations) has been made. The concept of vehicle-dependent costs is sufficient to model, for example, heterogeneous fleet and multi-depot problems.
3. *Resource-dependent* costs $c_{ij}(x) = c_{ij}(T_i^1, \dots, T_i^p, T_j^1, \dots, T_j^p)$ where $T_i^1, \dots, T_i^p$ and $T_j^1, \dots, T_j^p$ are resource variables of nodes $i \in V$ and $j \in V$. Time-dependent and load-dependent costs can be modeled in this way (Desaulniers et al., 1998). Resource-dependent costs cover the most general cases of intra-route constraints, but delay the cost computation of some/all routes to a point when the entire new route plan $x' = m(x)$ has been constructed. An example is the computation of the waiting costs of a route in problems with time windows, which depend on the departure time of all visited nodes (Desaulniers and Villeneuve, 2000).

The classification of a VRSP according to one of the three cases determines the possibility of allocating gains to partial moves at certain stages of the search procedure.

In order to implement efficient pruning rules in LS, it is necessary to allocate a gain $g(p_i, x)$ to each of the partial moves p_i, $i = 1, \dots, l$, depending only on the current solution x, but not on any intermediate solution. Let the move $m \in M$ be decomposed into partial moves $p_l \circ \cdots \circ p_2 \circ p_1$. For the gain functions it is desirable that the gain of a move m is the sum of the gains of its partial moves $p_1, \dots, p_l$. A decomposition $m = p_l \circ \cdots \circ p_2 \circ p_1$

is called *cost independent* if

$$g(m, x) = \sum_{i=1}^{l} g(p_i, x) \tag{1}$$

holds. If the equality is not fulfilled in all cases, then sufficient conditions which guarantee (1) can be given. These are called *legitimacy conditions*, cf. Glover (1996a). For example, legitimacy conditions might require that only compatible subsets of partial moves occur simultaneously or restrict the ordering of partial moves.

For the case $Y = Z$, the decomposition $m = p_l \circ \cdots \circ p_2 \circ p_1$ is *order-independent* if

$$m(x) = p_{\pi(l)} \circ \cdots \circ p_{\pi(2)} \circ p_{\pi(1)}(x)$$

holds for all solutions $x \in Z$ and all permutations π of $\{1, 2, \ldots, l\}$. It is called *cyclic independent* if the same holds for cyclic permutations π only.

It should be pointed out that even if the *exact* evaluation of a partial move has to be delayed when cost independence is not fulfilled, approximations or lower bounds may be used for pruning the search process.

2. Neighborhood types

For the description of different neighborhood types, we assume that VRSP solutions x are given by their giant-tour representation. Then, each solution $x \in Z$ can be transformed into any other solution $x' \in Z$ of a VRSP by deleting and adding a number of edges. Hence, every local search method for VRSPs could, in principle, be regarded as a special variant of an edge exchange. On the other hand, some transformations can be described better by considering nodes. Typical examples are the relocation of one or several nodes from one route to another or from their current positions to different positions within the same route.

We will use the term *edge exchange* when the number of directly involved nodes is larger than the number of directly involved edges, and we use the term *node exchange* when the opposite is true. Note that we refer to "edges" both for directed and undirected graphs, since this is very common in the vehicle routing literature.

In cases where the number of directly involved nodes and edges is approximately the same, the classification of a move as a node or an edge exchange may not be clear.

A third category of neighborhoods are the *combinatorial neighborhoods*. In these neighborhoods, the set of all feasible moves must correspond with the feasible region of some combinatorial optimization problem. Searching for the best improving neighbor is, therefore, equivalent to solving the associated combinatorial optimization problem. Examples of combinatorial optimization problems are the assignment problem, the shortest path problem and some of its extensions, and network flow problems. Furthermore, in some cases the decision variables of the combinatorial optimization problem are in one-to-one correspondence with appropriately defined partial moves. The constraints of the combinatorial optimization problems guarantee that only compatible partial moves are selected in order to produce new feasible solutions.

The fourth and last category of neighborhoods we consider are the *partially constructive neighborhoods*. In these neighborhoods, a number of nodes is removed from the current solution and re-inserted using some optimization algorithm or heuristic.

In the following, the major neighborhood structures for VRSPs are described, including the notation, the cardinality of the neighborhood, the decomposition of moves into partial moves (if interesting), and references to the original papers.

2.1. *Node-exchange neighborhoods*

In this paper, we will use the α^*-notation for describing the node-exchange neighborhoods. This notation is motivated by the (M, P)-notation suggested by Taillard (1993) and the cyclic transfer notation suggested by Thompson and Psaraftis (1993). It is based on the route plan definition given above. Let us assume that we are given ℓ disjoint segments (e.g., ℓ different routes) $s_1, \ldots, s_\ell$ of the route plan x. Denote by $|s_i|$ the number of nodes in segment s_i. Then a node exchange can be described by the ℓ-dimensional vector $\alpha = (\alpha_1, \ldots, \alpha_\ell)$, $\alpha_i \in \mathbb{N}, i = 1, \ldots, \ell$ where $\min\{\alpha_i, |s_i|\}$ nodes are moved from path s_i to path s_{i+1} if $i < \ell$ and from s_ℓ to s_1 if $i = \ell$. In many cases it is assumed that the nodes from path s_i take positions of the nodes removed from path s_{i+1}. If this is the case, we write $\underline{\alpha}_i$ instead of α_i. If it is allowed that less than $\min\{\alpha_i, |s_i|\}$ nodes are moved from s_i to s_{i+1}, then we write α_i^* instead of α_i.

The concept of node exchanges is not limited to those routing problems where requests can be described entirely by one node (such as in the case of the TSP and VRP). However, most implementations so far consider exactly this case. If one wants to handle the more general case, it is usually required that all nodes belonging to one request are moved simultaneously (e.g., the pickup and delivery node in the case of the PDP). The following subsections illustrate different node-exchange neighborhoods and the α^*-notation.

Note that the α^*-notation is not normalized w.r.t. cyclic shifts, e.g., the neighborhood given by (a, b, c) is identical to (b, c, a) and (c, a, b). In the following, we will give only one of these possible descriptions.

2.1.1. Relocation. Relocation, also called insertion, is the simplest and most basic node exchange. One node is moved from its current position and inserted into a different position. In α^*-notation this can be described by $(1, 0)$. The size of the relocation neighborhood is $\mathcal{O}(n^2)$, since there are n possible nodes that can be moved from their current position to $(n-2)$ other positions.

The relocation move m^{reloc} can be decomposed into two partial moves. In order to simplify the notation, we assume that the predecessor and successor of a node i are $i-1$ and $i+1$, respectively. First, a node i is removed from its current segment, which means removing the edges $(i-1, i)$ and $(i, i+1)$ and inserting the edge $(i-1, i+1)$. Now, node i is free (i.e., not connected to the giant tour) and, therefore, this partial move is called p_i^{free}. Second, the free node i is inserted into a second segment between consecutive nodes j and $j+1$. Consequently, the second partial move is denoted by $p_{i,j}^{\text{ins}}$, which removes the edge $(j, j+1)$ and inserts the edges (j, i) and $(i, j+1)$. The intermediate structure Y into (from) which

p_i^{free} ($p_{i,j}^{\text{ins}}$) maps, is the set

$$Y = Y^{\text{one-free-node}} = \{(i, \hat{x}) : i \in R, \hat{x} \text{ is a (feasible) route plan for } V \setminus \{i\}\}. \tag{2}$$

Of course, the composition $m_{ij}^{\text{reloc}} := p_{ij}^{\text{ins}} \circ p_i^{\text{free}}$ is only well-defined if $j \neq i$ and $j \neq i - 1$. For edge-dependent costs, the gain of the partial moves are $g(p_i^{\text{free}}, x) = c_{i-1,i} + c_{i,i+1} - c_{i-1,i+1}$ and $g(p_{ij}^{\text{ins}}, x) = c_{j,j+1} - c_{ji} - c_{i,j+1}$. The partial moves p_i^{free} and p_{ij}^{ins} are cost independent in the case of edge-dependent costs. The gain of the relocation move is $g(m_{ij}^{\text{reloc}}, x) = g(p_i^{\text{free}}, x) + g(p_{ij}^{\text{ins}}, x)$.

2.1.2. Exchange. In an *exchange move*, one node is moved from the first path to the second path and a second node is moved vice-versa. Some authors use the term interchange instead of exchange. We will only use the term exchange in order to avoid confusion with the λ-interchange move. The corresponding α^*-notation is (1, 1). Clearly, each (1, 1)-exchange move can be represented as the composition of two (partial) relocation moves, i.e., $m_{i_1,j_1,i_2,j_2}^{\text{exchange}} = m_{i_1,j_1}^{\text{reloc}} \circ m_{i_2,j_2}^{\text{reloc}}$. The size of the (1, 1)-exchange neighborhood $\hat{\mathcal{N}}(x)$ is $\mathcal{O}(n^4)$, since it is the combination of two relocation moves. For the exchange move it is easy to verify that its partial moves $m_{i_1,j_1}^{\text{reloc}}$ and $m_{i_2,j_2}^{\text{reloc}}$ are cost- and permutation-independent (in case of edge dependent costs), if the ten nodes $\{i_1 - 1, i_1, i_1 + 1, j_1, j_1 + 1, i_2 - 1, i_2, i_2 + 1, j_2, j_2 + 1\}$ are pairwise disjoint (these are sufficient legitimacy conditions).

In many implementations presented in the literature it is required that the nodes take the position of their counterparts, i.e., $(\underline{1},\underline{1})$ in α^*-notation. This is also called a *swap* move. A swap of nodes i and j is given by $m_{ij}^{\text{swap}} = m_{i,j-1}^{\text{reloc}} \circ m_{j,i-1}^{\text{reloc}}$ which is a decomposition into two *dependent* partial moves. Note that one can restrict swap moves to the case where $j \neq i - 1, i, i + 1$, since otherwise the resulting move is either the identity or a relocation move. The size of the neighborhood reduces to $\mathcal{O}(n^2)$ in the case of a swap. As we will see later when discussing efficient search techniques for the swap move, there are other independent decompositions into partial moves.

2.1.3. λ-Interchange. The term λ-interchange was introduced by Osman (1993). It is a generalization of the exchange move described above. In a λ-interchange one moves at most λ nodes from one segment to another and vice-versa, i.e., (λ^*, λ^*) in α^*-notation. In most implementations, also in the original paper of Osman, it is required that the nodes are inserted in the positions of the removed nodes, i.e., $(\underline{\lambda}^*, \underline{\lambda}^*)$ in α^*-notation. Note that the λ-interchange for $\lambda = 2$ includes the moves (1, 0), (1, 1), (2, 0), (2, 1) and (2, 2). In order to evaluate the size of the λ-interchange neighborhood for (λ^*, λ^*), we first consider the possible choices to remove $2k$ nodes ($k \leq \lambda$). There are $\binom{n}{2k}$ possibilities to select $2k$ different nodes. After having selected $2k$ nodes, there are $\mathcal{O}(n^{2k})$ (for k significantly smaller than n) possibilities to re-insert them into the segments. This gives a neighborhood size of $\mathcal{O}(\sum_{k=0}^{\lambda} \binom{n}{2k} n^{2k}) = \mathcal{O}(\binom{n}{2\lambda} n^{2\lambda}) = \mathcal{O}(n^{4\lambda})$. In the case of $(\underline{\lambda}^*, \underline{\lambda}^*)$, we still have $\binom{n}{2k}$ possibilities to select $2k$ different nodes. However, there are only $(2k)!$ (which is constant for small k) ways to re-insert the nodes, resulting in a neighborhood size of $\mathcal{O}(n^{2\lambda})$. We are only aware of papers where $(\underline{1}^*, \underline{1}^*)$ has been used. Note that in general, λ-interchange

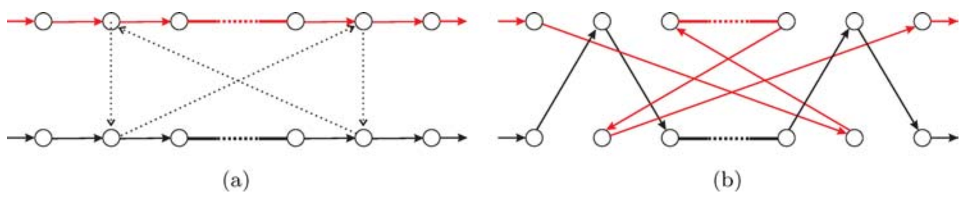

Figure 2. A 2-Interchange move that cannot be composed of exchange moves, (a) Before the move, dotted lines indicate which nodes are exchanged, (b) After the move.

moves cannot be represented by a composition of exchange moves. Figure 2 depicts a 2-interchange move that cannot be represented by exchange moves, since the involved nodes are not swapped pair-wise.

2.1.4. Node-ejection chains. Ejection chains are a powerful concept introduced by Glover for solving various combinatorial optimization problems (Glover, 1992, 1996a; Glover and Laguna, 1997). Ejection chains have been used to solve the TSP and VRP. Here we describe the use of node-ejection chains (NEC) in the context of the VRP, as a generalization of the concept implemented by Rego (1998).

In the first step of a NEC, a node i_1 is removed from its current position and inserted into another position, currently taken by another node i_2. In the second step, node i_2 is again removed from its current position and inserted to yet another node position i_3, etc. Rego considers two possibilities for performing the last move of such an ejection chain. In the first case, termed 'multi-node exchange process' (MNEP), the last ejected node is inserted into the position left empty by the first removed node. In the second case, termed 'multi-node insert process' (MNIP), the last node is inserted into another position without ejecting a node. In α^*-notation a NEC is given by $\alpha = (\alpha_1, \ldots, \alpha_\ell) = (\underline{1}, \ldots, \underline{1})$ for MNEP and by $\alpha = (\alpha_1, \ldots, \alpha_\ell) = (\underline{1}, \ldots, \underline{1}, 0)$ for MNIP. The number ℓ is the *depth* of the NEC. In the following we will consider MNEP only, since MNIP can easily be modelled as an *exchange* process by using an additional dummy node.

The NEC move decomposes naturally into alternating sequences of partial moves $p^{\text{free}}_{i_j}$ and $p^{\text{ins}}_{i_j,i_{j+1}}$ for $j = 1, \ldots, \ell - 1$, i.e.,

$$m^{\text{NEC}}_{i_1,\ldots,i_\ell}: \quad Z \xrightarrow{p^{\text{free}}_{i_1}} Y \xrightarrow{p^{\text{ins}}_{i_1,i_2}} Z \xrightarrow{p^{\text{free}}_{i_2}} Y \xrightarrow{p^{\text{ins}}_{i_2,i_3}} Z \xrightarrow{p^{\text{free}}_{i_3}} \cdots \xrightarrow{p^{\text{free}}_{i_\ell}} Y \xrightarrow{p^{\text{ins}}_{i_\ell,i_1}} Z. \tag{3}$$

Again, $Y = Y^{\text{one-node-free}}$ is the structure defined by (2). We assume that all nodes $i_1, i_2, \ldots, i_\ell$ are different.

In order to provide a basis for the description of appropriate LS procedures that will be presented in Section 3.1.2, we are now giving a more detailed analysis of the partial moves and their dependencies. We assume edge-dependent costs. At first we point out that the move $m^{\text{NEC}}_{i_1,\ldots,i_\ell}$ given by (3) is a cyclic shift of the ℓ nodes $i_1, i_2, \ldots, i_\ell$. The corresponding 2ℓ partial moves are cyclic independent as can be seen from figure 3. $m^{\text{NEC}}_{(i_1,\ldots,i_\ell)}$ moves are in one-to-one correspondence with cycles of partial moves (CPM). Neighborhood search (i.e.,

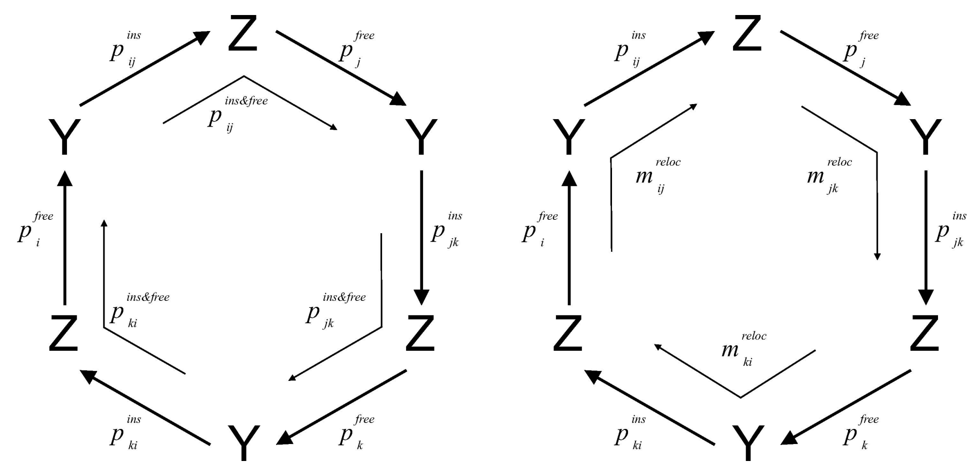

Figure 3. Partial moves in a Node-Ejection Chain $m^{\text{NEC}}_{(i,j,k)}$ with $l = 3$.

searching for moves with certain properties, e.g., improving moves) is, therefore, equivalent to searching for CPM with corresponding properties.

From Section 2.1.1 we know that p^{free}_i and its succeeding partial move p^{ins}_{ij} are cost independent (if $j \neq i-1, i, i+1$). In contrast, p^{ins}_{ij} and its successor p^{free}_j are *not* cost independent. To see this, note that the composition $p^{\text{ins\&free}}_{ij} := p^{\text{free}}_j \circ p^{\text{ins}}_{ij}$ adds the edges $(j-1, i), (i, j+1)$ and removes the edges $(j-1, j), (j, j+1)$ and, therefore, the gain is equal to $g(p^{\text{ins\&free}}_{ij}, x) = c_{j-1,j} + c_{j,j+1} - c_{j-1,i} - c_{i,j+1}$. This does not coincide with the sum of the gains of the partial moves p^{free}_j and p^{ins}_{ij}. As a result, the above decomposition of move $m^{\text{NEC}}_{(i_1,\ldots,i_\ell)}$ into 2ℓ partial moves is not cost independent.

However, it is possible to decompose a CPM into cost independent parts. With the above given gain $g(p^{\text{ins\&free}}_{ij}, x)$ it is easy to see that $p^{\text{ins\&free}}_{i_1,i_\ell} \circ p^{\text{ins\&free}}_{i_\ell,i_{\ell-1}} \circ \cdots \circ p^{\text{ins\&free}}_{i_2,i_3} \circ p^{\text{ins\&free}}_{i_1,i_2}$ is such a decomposition if the following legitimacy conditions are fulfilled. If a node i ejects node j, then in the remaining part of the NEC-move it is prohibited to move node i, the new predecessor $j-1$ or successor $j+1$. This implies that the nodes $i_1, \ldots, i_\ell$ have to be chosen such that $i_k \notin \{i_j - 1, i_j, i_j + 1\}$ holds for all $k \neq j$. From the cost independency it follows that finding improving moves m^{NEC} is equivalent to detecting CPM with a positive overall gain $\sum_{j=1}^{\ell} g(p^{\text{ins\&free}}_{i_j,i_{j+1}}, x)$. The point is that the atoms $p^{\text{ins\&free}}_{i_j,i_{j+1}}$ of this decomposition can be priced independently.

Next we determine the size of the NEC neighborhood. It follows from the legitimacy conditions that every ejection of a node j blocks the two nodes $j-1$ and $j+1$ from being ejected in subsequent partial moves. Thus, the maximum depth of a NEC is $\ell^{\max} = \lfloor \frac{n}{2} \rfloor$. For a given depth $\ell \in \{1, \ldots, \ell^{\max}\}$, there are $n(n-2)\cdots(n-2\ell+2) = \mathcal{O}(2^\ell \binom{\lfloor \frac{n}{2} \rfloor}{\ell})$ possibilities for selecting the ℓ nodes in the NEC. The order of these nodes can be permuted. Hence, the total neighborhood size is $\mathcal{O}(\sum_{\ell=1}^{\lfloor \frac{n}{2} \rfloor} 2^\ell \binom{\lfloor \frac{n}{2} \rfloor}{\ell} (\ell - 1)!)$.

2.1.5. Cyclic transfers. The concept of cyclic transfers (CTs), introduced in Thompson and Psaraftis (1993) is very similar to that of an ejection chain. Generally, in a cyclic transfer, m nodes are moved from route r^1 to route r^2, from route r^2 to route r^3 etc., until m nodes are moved from route r^b to route r^1. Such a general move is called a 'b-cyclic m-transfer'. In α^*-notation it is given by $\alpha = (\alpha_1, \dots, \alpha_b) = (m, \dots, m)$. In the original paper it is also allowed to move less than m nodes from one route to the next using so-called 'dummy demands'. In α^*-notation this is the move $\alpha = (\alpha_1, \dots, \alpha_b) = (m^*, \dots, m^*)$. A single CT move can be described by the b sets of nodes which are cyclicly shifted among their routes. Let S_i be the set of nodes which are removed from route r^i and inserted into route r^{i+1} (for abbreviation we set $r^{b+1} := r^1$ and $S_{b+1} := S_1$). Analogous to the NEC neighborhood, the cyclic-transfer move $m^{CT}_{(S_1,S_2,\dots,S_b)}$ can be decomposed into $2b$ partial moves $p^{\text{free}}_{S_i}$ and $p^{\text{ins}}_{S_i,S_{i+1}}$:

$$m^{CT}_{(S_1,S_2,\dots,S_b)}: \quad Z \xrightarrow{p^{\text{free}}_{S_1}} Y \xrightarrow{p^{\text{ins}}_{S_1,S_2}} Z \xrightarrow{p^{\text{free}}_{S_2}} Y \xrightarrow{p^{\text{ins}}_{S_2,S_3}} Z \xrightarrow{p^{\text{free}}_{S_3}} \cdots \xrightarrow{p^{\text{free}}_{S_b}} Y \xrightarrow{p^{\text{ins}}_{S_b,S_1}} Z.$$

The operation $p^{\text{free}}_{S_i}$ removes the nodes S_i from their current positions in the route r^i, while $p^{\text{ins}}_{S_i,S_{i+1}}$ inserts the nodes S_i into the route r^{i+1} under the assumption that the nodes S_{i+1} will be removed from r^{i+1} in the following partial move. The intermediate structure Y consists of pairs $(S, \hat{x})$, where S is a subset of nodes and $\hat{x}$ is a route plan on $V \backslash S$.

In contrast to the NECs described above, CT neighborhoods do not require that the inserted nodes take the place of the removed nodes. Rather, a new route $r^{i+1'}$ has to be determined which now includes the nodes $(V(r^{i+1}) \cup S_i) \backslash S_{i+1}$. For example, in the case of the VRP(TW), one would have to solve a TSP(TW) to generate r^{i+1}. Since solving these subproblems to optimality is, in general, too time-consuming, one usually resorts to an insertion heuristic.

In the following, we assume that the costs $c(x)$ of a route plan can be computed as the sum of costs $c(r^i)$ of the single routes. When inter-route resources (see p. 273) influence the cost of the route-plan this assumption might be violated. As in the case of NEC, $m^{CT}_{S_1,\dots,S_b}$ decomposes into b cost independent partial moves $p^{\text{ins\&free}}_{S_i,S_{i+1}} := p^{\text{free}}_{S_i} \circ p^{\text{ins}}_{S_i,S_{i+1}}$, $i = 1, \dots, b$. The gain of $p^{\text{ins\&free}}_{S_i,S_{i+1}}$ is $g(p^{\text{ins\&free}}_{S_i,S_{i+1}}, x) = c(r^{i+1}) - c(r^{i+1'})$ where $c(r^{i+1'})$ is the cost of the new route $r^{i+1'}$. If there does not exist a feasible route $r^{i+1'}$ for the node set $(V(r^{i+1}) \cup S_i) \backslash S_{i+1}$ (or no feasible route is found), then one sets $g(m^{CT}_{S_i,S_{i+1}}, x) = -\infty$.

CT neighborhoods are more general than NEC neighborhoods in the sense that several nodes might be shifted simultaneously within a single partial move, and the ordering of the nodes in the new route might be completely different from their previous ordering. On the other hand, CT require that node sets S_i, S_j of a move m^{CT} come from different routes. This assumption is not necessary in the general NEC approach. We will see later that this 'different route legitimacy condition' is very useful when the neighborhood is searched for improving solutions, since it allows intra-route constraints to be handled implicitly when solving a dynamic program or shortest path problem. Thus, routing problems with complex intra-route constraints can be handled using this approach.

In order to bound the size of the CT neighborhood, we assume that all H routes have approximately the length n/H. The number of different b-cyclic m-transfers is $\mathcal{O}\left(\binom{H}{b} \cdot 2^{bn/H}\right)$, since there are $\binom{H}{b}$ possibilities to choose the b routes and each route r^i has approximately

$2^{n/H}$ subsets S_i. Hence, the CT neighborhood is larger than the neighborhood of the NEC with the different routes legitimacy condition.

2.2. Edge-exchange neighborhoods

Edge-exchange neighborhoods are the most commonly used neighborhoods in heuristics for VRSPs. They can be described by a process of subsequently removing and re-inserting edges.

2.2.1. General description. Removing k different edges $d_1, \dots, d_k \in A$ from the giant tour x creates exactly k subpaths $(s_1, s_2, \dots, s_k)$ which are called *segments* in the context of edge exchanges. In order to construct a neighbor giant tour $x' = m(x)$ from these segments one has to add the same number k of edges, denoted by $a_1, a_2 \dots, a_k \in A$. (More generally, one can create a union of edge disjoint cycles which is then transformed back into a single tour.) The added edges have to connect pairs of nodes incident to at least one of the removed edges, i.e., start-nodes or end-nodes of the different segments. Each node is incident to the same number of removed and added edges. Therefore, the spanning graph $G(d_1, \dots, d_k, a_1, \dots, a_k)$ can be decomposed into *alternating cycles* of removed and added edges.

Deleting the edges $d_1, \dots, d_k$ and inserting new edges $a_1, \dots, a_k$ can be interpreted with respect to the following two aspects: First, when an edge a_i connects a segment s with another segment s', then these segments become predecessor and successor segments, respectively. Consequently, the inserted edges determine the *permutation* of the segments. Second, connecting one segment s with the first (last) node of segment s' implies that the second segment has to be traversed in its given (its reverse) order. Hence, the inserted edges also determine whether segments are *inverted* or not.

Therefore, each edge-exchange move m which deletes k and adds k edges can be considered as the subsequent execution of the following four operations:

1. k**-segmentation.** This operation removes k edges from the giant tour x resulting in k segments $s_1, \dots, s_k$.
2. k**-inversion.** This operation inverts a subset of the k segments, i.e., $s_i^{\pm 1}$.
3. k**-permutation.** This operation changes the order of the segments $s_{\pi(1)}^{\pm 1}, \dots, s_{\pi(k)}^{\pm 1}$.
4. k**-concatenation.** This operator concatenates the segments that result from applying the three first operators into a new giant tour x'.

When all inserted edges differ from the deleted edges, the move is called a *proper move*.

There are some details that we would like to point out. Any permutation π of the indices $\{1, 2, \dots, k\}$ can be represented in cycle-notation, e.g., as $(1, 3, 4)(2, 5)$ when the permutation maps 1 to 3, 3 to 4, 4 to 1, and exchanges 2 and 5. The result of applying π to $s_1^{\pm 1}, \dots, s_k^{\pm 1}$ and concatenating the permuted segments is a cycle (giant tour), if and only if π is a cyclic permutation. Non-cyclic permutations π transform x into multiple cycles. These multiple cycles do not necessarily correspond with feasible route plans. However, as we will see in Section 2.2.3, some of these structures are easily re-interpretable as cycles and might, therefore, generate "interesting" giant tours x'.

The set of moves, for which the operators inversion and permutation are identical, define a *move type* or *edge-exchange type*. In order to give a clear description of different edge-exchange types we introduce the *abc*-notation. In this notation, one uses the first k letters of the alphabet, where the ith letter corresponds to the ith segment of the giant tour. If the letter is uppercase, the corresponding segment is reversed. For example, the code $aBDc$ corresponds to moves where the second and the fourth segments are reversed and the third and fourth segments change position. To indicate that a new cycle starts we write "|", i.e., $aC|bd$ means that the first and the reversed third segment form one cycle while the second and the forth segments form the second non-reverse cycle (for the moment we do not care about the question of how to transform these two cycles back to a single cycle).

Next, we want the abc-notation to be unambiguous, i.e., the notation should not depend on the numbering of the segments within the giant tour. Let us assume that the segment "a" is fixed by an appropriate definition, e.g., stating that segment "a" includes the node with index 1. Any cyclic permutation π can be uniquely written with "a" as the *first* element of the cycle. This means that the cyclic permutation (a, c, d, b) *cannot* be written as (c, d, b, a) or (d, b, a, c) or (b, a, c, d). From this argument it follows that we have to distinguish $(k-1)!$ different cyclic permutations.

In the case of multiple cycles, the abc-notation first presents the cycle which contains segment a, followed by the cycle with the smallest remaining segment number (according to the letters in the alphabet), etc. For instance, instead of $aDf|c|eB$ one has to write $aDf|Be|c$.

Furthermore, in symmetric problems a giant tour x' and the reversed giant tour x'^{-1} represent identical solutions, e.g., Ab is identical to aB. In contrast, asymmetric problems have to take the orientation of the first segment a into account. In the following description of edge-exchange moves, we will focus on the symmetric case only. The results remain valid for the asymmetric case if we keep in mind the factor two for inverting the first segment.

Assignment of route-start and route-end nodes and inversion of segments. Recall that every route starts with a route-start node and ends with a route-end node. Any move that affects at least two routes may disrupt this configuration. The simplest case is the application of the so-called 2-opt move to two routes, an instance of the so-called cross exchange. This move is depicted in figure 4. The problem with this situation is that the feasible assignment

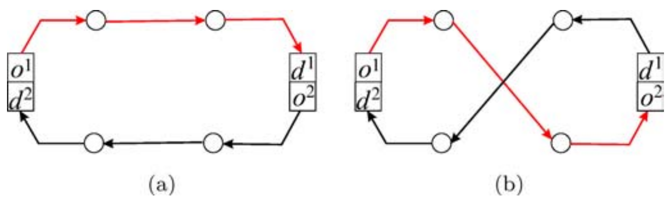

Figure 4. A cross move that disrupts the structure of the route, starting with a route-start node and terminating with a route-end node, (a) Before and (b) After the exchange.

of route-start and route-end nodes is disrupted by the move. Therefore, a new assignment of route-start and route-end nodes to the routes is necessary. However, this problem is an optimization problem by itself. Even in the case of two routes, there might be four possibilities (if all route-start and route-end nodes are compatible, i.e., $o^1 \sim d^1$, $o^1 \sim d^2$, $o^2 \sim d^1$ and $o^2 \sim d^2$ holds) of assigning a combination to the routes. A closer investigation of this problem shows that in general it can be modeled as a planar three-dimensional assignment problem (Magos and Miliotis, 1994).

In order to save the effort of solving an optimization problem, there is a simple way to implement the inversion of a segment s. First, if s solely consists of request nodes, then the the segment can be inverted directly. Second, if s contains a single (o, d)-pair, i.e., $s = (v_1, v_2, \dots, v_p, d, o, w_1, w_2, \dots, w_q)$, then the inverted segment should be defined as $s^{-1} = (w_q, \dots, w_2, w_1, d, o, v_p, \dots, v_2, v_1)$. This implies that after permutation and concatenation, the new giant tour still consists of routes starting (ending) with nodes $o \in O$ $(d \in D)$ including some request nodes. Third, if the segment s contains one or several routes $r^{h(i)}, r^{h(i)+1}, \dots, r^{h(j)}$, i.e., $s = (v_1, v_2, \dots, v_p, d, r^{h(i)}, r^{h(i)+1}, \dots, r^{h(j)}, o, w_1, w_2, \dots, w_q)$, then the inverted segment should be $s^{-1} = (w_q, \dots, w_2, w_1, d, r^{h(i)}, r^{h(i)+1}, \dots, r^{h(j)}, o, v_p, \dots, v_2, v_1)$, which means that the intermediate routes should not be inverted. The reason is that the *order of the single routes* in the giant tour is arbitrary, and a complete inversion of all included routes is 'more likely' to be infeasible if the problem at hand is asymmetric.

2.2.2. k-Opt. The oldest and most widely used neighborhood is the so-called k-opt neighborhood (see, e.g., Croes, 1958; Lin, 1965). In these neighborhoods, $k \geq 2$ edges are removed from the giant tour and k edges are inserted, resulting in a different route plan. The defining characteristic of k-opt moves is that the permutation π of the segments is a *cyclic* permutation. Although k-opt moves are usually employed within single-route problems, such as the TSP and TSPTW, they generalize to multiple-route problems in the giant-tour representation.

When deleted and inserted edges are not disjoint (an edge is first removed and later the same edge is added back to the tour (*delete-add*), or an edge is added and the same edge is deleted later (*add-delete*)) one gets a *non-proper* move which can also be found as a k'-opt move with $k' < k$.

For all $k \geq 2$, the size of the k-opt neighborhood can be derived from the above definition. There are $\binom{n}{k}$ possibilities to chose the edges for segmentation, 2^{k-1} possibilities for performing an inversion, and $(k-1)!$ possibilities for doing a cyclic permutation. The size of the neighborhood is, therefore, bounded by $\binom{n}{k}2^{k-1}(k-1)!$. This is an upper bound, since the segmentation may result in degenerated segments, which cannot be inverted. For $k \geq 3$, we have also included all neighborhoods from $2, \dots, k-1$ in the calculation. On the other hand, the number of move types for a given k is exactly $MT(k) := 2^{k-1}(k-1)!$, since the two operations k-inversion and k-permutation generate exactly this number of moves. The number of 2-opt and 3-opt move types is $MT(2) = 2$ and $MT(3) = 8$, respectively. The two 2-opt move types are the identity (which is a *non*-proper move) and the move where the edges $(i, i+1)$ and $(j, j+1)$ are replaced by (i, j) and $(i+1, j+1)$ and the segment between the nodes $i+1$ and j is inverted. The eight 3-opt move types depicted

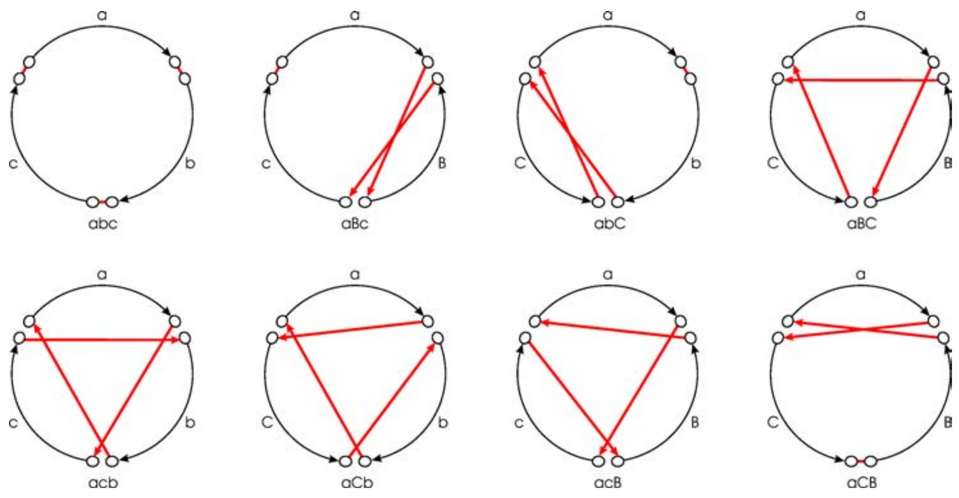

Figure 5. The eight 3-Opt move types.

in figure 5 contain one identical tour, three 2-opt move types, and only four proper 3-opt move types. Let us denote the number of proper move types for a given k by $PMT(k)$. We know that $PMT(2) = 1$ and $PMT(3) = 4$. For an arbitrary k, the value of $PMT(k)$ can be computed as follows. After having chosen the k edges to remove, we have to subtract from $MT(k)$ the number of proper moves that can be generated with fewer than k edges. For a given k and a given $i < k$, there are $\binom{k}{i}$ possibilities to select subsets with exactly i edges. Moreover, the identity move, where all k edges are re-inserted after deletion, also has to be substracted from the number of non-proper moves. This results in the recursive formula $PMT(k) = MT(k) - \sum_{i=2}^{k-1} \binom{k}{i} PMT(i) - 1, k \geq 3$ and $PMT(2) = 1$ for the number of proper k-opt moves. Table 1 depicts the values of $MT(k)$, $PMT(k)$ and $PMT(k)/MT(k)$ for selected values of k. This shows that the ratio between proper k-opt move types and the k-opt move types lies between 0.5 and 0.6 for 'relevant' values of k. It should also be noted that the ratio increases monotonically with k.

We would also like to point out that the restriction of considering only proper k-opt move types does not exclude the possibility of creating moves that can be composed of several k'-opt moves with $k' < k$ having all deleted and added edges disjoint. This occurs for the

Table 1. Growth of (proper) move types for k-opt moves.

k	2	3	4	5	6	7	8	9	10	50	100
$MT(k)$	2	8	48	384	3840	46080	645120	1.03E + 07	1.86E + 08	3.42E + 77	5.92E + 185
$PMT(k)$	1	4	25	208	2121	25828	365457	5.90E + 06	1.07E + 08	2.06E + 77	3.57E + 185
$\frac{PMT(k)}{MT(k)}$	0.5	0.5	0.52	0.54	0.55	0.56	0.57	0.57	0.57	0.6	0.6

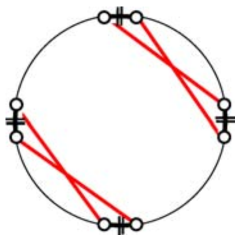

Figure 6. A proper 4-opt move type which can be composed of two independent 2-opt move types.

first time when $k = 4$. For example, the proper 4-opt move type $aBcD$ can be composed of the two independent 2-opt move types aB and cD, as can be seen from figure 6.

Or-opt. The only 3-opt move which does not reverse any segments is the move acb. This is an instance of the so-called *Or-opt* move (Or, 1976), where one usually requires that the number of nodes within at least one of the segments b and c is less than or equal to three. Therefore, the size of the Or-opt neighborhood is *not* cubic, but quadratic. Such restrictions on path lengths are a general way to speed up the serach within edge-exchange neighborhoods.

2.2.3. k-Opt*. A generalization of the k-opt moves for multiple-route problems was introduced by Potvin et al. (1989). In contrast to the k-opt neighborhood, the permutation defining the k-opt* move is *not necessarily a cyclic permutation* π. For example, $aE|Bc|d$ represents a move type which generates three subtours (cycles). The first one consists of segment a and the reversed segment e, the second one connects the reversed segment b with c, and the last one is formed by the single segment d. The k-opt* neighborhood includes the k-opt neighborhood.

The formation of a subtour is feasible if and only if the cycle represents a giant tour of the nodes it covers. This means that compatible pairs $o \sim d$ of tour-start and tour-end nodes have to form individual routes of the subtour. Furthermore, each cycle has to include at least one combination of route-start and route-end nodes. Figure 7 depicts such a replacement operation for a configuration resulting from a 2-opt* move.

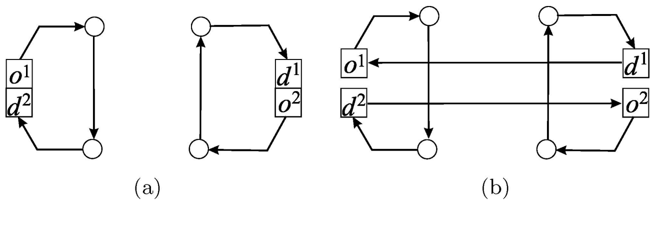

Figure 7. Transformation of the intermediate structure, (a) that results from the application of a 2-opt* move to (b) an equivalent giant tour.

The transformation of multiple cycles into one giant tour (which represents the route plan) is done by concatenating the individual routes in a cyclic way. The final ordering of these routes is arbitrary.

The size of the k-opt* neighborhood can be computed in analogy to the size of the k-opt neighborhood. The only difference is that here π is an arbitrary permutation. Thus, the size of the k-opt* neighborhood is bounded by $\binom{n}{k}2^{k-1}k!$, which is (approximately) k times the size of the k-opt neighborhood.

2.2.4. Special inter-route neighborhoods. Most heuristics and metaheuristics for VRSPs use a set of restricted k-opt operators that are presumably very efficient in this context. The advantage of using restricted k-opt operators is that the effort of searching is usually lower than the $\mathcal{O}(n^k)$-effort needed for searching the entire k-opt neighborhood. Most of these operators are structured so that no paths need to be reversed.

Cross exchange. A cross exchange is identical to applying a 2-opt or 2-opt* move to two routes. The case of applying the 2-opt* move corresponds to swapping the end segments between two routes. In most applications where the order of the nodes is important, cross exchanges are limited to this case.

String relocation. A segment, i.e., a 'string', is removed from one route and inserted into a different route. This is equivalent to applying the Or-opt move to two routes. If the length of the string is restricted, then the Or-opt neighborhood is quadratic.

String exchange Two strings from two different routes are exchanged. This is equivalent to applying the well-known 'double-bridge' move of the TSP (an 'adcb'-move) to two different routes. Due to different terminologies, this move is sometimes also referred to as a cross exchange. If the length of the two segments is restricted, then the string-exchange neighborhood can be searched in quadratic time. Without such restrictions, the size of the string-exchange neighborhood is bounded by $\mathcal{O}(n^4)$. Note that the cross exchange is equivalent to a special case of the string exchange in the single-depot case, where an endpoint of both of the strings is connected directly to the depot.

2.2.5. Lin-Kernighan neighborhood. Recall that in an edge-exchange move m, the added edges are denoted by $a_1, a_2, \ldots, a_k$, the deleted edges by $d_1, d_2, \ldots, d_k$, and that all these edges together implicitly define a set of alternating cycles. The alternating cycles completely determine the move m because they consist of the symmetric difference of the edges from the current giant tour x and the new giant tour x'. By SAC-k-opt we denote all k-opt moves whose deleted and added edges generate a *single alternating cycle* Funke, Grünert, and Irnich (2004). All 2-opt and 3-opt moves are contained in the SAC-2-opt and SAC-3-opt neighborhoods, respectively. However, it is well-known that the double-bridge move *acbd* is not contained in the SAC-4-opt neighborhood (see, e.g., Rego and Glover, 2002). Therefore, the SAC-k-opt neighborhood is a proper subset of the k-opt neighborhood. The Lin-Kernighan neighborhood (LK), introduced in Lin and Kernighan (1973), is a proper subset of SAC-k-opt, which further restricts the way edges can be added and deleted in the alternating cycle. It requires that the process of subsequent deletion and addition of edges satisfies some structural properties and that all added and deleted edges are disjoint. The corresponding moves have the property of decomposing naturally into cyclic independent partial moves (see p. 275). Lin and Kernighan have formulated the *gain criterion* for general cyclic independent moves, which makes the search in the TSP-case highly efficient (see Section 3.1.2).

In order to give a formal description of the neighborhood and its decomposition, we have to label the nodes that are incident with the added and deleted edges. Tracing along the alternating cycle shows that there exist $2k$ (not necessarily different) nodes $i_1, i_2, \dots, i_k$ and $j_1, j_2, \dots, j_k$ such that the deleted edges can be written as $d_p = (i_p, j_p)$ and the added edges can be written as $a_p = (j_p, i_{p+1})$ for all $p = 1, \dots, k$ (with the setting $i_{k+1} := i_1$). The move m is determined by the nodes $i_1, i_2, \dots, i_k$ and $j_1, j_2, \dots, j_k$ of the alternating cycle. The partial move p_{ij}^{del} deletes the edge (i, j) from the current tour (or more generally, from a given graph) and the partial move p_{ji}^{add} adds the edge (j, i). Using this notation and introducing the intermediate structures Z and Y, which will be explained below, the cyclic edge-exchange move m can be written as:

$$m^{LK}_{\substack{i_1,\dots,i_k,\\ j_1,\dots,j_k}} : \quad Z \overset{p^{\text{del}}_{i_1,j_1}}{\rightarrow} Y \overset{p^{\text{add}}_{j_1,i_2}}{\rightarrow} Z \overset{p^{\text{del}}_{i_2,j_2}}{\rightarrow} Y^* \overset{p^{\text{add}}_{j_2,i_3}}{\rightarrow} Z \overset{p^{\text{del}}_{i_3,j_3}}{\rightarrow} \cdots \overset{p^{\text{del}}_{i_k,j_k}}{\rightarrow} Y \overset{p^{\text{add}}_{j_k,i_1}}{\rightarrow} Z. \tag{4}$$

LK is a restricted neighborhood with moves m^{LK} of variable length, where the move always *starts with the deletion* of an edge. The alternative of starting with the addition of an edge leads to a so-called *edge-ejection chain* and has been studied by Glover (1992). LK restricts the steps of the decomposition (4) to the case where after the deletion of an edge, the generated structure is a *Hamiltonian path* y, i.e., Y is the set of all Hamiltonian paths. The intermediate objects $z \in Z$ are co-called *stem-and-cycles*. They result from the addition of an edge from an endpoint of a Hamiltonian path to another node. The stem-and-cycle reference structure Z was introduced in Glover (1996a, 1992).

Since all edges are pairwise disjoint, the LK neighborhood only contains moves that exchange exactly k edges with k other edges. Hence, LK is a subset of the neighborhood defined by the *proper* k-opt moves (see p. 281).

From our point of view, the above decomposition is the kernel of the famous Lin and Kernighan neighborhood. Its *variability* relies on the fact that at each add-step one can either add the edge (j_p, i_1), which transforms the current path into a (giant) tour (i.e., a closing partial move), or one may add a “short” edge (j_p, i_{p+1}) with $i_{p+1} \neq i_1$, which continues the alternating delete-add-process. The second alternative allows the generation of edge exchanges with variable length.

In Lin and Kernighan (1973), the authors allow the second deletion of an edge to either create a Hamiltonian path or a path with a single cycle. This slightly changes the LK-neighborhood into a larger neighborhood LK*. The motivation for this modification according to Lin and Kernighan (1973, p. 506) is that LK* contains all 3-opt moves, while LK does not. More specifically, the Or-opt move (type acb) is not contained in the LK neighborhood, but belongs to LK*. A comparison of neighborhoods of the Lin-Kernighan type and ejection-chain moves proposed in Glover (1991, 1996a) can be found in Funke, Grünert, and Irnich (2004).

2.3. *Combinatorial neighborhoods*

Large neighborhoods have the potential to contain more and better solutions than small neigborhoods. Ahuja et al. (1999) stress that large-scale neighborhoods do not necessarily produce more effective heuristics unless one can search the larger neighborhood in a very efficient manner. In this subsection, we focus on large neighborhoods for VRSP which can be searched effectively by a suitable algorithm, based on, e.g., dynamic programming or a matching approach. We call these neighborhoods *combinatorial neighborhoods (CN).*

2.3.1. Assignment-based neighborhoods. In the last few years, several researchers have considered the so-called *assignment neighborhood*, (see, e.g., Deĭneko and Woeginger, 2000). We give a VRSP specific presentation of the ideas here.

The first step of the procedure is to split the current giant tour x into $2k$ segments of the form $x = (f_1, e_1, f_2, e_2, \dots, f_k, e_k)$, i.e., a $2k$-segmentation. Subsequently one removes the segments $e_1, \dots, e_k$ from their current positions, permutes and reinserts them between the segments $f_1 \dots, f_k$, while keeping their ordering fixed. This yields a new giant tour $x' = (f_1, e_{\pi(1)}, f_2, e_{\pi(2)}, \dots, f_k, e_{\pi(k)})$. While the removed segments e_i might be empty, the fixed segments f_i have to include at least one node. The reason for this is that one wants to compute an insertion cost c_{ij} for inserting the removed segment e_i between the segments f_j and f_{j+1} (with $f_{k+1} := f_1$). Computing such cost coefficients c_{ij} is trivial in the case of edge-dependent costs and also applicable for vehicle-dependent costs.

Finding a minimum cost insertion of all removed segments amounts to solving an assignment problem on $(c_{ij})_{1 \le i,j \le k}$. The assignment problem is the least-cost selection of k different partial moves $p^{\text{assign}}_{i_t, j_t}$, $t = 1, \dots, k$ where the partial move $p^{\text{assign}}_{i,j}$ inserts segment e_i between f_j and f_{j+1}. Partial moves with different indices $i_t \neq i_s$ and $j_t \neq j_s$ for $t \neq s$ are cost independent and permutation-independent, i.e., fulfill the legitimacy condition.

The assignment neighborhood has been suggested in the context of the TSP with all segments consisting of a single node, i.e., $k = \lceil n/2 \rceil$ and possibly one empty segment. It means that every second node of the TSP-tour can be arbitrarily permuted. This choice of segments does *not* automatically generalize to other more complex routing problems, since one cannot model those constraints in the assignment framework. However, assignment neighborhoods are applicable to those VRSP where the feasibility of a route plan can be determined by considering all routes independently. Then, the splitting of the current giant tour into $2k$ segments has to be performed in a restricted way.

2.3.2. Partial order neighborhoods. Another type of combinatorial neighborhood is the one suggested by Balas and Simonetti (2001). Here, the current route plan is re-optimized under the condition that the change of the *relative position* of each node within the route does not exceed a certain pre-specified integer value. This condition implies a partial order on the nodes of the route plan. In Balas (1999) it was shown that in the case of the TSP, this problem can be solved efficiently by a shortest path calculation. However, when more complex constraints, such as time windows, have to be taken into account, the labeling procedures suffer from the well-known problem of a large number of undominated labels. In Balas and Simonetti (2001) it was suggested to alleviate this problem by heuristically limiting the number of labels kept at each node. For an analysis of the neighborhood size, see the original papers.

2.3.3. Pyramidal neighborhoods. Assume that the nodes in the current route plan are numbered consecutively from 1 to n. Then, any pyramidal neighbor of the current route plan is a route plan where the indices of the nodes first increase from 1 to n and then decrease again. For example, two pyramidal neighbors of the route plan $(1, 2, 3, 4, 5, 6, 7, 8, 9, 1)$ are $(1, 3, 4, 5, 9, 8, 7, 6, 2, 1)$ and $(1, 3, 4, 6, 7, 8, 9, 5, 2, 1)$. Cost optimal pyramidal neighbors can be found in $\mathcal{O}(n^2)$ using dynamic programming or shortest path computations (see, e.g., Gilmore, Lawler, and Shmoys, 1985). The size of the neighborhood is $\mathcal{O}(2^{n-1})$ (Carlier and Villon, 1990).

One drawback of the pyramidal neighborhood is that nodes 1 and 2 and nodes $n-1$ and n are directly connected by the edges $(1, 2)$ or $(2, 1)$ resp. $(n-1, n)$ or $(n, n-1)$. Therefore, Carlier and Villon (1990) suggested rotating the indices of the current route plan n times and to use every node once as the starting point of a cyclic numbering.

So far, the pyramidal neighborhood has only been used in the context of the TSP. However, as we will show below, this neighborhood can be generalized to take into account all types of intra-route constraints that can be modeled as resources, such as time windows and precedences.

2.3.4. Route-first cluster-second. Whereas all the other combinatorial neighborhoods mainly modify the *routing* of the current route plan, this neighborhood focuses on the *clustering* aspect of the routing problem. The route-first cluster-second approach of Beasley (1983) was developed as a constructive heuristic to solve VRSPs. Here we describe how the approach can be used within a local search context. Recently, a simplified version of this neighborhood has been employed within a Genetic Algorithm for the VRP by Prins (2003).

Suppose that all route-start and route-end nodes have been removed from the current route plan. This partitions the route plan into H segments, $p^1, \dots, p^H$. The route-first cluster-second neighborhood consists of all solutions that can be obtained by connecting the segments to a giant route and subsequently partitioning this giant route into H new routes. Note that we allow any number of routes to be empty. Consider, for example, the three routes $p_1 = (o^1, 2, 4, 7, d^1)$, $p_2 = (o^2, 3, 1, 9, 6, d^2)$ and $p_3 = (o^3, 5, 8, d^3)$. One neighbor solution can be found by first considering the permutation (p_2, p_1, p_3) that results in the giant route $(3, 1, 9, 6, 2, 4, 7, 5, 8)$ and then clustering this giant route into the three new routes $p'_1 = (o^1, 3, 1, 9, d^1)$, $p'_2 = (o^2, 6, 2, d^2)$ and $p'_3 = (o^3, 4, 7, 5, 8, d^3)$. Section 3.2.1 will explain how to determine a least cost clustering by solving a (possibly constrained) shortest path problem.

The size of this neighborhood can be computed as follows: There are $(H-1)!$ different cyclic permutations of the segments that result in a giant route. After patching the routes into a giant route and deleting the route-start and route-end nodes, $n-2H$ nodes remain. For a given giant route, any clustering with H routes can be described entirely by the set of first nodes that partition the giant route, e.g., the nodes 3, 6, 4 in the example above. Note that, for a given value of H, the clustering may lead to a solution containing between one and H new routes. Denote the number of new routes by $1 \leq \ell \leq H$. For a fixed value of ℓ, the number of clusters is $\binom{n-2H}{l}$. Thus, the total number of clusters is $\sum_{\ell=1}^{H} \binom{n-2H}{l}$, which is $\mathcal{O}(\binom{n-2H}{H})$. After having determined the clusters, it remains to assign pairs of compatible route-start and route-end nodes. There are $H!^2$ possibilities to perform such an assignment. Taking into account the $(H - 1)!$ possibilities of constructing the giant route, we obtain $\mathcal{O}(\binom{n-2H}{H}(H!)^3)$ as an estimate of the size of the neighborhood. Since usually the number of routes is much smaller than the number of nodes, we obtain the estimate $\mathcal{O}(n^H H!^3)$.

2.3.5. Special graph structures. For some routing problems, such as the TSP, optimal solutions can be found when the underlying routing graph has a special structure. The most famous of these graph structures are the so-called *Halin Graphs* (see Cornuéjols, Naddef, and Pulleyblank, 1983 for the case of TSP). Since optimal solutions for these problems can be found efficiently in $\mathcal{O}(n)$, a neighborhood of the current tour can be defined by adding some edges to the tour (a so-called *extension*) so that the resulting graph has the desired property and then considering all tours within the resulting graph as the neighborhood.

A similar approach is discussed by Glover and Punnen (1994). However, they do not solve the problem optimally over the specially defined subgraph but rather give algorithms that dominate an exponential-size set of tours w.r.t. their graph construction.

We do not consider these types of neighborhood in this paper since they are usually restricted to specialized problems such as the TSP and cannot be readily generalized to more complex VRSPs.

2.4. Partially destructive/constructive neighborhoods

Partially destructive/constructive neighborhoods (DCNs) can be characterized by a two-step process. In the first step, a number of nodes or segments are removed from the route plan. Then, in a second step, the removed nodes are inserted into the route plan using some type of constructive algorithm. This algorithm can be either an optimization algorithm, such as dynamic programming or branch-and-bound, or any heuristic, for example, an insertion method.

These types of neighborhoods do not fit nicely into the local search framework defined above, since they depend on the algorithm used for the construction of the new solution after the removal of the nodes. However, they are often used within a local search framework and we have, therefore, decided to include them in the survey.

In order to distinguish different types of DCNs, we suggest the following string $\alpha|\beta|\gamma$ notation: The parameter α gives the number of nodes that are removed from the route plan. If these nodes have to be consecutive within the current route plan, we write $\underline{\alpha}$. The second parameter β describes how the remaining disconnected route plan is treated. If $\beta = connect$, then the predecessor of the removed node is connected to the successor of the removed node. If $\beta = open$, then the segments that result after deletion of the nodes are left open. If $\beta = alg$, then the remaining segments of the route plan are re-connected using the algorithm alg. For example, if the segments are re-connected using a nearest neighbor algorithm, then we have $\beta = NN$. The third parameter γ describes how the removed nodes are re-inserted into the route plan. This is typically achieved using some insertion or optimization algorithm, such as branch-and-bound or shortest path calculations. If, for example, a cheapest insertion strategy is used, then we have $\gamma = CI$.

Without going into the details of this type of neighborhoods, we would like to mention several typical examples of destructive/constructive neighborhoods. The general idea of removing a set of nodes from the route plan and then re-inserting them again has been used by Russell and Gribbin (1991) and Russell (1995). Here, a total of five nodes is removed and re-inserted again by cheapest insertion. In our notation, the approach can be described by $5|connect|CI$. A similar approach has been suggested by Toth and Vigo (1996) for a solution of a dial-a-ride problem with time windows. They consider removing one or two requests (corresponding to a pickup and a delivery location) and to reinsert them in the cheapest possible position in another route, which corresponds to 1 or $2|connect|CI$ in our notation. A more general view of a partially destructive/constructive neighborhood is the so-called *ruin-and-recreate* metaheuristic suggested in Schrimpf et al. (2000). Here, the authors suggest to 'ruin' parts of the route plan using a controlled random search and to re-insert the nodes using an insertion heuristic. In contrast to many of the other approaches, the number of removed nodes is usually quite large in this approach.

An even simpler version of the remove-and-reinsert idea is suggested by Burke, Cowling, and Keuthen (2001). They introduce the concept of a *hyperedge*, which is simply a segment. The length λ of a hyperedge is equal to the number of edges (i.e., the number of nodes less one) in the segment. A *HyperOpt* move consists of removing k hyperedges of length λ and re-inserting them in an optimal way (by dynamic programming) into the route plan, while keeping the remaining paths fixed. Thus, this type of neighborhood can be abbreviated as $k\underline{(\lambda - 1)}|open|DynProg$. A similar neighborhood of the type $k(\lambda - 1)|open|random$ is called hyper-shake and is used for diversification of the search.

Finally, we would like to mention the US move. Unstringing-and-Stringing (US) is used within the GENIUS heuristic for the TSP(TW) (Gendreau, Hertz, and Laporte, 1992; Gendreau et al., 1998). The idea of the US procedure is to remove a node from the current tour, re-connect the remaining nodes by a reverse generalized insertion (GENI) procedure and re-insert the removed node, again using the GENI procedure. In our notation, the US neighborhood can be described by $1|reverseGENI|GENI$.

3. Search techniques

In this section we will study the following problem: Given a neighborhood in a VRSP, how can one search efficiently in order to reach a local optimum as quickly as possible? We will distinguish two generic approaches: *direct search by enumeration* and *indirect search by optimization.* By direct search, we mean approaches that subsequently add and/or delete edges or nodes and evaluate the result of these operations directly. Usually one can consider these approaches as some type of tree search method. Indirect search methods try to map the problem of finding a best improving solution in the neighborhood into some optimization problem, such as a shortest path, assignment or set packing problem. The search for improving neighbor solutions is then equivalent to solving the optimization algorithm exactly or by a heuristic.

Before going into the details of neighborhood exploration, it should be noted that the search methods are only relevant for algorithms that scan the entire neighborhood (at least in a 'first improve' manner). Some metaheuristics, like Simulated Annealing and

Genetic/Evolutionary Algorithms do not scan the neighborhood but rather sample from it by choosing neighbor solutions randomly. As outlined for the TSP in Johnson and McGeoch (1997), a method that uses a controlled random experiment by sampling from neighbor lists outperforms a total random sampling. However, these issues will not be discussed any further in this paper.

In order to suggest a suitable search technique not only the type of neighborhood but also the objective/cost function and the relevant constraints have to be considered. The objective function influences whether a cost independent decomposition of moves can exist or not. In case of edge-dependent and vehicle-dependent costs, this only depends on the neighborhood. If the costs depend on resource variables, no cost independent decomposition exists. Considering constraints, one has to distinguish between methods that guarantee feasibility of neighbor solutions by checking the constraints during the search and other methods that obtain the neighbor solution by composition of partial moves that are guaranteed to be feasible. A third group of algorithms is driven mainly by cost improvement (gain) considerations. They only work for loosely constrained problems, since the search method will mostly determine infeasible solutions with high profitability if the problem is highly constrained.

3.1. Direct search techniques

Direct search techniques are used within neighborhoods that move or exchange edges or nodes. Suppose that we are given such a neighborhood of size $\mathcal{O}(n^k)$. In order to search this neighborhood efficiently, we would like to reduce the effort (at least on average) by pruning early in the search. There are several techniques available to avoid spending this effort in every iteration. However, we will focus on the basic search methods, whereas further acceleration techniques are beyond the scope of this paper (e.g., *fixing edges*, Lin and Kernighan, 1973; Walshaw, 2002, *treating segments as nodes*, Kindervater and Savelsbergh, 1997; Funke, 2003, *candidate lists*, Glover, 1991; Rego and Glover, 2002, *don't look bits*, Bentley, 1992; Johnson and McGeoch, 1997; Gambardella, Taillard, and Agazzi, 1999; Cordone and Wolfer Calvo, 2001).

Consider the problem of exchanging k edges (resp. nodes) with k other edges (nodes). All direct search methods start by selecting the first edge (node), then the second edge (node) etc. until the k-th edge (node) has been selected so that a complete exchange has been specified. The goal of a search method should be to limit the number of possible edges or nodes at each stage $1 \leq i \leq k$ of this general approach as much as possible in order to keep the effort low. Sometimes it is possible to prove that after the i-th stage, there are no or only a small number of edges or nodes available for selection. This can reduce the search effort drastically. The two main criteria for a reduction of the search space are *cost* and *feasibility*. The idea of cost-based reductions is to prove at an early stage $i < k$ that no improvement can be found that includes the nodes or edges of the stages $1, \ldots, i$. Feasibility reductions use the same idea but try to prove at an early stage $i < k$ that no feasible exchange exists that includes the nodes or edges of the stages $1, \ldots, i$. Unfortunately, the options for using both approaches simultaneously are very limited. Therefore, one usually has to decide whether to apply cost *or* feasibility reductions in the search. In the sequel, we introduce two direct search approaches for k-opt. *Lexicographic search* is motivated by feasibility reductions whereas *sequential search* is based on cost reductions.

3.1.1. Lexicographic search. The natural approach for developing an algorithm that exchanges k elements with k other elements is to specify the k elements by a set of k nested loops. The first loop considers the elements $i_1 = 1, \dots, n$ (n is the number of elements), the second loop the elements $i_2 := i_1 + 1, \dots, n$ and the k-th loop the elements $i_k := i_{k-1} + 1, \dots, n$. Since the iterator of an inner loop is always larger than the iterator of an outer loop, i.e., $i_{l+1} > i_l, l = 1, \dots, n-1$, this search approach is referred to as *lexicographic search*. In the following, we will assume that the elements are nodes or edges and numbered according to their position in the current route plan. Given the k elements, the *neighborhood type* specifies the possible moves that can be performed.

k-Opt. Let us first consider the case of k-opt moves and a fixed given move type. Given the k edges to be removed, the *segmentation* is fixed. The determination of the k segments is implemented using k nested loops to fix the edges for removal. Let us assume that the algorithm is in its ith loop, i.e., the first i segments are fixed. Pruning the search can be performed by the following *feasibility reductions*: Knowing the first i segments it can sometimes be shown that the concatenation of some of the segments result in infeasible subpaths, independent of the remaining segments. For example, in the k-opt move type $aB\dots$, the concatenation of the segments a and B to aB can be infeasible regardless of the remaining segments. This means that the search can be terminated *before* all the k edges have been specified.

If one wants to implement lexicographic search for *all move types* and a given value of k, then it is useful to identify all move types a priori and to store them in a move-type table. Now consider the same situation as above, i.e., that $i < k$ edges have been fixed. Then one can apply the termination argument for *all* move types in the move type table and terminate the search when all move types have been identified as infeasible in one of the stages $1, \dots, i$.

λ-Interchange. As a second example, consider λ-interchange, where the nodes need to take the positions of a removed node when inserted again, i.e., the case $(\underline{\lambda}^*, \underline{\lambda}^*)$ in α^*-notation. We assume that each set of λ nodes has to come from one of the H routes of the current route plan. We also assume that the number of nodes in the i-th route ($i = 1, \dots, H$) is given by n^i.

The interchange procedure can be implemented by nested loops. Two outer loops control the selection of the two routes that exchange nodes. Let us assume that we are inside these loops and that the two routes r^1 and r^2 have been selected. The first inner loop iterates over the nodes $i_1 := 2, \dots, n^1 - \lambda$ of the first route (it is assumed that the route-start and route-end nodes cannot be contained in an interchange), the second loop over the nodes $i_2 := i_1 + 1, \dots, n^1 - \lambda + 1$ and the λ-th loop over $i_\lambda := i_{\lambda-1} + 1, \dots, n^1 - 1$. The same set of λ loops selects the λ nodes from the second route. After the 2λ nodes have been fixed for possible exchange, one needs to compute all $(\lambda!)^2$ possible ways to re-insert the nodes into the open positions.

Summary. The major advantage of lexicographic search is the systematic way in which the segments are built. In every loop, exactly one element is added to or deleted from a segment. Thus, for every segment one can efficiently update all necessary information regarding resources, such as, e.g., load, arrival times, waiting times, precedences etc. This information can be used, in many cases, to prune the search at an early stage as well as to check *feasibility* of the move in constant time. Most of these ideas have been published by Savelsbergh and his co-authors. For a survey, see Kindervater and Savelsbergh (1997). In some cases, one knows a priori that some of the segments need to be reversed. Again, the information regarding the segments to be inverted can be updated efficiently during the lexicographic search and even used for early termination whenever a segment to be inverted becomes infeasible due to, e.g., a time window or precedence constraint violation.

3.1.2. Sequential search. Sequential search has been developed for problems where k elements (edges or nodes) are replaced by k other elements. It is a cost or gain oriented search algorithm that exploits the cost independence of moves and cyclic independence of neighborhoods. The basic idea of this approach is to consider all relevant partial moves of a cyclic independent neighborhood recursively. In contrast to lexicographic search, the elements in sequential search are *not* selected according to any order specified by the current route plan but rather considering *candidate lists*. These candidate lists are sorted in the order of increasing cost of the elements. Sequential search is particularly attractive if the number of elements in every selection step (i.e., the length of the candidate list) is reduced from $\mathcal{O}(n)$ to some small constant. The major disadvantage of sequential search is that the *feasibility* of a move cannot be checked before all k elements have been specified *and* the check will (in the case of non-trivial constraints) require at least linear time. Therefore, there is a trade-off between the reduction of the search space in sequential search and the extra effort that has to be paid to check feasibility. A particular bad scenario is the discovery of many potentially improving, but infeasible moves.

In order to search a neighborhood entirely by sequential search, it has to be *cyclic independent*, although it can be used heuristically with non-cyclic-independent neighborhoods, as discussed below for the Lin-Kernighan heuristic. The second requirement is that the partial moves have to be *cost independent*.

Many of the node and edge-exchange neighborhoods considered in this paper have a cyclic independent decomposition into partial moves. Examples are the λ-interchange, NEC, and cyclic-transfer neighborhoods in the case of node exchanges, and a large fraction of the k-opt move types as well as the edge-ejection chains.

Recall that cost independency of partial moves depends on the objective function under consideration and thus a neighborhood that decomposes into partial moves can be used for sequential search if and only if the cost independency assumption holds. In this subsection we will assume that cost independency holds for the moves we study.

The attractiveness of sequential search in cyclic neighborhoods is due to the following theorem of Lin and Kernighan (1973):

Theorem 1. *If a sequence of numbers* $(g_i)_{i=1}^k$ *has a positive sum* $\sum_{i=1}^k g_i > 0$, *there is a cyclic permutation* π *of these numbers such that* every partial sum *is positive, i.e.*, $\sum_{i=1}^p g_{\pi(i)} > 0$ *for all* $1 \leq p \leq k$.

Proof: Let q be the largest index for which $\sum_{i=1}^{q-1} g_i$ is minimum. Choose π such that $\pi(1) = q$. If $q \leq p \leq n$, $g_q + \cdots + g_p = (g_1 + \cdots + g_p) - (g_1 + \cdots + g_{q-1}) > 0$. If $1 \leq p < q$, then $g_q + \cdots + g_n + g_1 + \cdots + g_p \geq g_q + \cdots + g_n + g_1 + \cdots + g_{q-1} > 0$. $\square$

The theorem implies that, for finding an improving move $m = p_k \circ \cdots \circ p_1$ within a given neighborhood which decomposes into cyclic independent partial moves, we need to consider those moves only where $G_i := \sum_{l=1}^{i} g(p_i, x) > 0$ holds for all $i = 1, \cdots, k$. The direct implication is that at stage i of the search we need to consider moves with a gain $g(p_i, x) > -G_{i-1}$ only. Thus, the total gain at stage $i-1$ limits the choice of a partial move at stage i. We refer to this rule as the *gain criterion*. The gain criterion is fundamental for the effectiveness of k-opt and Lin-Kernighan algorithms for the TSP, see Section 2.2.5 and, e.g., (Aarts and Lenstra, 1997, p. 238ff; Bentley, 1992). Interestingly, the path reference structure used within the Lin-Kernighan neighborhood is *not* cyclic independent. Therefore, Theorem 1 does not apply. Rather, it is used *heuristically* with excellent results. This should be kept in mind when considering the exploitation of the gain criterion within a search algorithm. The gain criterion is also exploited in effective algorithms for 2-opt and 3-opt for the TSP (Bentley, 1992; Johnson and McGeoch, 1997). In Irnich, Funke, and Grünert (2004) the use sequential search for capacitated VRP is investigated. A comparison of lexicographic and sequential search showed substantial speedups for the sequential search approach. However, we are not aware of any of the 'modern' metaheuristics for VRSPs that exploit this very important speedup technique.

In order to describe a generic sequential search algorithm, consider the decomposition $m = p_k \circ \cdots \circ p_2 \circ p_1$ of a move m into $k \geq 2$ partial moves. The cyclic independence of the neighborhood implies that any sequence $p_{j-1} \circ \cdots \circ p_1 \circ p_k, \ldots, p_j$ with $1 \leq j \leq k$ represents the same move m. For the gain criterion to be applicable, the search algorithm has to guarantee that every cyclic permutation of the moves is generated.

Hence, the algorithm has to generate all the partial moves $p_1, \ldots, p_k$ on the first search level. Each of these partial moves has to satisfy the gain criterion, otherwise, it will be discarded immediately. On the second level, the composed partial moves $p_2 \circ p_1$, $p_3 \circ p_2, \ldots, p_k \circ p_{k-1}$, $p_1 \circ p_k$ that extend non-discarded partial moves from the first level have to be generated. Again, all move compositions on the second level have to satisfy the gain criterion and can otherwise be discarded. The same is true for the third level and so on until all k levels have been investigated.

This implies that the sequential search algorithm has to meet two requirements: First, it has to generate all compositions of partial moves according to the cyclic decomposition of the neighborhood. Second, it can apply the gain criterion to all compositions of partial moves in order to fathom the search as much as possible. The first requirement directly shows the close relationship between the decomposition of the neighborhood and efficient search, while the second requirement directly implies the cost independence of partial moves. We now give some examples of how sequential search can be applied efficiently in the context of vehicle routing and scheduling.

2-Opt. The 2-opt moves are single alternating cycle moves and, therefore, cyclic. We decompose the 2-opt neighborhood into two partial moves. The partial move $p_1 := p^{\text{add}}_{j_1,i_2} \circ$

$p^{\text{del}}_{i_1,j_1}$ consists of deleting the edge $d_1 = (i_1, j_1)$ and subsequently adding the edge $a_1 = (j_1, i_2)$. The partial move $p_2 := p^{\text{add}}_{j_2,i_1} \circ p^{\text{del}}_{i_2,j_2}$ deletes the edge $d_2 = (i_2, j_2)$ and finally adds the edge $a_2 = (j_2, i_1)$. Let us denote the length of the involved edges by $|d_l|$ and $|a_l|$, $l = 1, 2$, respectively. The partial gains of the two partial moves are equal to $g_1 := |d_1| - |a_1|$ and $g_2 := |d_2| - |a_2|$ and the gain criterion requires that both $g_1 > 0$ and $g_1 + g_2 > 0$ hold. Now suppose that we want to employ sequential search to find an improving 2-opt move. First, we loop over $i_1 := 1, \ldots, n$ to determine the node i_1. Next, we consider one of the edges incident with i_1 in the tour to obtain the edge $d_1 := (i_1, j_1)$. This determines the length of the edge d_1 and from the gain criterion, $g(p_1, x) > 0$, we know that we only have to consider adding edges $a_1 := (j_1, i_2)$ that satisfy $|a_1| < |d_1|$. This is usually done efficiently by storing a list of edges incident with every node ordered by increasing length (in asymmetric problems, ingoing and outgoing edges are stored separately). Suppose that we have found an edge on this list which satisfies $|a_1| < |d_1|$. Then, the entire move has been specified, since there is only one possibility of obtaining a feasible 2-opt move. After having checked this possibility, we backtrack to the first level and check the other edge incident to i_1. This is necessary to generate *all cyclic permutations* of the moves p_1 and p_2. Figure 8 depicts the generation of 2-opt moves by this procedure. Figure 8(a) shows the situation after the application of p_1 as the first partial move according to $m = p_2 \circ p_1$, i.e., with $i_1 = v_1$, $j_1 = w_1$, and $i_2 = v_2$. This move is generated by taking j_1 as the node adjacent to i_1 in the *clockwise direction*. Figure 8(b) shows the situation after applying the move p_2 first, according to $m = p_1 \circ p_2$, i.e., with $i_1 = v_2$, $j_1 = w_2$, and $i_2 = v_1$. Here, the node j_1 is obtained by taking the node adjacent to i_1 in the *counterclockwise direction*. It is therefore necessary to consider both directions—clockwise and counterclockwise—when selecting j_1.

Lin-Kernighan. As a second example we consider the Lin-Kernighan partial moves based on the path reference structure discussed in Section 2.2.5. Here edges are deleted and added subsequently and the edge to be deleted next is always incident to the endpoint of the last added edge and vice versa. At stage i, the edges $d_1, \ldots, d_i$ have been deleted and the edges $a_1, \ldots, a_i$ have been added. The gain of each move is $g_i := |d_i| - |a_i|$. Now consider stage $i+1$. The next edge to delete is d_{i+1} and there is no choice due to the path reference structure.

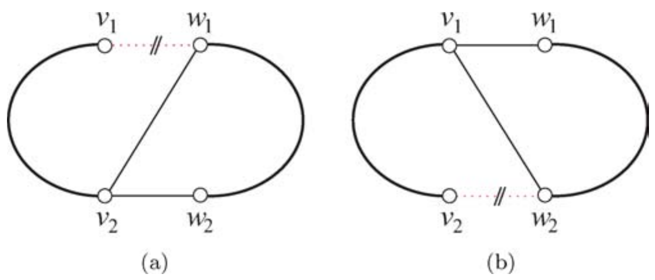

Figure 8. Using sequential search for finding improving 2-opt moves, (a) The partial move $p_1 = p^{\text{add}}_{w_1,v_2} \circ p^{\text{del}}_{v_1,w_1}$ is applied first, (b) The partial move $p_2 = p^{\text{add}}_{w_2,v_1} \circ p^{\text{del}}_{v_2,w_2}$ is applied first.

From the endpoint of d_{i+1} we can add an edge a_{i+1} that satisfies $|a_{i+1}| \leq G_i - |d_{i+1}|$. If no such edge exists, the search can be terminated at this stage. This implies that the search depth of the algorithm is adjusted dynamically by the gain criterion. The Lin-Kernighan algorithm does not search the entire neighborhood of the path reference structure but rather does a depth-first search, where in every iteration, the next added edge is selected according to a greedy criterion. Another feature of the algorithm is that it checks the tour length resulting from a closing partial move at each iteration and finally selects the move that results in the minimum tour length over all levels of the search. Recall that the decomposition of moves that results from the path reference structure is not cyclic. Therefore, although it is the first efficient application of the gain criterion within a local search algorithm, the theorem provided by Lin and Kernighan does not apply to this neighborhood in an exact sense, but rather is a *heuristic* criterion to cut off branches of the search tree (albeit a very efficient one).

Swap. The swap move m_{ij}^{swap} replaces a node i of the giant tour by a node j and vice versa. Consequently, the four edges $(i-1, i), (i, i+1), (j-1, j), (j, j+1)$ are deleted and the four edges $(i-1, j), (j, i+1), (j-1, i), (i, j+1)$ are added to the current solution x. The swap move was presented in Section 2.1.2 as composed of two *dependent* relocation partial moves $m_{i,j-1}^{\text{reloc}} \circ m_{j,i-1}^{\text{reloc}}$.

There also exist several decompositions of m_{ij}^{swap} into two cost independent partial moves of the same type. One possibility is to define p_{ij} as a partial move which deletes the edges $(i-1, i), (i, i+1)$ and then adds the edges $(j-1, i), (i, j+1)$. Then, $m_{ij}^{\text{swap}} = p_{ij} \circ p_{ji} = p_{ji} \circ p_{ij}$ is a cyclic decomposition into cost independent partial moves. The partial gain of p_{ij} is $g(p_{ij}, x) = c_{i-1,i} + c_{i,i+1} - c_{j-1,i} - c_{i,j+1}$. When looking for improving swap moves by sequential search, the gain criterion tells us that we can restrict our attention to a first partial move p_{ij} with positive gain. We propose to search for those p_{ij} with positive gain by first considering all nodes $i \in V$. The task is then to restrict the search for possible nodes j under the condition that i is known. This can be done with candidate lists of edges incident to node i ordered increasingly by their cost. Let $\alpha := \frac{c_{i-1,i}+c_{i,i+1}}{2}$, which is a fixed constant when node i is chosen. The condition $g(p_{ij}, x) > 0$ is equivalent to $(c_{j-1,i} - \alpha) + (c_{i,j+1} - \alpha) < 0$ which implies $c_{j-1,i} < \alpha$ or $c_{i,j+1} < \alpha$. As a consequence, only ingoing edges $(j-1, i)$ of cost less than α and outgoing edges $(i, j+1)$ of cost less than α have to be considered. Finally, when j is chosen, the total gain $g(p_{ij}, x) + g(p_{ji}, x)$ can be checked in constant time.

Another decomposition of m_{ij}^{swap} is to define a partial move q_{ij} that removes the edges $(i-1, i), (j-1, j)$ and adds the edges $(i-1, j), (j-1, i)$. Then, $m_{ij}^{\text{swap}} = q_{ij} \circ q_{i+1,j+1} = q_{i+1,j+1} \circ q_{ij}$. This is also a decomposition into two cost independent partial moves of the same type and, therefore, the gain criterion applies in a similar way.

Granular tabu search. Granular Tabu Search, suggested by Toth and Vigo (2002a), uses a variant of the idea of sequential search. In this approach the following four simple moves are used: 2-opt, relocation, Or-opt with a segment length of two, and swap. In all these moves the algorithm requires that the first added edge is ‘short’, i.e., belongs to a candidate list. This candidate list contains edges, whose length is below a certain threshold. The threshold is varied during the search in order to intensify or diversify the search. Since all the moves that are used in this algorithm can be specified entirely by the first added edge, this can be considered as a very simple version of sequential search.

Summary. Sequential search is a very powerful technique that could be used successfully in solving a number of VRSPs. If a cost-based strategy is used for constructing the candidate lists, it is usually required that the objective function can be decomposed into partial gains and that each partial move contributes a fixed amount to the objective function. This requirement for cost independency is not generally fulfilled. For example, if the duration of the trips or the latest arrivals should be minimized, then this requirement is not met. However, in problems where distance or an equivalent measure has to be minimized, this is an interesting alternative to other search methods that could be the basis for a number of highly efficient algorithms.

3.2. Optimization-based indirect search techniques

The idea of indirect search techniques is to re-formulate the problem of finding the best move in a given neighborhood so that suitable optimization algorithms or heuristics can be used for the search. Two main approaches have been discussed in the literature. The first approach is *move-composition*, which is based on the idea of explicitly mapping the partial moves which can be applied to the current solution to the decision variables of an auxiliary problem. The optimal solution to this auxiliary problem then defines a gain-maximizing collection of partial moves that can be applied to the current solution. The second type of approaches are the *combinatorial structure approaches*. They implicitly restrict the neighborhood of the current solution to satisfy some combinatorial requirements that allow the search for improving neighboring solutions to be implemented very efficiently.

Both approaches employ a type of indirect search, which is usually referred to as *very large-scale neighborhood search* (Ahuja et al., 1999). It can be described generically by the following algorithm:

Algorithm 2: Optimization-based Indirect Local Search

```
1: Initialize the algorithm with a feasible solution $x^0$ and set the iteration counter $t := 0$.
2: REPEAT
3:    Generate an instance $P(x^t)$ of the optimization problem corresponding to the
      neighborhood $\mathcal{N}(x^t)$
      (either from scratch or using knowledge about the last move).
4:    Find a neighbor solution $x'$ by searching for the optimal or an improving solution of
      $P(x^t)$.
5:    If an improving solution $x'$ has been found, then set $t := t + 1$ and $x^{t+1} := x'$.
6: UNTIL no more improvements are found.
```

The time-consuming parts of this algorithm are the generation of the optimization problem instance and its solution by a convenient algorithm. Indeed, the optimization problem may be $\mathcal{NP}$-hard, thus requiring suitable heuristics to be used for search in step 4.

3.2.1. Move-composition approaches. The general idea of these approaches is the decomposition of a very large neighborhood into a set of partial moves. These partial moves are used to define an optimization problem (often on a network), which can be solved by an appropriate algorithm. The result is a selection of moves, whose composition is both *feasible* and *gain-maximal* in the (very large) neighborhood. If the auxiliary optimization problem is based on a graph, we call this graph an *improvement graph*. This concept was first suggested by Thompson and Psaraftis (1993). However, as shown below, other optimization problems may be formulated within move-composition approaches. Since the optimal solution of the auxiliary problem implicitly dominates a large (usually exponential) number of other solutions, this approach gives rise to the so-called *combinatorial leverage effect*, introduced in Glover and Punnen (1994), that considers the size of the neighborhood in relation to the effort required to scan it.

Cyclic improvement graphs. The concept of cyclic improvement graphs goes back to Thompson and Psaraftis (1993) and is closely related to the concept of *cyclic transfers* described earlier. Here we give a slightly generalized description of the original concept. The cyclic improvement graph models the ejection of a node or a set of nodes by another node or set of nodes via arcs. By definition of cyclic transfers, these node sets belong to different routes. Let the node set of the improvement graph contain all nodes or sets of nodes that could be part of an ejection chain. Let two nodes of the improvement graph be S_i and S_j. An arc (S_i, S_j) of the improvement graph models the operation of removing the node set S_j from its current route and inserting the node set S_i into that route. The gain $G(S_i, S_j)$ depends on how the insertion cost is computed. This can be done by any heuristic or optimization algorithm for the single route problem. In order to reduce complexity, simple heuristics are used in most cases. The cost or length of an arc (S_i, S_j) is equal to the negative gain $-G(S_i, S_j)$. Then, it is easy to see that any cycle with negative length in the improvement graph corresponds to a cost improving cyclic exchange. Unfortunately, the problem of detecting such a cycle is $\mathcal{NP}$-hard, since it again corresponds to a resource-constrained shortest path problem. In such a problem, one introduces a resource for each route and requires that this resource is only used once in a solution. Several heuristics have been developed for this problem (Ahuja, Boland, and Dumitrescu, 2001a). It should be noted that the case where a node or a set of nodes is ejected by an empty set can easily be handled by introducing a dummy node in the improvement graph (Ahuja, Orlin, and Sharma, 2001b). This allows insertion moves to be handled in addition to exchange moves.

Note that, since the cyclic-transfer neighborhood is cyclic, the gain criterion can be employed. This means that in any negative length cycle, the total length of any cyclic subset of edges must be negative.

Cluster-partition graphs. Cluster-partition graphs were introduced in Beasley (1983) in order to solve the clustering problem in the route-first cluster-second heuristic for the vehicle routing problem. Here, we consider the solution of finding the best improving solution in a neighborhood as described in Section 2.3.4, where the initial routing, i.e., the order of the routes, is fixed.

Suppose that all depot nodes $o \sim d$ have been removed from the route plan and that the resulting single-route segments have been ordered and concatenated to yield a giant route where the nodes are numbered consecutively from 1 to n. The cluster-partition graph is based on the giant-route representation and can be defined as follows: Let the node set be $V = \{1, \ldots, n+1\}$. For every pair (i, j) with $j > i$ in the cluster-partition graph, there exists an arc (i, j) for vehicle $h \in H$ if the route $(o^h, i, i+1, \ldots, j-1, d^h)$ is feasible. If several vehicles have identical characteristics and the same depot combination, then they can be handled as identical vehicles. The cost C^h_{ij} of the arc (i, j) is equal to the cost of this route. It is easy to see that an optimal clustering w.r.t. the given giant route can be found by solving a (possibly resource-constrained) shortest path problem from node 1 to node $n+1$.

The composition of complete moves. The composition of complete moves, such as the 2-opt, Or-opt, or swap moves into a compound move, was suggested by Congram, Potts, and van de Velde (2002), Ergun, Orlin, and Steele-Feldman (2002). Let us first consider the 2-opt neighborhood to explain the idea behind this type of large-scale neighborhood search. A feasible, cost independent combination of several 2-opt moves $m = m^{\text{2-opt}}_{i_1 j_1} \circ m^{\text{2-opt}}_{i_2 j_2} \circ \cdots \circ m^{\text{2-opt}}_{i_t j_t}$ (a so-called compound move) with "small" individual improvements is sometimes superior to the single best move $m^{\text{2-opt}}_{i_{\text{best}} j_{\text{best}}}$ in the neighborhood. Feasibility constraints and/or a legitimacy condition might forbid the execution of the moves with small improvement after the best improving move has been executed.

For simplicity, we restrict the following description to the 2-opt moves case. An improvement graph for this neighborhood can be constructed by re-labeling the nodes in the current solutions by $1, \ldots, n$. One 2-opt move can be described entirely by two nodes $(i, j) \in \{1, \ldots, n\}, i+2 \leq j$. This corresponds to the deletion of the edges $d_1 = (i, i+1)$ and $d_2 = (j-1, j)$, the addition of the edges $a_1 = (i, j-1)$ and $a_2 = (i+1, j)$, and the reversal of the segment $(i+1, j-1)$. The alternative of reversing the paths $(1, \ldots, i)$ and $(j, \ldots, n)$ is not considered here. Let the total gain of this 2-opt move be $G_2(i, j)$. Then, the following improvement graph can be constructed: Let $V := \{1, \ldots, n+1\}$ be the set of nodes, where $n+1$ corresponds to a copy of node 1. The arc set consists of all feasible 2-opt moves, $A = \{(i, j) : \text{the 2-opt move } (i, j) \text{ is feasible}\}$. Moreover, the improvement graph contains the arcs $(i, i+1), i = 1, \ldots, n$, that correspond to leaving the edge $(i, i+1)$ unchanged by the compound move. The cost of an arc $(i, j) \in A, j \neq i+1$ is equal to $-G_2(i, j)$ and the cost of an arc $(i, i+1)$ is equal to 0. Then the shortest path from 1 to $n+1$ corresponds to the composition of 2-opt moves that lead to the maximal cost reduction. If the length of this path is negative, then an improving compound move is found, otherwise, the current solution is 2-optimal.

Note that the concept of this type of improvement graph is to define moves by the segment they affect. Clearly, this concept is not limited to 2-opt moves but can readily be generalized to include relocation and exchange moves as well as other moves, e.g., 2-opt*, Or-opt, and double-bridge moves. The direct generalization to swap moves requires that two swaps (i, j) and (k, l) satisfy the legitimacy condition $i < j, j+1 < k < l$, i.e., do not overlap (Congram, Potts, and van de Velde, 2002; Ergun, Orlin, and Steele-Feldman, 2002).

A pre-requisite for this approach is that all relevant moves be enumerated a priori by a convenient direct search method. Both lexicographic and sequential search can be used for this purpose. In both cases, one has to check feasibility of the moves when intra-route constraints are relevant in the VRSP under consideration. However, when complex resources, such as time windows, are relevant, then the composition of several feasible moves does not necessarily result in a compound feasible move. In this case it is necessary to store the change of resource variables (e.g., load, arrival times) as a function of the involved nodes. Instead of solving a simple shortest path problem, a *resource constrained shortest path problem* has to be solved. Unfortunately, this problem is $\mathcal{NP}$-hard, so heuristics may be used instead (Ahuja, Boland, and Dumitrescu, 2001a).

Ergun, Orlin, and Steele-Feldman (2002) relax the independence requirement of moves by allowing two subsequent moves to overlap slightly. If this is the case, a new move is defined, which is different from the move that would be obtained if the two single moves had been applied independently. Thus, a correction term has to be added to the evaluation. This is accounted for by storing the corresponding information for all pairs of arcs in the improvement graph and solving a shortest path problem *with turn penalties* instead of a regular shortest path problem.

A problem of this approach in the multiple-route case is that it requires all arcs (i, j) to be ordered so that $j > i$. Otherwise, the independence of moves cannot be guaranteed easily. Thus, in multiple-route problems, the order of the routes affects the type of compound moves that can be generated using this neighborhood.

Assignment-based approaches. Obviously, assignment-based neighborhoods, as described in Section 2.3.1 can implicitly be searched by solving the corresponding assignment problem.

Finding the best double-bridge move in $\mathcal{O}(n^2)$. The composition of the double bridge move as two cost independent, *crossed* 2-opt* moves is described in Glover (1996b). The best improving double bridge move of a TSP can be determined with $\mathcal{O}(n^2)$ effort. Assume that the nodes of the giant tour are numbered $1, 2, \ldots, n$ such that all edges are of the form $(i, i+1)$. A double bridge move m^{DB} is determined by its two alternating cycles of deleted and added edges, i.e., two 2-opt* moves p_{i_1,i_2} and p_{i_3,i_4}. The trick of Glover is to build an acyclic digraph D with $\mathcal{O}(n^2)$ nodes and arcs in which each feasible combination $1 \le i_1 < i_3 < i_2 < i_4 \le n$ is uniquely coded as a path $P(i_1, i_2, i_3, i_4)$. Furthermore, the length of such a path $P(i_1, i_2, i_3, i_4)$ is the cost of the double bridge move.

3.2.2. Combinatorial structure approaches. Most of the combinatorial structure appro-break aches have so far been developed for the TSP (Cornuéjols, Naddef, and Pulleyblank, 1983; Glover and Punnen, 1994). As discussed above, we will not go into the details of these approaches, since they do not generalize to more complex VRSPs. We do, however, believe that these types of approaches have a big potential also in the context of VRSPs and that more research is necessary in order to develop more generally applicable algorithms than those available today. Two approaches that can be generalized to some extent include partial order neighborhoods and pyramidal tour improvement neighborhoods.

Partial order neighborhoods. The restrictive possibilities for moving a node relative to its current position in the route lies at the heart of the dynamic programming approach by Balas (1999) and Balas and Simonetti (2001). The direct exploitation of these possibilities enables one to reduce the number of possible states in a dynamic programming approach from exponential to linear. The algorithm of Balas and Simonetti (2001) exploits this fact and also provides a forward dynamic programming recursion, which can be considered a shortest path approach. For the details of the implementation, we refer to the original paper.

Pyramidal tour improvement graphs. An optimal pyramidal tour neighbor for the TSP can be found in $\mathcal{O}(n^2)$ using dynamic programming or shortest path calculations (Gilmore, Lawler, and Shmoys, 1985). Here we are interested in its generalization to more complex VRSPs and first describe the corresponding improvement graph suggested in Ahuja et al. (1999). The improvement graph—a bipartite digraph—consists of $2n$ nodes $V = V^1 \cup V^2$, where $V^1 = \{1, \dots, n\}$ and $V^2 = \{1', \dots, n'\}$. Let us call the nodes in V^1 forward nodes and the nodes in V^2 backward nodes. All arcs in the improvement graph connect forward nodes with backward nodes or vice versa. Their interpretation is as follows: Consider the arc $(i, j'), i \in V^1, j' \in V^2, j \geq i$. This arc corresponds to the case that the segment $(i, \dots, j)$ in the current route is fixed and visited in this order in the first part of the route. The arc $(j', k), j' \in V^2, k \in V^1, k > j + 1$, corresponds to the case where the segment $(j+1, \dots, k-1)$ is skipped in the first part and visited in the reverse order $(k-1, \dots, j+1)$ in the second half of the route, i.e., the edges $(j, j+1)$ and $(k-1, k)$ are deleted and the edge (j, k) is introduced.

Let the cost of the edge (f, g) in the original routing problem be c_{fg}. Then, the gain $G(i, j')$ that corresponds to an arc $(i, j'), i \in V^1, j' \in V^2, j \geq i$ in the improvement graph can be computed using Table 2.

The cost of the arc $(j', k), j' \in V^2, k \in V^1, k > j + 1$, is equal to $G(j', k) = c_{jk} - \sum_{\ell=j}^{k-1} c_{\ell,\ell+1} + \sum_{\ell=k-1}^{j+2} c_{\ell,\ell-1}$. Any path P from 1 to n' in the improvement graph corresponds to a pyramidal tour x', and the cost of such a path is $-g(x, x')$, i.e., a shortest path corresponds with a best (improving) neighbor solution. Note that if the segment $(j+1, \dots, k-1)$ which is reversed contains one or more entire routes, then inversion can be avoided for these routes. Of course, the cost of the arcs in the improvement graph have to take this into account. If a reversion causes a segment to be infeasible, then the corresponding arc can be removed from the improvement graph.

In case complex resources play a role in the VRSP under consideration, a resource-constrained shortest path problem from node 1 to node n' can be solved in order to check for an improving pyramidal neighbor. It can be verified that any path with negative costs corresponds to an improving neighbor.

Clearly, the order of the routes in the giant route determine the feasible pyramidal tours and, thus, the results of the algorithm. Again, one has to use heuristics to find 'good' orders of the routes.

Table 2. Costs of the arcs in the shortest path problem for finding optimal pyramidal tours.

$G(i, j')$	$j' = 1$	$j' = i, i+1, \dots, n-1$	$j' = n$
$i = 1$	$-c_{j,j+1} + c_{j+1,1}$	$-c_{j,j+1} + c_{j+1,1}$	0
$i = 2, 3, \dots, n-1$	—	$c_{j+1,i-1}$	$-c_{n,1} + c_{n,i-1}$
$i = n$	—	—	$-c_{n-1,n} + c_{n,n-1}$

Although applicable to solve VRSPs with complex constraints, the potential of the pyramidal neighborhood is quite limited in cases where time windows and/or precedences occur. The reason is that some nodes that occur early in the route will have to occur late in a neighbor route if any improvement is to be found. However, it is unlikely that such an operation is feasible when time windows and precedences occur. This problem can be avoided by considering pyramidal tours that are defined only for parts of the original tour, so that nodes are not moved too 'far' from their current positions.

Conclusions

This paper has provided an overview and a conceptual integration of different classical and modern approaches of local search for solving complex VRSPs. The formal description is based on the so-called giant-tour representation, which allows problems with complex constraints to be modeled within one conceptual framework. The problems include those that can be modeled by resource variables, e.g., heterogeneous fleet problems with time windows, route-duration constraints, and simultaneous pickups and deliveries.

The basis of all local search algorithms is the definition of a neighborhood and the design of an appropriate search method. In order to obtain a theoretical basis for the description of the neighborhood, we have introduced the concept of moves and their decomposition into partial moves. Different decompositions can result in different types of move independence. Depending on the objective function, partial moves may give rise to partial gains, thus enabling the search algorithm to prune regions of the search space.

Search methods can be classified into direct and indirect search methods. Direct search methods can be classified further into sequential search methods, which are based on the gain criterion and lexicographic search methods that are based on constraint evaluations. Most indirect search methods have so far been studied in the context of the TSP. Here, we have assessed their prospects w.r.t. an extension to complex VRSPs.

For a given VRSP, the design of a suitable local search algorithm should be based on an analysis of not only cost independence and cyclic independence of the partial moves but also the structural properties of the constraints. In addition to giving an overview of the most widely used neighborhoods, the paper also provides a theoretical basis for judging the applicability of different local search methods. This should enable researchers to design new and efficient local search algorithms both for 'classical' and more complex real-world problems.

References

Aarts, E. and J. Lenstra. (1997). *Local Search in Combinatorial Optimization*. Wiley, Chichester.

Ahuja, R., N. Boland, and I. Dumitrescu. (2001a). "Exact and Heuristic Algorithms for the Subset Disjoint Minimum Cost Cycle Problem." Technical report, MIT, Boston.

Ahuja, R., O. Ergun, J. Orlin, and A. Punnen. (1999). "A Survey of Very Large-Scale Neighborhood Search Techniques." Technical report, Department of Industrial & Systems Engineering, University of Florida, Gainesville, FL 32611.

Ahuja, R., J. Orlin, and D. Sharma. (2001b). "Multi-Exchange Neighborhood Structures for the Capacitated Minimum Spanning Tree Problem." *Mathematical Programming, Series A* 91(1), 71–97.

Balas, E. and N. Simonetti. (2001). "Linear Time Dynamic-Programming Algorithms for New Classes of Restricted TSPs: A Computational Study." *INFORMS Journal on Computing* 13(1), 56–75.

Balas, E. (1999). "New Classes of Efficiently Solvable Generalized Traveling Salesman Problems." *Annals of Operations Research* 86, 529–558.

Beasley, J. (1983). "Route First—Cluster Second Methods for Vehicle Routing. *OMEGA International Journal of Management Science* 11(4), 403–408.

Bentley, J. (1992). "Fast Algorithms for Geometric Traveling Salesman Problems." *Operations Research Society of America* 4(4), 387–411.

Bodin, L.D., B. Golden, A. Assad, and M. Ball. (1983). "Routing and Scheduling of Vehicles and Crews—The State of the Art." *Computers & Operations Research* 10, 63–211.

Bräysy, O. and M. Gendreau. (2005a). "Vehicle Routing with Time Windows, Part II: Metaheuristics." *Transportation Science* 39(1), 119–139.

Bräysy, O. and M. Gendreau. (2005b). "Vehicle Routing with Time Windows, Part I: Route Construction and Local Search Algorithms." *Transportation Science* 39(1), 104–118.

Burke, E., P. Cowling, and R. Keuthen. (2001). "Effective Local and Guided Variable Neighbourhood Search Methods for the Asymmetric Travelling Salesman Problem." In E. Boers, J. Gottlieb, P. Lanzi, R. Smith, S. Cagnoni, E. Hart, G. Raidl, and H. Tijink, (eds.), *Applications of Evolutionary Computing*, Springer Verlag, Berlin, pp. 203–212.

Carlier, J. and P. Villon. (1990). "A New Heuristic for the Traveling Salesman Problem." *RAIRO—Recherche opérationelle/Operations Research* 24(3), 245–253.

Christofides, N. and S. Eilon. (1969). "An Algorithm for the Vehicle-Dispatching Problem." *Operational Research Quarterly* 20(3), 309–318.

Congram, R., C. Potts, and S. van de Velde. (2002). "An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Sceduling Problem." *INFORMS Journal on Computing* 14(1), 52–67.

Cordeau, J., G. Desaulniers, J. Desrosiers, M. Solomon, and F. Soumis. (2002a). "VRP with Time Windows." In Toth and Vigo (eds.), (2002c), chapter 7, pp. 155–194.

Cordeau, J., M. Gendreau, G. Laporte, J. Potvin, and F. Semet. (2002b). "A Guide to Vehicle Routing Heuristics." *Journal of the Operational Research Society* 53, 512–522.

Cordone, R. and R. Wolfer Calvo. (2001). "A Heuristic for the Vehicle Routing Problem with Time Windows." *Journal of Heuristics* 7, 107–129.

Cornuéjols, G., D. Naddef, and W. Pulleyblank. (1983). "Halin Graphs and the Traveling Salesman Problem." *Mathematical Programming* 26, 287–294.

Croes, G. (1958). "A Method for Solving Traveling-Salesman Problems." *Operations Research* 6, 791–812.

Desaulniers, G., J. Desrosiers, A. Erdmann, M. Solomon, and F. Soumis. (2002). "VRP with Pickup and Delivery." In Toth and Vigo (eds.), (2002c) chapter 9, pp. 225–242.

Desaulniers, G., J. Desrosiers, I. Ioachim, M. Solomon, F. Soumis, and D. Villeneuve. (1998). "A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems." In T. Crainic and G. Laporte (eds.), *Fleet Management and Logistics*, chapter 3. Boston, Dordrecht, London: Kluwer Academic Publisher, pp. 57–93.

Desaulniers, G. and D. Villeneuve. (2000). "The Shortest Path Problem with Time Windows and Linear Waiting Costs." *Transportation Science* 34(3), 312–319.

Desrosiers, J., Y. Dumas, M. Solomon, and F. Soumis. (1995). "Time Constrained Routing and Scheduling." In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser (eds.), *Handbooks in Operations Research and Management Science, Vol. 8, Network Routing*, chapter 2. Amsterdam: Elsevier, pp. 35–139.

Deĭneko, V. and Woeginger, G. (2000). "A Study of Exponential Neighborhoods for the Travelling Salesman Problem and for the Quadratic Assignment Problem." *Mathematical Programming* 87(3), 519–542.

Ergun, O., J. Orlin, and A. Steele-Feldman. (2002). "Creating Very Large Scale Neighborhoods Out of Smaller Ones by Compounding Moves: A Study on the Vehicle Routing Problem." Technical report, Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 30332-0205, USA.

Funke, B., T. Grünert, and S. Irnich. (2004). "A Note on Single Alternating Cycle Neighborhoods for the TSP." *Journal of Heuristics* (to appear).

Funke, B. (2003). "Effiziente Lokale Suche für Vehicle Routing und Scheduling Probleme mit Ressourcenbeschränkungen." PhD thesis, Fakultät für Wirtschaftswissenschaften, RWTH Aachen, Templergraben 64, 52062 Aachen.

Gambardella, L., É. Taillard, and G. Agazzi. (1999). "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows." In D. Corne, M. Dorigo, and F. Glover (eds.), *New Ideas in Optimization*, chapter 5. London: McGraw-Hill, pp. 63–76.

Gendreau, M., A. Hertz, G. Laporte, and M. Stan. (1998). "A Generalized Insertion Heuristics for the Traveling Salesman Problem with Time Windows." *Operations Research* 43(3), 330–335.

Gendreau, M., A. Hertz, and G. Laporte (1992). "New Insertion and Postoptimization Procedures for the Traveling Salesman Problem." *Operations Research* 40(6), 1086–1094.

Gilmore, P., E. Lawler, and D. Shmoys. (1985). "Well-Solved Special Cases." In Lawler et al., (eds.), (1985), chapter 4. pp. 87–143.

Glover, F. and M. Laguna. (1997). *Tabu Search*. Dortrecht: Kluwer.

Glover, F. and A. Punnen. (1994). "The Traveling Salesman Problem: Linear Time Heuristics with Exponential Combinatorial Leverage." Technical report, US West Chair in Systems Science, University of Colorado, Boulder, School of Business, Campus Box 419, Boulder, CO, 80309.

Glover, F. (1991). "Multilevel Tabu Search and Embedded Search Neighborhoods for the Travling Salesman Problem." Technical report, US West Chair in Systems Science, University of Colorado, Boulder, School of Business, Campus Box 419, Boulder, CO, 80309.

Glover, F. (1992). "New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems." In O. Balci, R. Sharda, and S. Zenios (eds.), *Computer Science and Operations Research—New Developments in their Interfaces*. Pergamon Press, pp. 491–508.

Glover, F. (1996a). "Ejection Chains, Reference Structures and Alternating Path Structures for Traveling Salesman Problems." *Discrete Applied Mathematics* 65, 223–253.

Glover, F. (1996b)."Finding a Best Traveling Salesman 4-opt Move in the Same Time as a Best 2-opt Move." *Journal of Heuristics* 2, 169–179.

Gutin, G. and A. Punnen (eds.). (2002). *The Traveling Salesman Problem and Its Variations*, Vol. 12 of *Combinatorial Optimization*. Dordrecht: Kluwer.

Irnich, S. and G. Desaulniers. (2004). "Shortest Path Problems with Resource Constraints." Technical Report G-2004-11, Les Cahiers du GERAD, HEC Montréal, Montréal, Quebec, Canada.

Irnich, S., B. Funke, and T. Grünert. (2004). "Sequential Search and Its Aplication to Vehicle-Routing Problems." *Computers & Operations Research* (to appear).

Johnson, D. and L. McGeoch. (1997). "The Traveling Salesman Problem: A Case Study in Local Optimization." In E. Aarts and J. Lenstra (eds.), *Local Search in Combinatorial Optimization*, chapter 8. Chichester: Wiley, pp. 215–310.

Kernighan, B. and S. Lin. (1970). "An Efficient Heuristic Procedure for Partitioning Graphs." *Bell Syst. Tech. J.* 49, 291–307.

Kindervater, G. and M. Savelsbergh. (1997). "Vehicle Routing: Handling Edge Exchanges." In E. Aarts and J. Lenstra (eds.), *Local Search in Combinatorial Optimization*, chapter 10. Chichester: Wiley, pp. 337–360.

Lawler, E., J. Lenstra, A. Rinnooy Kan, and D. Shmoys, (eds.). (1985). "The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization." Wiley-Interscience Series in Discrete Mathematics. Chichester: Wiley.

Lin, S. and B. Kernighan. (1973). "An Effective Heuristic Algorithm for the Traveling-Salesman Problem." *Operations Research* 21, 498–516.

Lin, S. (1965). "Computer Solutions of the Traveling Salesman Problem." *Bell System Technical Journal* 44, 2245–2269.

Magos, D. and T. Miliotis. (1994). "An Algorithm for the Planar Three-Index Assignment Problem." *European Journal of Operational Research* 77(1), 141–153.

Or, I. (1976). "Traveling Salesman-Type Problems and their Relation to the Logistics of Regional Blood Banking." PhD thesis, Department of Industrial Engineering and Management Sciences. Northwestern University, Evanston, IL.

Osman, I. (1993). "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem." *Annals of Operations Research* 41, 421–451.

Potvin, J., G. Lapalme, and J. Rousseau (1989). "A Generalized *k*-opt Exchange Procedure for the MTSP." *Information Systems and Operations Research* 27(4), 474–481.

Prins, C. (2003). "A Simple and Effective Evolutionary Algorithm for the Vehicle Routing Problem." *Computers & Operations Research* 1–18 (to appear).

Rego, C. and F. Glover. (2002). "Local Search and Metaheuristics." In G. Gutin and A. Punnen (eds.), *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimization*, chapter 8. Dordrecht: Kluwer, pp. 309–368.

Rego, C. (1998). "A Subpath Ejection Method for the Vehicle Routing Problem." *Management Science* 44(10), 1447–1459.

Russell, R. (1995). "Hybrid Heuristics for the Vehicle Routing Problem with Time Windows." *Transportation Science* 29(2), 156–166.

Russell, R. and D. Gribbin. (1991). "A Multiphase Approach to the Period Routing Problem." *Networks* 21, 747–765.

Savelsbergh, M. and M. Sol. (1985). "The General Pickup and Delivery Problem." *Transportation Science* 29(1), 17–29.

Schrimpf, G., J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. (2000). "Record Breaking Optimization Results Using the Ruin and Recreate Principle." *Journal of Computational Physics* 159, 139–171.

Taillard, É. (1993). "Parallel Iterative Search Methods for Vehicle Routing Problems." *Networks* 23, 661–676.

Thompson, P. and H. Psaraftis. (1993). "Cyclic Transfer Algorithms for Multivehicle Routing and Scheduling Problems." *Operations Research* 41(5), 935–946.

Toth, P. and D. Vigo. (1996). "Fast Local Search Algorithms for the Handicapped Persons Transportation Problem." In I. Osman and J. Kelly (eds.), *Meta-Heuristics: Theory & Application*, chapter 41. Boston: Kluwer Academic, pp. 677–690.

Toth, P. and D. Vigo. (2002a). "Branch-and-Bound Algorithms for the Capacitated VRP." In Toth and Vigo (eds.), *The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications* (2002c), chapter 2, pp. 29–51.

Toth, P. and D. Vigo. (2002b). "An Overview of Vehicle Routing Problems." In Toth and Vigo (eds.), *The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications* (2002c), chapter 1, pp. 1–23.

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

Voß, S., S. Martello, I. Osman, and C. Roucairol. (1999). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Boston: Kluwer Academic.

Walshaw, C. (2002). "A Multilevel Approach to the Travelling Salesman Problem." *Operations Research* 50(5), 862–877.