Sequential Search and its Application to Vehicle-Routing Problems

Stefan Irnich [a,*] Birger Funke [b] Tore Grünert [b]

[a] *Deutsche Post Lehrstuhl für Optimierung von Distributionsnetzwerken, RWTH Aachen University, Templergraben 64, D-52056 Aachen, Germany.*

[b] *GTS Systems and Consulting GmbH, Raiffeisenstr. 10, D-52134 Herzogenrath, Germany.*

Abstract

Local search is the most frequently used heuristic technique for solving combinatorial optimization problems. It is also the basis for modern metaheuristics, like, e.g., Tabu Search, and Variable Neighborhood Search. The paper introduces *sequential search* as a generic technique for the efficient exploration of local-search neighborhoods. One of its key concepts is the systematic decomposition of moves, which allows pruning within local search based on associated partial gains. The application of theoretical concepts to several well-known neighborhoods of the vehicle-routing problem (VRP) is demonstrated. Computational tests show substantial speedup factors, e.g., up to 10 000 for the 3-opt* neighborhood. This underlines the superiority of sequential search over straightforward techniques in the VRP context.

Key words: local search, vehicle-routing problem

1 Introduction

Local search is the most frequently used heuristic technique for solving combinatorial optimization problems. It is also the basis for modern metaheuristics, like, e.g., Tabu Search and Variable Neighborhood Search. Most of the effort spent within a local-search algorithm is used for scanning the neighborhood,

* Corresponding author.
Email addresses: sirnich@or.rwth-aachen.de (Stefan Irnich), funke@gts-systems.de (Birger Funke), gruenert@gts-systems.de (Tore Grünert).

i.e., the set of solutions close to the current solution. It is, therefore, desirable to use efficient algorithms within local search to speed up the subroutine that performs the scan.

Sequential search is a technique that allows neighborhoods within local-search algorithms to be investigated in a highly efficient way. It was discovered independently in the 1970's by Christofides and Eilon (1972) and Lin and Kernighan (1973) in algorithms for the traveling-salesman problem (TSP) and the graph-partitioning problem, Kernighan and Lin (1970). Although the Lin-Kernighan algorithm and its extensions still belong to the best heuristics for solving TSPs, little attention has been paid to applying sequential search to other problems. This might be attributed to the fact that the definition of a sequential search algorithm for a given neighborhood is not a straightforward task and that the principle requires some assumptions that are not met for all kinds of problems and neighborhoods.

In this paper we give a generic description of sequential search and show when and how it can be applied within local-search algorithms. As an example we use classical neighborhoods of the *capacitated vehicle-routing problem (CVRP)*. The CVRP is a good candidate to study, since it is more constrained than the TSP and much more difficult to solve in practice. The intention of the paper is not to present a new and better algorithm for the CVRP. Instead, the paper presents sequential search as a generic technique. It is compared to traditional algorithms for the exploration of the neighborhood, which one can characterize as *lexicographic search* approaches.

The paper is structured as follows. In Section 2, we define the CVRP and review classical and some recent papers on this problem. Section 3 introduces the concepts and terminology of local search. Based on the definition of moves, move decompositions, and gains, a generic description of lexicographic search and sequential search is provided. In Section 4, the general principles of lexicographic and sequential search are applied to different, classical, neighborhoods of the CVRP. This enables a direct comparison between the two methods. In order to show the effectiveness of sequential search, we provide computational results for implementations of both lexicographic and sequential search algorithms for the different neighborhoods of the CVRP in Section 5. Final conclusions are given in Section 6.

2 The Capacitated Vehicle-Routing Problem

The CVRP is one of the basic problems of vehicle routing. In this paper, we study the undirected version of the problem. Let N be a set of *customers* and 0 a depot. Define the undirected graph $G = (V, E)$, where $V = N \cup \{0\}$ is

the node set and E is the edge set. A *tour* $R = (v_0, v_1, \ldots, v_\ell, v_{\ell+1})$ is a cycle in G, which starts at the depot $v_0 = 0$, visits the customers $v_1, \ldots, v_\ell \in N$, and ends at the depot $v_{\ell+1} = 0$. Every customer $i \in N$ has a positive integer *demand* q_i. The demand must be satisfied by a set of F identical *vehicles* with positive integer *capacity* Q. A tour is *(capacity-)feasible* if $\sum_{i=1}^{\ell} q_{v_i} \leq Q$, i.e., if the total demand of the customers does not exceed the capacity of the vehicle. Let the cost of using edge $\{i, j\} \in E$ be c_{ij}. Then the cost of a tour R is $c(R) = \sum_{i=0}^{\ell} c_{v_i, v_{i+1}}$. A set $\{R^1, \ldots, R^F\}$ of F tours is a *tour plan* if it covers each customer exactly once. A tour plan is feasible (i.e., it provides a feasible solution to the CVRP) if all of its tours are feasible.

A variant of the CVRP is the *distance constrained VRP (DCVRP)*. In the DCVRP demand is associated with edges, see e.g. Li *et al.* (1992). Denote by q_{ij} the *travel time/distance* of traversing edge $\{i, j\}$ and by Q the *maximum travel time/distance*. A tour $R = (v_0, v_1, \ldots, v_\ell, v_{\ell+1})$ is *(travel-time-)feasible* if $q(R) = \sum_{i=0}^{\ell} q_{v_i, v_{i+1}} \leq Q$ holds. Given the number F of vehicles, the feasibility problem of a CVRP is equivalent to solving a bin-packing problem, see e.g. Toth and Vigo (1998). However, in cases where travel-time-feasibility is relevant, no such correspondence exists. Two conflicting objective functions have been studied for the CVRP and DCVRP: minimizing the *number of tours* and minimizing *costs*. In this paper we only consider cost minimization, i.e., the objective function $c(R^1, \ldots, R^F) = \sum_{k=1}^{F} c(R^k)$. For the sake of brevity, the paper mainly focuses on the presentation of the CVRP, but all search techniques are also applicable to the CVRP with distance constraints.

The CVRP is one of the most studied problems in combinatorial optimization. Since its first description by Dantzig and Ramser (1959), numerous papers have been published describing models, algorithms, and extensions. Starting in the 1960's with the well-known savings algorithm by Clarke and Wright (1964), the development of so-called classical heuristics continued until the 1980's. These heuristics are classified, described and empirically studied in the paper by Laporte and Semet (2002). Better solutions to the CVRP can be obtained with metaheuristics. Classical papers are the ones by Taillard (1993), Osman (1993), Gendreau *et al.* (1994), and Rochat and Taillard (1995). The main drawback of these algorithms is the heavy computational effort required to get good solutions and the large number of parameters that have to be adjusted. As pointed out in Cordeau *et al.* (2002), the goal has now switched towards obtaining good solutions for larger instances consistently within an acceptable time limit using metaheuristics with a small number of parameters. Recent algorithmic developments by, e.g., Cordeau *et al.* (2001) and Toth and Vigo (2003) support this trend. An overview of metaheuristics for the CVRP can be found in the paper by Gendreau *et al.* (2002).

Exact solutions to the CVRP are obtained by branch-and-bound and branch-and-cut algorithms. Branch-and-bound algorithms are based on different re-

laxations, sometimes within an additive bounding scheme. An overview of different approaches can be found in the paper by Toth and Vigo (2002a). Recently, some advances have been made with LP-based models embedded within the branch-and-cut-scheme. An overview is given in the paper by Naddef and Rinaldi (2002). A promising branch-and-price-and-cut approach is reported in Fukasawa *et al.* (2003). However, none of these exact methods can consistently solve instances with more than 100 customers. Therefore, heuristics and metaheuristics are required for solving larger instances.

3 Local Search

Combinatorial optimization problems can be stated as $\min_{x \in X} c(x)$, where X is the set of feasible solutions and c the cost function. For practically relevant instances of $\mathcal{NP}$-hard combinatorial optimization problems, the set X is in general too large to be searched exhaustively. One of the most popular techniques for searching a subset of feasible solutions is *local search.* Local search is a general heuristic approach that starts with a feasible solution as the current solution and iteratively replaces the current solution by a better and similar, so-called neighbor solution, until no better neighbor solution can be found.

The heart of a local-search procedure is the definition of a *neighborhood* $\mathcal{N}$, which is a mapping $\mathcal{N} : X \rightarrow 2^X : \mathcal{N}(x) \subset X$. Each element $x' \in \mathcal{N}(x)$ is called *neighbor of* x. Neighbors x' with cost $c(x') < c(x)$ are called *improving neighbors*. Local search starts with an initial feasible solution $x^0 \in X$. In each iteration t it replaces the current solution x^t by an improving neighbor $x^{t+1} \in \mathcal{N}(x^t)$, if such an improving neighbor exists. The local-search procedure terminates with a *local optimum*, i.e., a solution x^t for which the neigborhood $\mathcal{N}(x^t)$ contains no improving solution.

Algorithm 1 Generic Local Search

1: **Input:** A feasible solution $x^0 \in X$. LET $t = 0$.
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 $x' \in \mathcal{N}(x^t)$,
5: THEN LET $x^{t+1} = x'$ and $t = t + 1$.
6: UNTIL no more improvements can be found.
7: **Output:** A local optimum x^t.

For further details of local search, we refer the reader to the books of Aarts and Lenstra 1997 and Rayward-Smith *et al.* 1996.

There are several options for choosing improving neighbor solutions in step 3. If the search method is enumerative (i.e., neighbor solutions $x' \in \mathcal{N}(x^t)$ and their costs $c(x')$ are evaluated one after another), taking *the first improving solution* or taking a *best improving solution* are two extreme strategies known as *first*

improvement and *best improvement*. Another well-known strategy, called *d-best improvement*, terminates the search when d improving neighbor solutions have been found. Then the best solution from this set is taken as the next solution. 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.

3.1 Moves and their Decomposition

Neighborhoods are often defined implicitly by a set of *moves*. A move m transforms a solution into a neighbor solution. Some of the moves $m \in M$ might transform a feasible solution x into an object $m(x)$, which has a structure similar to a feasible solution, but does not necessarily satisfy all constraints that define feasible solutions. In the following, we will call such an object a *solution*. An example in the case of the VRP is the swap of two customers between two tours, which might violate a capacity constraint.

For a formal definition of a *move*, it is helpful to consider the set of all *solutions*, $Z \supseteq X$. In general, we denote by M the *set of moves* where a move $m \in M$ is a map from Z to itself, i.e., $m : Z \to Z$. From the above discussion it is clear that a move m maps solutions to solutions. For a given $x \in Z$, the *extended neighborhood* contains all neighbors of x, either feasible or infeasible. Every move, $m \in M$, with $m(x) \in X$ is called a *feasible move* w.r.t. x.

In order to analyze moves, we will decompose them into smaller parts, the so-called *partial moves*. A given decomposition $m = p_\ell \circ \ldots \circ p_2 \circ p_1$ of a move m into $\ell \geq 2$ partial moves $p_1, p_2, \ldots, p_\ell$ means that an $x \in Z$ is first transformed to $p_1(x)$, second $p_1(x)$ is transformed to $p_2(p_1(x))$, and so on. Note that the decomposition of a move into partial moves is *not* uniquely defined by the move itself. In general, there exist various decompositions for the same move, differing in the number ℓ of stages and the structure of the partial moves. In the case of the VRP, there exist two basic types of partial moves. They are the building blocks of more complex (partial) moves, which we will consider later. The partial move p_{ij}^{add} adds the edge $\{i, j\}$ to the current solution and the partial move p_{kl}^{del} deletes the edge $\{k, l\}$ from the current solution.

3.2 Costs and Gains of (Partial) Moves

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 the CVRP, the gain $g(m, x)$ only depends on the added and deleted edges and thus only on the symmetric difference of x and $m(x)$. 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, $i = 1, \ldots, \ell$, depending only on the current solution x and not on the intermediate solutions generated by the partial moves $p_1, \ldots, p_{i-1}$. Let the move $m \in M$ be decomposed into the partial moves $p_\ell \circ \ldots \circ p_2 \circ p_1$. For efficiency reasons it is desirable that the gain of a move m is the sum of the gains of its partial moves $p_1, \ldots, p_\ell$. A corresponding decomposition $m = p_\ell \circ \ldots \circ p_2 \circ p_1$ is called *cost-independent* if

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

holds. If the equality is not fulfilled for all decompositions, then sufficient conditions, which guarantee (1) can be defined. These are referred to as *legitimacy conditions*, cf. Glover (1992). For instance, legitimacy conditions can require that only compatible subsets of partial moves occur simultaneously or restrict the ordering of partial moves.

The decomposition $m = p_l \circ \ldots \circ p_2 \circ p_1$ is *order-independent* if $m(x) = p_{\pi(l)} \circ \ldots \circ p_{\pi(2)} \circ p_{\pi(1)}(x)$ 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 only for cyclic permutations π. We will show below that cyclic-independent move decompositions are essential in the development of efficient search algorithms.

3.3 Search Techniques

In this paper we study two generic search techniques, lexicographic search and sequential search. In order to explain both approaches, we consider the following *generic search problem*: A neighborhood $\mathcal{N}(x)$ of size $\mathcal{O}(n^k)$ is implicitly given by moves $m_{i_1, i_2, \ldots, i_k}$ with $\{i_1, i_2, \ldots, i_k\}$ a subset of $\{1, \ldots, n\}$ of cardinality k.

3.3.1 Lexicographic Search

A natural way to determine the k different elements $i_1 < i_2 < \ldots < i_k$ is to implement k nested loops. The first loop considers the elements $i_1 \in \{1, \ldots, n\}$, the second loop the elements $i_2 \in \{i_1 + 1, \ldots, n\}$, and, in general the p-th loop the elements $i_p \in \{i_{p-1} + 1, \ldots, n\}$. The iterator of an inner loop is always larger than the iterator of an outer loop, i.e., $i_{l+1} > i_l$ holds for all $l \in \{1, \ldots, n-1\}$. Hence, this approach is referred to as *lexicographic search*.

Algorithm 2 Generic Lexicographic Search

1: **Input:** $x \in X$; $G_{min} \in \mathbb{R}$ minimum gain.
2: LET $G^* = G_{min}$.
3: LOOP $i_1 \in \{1, 2, \ldots, n\}$
4: LOOP $i_2 \in \{i_1 + 1, 2, \ldots, n\}$
5: ...
6: LOOP $i_k \in \{i_{k-1} + 1, 2, \ldots, n\}$
7: IF ($m_{i_1,\ldots,i_k}(x) \in X$ AND $G^* < g(m_{i_1,\ldots,i_k}, x)$)
8: LET $G^* = g(m_{i_1,\ldots,i_k}, x)$.
9: LET $(i_1^*, i_2^*, \ldots, i_k^*) = (i_1, i_2, \ldots, i_k)$.
10: **Output:** IF $(G^* > G_{min})$ THEN RETURN $(i_1^*, i_2^*, \ldots, i_k^*)$.

The above search algorithm finds the best (w.r.t. the objective c), not necessarily improving neighbor solution $x' \in \mathcal{N}(x)$. With an aspiration level given by a minimum gain $G_{min} = 0$, the algorithm either finds an improving neighbor $x' = m_{i_1^*,i_2^*,\ldots,i_k^*}(x)$ or returns the information that no such solution exists, i.e., $G^* = 0$.

Step 7 of Algorithm 2 checks whether the neighbor solution is feasible and improving. In order to obtain an efficient search algorithm, it is crucial that the feasibility check and the computation of the gain can be done in constant time. Therefore, a preprocessing for computing additional aggregated information (about substructures of the current solution x) might be necessary to guarantee an $\mathcal{O}(n^k)$ algorithm. We will come back to this important issue in Section 4.8.

3.3.2 Sequential Search

Cyclic-independent neighborhoods are closely linked to the concept of *sequential search*, first developed by Christofides and Eilon (1972) and Lin and Kernighan (1973). The basic idea of this approach is to consider all relevant partial moves of a cyclic independent neighborhood recursively. Sequential search accelerates the search for an improving neighbor solution by pruning the search as early as possible so that (on average) only a small fraction of the entire neighborhood has to be scanned. Pruning is based on an evaluation of partial moves and their corresponding partial gains.

For the description of sequential search we assume that all moves $m_{i_1,i_2,\ldots,i_k}$ have a cyclic-independent and cost-independent decomposition into k partial moves, according to $m_{i_1,i_2,\ldots,i_k} = p_{i_k,i_1} \circ p_{i_{k-1},i_k} \circ \ldots \circ p_{i_2,i_3} \circ p_{i_1,i_2}$. In the case of $k = 3$, it means that m_{i_1,i_2,i_3} decomposes into $p_{i_3,i_1} \circ p_{i_2,i_3} \circ p_{i_1,i_2} = p_{i_2,i_3} \circ p_{i_1,i_2} \circ p_{i_3,i_1} = p_{i_1,i_2} \circ p_{i_3,i_1} \circ p_{i_2,i_3}$. Such a decomposition seems complicated at first, but it makes sense for many types of moves: Typically the gain of a move $m_{i_1,i_2,\ldots,i_k}$ does not depend directly on the individual indices i_l, $l = 1, 2, \ldots, k$, but on combinations of consecutive indices, i.e., (i_j, i_{j+1}) for $j = 1, \ldots, k$ (with the definition $j_{k+1} = j_1$). Examples are partial moves $p_{i_j,i_{j+1}}$, where one element i_j replaces another element i_{j+1} so that the partial gain $g(p_{i_j,i_{j+1}}, x)$ depends on

both, i_j and i_{j+1}. This sequential arrangement of indices has motivated us to call the search technique that exploits this property sequential search.

The attractiveness of sequential search in cyclic- and cost-independent 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* $\sum_{i=1}^k g_i > 0$, *then there is a cyclic permutation* π *of these numbers such that* every partial sum *is positive, i.e.,* $\sum_{i=1}^{\ell} g_{\pi(i)} > 0$ *for all* $1 \leq \ell \leq k$.

The theorem implies that all improving moves $m_{i_1,i_2,\ldots,i_k}$ can be constructed as a sequence of partial moves, where the sum of the partial gain is positive for all sub-sequences. The direct implication is that at stage 1 of the search, we need only consider partial moves with a positive gain, i.e., $g(p_{i_1,i_2},x) > 0$ or $g(p_{i_2,i_3},x) > 0$ or $\ldots g(p_{i_k,i_1},x) > 0$. The cyclic-independence and the symmetry of the decomposition allow us to re-formulate this condition to $g(p_{i_1,i_2},x) > 0$. Note that one can always exchange the indices in a cyclic way such that the first pair is denoted by (i_1,i_2). However, we can no longer postulate $i_1 < i_2$. In general at stage p, one can restrict the search to those partial moves $p_{i_p,i_{p+1}}$ having $g(p_{i_p,i_{p+1}},x) > -G_{p-1}$, where $G_{p-1} = \sum_{l=1}^{p-1} g(p_{i_l,i_{l+1}},x)$ is the accumulated partial gain of the preceding stages. Thus, the total gain G_{p-1} at stage $p-1$ limits the choice of a partial move at stage p. Lin and Kernighan refer to this rule as the *gain criterion*.

An implementation of the gain criterion has to guarantee that the pruning of the search for promising partial moves can be performed efficiently. At stage p, where $i_1, i_2, \ldots, i_{p-1}$ are known, the new index i_p has to be chosen such that $g(p_{i_{p-1},i_p},x) > -G_{p-1}$ holds. For some neighborhoods, the limitation of the search can be done on-the-fly. For typical routing neighborhoods, the search for promising i_p is performed with help of a *neighbor list* associated with i_{p-1}. The neighbor list $NL(i_{p-1})$ is a data structure which stores elements i_p by increasing values of $g(p_{i_{p-1},i_p},x)$. Neighbor lists are computed in a preprocessing step, i.e., before LS is started. The following algorithm summarizes the above description of sequential search.

Algorithm 3 Generic Sequential Search

Input: $x \in X$; $G_{min} \in \mathbb{R}$ minimum gain.
LET $G^* = G_{min}$.
LOOP $i_1 \in \{1,2,\ldots,n\}$
  COMPUTE B_1 (based on i_1).
  LOOP $i_2 \in NL(i_1)$ as long as $g(p_{i_1 i_2},x) > B_1$
    COMPUTE B_2 (based on $g(p_{i_1,i_2},x)$ and i_2).
    LOOP $i_3 \in NL(i_2)$ as long as $g(p_{i_1,i_2},x) + g(p_{i_2,i_3},x) > B_2$
      ...
        COMPUTE B_{k-1} (based on $\sum_{l=1}^{k-2} g(p_{i_l,i_{l+1}},x)$ and i_{k-1}).
        LOOP $i_k \in \{i_{k-1}+1, 2, \ldots, n\}$ as long as $\sum_{l=1}^{k-1} g(p_{i_k,i_{k+1}},x) > B_{k-1}$
          IF ($m_{i_1,\ldots,i_k}(x) \in X$ AND $g(m_{i_1,\ldots,i_k},x) > G^*$)
            LET $G^* = g(m_{i_1,\ldots,i_k},x)$.
            LET $(i_1^*, i_2^*, \ldots, i_k^*) = (i_1, i_2, \ldots, i_k)$.

14: **Output:** IF ($G^* > G_{min}$) THEN RETURN $(i_1^*, i_2^*, \ldots, i_k^*)$.

The crucial point for the efficiency of Algorithm 3 is the computation of the bounds $B_1, B_2, \ldots, B_{k-1}$ in the steps 4, 6, and 9. Bound B_1 takes only index i_1 into account in order to limit the search for i_2 resp. p_{i_1,i_2} with positive gain $g(p_{i_1,i_2}, x) > 0$. In general, at stage ℓ the bound B_ℓ is computed based on the already known partial gains and the index $i_{\ell-1}$ in order to restrict the exploration to indices i_ℓ.

The following corollary provides two extensions of the gain criterion of Lin and Kernighan: First, it allows to search for moves with a (not necessarily positive) gain of at least G^*. Second, whenever a feasible move with gain G^* has been found, the exploration at stage ℓ can be limited to partial moves such that the sum of the partial gains is at least $\ell G^*/k$.

Corollary 1 *Given a number G^*. If a sequence of numbers $(g_i)_{i=1}^k$ fulfills $\sum_{i=1}^k g_i > G^*$, then there is a cyclic permutation π of these numbers such that for* every partial sum $\sum_{i=1}^{\ell} g_{\pi(i)} > \ell G^*/k$ *holds for all $1 \leq \ell \leq k$.*

Proof: Define $g_i' = g_i - G^*/k$ for $i \in \{1, \ldots, k\}$ and use Theorem 1 for the sequence $(g_i')_{i=1}^k$. ◇

4 Neighborhoods and Search Algorithms for the VRP

Local-search procedures for the VRP which replace one tour plan by another have a long tradition, see Kindervater and Savelsbergh (1997). In this section we provide descriptions of both sequential and lexicographic search implementations for different VRP neighborhoods. Since the implementation of lexicographic search is rather straightforward, we will focus on the implementation of sequential search. In the following we will use a representation of tour plans as Hamiltonian cycles. It allows to handle inner-tour moves (i.e., moves, which do only change a single tour) as well as inter-tour moves (i.e., moves which change the assignment of customers to routes) in a unifying way.

4.1 Giant Tour Representation of VRP Solutions

For the definition of neighbor solutions it is important to have a well-defined representation of all elements $x \in X$. This subsection formalizes the *giant tour representation* for the VRP, which has also been used to represent *multiple salesman TSP, MTSP*, see Bellmore and Hong (1974) and Rao (1980). Given a tour plan with F different tours $R^i = (0, v_1^i, v_2^i, \ldots, v_{p_i}^i, 0)$, $i \in \{1, 2, \ldots, F\}$,

$p_i \in \mathbb{N}_0$, the corresponding giant tour uses F copies $0_1, \ldots, 0_F$ of the depot and is defined as $x = (0_1, v_1^1, \ldots, v_{p_1}^1, 0_2, v_1^2, \ldots, v_{p_2}^2, 0_3, v_1^3, \ldots, \ldots, 0_F, v_1^F \ldots, v_{p_F}^F, 0_1)$. Note that a giant tour representation is not unique since tours can be arbitrarily permuted and inverted (in the case of symmetric problems). Any Hamiltonian cycle in the extended graph $G_0 = (V_0, E_0)$, with $V_0 = \{0_1, \ldots, 0_F\} \cup N$ and $E_0 = (E \setminus \{\{0, j\} : j \in N\} \cup \{\{0_i, j\} : i = 1, \ldots, F, j \in N\} \cup \{\{0_i, 0_j\} : i, j = 1, \ldots, F, i \neq j\}$, corresponds to a tour plan. The tour plan is feasible if and only if all sub-paths between two consecutive depot nodes represent feasible tours.

In the context of local search, we assume that a current solution $x = x^t$ is given. In order to describe the associated giant tour, one builds two arrays of length $|V_0|$ which contain the following information: The array *node* is indexed by positions $i \in \{1, 2, 3, \ldots, |V_0|\}$ and $node[i] \in V_0$ is the node of the giant tour at position i. For the sake of convenience, we assume that one can access the elements in a wrap-around fashion, e.g., $node[0] = node[|V_0|]$, $node[|V_0| + 1] = node[1]$, etc. Contrarily, the array *pos* gives for each node $t \in V_0$ the current position $pos[t] \in \{1, 2, 3, \ldots, |V_i|\}$ of the node t in the giant tour.

4.2 Construction of Neighbor Lists

For a node $t \in V_0$, the neighbor list $NL(t)$ stores the nodes $t' \in V_0 \setminus \{t\}$ ordered by non-decreasing values $c_{tt'}$. If only the K first elements of the neighbor list are considered, we obtain the list denoted by $NL^K(t)$ and call it the *candidate list* of node t.

4.3 The 2-Opt Neighborhood*

By replacing two edges from the giant tour by two edges which do not belong to the giant tour, one can construct two types of neighbor tour plans. The union of all these neighbor tour plans define the 2-opt* neighborhood of the current tour plan. We will now describe the two types of neighborhoods by their corresponding moves. The first type of moves, well-known as 2-opt moves and first described by Croes (1958), produces a Hamiltonian cycle. The second type of moves transforms the Hamiltonian cycle into two subtours. These so-called *special 2-opt** or *crossover moves* are easier to describe in the context of sequential search. We, therefore, start with their description.

A special 2-opt* move removes two edges and adds two incident edges in such a way that the giant tour is transformed into two subtours. While such an operation is not feasible in the TSP context, it implies a new tour plan whenever both subtours contain a depot node. The special 2-opt* neighborhood and its generalization of removing and adding k edges, called k-opt* neighborhood, was first introduced by Potvin *et al.* (1989).

Figure 1 shows that a special 2-opt* move is completely determined by two positions i_1, i_2 in the giant tour, since the four involved nodes t_1, t_2, t_3, t_4 are located at the positions i_1, $i_1 + 1$, i_2 and $i_2 + 1$. The symmetry implies a

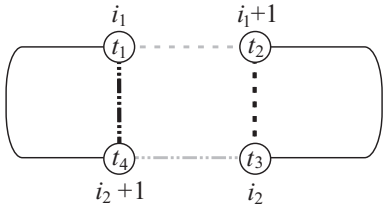

Fig. 1. Principle of a Special 2-Opt*/Crossover Move. Deleted Edges are grey, added Edges black.

decomposition of the special 2-Opt* move $m^{2Opt*}_{i_1,i_2}$ into two partial moves, i.e., $m^{2Opt*}_{i_1,i_2} = p_{i_1,i_2} \circ p_{i_2,i_1} = p_{i_2,i_1} \circ p_{i_1,i_2}$. The partial move p_{i_1,i_2} removes the tour edge linking position i_1 with $i_1 + 1$ and adds the non-tour edge linking the positions $i_1 + 1$ and i_2. Hence, its partial gain is $g(p_{i_1,i_2}, x) = c_{node[i_1],node[i_1+1]} - c_{node[i_1+1],node[i_2]}$. Note that this decomposition is both symmetric and cost-independent. Given position i_1 and the condition that p_{i_1,i_2} has a positive partial gain, the search for suitable positions i_2 can be performed using node-neighbor lists. Let $t_1 = node[i_1]$ and $t_2 = node[i_1+1]$. The node t_3 at position i_2 has to be a neighbor node of node t_2 with $c_{t_2,t_3} < c_{t_1,t_2}$. Therefore, position i_1 defines a bound $B_1 = c_{t_1,t_2}$ and t_3 has to be found among all neighbors of t_2 closer than B_1. The following two algorithms show how the best special 2-opt* neighbor of a given tour plan $x \in X$ can be found either using lexicographic search or sequential search.

Algorithm 4 Lexicographic Search for Special 2-Opt*

```
1:  Input: x ∈ X; G_min ∈ ℝ minimum gain.
2:  LET G* = G_min.
3:  LOOP i1 ∈ {1,2,...,n}
4:      LET t1 = node[i1].
5:      LET t2 = node[i1 + 1].
6:
7:      LOOP i2 ∈ {i1 + 1, i1 + 2,...,n}
8:          LET t3 = node[i2].
9:          LET t4 = node[i2 + 1].
10:         LET G = c_{t1,t2} − c_{t2,t3} + c_{t3,t4} − c_{t4,t1}.
11:         IF ( G > G* and feasible )
12:             LET G* = G.
13:             LET (i1*, i2*) = (i1, i2).
14: Output: IF (G* > G_min) THEN
15:             RETURN (i1*, i2*).
```

Algorithm 5 Sequential Search for Special 2-Opt*

```
1:  Input: x ∈ X; G_min ∈ ℝ minimum gain.
2:  LET G* = G_min.
3:  LOOP i1 ∈ {1,2,...,n}
4:      LET t1 = node[i1].
5:      LET t2 = node[i1 + 1].
6:      LET B1 = c_{t1,t2} − G*/2.
7:      LOOP t3 ∈ NL(t2) as long as c_{t2,t3} < B1
8:          LET i2 = pos[t3].
9:          LET t4 = node[i2 + 1].
10:         LET G = c_{t1,t2} − c_{t2,t3} + c_{t3,t4} − c_{t4,t1}.
11:         IF ( G > G* and feasible )
12:             LET G* = G.
13:             LET (i1*, i2*) = (i1, i2).
14: Output: IF (G* > G_min) THEN
15:             RETURN (i1*, i2*).
```

In step 11 one has to check whether each subtour contains at least one depot and the two new tours respect the capacity constraints. Note that the computation of the bound B_1 uses Corollary 1 with $k = 2$ and $\ell = 1$.

4.3.2 The 2-Opt Neighborhood

A 2-opt move partitions the given (giant) tour into two segments, inverts one of the segments, and rejoins the segments so that a new tour results. Figure 2 shows that a 2-opt move is determined by two edges associated with the positions i_1 and i_2. It can be decomposed into two partial moves $m^{2Opt}_{i_1,i_2} =$

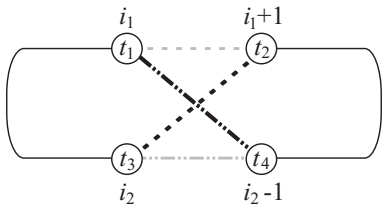

Fig. 2. Principle of a 2-Opt Move.

$p^{+1}_{i_1,i_2} \circ p^{-1}_{i_2,i_1} = p^{-1}_{i_2,i_1} \circ p^{+1}_{i_1,i_2}$, where $p^{\sigma}_{i_1,i_2}$, $\sigma \in \{-1,+1\}$ removes the tour edge linking nodes at the positions i_1 and $i_1+\sigma$ and adds the edge between positions $i_1 + \sigma$ and i_2. In contrast to the 2-opt* move, this decomposition of the 2-opt move is not fully symmetric. Given the partial move defined above with the inserted edge $\{u, v\}$ and the deleted edge $\{v, w\}$, the node w is either the predecessor or the successor of v in the tour. While lexicographic search algorithms can directly take this slight asymmetry into account, sequential search algorithms have to handle it explicitly. The fact that either p^{+1} or p^{-1} has to be improving is taken into account by a double outer loop to determine i_1 (which is the symmetric counterpart of i_2) and $\sigma \in \{-1,+1\}$. The following two algorithms illustrate the similarities and differences in the two implementations of 2-opt lexicographic search and sequential search. They both compute the 2-opt move $p^{\sigma^*}_{i_1^*,i_2^*} \circ p^{-\sigma^*}_{i_2^*,i_1^*}$ with gain at least G^*, if such a move exists.

Algorithm 6 Sequential Search for 2-Opt

1: **Input:** $x \in X$; $G_{min} \in \mathbb{R}$ minimum gain.
2: LET $G^* = G_{min}$.
3: LOOP $i_1 \in \{1, 2, \ldots, n\}$,
4: $\sigma \in \{-1, +1\}$
5: LET $t_1 = node[i_1]$.
6: LET $t_2 = node[i_1 + \sigma]$.
7: LET $B_1 = c_{t_1,t_2} - G^*/2$.
8: LOOP $t_3 \in NL(t_2)$ as long as $c_{t_2,t_3} < B_1$
9: LET $i_2 = pos[t_3]$.
10: LET $t_4 = node[i_2 - \sigma]$.
11: LET $G = c_{t_1,t_2} - c_{t_2,t_3} + c_{t_3,t_4} - c_{t_4,t_1}$.
12: IF ($G > G^*$ and feasible)
13: LET $G^* = G$.
14: LET $(i_1^*, i_2^*, \sigma^*) = (i_1, i_2, \sigma)$.
15: **Output:** IF $(G^* > G_{min})$ THEN
16: RETURN (i_1^*, i_2^*, σ^*).

Algorithm 7 Lexicographic Search for 2-Opt

1: **Input:** $x \in X$; $G_{min} \in \mathbb{R}$ minimum gain.
2: LET $G^* = G_{min}$.
3: LOOP $i_1 \in \{1, 2, \ldots, n\}$
4:
5: LET $t_1 = node[i_1]$.
6: LET $t_2 = node[i_1 + 1]$.
7:
8: LOOP $i_2 \in \{i_1 + 3, i_1 + 4, \ldots, n\}$
9: LET $t_3 = node[i_2]$.
10: LET $t_4 = node[i_2 - 1]$.
11: LET $G = c_{t_1,t_2} - c_{t_2,t_3} + c_{t_3,t_4} - c_{t_4,t_1}$.
12: IF ($G > G^*$ and feasible)
13: LET $G^* = G$.
14: LET $(i_1^*, i_2^*, \sigma^*) = (i_1, i_2, +1)$.
15: **Output:** IF $(G^* > G_{min})$ THEN
16: RETURN (i_1^*, i_2^*, σ^*).

The feasibility check in step 12 has to distinguish two cases. If one of the segments does not contain a depot (i.e., both removed edges belong to the same tour), then the new solution is always capacity-feasible. If both segments contain a depot, the feasibility check is similar to the one for 2-opt* moves.

Since the implementation of lexicographic search for the remaining neighborhoods follows the same pattern as in the case of 2-opt* and 2-opt, we only present sequential search algorithms in the following.

4.4 The Swap Neighborhood

The swap move m_{i_1,i_2}^{swap} replaces the node at position i_1 by a node at position i_2 and vice versa. Consequently, the four edges linking the positions $(i_1 - 1, i_1), (i_1, i_1 + 1), (i_2 - 1, i_2), (i_2, i_2 + 1)$ are deleted and the four edges linking the positions $(i_1 - 1, i_2), (i_2, i_1 + 1), (i_2 - 1, i_1), (i_1, i_2 + 1)$ are added to the current solution, see Figure 3.

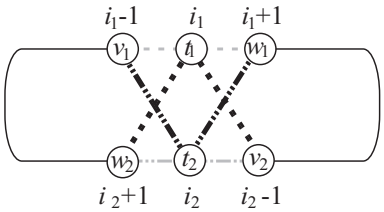

Fig. 3. Principle of a Swap Move.

There exist several decompositions of m_{i_1,i_2}^{swap} into two cost-independent partial moves of the same type. One possibility is to define p_{i_1,i_2} as a partial move which deletes the edges between positions $(i_1 - 1, i_1), (i_1, i_1 + 1)$ and then adds $(i_2 - 1, i_1), (i_1, i_2 + 1)$. Then, $m_{i_1,i_2}^{swap} = p_{i_1,i_2} \circ p_{i_2,i_1} = p_{i_2,i_1} \circ p_{i_1,i_2}$ is a cyclic independent decomposition into cost-independent partial moves. To make the

notation more convenient, define $t_1 = node[i_1]$, $t_2 = node[i_2]$ and v_1, v_2 (resp. w_1, w_2) as the corresponding predecessor (successor) nodes in the giant tour. The partial gain of p_{i_1,i_2} is $g(p_{i_1,i_2}, x) = c_{v_1,t_1} + c_{t_1,w_1} - c_{v_2,t_1} - c_{t_1,w_2}$. 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_{i_1,i_2} with positive gain (but possibly $i_1 > i_2$).

We propose to search for partial moves p_{i_1,i_2} with positive partial gains by first considering all positions $i_1 \in \{1, \ldots, n\}$. The task is then to restrict the search for possible positions i_2 under the condition that i_1 is known. This can be done with neighbor lists. Let $B_1 = (c_{v_1,t_1} + c_{t_1,w_1})/2$, which is a fixed constant when position i_1 is chosen. The condition $g(p_{i_1,i_2}, x) > 0$ is equivalent to $(c_{t_1,v_2} - B_1) + (c_{t_1,w_2} - B_1) < 0$ which implies $c_{t_1,v_2} < B_1$ or $c_{t_1,w_2} < B_1$. This prunes the search for v_2 (resp. w_2) to candidate edges of length less than B_1. Position i_2 is determined as the successor position of node v_2 (resp. the predecessor position of node w_2).

Algorithm 8 Sequential Search for Swap

1: **Input:** $x \in X$; $G_{min} \in \mathbb{R}$ minimum gain.
2: LET $G^* = G_{min}$.
3: LOOP $i_1 \in \{1, 2, \ldots, n\}$,
4: $\sigma \in \{-1, +1\}$
5: LET $v_1 = node[i_1 - 1]$, $t_1 = node[i_1]$, $w_1 = node[i_1 + 1]$.
6: LET $B_1 = (c_{v_1,t_1} + c_{t_1,w_1})/2 \; - \; G^*/2$.
7: LOOP $t \in NL(t_1)$ as long as $c_{t_1,t} < B_1$
8: LET $i_2 = pos[t] + \sigma$.
9: LET $v_2 = node[i_2 - 1]$, $t_2 = node[i_2]$, $w_2 = node[i_2 + 1]$.
10: LET $G = c_{v_1,t_1} + c_{t_1,w_1} + c_{v_2,t_2} + c_{t_2,w_2} - c_{v_1,t_2} - c_{t_2,w_1} - c_{v_2,t_1} - c_{t_1,w_2}$.
11: IF ($G > G^*$ and feasible)
12: LET $G^* = G$.
13: LET $(i_1^*, i_2^*) = (i_1, i_2)$.
14: **Output:** IF $(G^* > G_{min})$ THEN
15: RETURN (i_1^*, i_2^*).

The feasibility check in step 11 has to guarantee both the legitimacy conditions of the decomposition and the feasibility of the resulting tours. The feasibility of the resulting tours only needs to be checked if the nodes t_1 and t_2 belong to different tours. Otherwise the swap only exchanges the positions of the nodes within a single tour, which does not affect the capacity constraints. The legitimacy conditions require that the nodes t_1 and t_2 are not adjacent in the giant tour (i.e., $i_1 \neq i_2 \pm 1$) since if $i_2 = i_1 \pm 1$, the swap move $m_{p,p\pm1}^{swap}$ is feasible, but only exchanges two edges. In this case the computed partial gains do not equal the gain $g(m_{p,p\pm1}^{swap}, x)$. Therefore, the legitimacy conditions forbid this feasible swap move to be computed as a composition of two partial moves. However, it can be shown that this case corresponds to a special 2-opt move and therefore can be computed within a search algorithm for 2-opt or in linear time within an additional loop checking only this special case.

As the name suggests, a string-exchange move takes two subpaths (=strings) of the giant tour and exchanges them. Typically, one restricts the length of the two strings to a small value $k \in \mathbb{N}$ implying a neighborhood of size $\mathcal{O}(k^2 \cdot n^2)$, which is quadratic for fixed values of k. The case $k = 1$ coincides with the swap move. For $k \geq 2$, there exist two variants of string-exchange moves, either inverting segments or not. Figure 4 shows the principle of a string-exchange with inversion. The other case can be handled analogously.

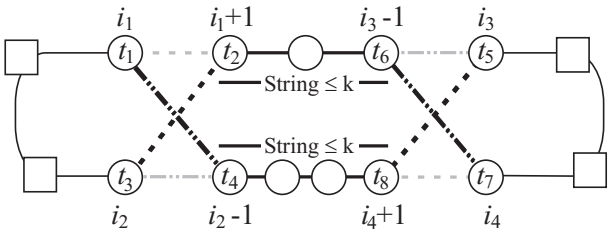

Fig. 4. Principle of a String-Exchange Move. Variant with both Strings inverted.

From Figure 4 one can see that the string-exchange move can be decomposed into four partial moves, i.e., $m^{str-exch}_{i_1,i_2,i_3,i_4} = p^{+1}_{i_1,i_2} \circ p^{-1}_{i_2,i_1} \circ p^{+1}_{i_4,i_3} \circ p^{-1}_{i_3,i_4}$. This decomposition is order-independent. Since the string-exchange move consists of two 2-opt moves $m^{str-exch}_{i_1,i_2,i_3,i_4} = m^{2-opt}_{i_1,i_2} \circ m^{2-opt}_{i_4,i_3}$, the string-exchange move is not completely symmetric in all four partial moves. However, there is a symmetry between the two pairs (i_1, i_2) and (i_4, i_3). This symmetry allows us to restrict the exploration to find an improving partial move $p^{\pm 1}_{i_1,i_2}$ in the first stage (this covers the symmetric counterpart $p^{\pm 1}_{i_4,i_3}$).

Notice that we take the asymmetry in i_1 and i_2 into account considering both $p^{+1}_{i_1,i_2}$ and $p^{-1}_{i_2,i_1}$ at the first stage. Once that i_1 and i_2 are determined, there are only k^2 possibilities to choose i_3 and i_4. The resulting sequential search algorithm can be summarized as follows.

Algorithm 9 Sequential Search for String-Exchange with Segment Inversion

1: **Input:** $x \in X$; $G_{min} \in \mathbb{R}$ minimum gain.
2: LET $G^* = G_{min}$.
3: LOOP $i_1 \in \{1, 2, \ldots, n\}$,
4: $\sigma \in \{-1, +1\}$
5: LET $t_1 = node[i_1]$, $t_2 = node[i_1 + \sigma]$.
6: LET $B_1 = c_{t_1,t_2} - G^*/4$.
7: LOOP $t_3 \in NL(t_2)$ as long as $c_{t_2,t_3} < B_1$
8: LET $i_2 = pos[t_3]$, $t_4 = node[i_2 - \sigma]$.
9: LET $B_2 = c_{t_1,t_2} - c_{t_2,t_3} + c_{t_3,t_4} - c_{t_4,t_1} - G^*/2$.
10: IF $B_2 > 0$
11: LOOP $i_3 \in \{i_1 + 2\sigma, \ldots, i_1 + (k+1)\sigma\}$
12: LET $t_5 = node[i_3]$, $t_6 = node[i_3 - \sigma]$.
13: LOOP $i_4 \in \{i_2 - 2\sigma, \ldots, i_2 - (k+1)\sigma\}$
14: LET $t_7 = node[i_4]$, $t_8 = node[i_4 + \sigma]$.
15: LET $G = c_{t_1,t_2} - c_{t_2,t_3} + c_{t_3,t_4} - c_{t_4,t_1} + c_{t_5,t_6} - c_{t_6,t_7} + c_{t_7,t_8} - c_{t_8,t_5}$.
16: IF ($G > G^*$ and feasible)
17: LET $G^* = G$.
18: LET $(i_1^*, i_2^*, i_3^*, i_4^*, \sigma^*) = (i_1, i_2, i_3, i_4, \sigma)$.
19: **Output:** IF $(G^* > G_{min})$ THEN
20: RETURN $(i_1^*, i_2^*, i_3^*, i_4^*, \sigma^*)$.

The resulting string-exchange move is $p^{\sigma^*}_{i_1^*,i_2^*} \circ p^{-\sigma^*}_{i_2^*,i_1^*} \circ p^{\sigma^*}_{i_4^*,i_3^*} \circ p^{-\sigma^*}_{i_3^*,i_4^*}$. In our implementation, the exchanged strings have to consist of customer nodes only. Therefore, the feasibility check in step 16 reduces to a comparison of the demands of the strings with the residual capacities of the two tours.

4.6 The k-Opt and k-Opt Neighborhoods*

We now consider the generalization of 2-opt* moves to k-opt* moves with $k > 2$. A k-opt move deletes k different edges from a (giant) tour and inserts k other edges so that the result is a Hamiltonian cycle. k-opt* moves allow that the new solution decomposes into at most k subtours. The new solution can be re-interpreted as a new tour plan x' of the VRP if and only if each subtour contains at least one depot node.

Given a k-opt* move m and a tour plan $x \in X$, the symmetric difference of x and $m(x)$ can be interpreted as the result of one or several sequences where incident edges of the solution graph are subsequently deleted and added. Every sequence forms a so-called *alternating cycle*. If a move can be represented as one sequence it corresponds to a *single* alternating cycle. Otherwise, it corresponds to *multiple* alternating cycles. If a move corresponds to a single alternating cycle, it can be decomposed into cyclic independent partial moves of the add-delete type described above. Otherwise, other cyclic independent move decompositions may exist, but give rise to more complex implementations of sequential search that are beyond the scope of this paper. All k-opt* moves with $k \leq 3$ can be represented by single alternating cycles. This is not the case for $k \geq 4$. Therefore, only a subset of the k-opt* moves can be found using sequential search with the delete-add moves described above. For

a more detailed analysis of the so-called *single alternating cycle neighborhoods*, see Funke *et al.* (2004).

4.6.1 The 3-Opt and 3-Opt Neighborhood*

All types of 3-opt and 3-opt* moves define single alternating cycle neighborhoods, i.e., deleted and added edges of these moves form a single alternating cycle $C = (t_1, t_2, t_3, t_4, t_5, t_6, t_1)$, where $\{t_1, t_2\}, \{t_3, t_4\}, \{t_5, t_6\}$ are removed from the current giant tour x and $\{t_2, t_3\}, \{t_4, t_5\}, \{t_6, t_1\}$ are added to the resulting tour. Assuming that the nodes t_1, t_3 and t_5 with odd indeces are at positions i_1, i_2 and i_3, all 3-opt* moves decompose into $m_{i_1,i_2,i_3}^{\sigma_1,\sigma_2,\sigma_3} = p_{i_1,i_2}^{\sigma_1} \circ p_{i_2,i_3}^{\sigma_2} \circ p_{i_3,i_1}^{\sigma_3}$. Figure 5 visualizes the above description. Again, the decomposition is cyclic-

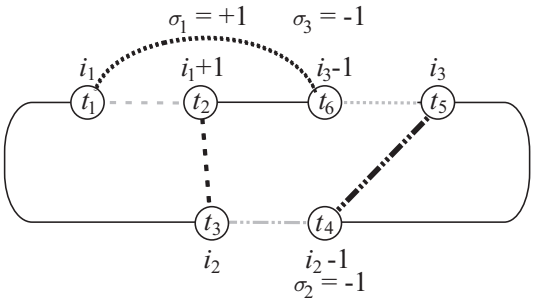

Fig. 5. Example of a 3-Opt* Move. Move depicted here produces two Subtours.

independent. The sequential search algorithm is now easy to formulate:

Algorithm 10 Sequential Search for 3-Opt*

1: **Input:** $x \in X$; $G_{min} \in \mathbb{R}$ minimum gain.
2: LET $G^* = G_{min}$.
3: LOOP $i_1 \in \{1, 2, \ldots, n\}$,
4: $\sigma_1 \in \{-1, +1\}$
5: LET $t_1 = node[i_1]$, $t_2 = node[i_1 + \sigma_1]$.
6: LET $B_1 = c_{t_1,t_2} - G^*/3$.
7: LOOP $t_3 \in NL(t_2)$ as long as $c_{t_2,t_3} < B_1$
8: $\sigma_2 \in \{-1, +1\}$
9: LET $i_2 = pos[t_3]$, $t_4 = node[i_2 + \sigma_2]$.
10: LET $B_2 = c_{t_1,t_2} - c_{t_2,t_3} + c_{t_3,t_4} - 2G^*/3$.
11: LOOP $t_5 \in NL(t_4)$ as long as $c_{t_3,t_4} < B_2$
12: $\sigma_3 \in \{-1, +1\}$
13: LET $i_3 = pos[t_5]$, $t_6 = node[i_3 + \sigma_3]$.
14: LET $G = c_{t_1,t_2} - c_{t_2,t_3} + c_{t_3,t_4} - c_{t_4,t_5} + c_{t_5,t_6} - c_{t_6,t_1}$.
15: IF ($G > G^*$ and feasible)
16: LET $G^* = G$.
17: LET $(i_1^*, i_2^*, i_3^*, \sigma_1^*, \sigma_2^*, \sigma_3^*) = (i_1, i_2, i_3, \sigma_1, \sigma_2, \sigma_3)$.
18: **Output:** IF $(G^* > G_{min})$ THEN
19: RETURN $(i_1^*, i_2^*, i_3^*, \sigma_1^*, \sigma_2^*, \sigma_3^*)$.

The feasibility check is difficult to implement: The three positions i_1, i_2, i_3 and directions $(\sigma_1, \sigma_2, \sigma_3)$ split the giant tour into three segments. One or several subtours are built by concatenation and one has to check whether these subtours form one or several feasible VRP tours. Additional legitimacy conditions have to be checked in step 15. They require that all added and deleted edges are disjoint. However, Section 4.8 will show that checking the

feasibility of the concatenation of several tour sub-paths can always be done in constant time.

4.6.2 Or-Opt and Relocation Neighborhoods

Or-opt and relocation moves are special 3-opt moves, which relocate a short segment, i.e., a segment is first removed and subsequently inserted at different position in the (giant) tour. While in a *relocation move*, which is also called 2.5-opt in Bentley (1992); Johnson and McGeoch (1997), the short segment is restricted to contain a single node, Or-opt moves (Or, 1976) relocate a string of length k, typically with $k \in \{1, 2, 3\}$. Hence, for fixed k the neighborhood is quadratic of size $\mathcal{O}(k \cdot n^2)$. The move decomposes into $m^{or-opt}_{i_1,i_2,i_3} = p^{+1}_{i_1,i_2} \circ p^{+1}_{i_2,i_3} \circ p^{+1}_{i_3,i_1}$.

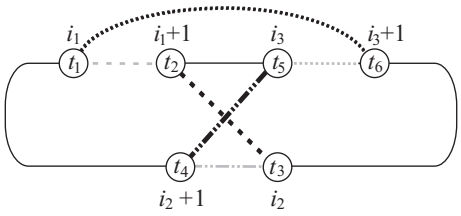

Fig. 6. Principle of an Or-Opt Move. At least one Segment has to be short, w.l.o.g. the Segment $(t_2, \ldots, t_5)$.

Searching for a best neighbor solution $x' \in \mathcal{N}(x)$ in the Or-opt (resp. relocation, $k = 1$) neighborhood has to take the asymmetry into account, i.e., that the length of one of the segments does not exceed k. Without loss of generality one can assume that the segment $(t_2, \ldots, t_5)$ is short, which means that $i_3 \in \{i_1 + 1, \ldots, i_1 + k\}$ or equivalently $i_1 \in \{i_3 - k, \ldots, i_3 - 1\}$ holds, see Figure 6. As before, the gain criterion tells us that for an Or-opt move to be improving, at least one of the three partial moves must have a positive gain, i.e., $g(p^{+1}_{i_1,i_2}, x) > 0$ or $g(p^{+1}_{i_2,i_3}, x) > 0$ or $g(p^{+1}_{i_3,i_1}, x) > 0$. This can be used in three different loops leading to the following algorithm.

Algorithm 11 Sequential Search for Or-Opt

```
1:  Input: x ∈ X; G_min ∈ ℝ minimum gain.
2:  LET G* = G_min.
3:  LOOP i_1 ∈ {1,2,...,n},
4:      LET t_1 = node[i_1], t_2 = node[i_1 + 1], B_1 = c_{t_1,t_2} − G*/3.
5:      LOOP t_3 ∈ NL(t_2) as long as c_{t_2,t_3} < B_1
6:          LET i_2 = pos[t_3], t_4 = node[i_2 + 1].
7:          LOOP i_3 ∈ {i_1 + 1,...,i_1 + k}
8:              LET t_5 = node[i_3], t_6 = node[i_3 + 1].
9:              ... Steps 14–17 of Algorithm 10
10: LOOP i_2 ∈ {1,2,...,n},
11:     LET t_3 = node[i_2], t_4 = node[i_2 + 1], B_1 = c_{t_3,t_4} − G*/3.
12:     LOOP t_5 ∈ NL(t_4) as long as c_{t_4,t_5} < B_1
13:         LET i_3 = pos[t_5], t_6 = node[i_3 + 1].
14:         LOOP i_1 ∈ {i_3 − k,...,i_3 − 1}
15:             LET t_1 = node[i_1], t_2 = node[i_1 + 1].
16:             ... Steps 14–17 of Algorithm 10
17: LOOP i_3 ∈ {1,2,...,n},
18:     LET t_5 = node[i_3], t_6 = node[i_3 + 1].
19:     LOOP i_1 ∈ {i_3 − k,...,i_3 − 1}
20:         LET t_1 = node[i_1].
21:         IF c_{t_5,t_6} − c_{t_6,t_1} > G*/3 THEN
22:             LET t_2 = node[i_1 + 1].
23:             LET B_2 = c_{t_5,t_6} − c_{t_6,t_1} + c_{t_1,t_2} − 2G*/3.
24:             LOOP t_3 ∈ NL(t_2) as long as c_{t_2,t_3} < B_2
25:                 LET i_2 = pos[t_3], t_4 = node[i_2 + 1].
26:                 ... Steps 14–17 of Algorithm 10
27: Output: IF (G* > G_min) THEN
28:             RETURN (i_1*, i_2*, i_3*).
```

Note that the inner loop in step 7 (resp. 14, 19) implies only a constant effort $\mathcal{O}(k) = \mathcal{O}(1)$. Therefore, each of the three blocks requires $\mathcal{O}(n^2)$ operations in the worst-case, but fewer operations on average due to the gain criterion. The feasibility test has to check the legitimacy condition for Or-opt moves in all three blocks. This means that the positions i_1, i_2 and i_3 fulfill either $i_1 < i_3 < i_2$ or $i_3 < i_2 < i_1$ or $i_2 < i_1 < i_3$.

4.7 Other Neighborhoods

The report Funke *et al.* (2003) shows that node-ejection-chains and cyclic-transfers are additional examples of cyclic-independent and cost-independent neighborhoods. Hence, the gain criterion and sequential search are applicable.

4.8 Feasibility Checking in Constant Time

All CVRP moves presented in this section can be described as edge exchanges, i.e., the splitting of the giant tour into $\ell \geq 2$ segments and the concatenation of (possibly inverted) segments to one or several subtours of the giant tour. For a subtour to be feasible, it has to contain at least one depot node. In order to check whether a segment $S = (t, \ldots, t')$ contains a depot node, one performs a linear time preprocessing to build a vector *tour*, which gives, for each node t,

the corresponding tour index in the giant tour. Hence, $S = (t, \ldots, t')$ contains a depot if one of the endpoints t or t' is a depot node or $tour[t] \neq tour[t']$.

The feasibility of the concatenation of segments $S_1, S_2, \ldots, S_\ell$ depends on the accumulated demand in each subpath joining two depot nodes. Assume (w.l.o.g) that S_1 contains a depot and that S_p, $p \in \{2, \ldots, \ell\}$ is the next segment, which also contains a depot node (if all remaining segments do not contain a depot, let $S_{\ell+1} := S_1$ and $p = \ell + 1$). Now, the concatenation of $S_1, S_2, \ldots, S_p$ is feasible w.r.t. the vehicle capacity if

- the accumulated demand $q^{end}(S_1)$ from the last depot contained in S_1 to the last node of S_1,
- plus the sum of the accumulated demands $q^{all}(S_q)$ along the segments S_q, $q \in \{2, \ldots, p-1\}$,
- plus the accumulated demand $q^{start}(S_p)$ from the first node of S_p to the first depot contained in S_p

does not exceed the vehicle capacity, i.e., $q^{end}(S_1) + \sum_{q=2}^{p-1} q^{all}(S_q) + q^{start}(S_1) \leq Q$ holds. In order to have constant-time access to the above values, we propose to build two vectors qs and qe, indexed by the nodes V of the giant tour, containing the following values: $qs[t]$ is the accumulated demand (accumulated along the giant tour) of the path starting at node t and ending at the next succeeding depot node. Similarly, $qe[t]$ is the accumulated demand of the path starting at the last depot node, which precedes node t, to node t. The computation of the vectors qs and qe requires linear time only when nodes are considered in ascendind/descending order of the giant tour. For any segment $S = (t, \ldots, t')$ one has $q^{start}(S) = qs[t]$ and $q^{end}(S) = qe[t']$. If S does not contain a depot node, then $q^{all}(S) = qs[t'] - qs[t] + q_t = qe[t] - qe[t'] + q_{t'}$ holds. With $tour$, qs and qe a priori computed, the concatenation of $S_1, \ldots, S_p$ can be checked in constant time. Subsequent concatenations of segments $S_p, S_{p+1} \ldots, S_k$ can be handled analogously, so that the overall feasibility check requires constant time only.

It is worth to mention that the same kind of techniques allow constant-time feasibility checks for the DCVRP. The only difference to the CVRP is that one has to consider inter-connection times and distances in the concatenation of segments.

5 Computational Results

The following computational results are based on a large number of randomly generated large-scale CVRP instances. We have examined the running time of sequential and lexicographic search algorithms, the impact of

best-improvement and first-improvement stopping rules on running time and solution quality, and similarly, the impact of using candidate lists.

5.1 Test Instances

The benchmark instances are generated by varying the number of customers, the customer distribution within the Euclidian plane, the demand distribution of customers, and the vehicle capacity. Altogether there are 600 instances grouped into 10 series of 60 instances each. Any series include instances of 15 different sizes, i.e., number $n = |N|$ of customers, ranging from $n = 250$ to $n = 2500$. For each size there are four different demand distributions. Table 1 summarizes the series. Customers $i \in N$ are located at integers point (x_i, y_i) of

Series	Location (x_i, y_i) uniform from	Demand q_i uniform from	Capacity Q
1	$[-100, +100]^2$	$[10, 30]$	$\{500, 1000, 1500, 2000\}$
2	$[-100, +100]^2$	$[10, 50]$	$\{750, 1500, 2250, 3000\}$
3	$[-100, +100]^2$	$[10, 90]$	$\{1250, 2500, 3750, 5000\}$
4	$[-100, +100]^2$	$[1, 99]$	$\{1250, 2500, 3750, 5000\}$
5	$[-100, +100]^2$	$[90, 110]$	$\{2500, 5000, 7500, 10000\}$
6	$[-1000, +1000]^2$	$[1, 1]$	$\{25, 50, 75, 100\}$
7	$[-1000, +1000]^2$	$[1, 3]$	$\{50, 100, 150, 200\}$
8	$[-1000, +1000]^2$	$[50, 150]$	$\{2500, 5000, 7500, 10000\}$
9	$[-1000, +1000]^2$	$[8, 12]$	$\{250, 500, 750, 1000\}$
10	$[-1000, +1000]^2$	$[100, 300]$	$\{5000, 10000, 15000, 20000\}$

Table 1
Generation of CVRP Test Instances, Distribution of Customers and Demands, Choice of Capacities

the 2-dimensional Euclidian plane using uniform numbers from $[-100, +100]^2$ or from $[-1000, +1000]^2$.

The cost c_{ij} is the Euclidian distance rounded to the next integer, as described in Reinelt (1991). The demands q_i are generated using different integer uniform distributions as, e.g., demands in a small range $[90, 110]$, or in a wide range $[1, 99]$, or unit demands with $q_i = 1$ for all $i \in N$. The vehicle capacity Q is determined by multiplying the average demand by factors $f = 25$, 50, 75, and 100. Hence, the generated instances have an average number of f customers on each tour. All instances are also available at `www.dpor.rwth-aachen.de/vrp-instances`.

The first part of the computational study compares the times for searching the neighborhoods discussed in Section 4 with either a sequential search or a lexicographic search algorithm. The comparison is based on first computing four different starting solutions for each of the 600 instances using a parametrized savings algorithm, see Clarke and Wright (1964) and Paessens (1988).

For each of the starting solutions a local optimum w.r.t. all neighborhoods is computed in the following way. Given the current solution x, one searches for a best improving 2-opt neighbor solution $x' \in \mathcal{N}(x)$. This is done by applying both the sequential and the lexicographic search variant of the 2-opt algorithm to the current solution x. If an improving neighbor is found, it becomes the new current solution x. Next, the remaining neighborhoods, i.e., 2-opt*, swap, relocation, Or-opt, and string-exchange, are searched in the same manner. Then, 2-opt will be repeated and so on. Sequential and lexicographic algorithms are always compared using the same current solution x. If the corresponding gain is positive, the move is performed. (Note: Sequential search and lexicographic search algorithms do not necessarily find the same neighbor with maximum gain due to degeneracy. The next iteration was always continued with the sequential search solution.) The whole run terminates when a solution is found which cannot be improved by any neighborhood under consideration. This search strategy is called variable neighborhood descent (VND, see Mladenović and Hansen (1997); Hansen and Mladenović (2001, 2002)). In order to make the search more symmetric, we decided to perform the neighborhood selection in a cyclic way (which is in fact a small variation of VND).

All algorithms were coded in C++, compiled in release mode (using MS Visual Studio 6.0), and run on a standard PC (Intel x86 family 15 model 2, 2.4 GHz, 1GB main memory, on MS-Win 2000).

Figure 7 depicts the *acceleration factor*, i.e., the ratio of the time spent in lexicographic search divided by the time of sequential search, for each of the six quadratic neighborhoods under consideration. Each point (=factor) in the diagram corresponds to a fixed pair of size n and average number f of customers in a tour. It is computed considering several thousands of runs (10 series, 4 starting solutions, from 35 up to 500 iterations). Note that we have chosen the maximum string length $k = 3$ for Or-opt and relocation neighborhoods.

All six diagrams show that there is a substantial speedup when the classical lexicographical search approach is replaced by a sequential search procedure. The most remarkable insight is that for all neighborhoods considered, the acceleration factor mainly depends on the average number f of customers in a route. The smaller f is, the more constrained is the problem instance and the

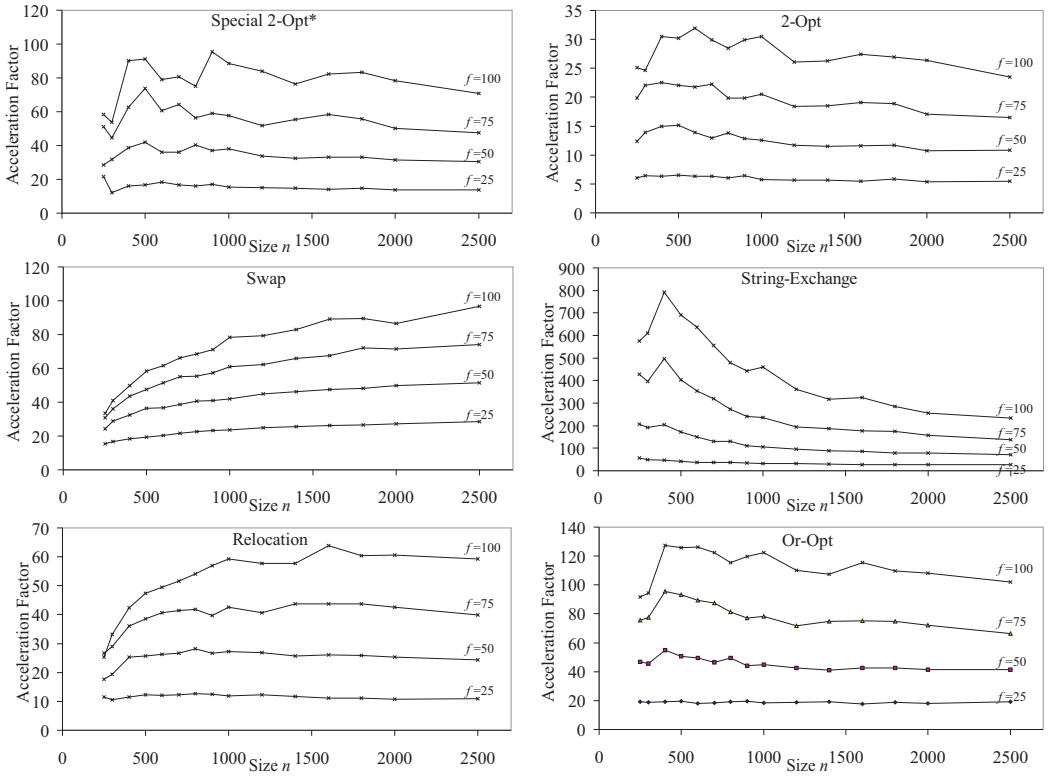

Fig. 7. Acceleration Factor Comparing the Running Times of Lexicographic Search and Sequential Search Algorithms.

smaller is the acceleration factor. One may interpret the results in the following way. In more constrained problems (i.e., with smaller values f) there are more improving moves which are not necessarily feasible. Since sequential search prunes the search only based on cost (and not on feasibility) considerations, it is less effective in problems with tighter constraints.

A second aspect to analyze is the correlation between the acceleration factor and the size of the instances. Considering the diagrams it can be recognized that there is a positive correlation between the acceleration factor and n for the swap and relocation neighborhoods. For special 2-opt*, 2-opt, and Or-opt neighborhoods, there is obviously no significant correlation. That means that there is no decrease in the speedup. Large acceleration factors can be observed when comparing lexicographic and sequential search algorithms for the string-exchange neighborhood. The factor is in the range between 24 and more than 750, while the maximum string length was chosen as $k = 3$. The string-exchange neighborhood is the only neighborhood under consideration, for which the acceleration drastically decreases with the size n of the instances. At the first glance it seems that for larger instances, the sequential search approach might become slower than the lexicographical search approach. This is not the case. Additional computational tests for larger instances with up to 5000 customers have shown that the acceleration never falls below a certain

threshold (never below factor 20 for $f = 25$). The decrease of the acceleration factor with increasing n depends on the check in step 10 of Algorithm 8. In an earlier implementation, with corresponding results presented in Irnich *et al.* (2004), we forgot to prune the search according to the criterion $B_2 > 0$. As a result, the acceleration factor was nearly constant for fixed f, but substantially smaller. It is an open research problem why the behavior differs for the types of neighborhoods we considered.

Since the lexicographic search for improving 3-opt* neighbors is too time-consuming, we modified the computational tests in the following way. The 3-opt* search procedures are only applied to current solutions which are local optima w.r.t. all other (quadratic) neighborhoods. If an improving 3-opt* neighbor is found, the search is again directed to determine a local optimum of the quadratic neighborhoods. Consequently, the number of computationally costly searches within the 3-opt* neighborhood is small in comparison to the number of searches in the quadratic neighborhoods. However, due to the fast growing effort of lexicographic search in the 3-opt* neighborhood, we considered instances with $n \leq 500$ only. Results for the 3-opt* neighborhood are depicted in Figure 8. The acceleration factor is between 300 for $f = 25$

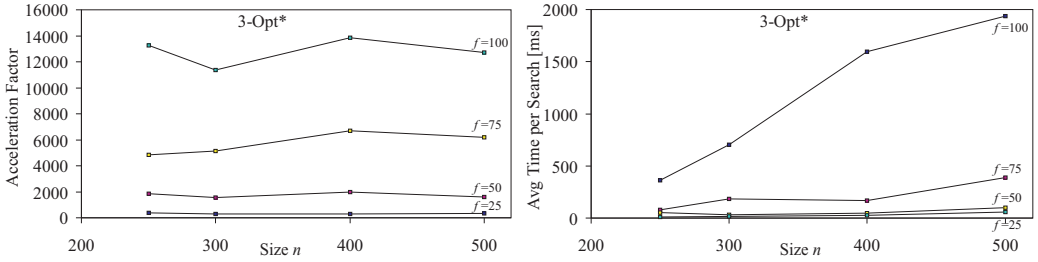

Fig. 8. Acceleration Factor Comparing Lexicographic Search and Sequential Search Algorithms, Average Running Times of a Single Sequential Search Iteration for 3-Opt* moves.

and more than 10.000 for $f = 100$. The time for a single sequential search iteration for the 3-opt* neighborhood (see also Section 5.3) ranges from a few milliseconds to about 2 seconds for large-scale instances with 500 customers and $f = 100$. Both, the acceleration factor and the time for a single iteration, explain why lexicographic search is computationally intractable for even larger problem instances.

5.3 Running Time of Sequential Search Procedures

Next we analyze the average running time of a single sequential search step (i.e., step 3 in Algorithm 1) depending on both, the size n of the instance and the average number f of customers in a tour. Figure 9 summarizes the results.

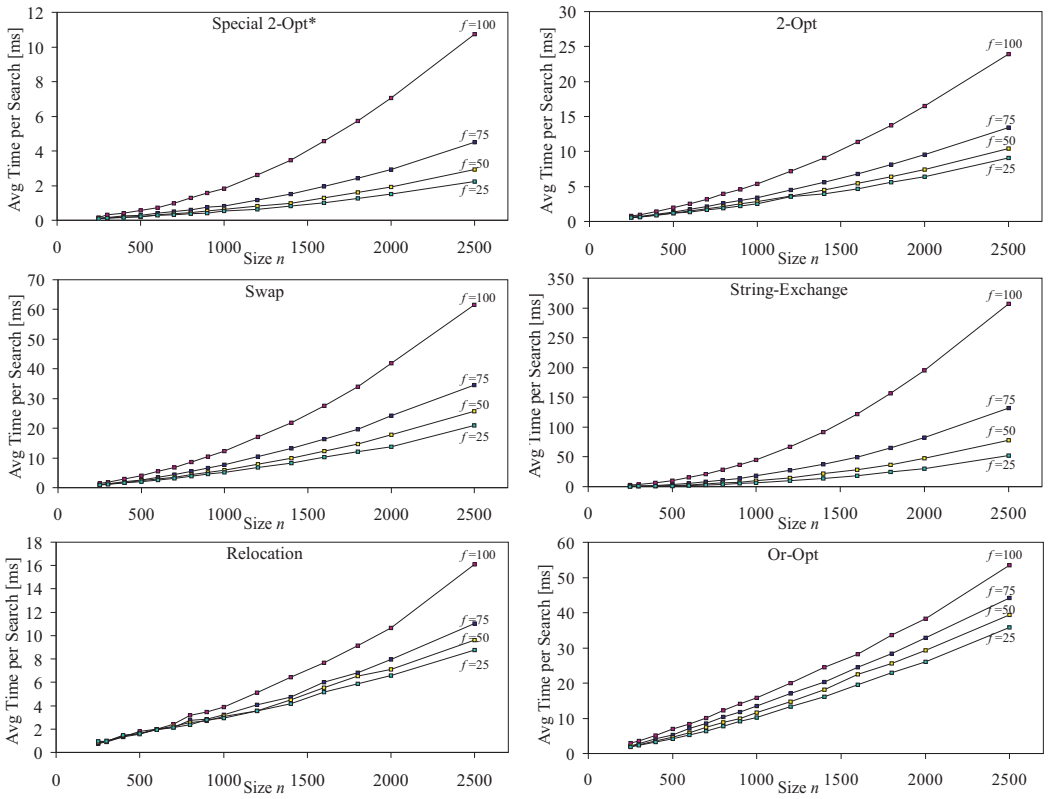

Fig. 9. Average Running Times of a Single Sequential Search Iteration Depending on the Size n and the Number f of Customers per Tour.

These curves can be analyzed with different regression models, e.g., quadratic $t(n) = \alpha n^2 + \beta n$, potential $t(n) = \alpha n^\beta$, log-linear $t(n) = \alpha n + \beta n \log n$, etc. In all cases a quadratic regression model with a very small coefficient for the quadratic term fits well (with $R^2 > 0.99$). Our interpretation of that finding is that always a small but quadratic number (growing with n) of moves has to be scanned, in particular when they have a positive gain but are not improving (i.e., they seem promising but are infeasible w.r.t. the current solution).

For some types of moves, a potential regression model gives a tight approximation saying, e.g., that the average running time of 2-opt is $t^{2-opt}(n) \approx \alpha n^{1.65}$, of Or-opt* is $t^{Or}(n) \approx \alpha n^{1.35}$, and of string-exchange is $t^{str-exch}(n) \approx \alpha n^2$.

5.4 First-Improvement versus Best-Improvement

The previous computational results were all based on a best-improvement (BI) strategy, see also Section 3. In this section we will quantify the relationship of best-improvement and first-improvement (FI) sequential search algorithms w.r.t. running time and quality of the computed local optima. Hansen and Mladenović (1999) already made comparisons for the symmetric TSP and their

findings were the following. Starting from a random solution, FI computes better results that BI. Conversely, starting local search from a given greedy or nearest-neighbor solution, BI is better and faster than FI on average.

Here, we will analyze the running time and solutions quality for both strategies applied to savings solutions of the CVRP. We use the 40 different instances with $n = 1000$ customers (10 series, $f = 25, 50, 745, 100$) and compute 18 (different) starting solutions using a parametrized savings algorithm. Starting with the same initial solutions, BI and FI are separately applied in 2-opt, 2-opt*, swap, relocation, Or-opt, and string-exchange local-search procedures until a local optimum to all neighborhoods is found.

First, in contrast to the results of the previous sections, the value of f has no significant impact on the solution quality or running time comparing BI and FI. In about 85% of all 720 cases, FI was faster than BI. However, the average running time of BI (=14.1s) exceeds the average running time of FI (=12.9s) by only 9%. The reason for this is that each iteration of the FI local search takes less time, but more iterations are needed to reach a local optimum.

In 55% of the cases, BI was better than FI. On average, BI terminates in a local optimum 3.11% and FI 3.18% above the best known solution. This average improvement of BI over FI of 0,07% is small. Hence, the results of Hansen and Mladenović (1999) of the STSP cannot be projected to the CVRP.

5.5 Candidate Lists

In order to further accelerate local search one might use candidate lists $NL^K(t)$ instead of a complete neighbor list $NL(t)$ containing all nodes $V \setminus \{t\}$. With the same setup as in Section 5.4, we compare implementations using candidate lists $NL^K(t)$ with $K \in \{10, 20, 40, 80\}$ and a complete neighbor list $NL(t)$, i.e., $K = n + F - 1$. The tradeoff between running time and solution quality is measured by comparing the average running time *avg time* (until a local optimum w.r.t. all quadratic neighborhoods is found) with the average ratio $\frac{obj-best}{best}$. Herein, *obj* is the cost of the computed local optimum and *best* the cost of the best known solution of the instance. Figure 10 shows the results. Using candidate list of moderate size can further reduce the running time, while the solution quality decreases slowly. For instance, for $K = 40$ and $n = 1000$ customer instances under consideration, the time reduces on average by 60% compared to the full neighbor list implementation and the loss in solution quality is about 0.1%.

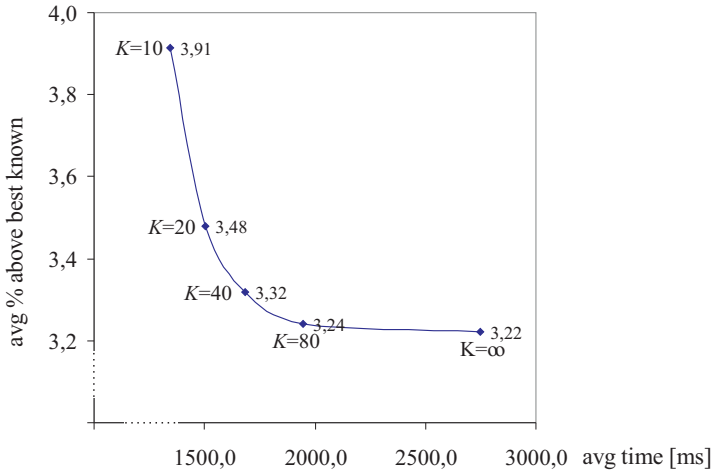

Fig. 10. Comparison of Running Time and Solution Quality for Candidate Lists $NL^K(t)$ of Different Length K

6 Conclusions

The paper has introduced sequential search as general and highly efficient technique for scanning neighborhoods within local-search algorithms. It can be applied both within classical local-search algorithms and modern metaheuristics based on local search. These include tabu search (Glover and Laguna 1997), variable neighborhood descent and variable neighborhood search (Mladenović and Hansen 1997; Hansen and Mladenović 2001, 2002), GRASP (Festa and Resende 2004) and large-step Markov chains (Martin *et al.* 1992).

Based on the formal description of moves and move decompositions, it was shown that the necessary conditions for applying sequential search depend on both the problem and the partial moves that constitute the neighborhood. The condition of cost-independence depends on the objective function of the problem, whereas the condition of cyclic independence follows from the definition of the partial moves.

The paper shows that sequential search algorithms can be developed for the classical neighborhoods of the CVRP. As indicated by the computational results, the efficiency is increased significantly in comparison to classical lexicographic search implementations.

The main challenge in the development of new sequential search algorithms is to find a decomposition of moves into partial moves that satisfy the necessary conditions for applying sequential search. If successful, this can lead to the development of new and significantly faster local-search algorithms and metaheuristics for many types of combinatorial optimization problems.

References

Aarts, E. and Lenstra, J. (1997). *Local search in combinatorial optimization*. Wiley, Chichester.

Bellmore, M. and Hong, S. (1974). Transformation of multisalesmen problem to the standard traveling salesman problem. *Journal of the Association for Computing Machinery*, **21**(3), 500–504.

Bentley, J. (1992). Fast algorithms for geometric traveling salesman problems. *Operations Research Society of America*, **4**(4), 387–411.

Christofides, N. and Eilon, S. (1972). Algorithms for large-scale travelling salesman problems. *Operational Research Quarterly*, **23**(4), 511–518.

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

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

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

Croes, G. (1958). A method for solving traveling-salesman problems. *Operations Research*, **6**, 791–812.

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

Festa, P. and Resende, G. (2004). An annoted bibliography of GRASP. Technical report TD-5WYSEW, AT&T Labs Research.

Fukasawa, R., Marcus Poggi de Aragão, M. R., and Uchoa, E. (2003). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Relatórios de Pesquisa em Engenharia de Produção Vol.3, no.8, Departamento de Engenharia de Produção, Universidade Federal Fluminense R. Passo da Pátria, 156, Bloco E, sala 440, Niterói, Brasil, 24210-240.

Funke, B., Grünert, T., and Irnich, S. (2003). Local search for vehicle routing and scheduling problems: Review and conceptual integration. Technical report, Lehr- und Forschungsgebiet Operations Research und Logistik Management, RWTH Aachen, Templergraben 64, 52056 Aachen.

Funke, B., Grünert, T., and Irnich, S. (2004). A note on single alternating cycle neighborhoods for the TSP. *Journal of Heuristics*, **to appear**.

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

Gendreau, M., Laporte, G., and Potvin, J. (2002). Metaheuristics for the capacitated VRP. In Toth and Vigo (2002b), chapter 6, pages 129–154.

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

Glover, F. (1992). Ejection chains, reference structures and alternating path methods for traveling salesman problems. Technical report, US West Chair in Systems Science, University of Colorado, Boulder, School of Business, Campus Box 419, Boulder, CO, 80309.

Hansen, P. and Mladenović, N. (1999). First vs. best improvement: An empirical study. Technical Report Les Cahiers du GERAD G-99-40, École des Hautes Études Commerciales, Montréal, Canada. Revised: January 2004.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, **130**(1), 449–467.

Hansen, P. and Mladenović, N. (2002). Developments of variable neighborhood search. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, Operations Research/Computer Science Interfaces Series, chapter 19, pages 415–439. Kluwer, Boston.

Irnich, S., Funke, B., and Grünert, T. (2004). Efficient local search for cvrp, move decomposition and sequential search. Extended Abstract, TRISTAN V, Fifth Triennial Symposium on Transportation Analysis, Guadeloupe, France.

Johnson, D. and McGeoch, L. (1997). The traveling salesman problem: A case study in local optimization. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 8, pages 215–310. Wiley, Chichester.

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

Kindervater, G. and Savelsbergh, M. (1997). Vehicle routing: Handling edge exchanges. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 10, pages 337–360. Wiley, Chichester.

Laporte, G. and Semet, F. (2002). Classical heuristics for the capacitated VRP. In Toth and Vigo (2002b), chapter 5, pages 109–128.

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

Li, C., Simchi-Levi, D., and Desrochers, M. (1992). On the distance constrained vehicle routing problem. *Operations Research*, **40**(4), 790–799.

Martin, O., Otto, S., and Felten, E. (1992). Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, **11**(4), 219–224.

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

Naddef, D. and Rinaldi, G. (2002). Branch-and-Cut algorithms for the capacitated VRP. In Toth and Vigo (2002b), chapter 3, pages 53–84.

Or, I. (1976). *Traveling Salesman-Type Problems and their Relation to the Logistics of Regional Blood Banking*. Ph.D. 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.

Paessens, H. (1988). The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, **34**, 336–344.

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

Rao, M. (1980). A note on multiple traveling salesman problem. *Operations Research*, **28**(3 *PartI*), 628–632.

Rayward-Smith, V., Osman, I., Reeves, C., and Smith, G. (1996). *Modern heuristic search methods*. Wiley, Cichester.

Reinelt, G. (1991). TSPLIB - A traveling salesman problem library. *ORSA Journal on Computing*, **3**(4), 376–384.

Rochat, Y. and Taillard, É. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, **1**, 147–167.

Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, **23**, 661–676.

Toth, P. and Vigo, D. (1998). Exact solution of the vehicle routing problem. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 1, pages 1–32. Kluwer Academic Publishers, Norwell, Massachusetts and Dordrecht, The Netherlands.

Toth, P. and Vigo, D. (2002a). Branch-and-Bound algorithms for the capacitated VRP. In Toth and Vigo (2002b), chapter 2, pages 29–51.

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

Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, **15**(4), 333–346.