Vehicle Routing 2
Handling Side Constraints

G.A.P. Kindervater
Erasmus University
P.O. Box 1738, 3000 DR Rotterdam
The Netherlands

M.W.P. Savelsbergh
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205
U.S.A.

1 INTRODUCTION

Physical distribution management presents a variety of decision making problems at the three levels of strategic, tactical and operational planning. Decisions relating to the location of facilities (plants, warehouses or depots) may be viewed as strategic, while problems of fleet size and fleet mix determination can be termed tactical. On the operational level, two problems prevail: the routing of capacitated vehicles through a collection of customers to pickup or deliver goods, and the scheduling of vehicles to meet time or precedence constraints imposed upon their routes.

The importance of effective and efficient distribution management is evident from its associated costs. Physical distribution management at the operational level, which is considered in this paper, is responsible for an important fraction of the total distribution costs. Small relative savings in these expenses could already account for substantial savings in absolute terms. The significance of detecting these potential savings has become increasingly apparent due to the escalation of the costs involved, such as capital and fuel costs and driver salaries.

Not surprisingly, there is a growing demand for planning systems that produce economical routes. Although cost optimization is often the primary objective for purchasing computerized systems for physical distribution management, there are other benefits that should not be underestimated. The introduction of such systems enables companies to maintain a higher level of service towards their customers, it makes them less dependent of their planners, it supplies better management information facilities, and it makes the conduct of work faster and simpler.

Besides its obvious practical importance, physical distribution management also provides some fascinating basic models, such as the *traveling salesman problem* (TSP) and the *vehicle routing problem* (VRP). Consequently, researchers from the fields of operations research, mathematics, and computer science, have spent man-centuries in trying to develop solution approaches for these problems. For an extensive survey of models and solution methods in vehicle routing and scheduling see Bodin, Golden, Assad & Ball [1983], Laporte & Norbert [1987], and Golden & Assad [1988].

In the last decade, enormous theoretical as well as practical advances have been made. Certainly one of the most important advances is the incorporation of 'real world' characteristics, such as time windows and precedence relations, into the basic models.

In the *vehicle routing problem with time windows* (VRPTW) [Desrochers, Lenstra, Savelsbergh & Soumis, 1988; Solomon & Desrosiers, 1988] a number of vehicles, each with a given capacity, is located at a single depot and must serve a number of geographically dispersed customers. Each customer has a given demand and must be served within a specified time window. The objective is to minimize the total cost of travel.

In the *pickup and delivery problem* (PDP) [Savelsbergh & Sol, 1995], there is again a single depot, a number of vehicles with given capacities, and a number of customers with given demands. Each customer must be served to pick up goods at his origin during a specified time window, and to deliver the items at his destination during another specified time window. The objective is to minimize total travel cost. The special case in which all customer demands are equal is called the *dial-a-ride problem* (DARP). It arises in transportation systems for the handicapped and the elderly. In these situations, the temporal constraints imposed by the customers strongly restrict the total vehicle load at any point in time, and the capacity constraints are of secondary importance. The cost of a route is a combination of travel time and customer dissatisfaction.

An important consideration in the formulation and solution of vehicle routing and scheduling problems is the required computational effort associated with various solution techniques. Virtually all vehicle routing and scheduling problems belong to the class of *NP*-hard problems. This indicates that it is difficult to solve even small instances of a problem to optimality with a reasonable computational effort. As a consequence, when we have to solve real-life problems, we should not insist on finding an optimal solution, but instead on finding an acceptable solution within an acceptable amount of computation time. To accomplish this we have to resort to approximation algorithms.

Approximation algorithms for vehicle routing and scheduling problems usually have two phases: a *construction phase*, in which an initial feasible solution is constructed, and an *improvement phase*, in which an attempt is made to improve that initial solution by repeatedly searching a specified neighborhood for a better one.

To the best of our knowledge, most iterative improvement methods that have been applied to the vehicle routing and scheduling problems are extensions of the well-known k-exchange algorithms, which have been studied extensively in the context of the traveling salesman problem, see for instance Johnson & McGeogh [1995]. Extensions are necessary to handle multiple routes and various side constraints.

Obviously, traditional k-exchange algorithms can be used to improve a vehicle routing and scheduling solution by considering routes one at a time. However, the multiple routes structure offers additional opportunities. In Section 2, we focus on edge-exchange neighborhoods in the single route situation and on neighborhoods that have been proposed to exploit the multiple routes structure.

Real-life vehicle routing and scheduling problems are complicated by various side constraints, such as precedence relations between vertices, collections or deliveries at vertices, and time windows at vertices. Clearly, the traditional k-exchange algorithms have to be modified to ensure the feasibility of a route after an exchange has been made. Unfortunately, straightforward extensions lead to prohibitive computation times. In Section 3, we focus on the techniques that have been proposed to handle these various side constraints efficiently and present several computational experiments that we have conducted to evaluate the benefits of these techniques.

Because efficiency is a key consideration in the development of iterative improvement methods, we also investigate the potential of parallel computations and discuss our findings in Section 4.

Finally, in Section 5, we observe that people have started to apply recently developed variants of local search, such as simulated annealing and tabu search, to vehicle routing and scheduling problems.

2 EDGE-EXCHANGE NEIGHBORHOODS

Most iterative improvement methods that have been applied to vehicle routing and scheduling problems are edge-exchange algorithms. We distinguish between edge-exchange neighborhoods for a single route and edge-exchange neighborhoods for multiple routes. We will assume that the routes start and end at the depot, and that there are $n-1$ customers to be served. Further, we will use the notation pre_i and suc_i to denote the predecessor and successor of vertex i in de current route.

2.1 Edge-exchange neighborhoods for a single route

The edge-exchange neighborhoods for a single route are sets of tours that can be obtained from an initial tour by replacing a set of k of its edges by another set of k edges. Such replacements are called *k-exchanges*, and a tour that cannot be improved by a k-exchange is said to be *k-optimal*. Verifying k-optimality requires $O(n^k)$ time.

The k-exchange neighborhoods defined above are sets of tours that can be obtained from an initial tour by replacing a set of edges of fixed cardinality k by another set of k edges. Not surprisingly, the quality of an exchange may improve considerably, when the neighborhood is enlarged by increasing the cardinality of the set of edges to be replaced. Unfortunately, the computational effort required to search the neighborhood becomes much greater as well. Lin & Kernighan [1973] developed a variable-depth exchange procedure by dynamically determining the cardinality of the set of edges to be replaced, thus finding a good balance between the computational effort and the quality of the solution.

For two reasons, we restrict our attention to k-exchanges for $k \leq 3$. First, k-exchanges for $k > 3$ are seldom used in iterative improvement methods for vehicle routing and scheduling problems. Second, k-exchanges for $k \leq 3$ suffice to illustrate the techniques we want to discuss. In the end, it should be clear how the presented techniques can be extended to the general case as well as the Lin-Kernighan algorithm. The following two neighborhoods will be considered in detail:
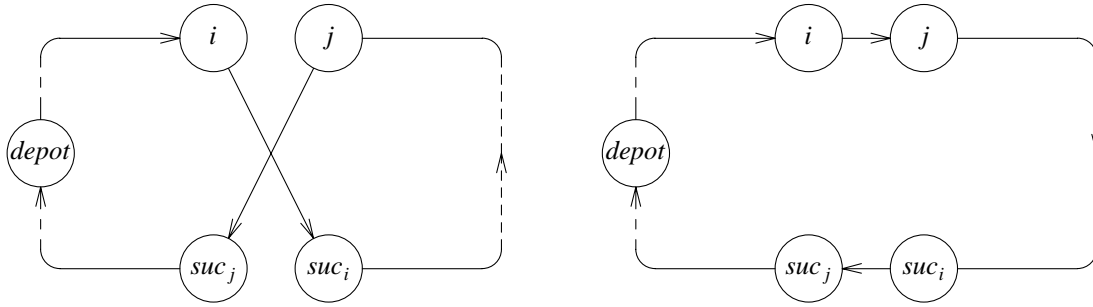

Figure 1 A 2-exchange.

(1) Try to improve the tour by replacing 2 of its edges by 2 other edges, i.e., a 2-exchange, and iterate until no further improvement is possible. An example of a 2-exchange is given in Figure 1.
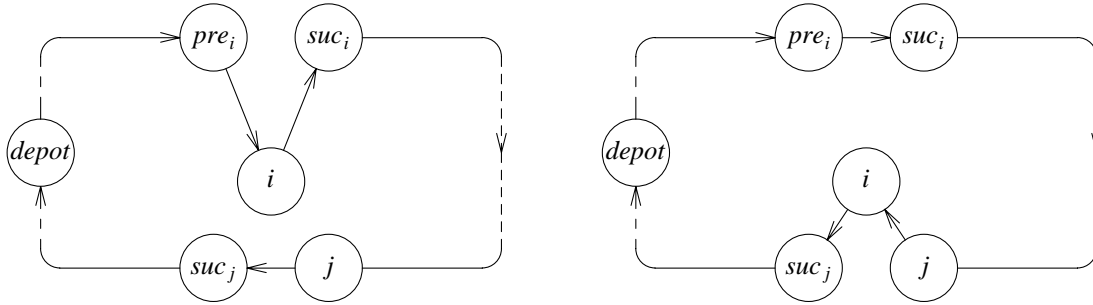

Figure 2 A forward Or-exchange.

(2) Try to improve the tour by relocating a chain of l consecutive vertices and iterate until no further improvement is possible. This type modification of the tour involves the replacement of a set of three edges, with a restriction on the choice of edges. Hence, the neighborhood is a subset of the 3-exchange neighborhood. Since the procedures involved are conceptually the same for all values of l, we will restrict ourselves to $l = 1$. This type of exchanges was first considered by Or [1976], and are, therefore, called *Or-exchanges*. We

speak of a *forward* Or-exchange if the vertex is moved to a place further on the tour (cf. Figure 2), and of a *backward* Or-exchange otherwise (cf. Figure 3). Testing optimality over the Or-exchange neighborhood requires $O(n^2)$ time.
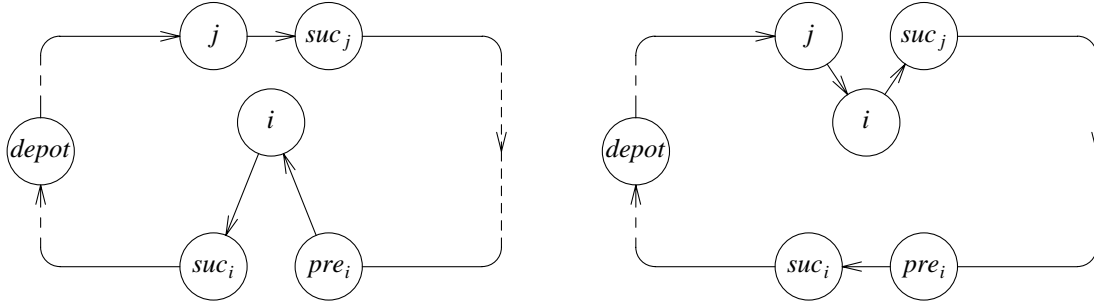

Figure 3 A backward Or-exchange.

2.2 Edge-exchange neighborhoods for multiple routes

Two types of decisions have to be made to obtain a solution to a vehicle routing and scheduling problem: *assignment* decisions to determine which vehicle will serve which customers and *routing* decisions to determine in which order the customers assigned to a vehicle will be visited. The local search methods in the previous section try to improve the routing decisions. Another possibility is to improve the assignment decisions. Improving assignment decisions may even be more effective when we consider the fact that in most instances of the VRP the number of customers per vehicle is fairly small, i.e., the resulting instances of the TSP are fairly simple. It is uncommon to find instances where the number of customers per vehicle exceeds 30 customers. More often we are in a situation where the number of customers per vehicle is much less; in the order of 5 to 15. In that case, it is unrealistic to hope for major improvements when routing decisions are revised. However, the total number of customers is usually large. Therefore, there is much more potential for improvement when assignment decisions are revised. In view of the above, it is surprising to see that there is very little known about local search methods that revise assignment decisions.

The three basic k-exchange neighborhoods for the VRP (see for example Savelsbergh [1992]) relocate vertices between *two* routes. The neighborhoods are chosen such that testing optimality over the neighborhood requires $O(n^2)$ time. For presentational convenience, we will first describe relocations of single vertices. In the figures, we will split the depot.
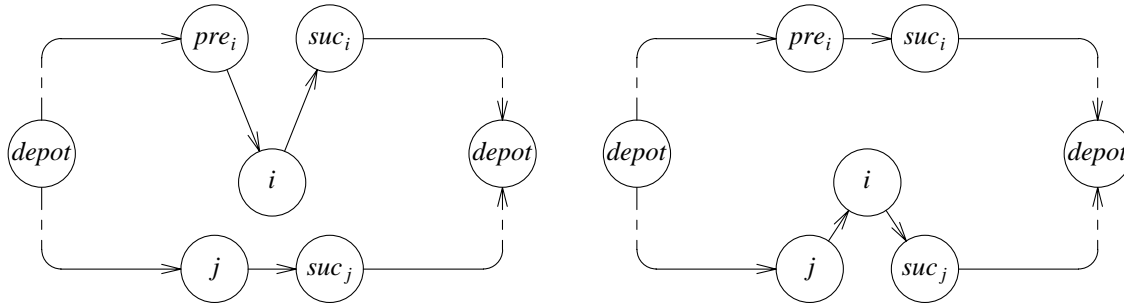

Figure 4 A relocation.

Relocation. The edges $\{pre_i, i\}$, $\{i, suc_i\}$ and $\{j, suc_j\}$ are replaced by $\{pre_i, suc_i\}$, $\{j, i\}$ and $\{i, suc_j\}$, i.e., vertex i from the origin route is placed in the destination route. A relocation is pictured in Figure 4.
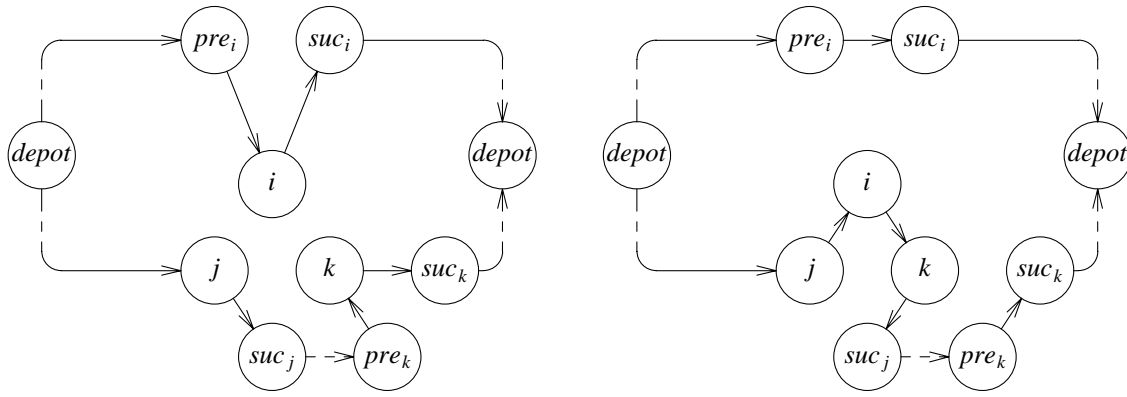
Figure 5 An insertion between two non-consecutive vertices.

Note that the vertex being relocated is inserted between two consecutive vertices on the destination route. Gendreau, Hertz & Laporte [1994] propose an extension of the relocate neighborhood in which a vertex can also be inserted between the two vertices on the destination route that are nearest to it, even if these vertices are not consecutive. Insertions of this type have been used with great success in an insertion algorithm for the TSP [Gendreau, Hertz & Laporte, 1992]. An example is shown in Figure 5. Observe that it is no longer obvious how to obtain a tour again after the edges $\{j, i\}$ and $\{i, k\}$ have been added. One possibility is to delete edges $\{j, suc_j\}$ and $\{pre_k, k\}$ and to relocate the path $(suc_j, \ldots, pre_k)$.

Exchange. The edges $\{pre_i, i\}$, $\{i, suc_i\}$, $\{pre_j, j\}$ and $\{j, suc_j\}$ are replaced by $\{pre_i, j\}$, $\{j, suc_i\}$, $\{pre_j, i\}$ and $\{i, suc_j\}$, i.e., two vertices from different routes are simultaneously placed into the other routes. An exchange is pictured in Figure 6.
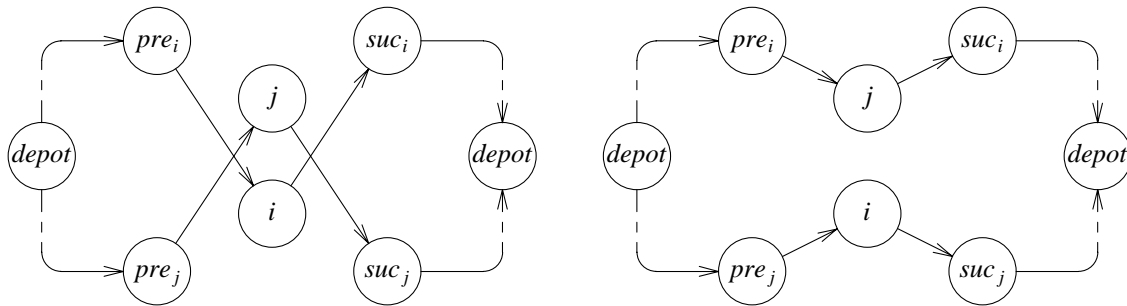
Figure 6 An exchange.

Crossover. The edges $\{i, suc_i\}$ and $\{j, suc_j\}$ are replaced by $\{i, suc_j\}$ and $\{j, suc_i\}$, i.e., the crossing links of two routes are removed. A crossover is pictured in Figure 7. As a special case, it can combine two routes into one if edge $\{i, suc_i\}$ is the first one on its route and edge $\{j, suc_j\}$ the last one on its route or vice versa. Note that if a crossover is actually performed, the last part of either route will become the last part of the other.

Some of the above described iterative improvement methods can easily be extended to larger neighborhoods by the introduction of paths instead of vertices. This does not influence the worst case time complexity of testing optimality, as long as the length of the paths is fixed. Figure 8 illustrates possible extensions.
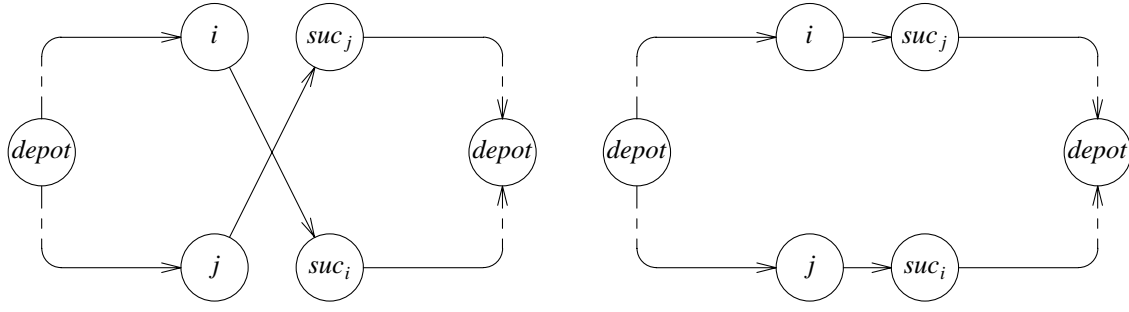
Figure 7 A crossover

All of the above neighborhoods can be viewed as special cases of cyclic transfers proposed for vehicle routing and scheduling problems by Thompson & Psaraftis [1993]. The central idea behind cyclic transfers is to attempt to improve a solution to a partitioning problem by transferring small numbers of elements among cycles of sets. More formally, let $\{I^1, I^2, \ldots, I^m\}$ be a partition of a set of elements S, and let ρ be a cyclic permutation of a subset of $(1, 2, \ldots, m)$. Then, a cyclic transfer is the simultaneous transfer of elements from I^j to $I^{\rho(j)}$ for each j.

Several special cases of cyclic transfers are of interest. Cyclic k-transfers transfer exactly k elements from I^j to $I^{\rho(j)}$ for each j and some integer k. The flexibility of k-transfers can be further increased by allowing dummy elements to be transferred, i.e., cyclic transfers are obtained in which the number of elements to be transferred from I^j to $I^{\rho(j)}$ for each j is bounded from above by k. If, in addition, the cyclic permutation has cardinality b, the set of transfers is called a b-cyclic k-transfer. Observe that an exchange is a 2-cyclic 1-transfer and a relocation is a 2-cyclic 1-transfer in which a dummy element is transferred from one of the two sets.

The cost of a cyclic transfer is the change in the objective function caused by the cyclic transfer and a solution is cyclic-transfer-optimal if all cyclic transfers have nonnegative cost. Thompson & Orlin [1989] develop a general methodology for searching cyclic transfer neighborhoods. They transform the search for negative cost cyclic transfers into a search for negative cost cycles on an auxiliary graph. Due to the computational intensity of searching cyclic transfer neighborhoods only b-cyclic k-transfers with only small numbers b and k are considered, i.e., $b \leq 3$ and $k \leq 2$.

3 TECHNIQUES TO HANDLE SIDE CONSTRAINTS

Real-life vehicle routing and scheduling problems are complicated by various side constraints. The main thrust of research on modifying edge-exchange algorithms to handle these side constraints has not focused on how to incorporate the side constraints, which is trivial, but on how to do this efficiently, which is non-trivial. We restrict our discussion to the following side constraints:

- precedence relations between vertices;
- collections or deliveries at vertices;
- time windows at vertices.

We will consider each of these side constraints separately and we will describe the techniques to efficiently handle side constraints only for edge exchange neighborhoods involving a single route. However, it should be clear in the end that the techniques presented can easily be used to produce an edge exchange algorithm that can handle any combination of side constraints for edge exchange neighborhoods involving either a single route or multiple routes.
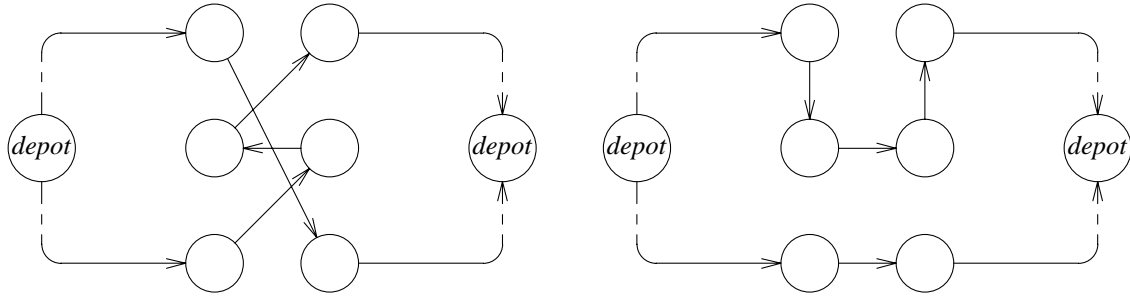
We start by introducing the three variants of the TSP for which we will study the exchange procedures in more detail.
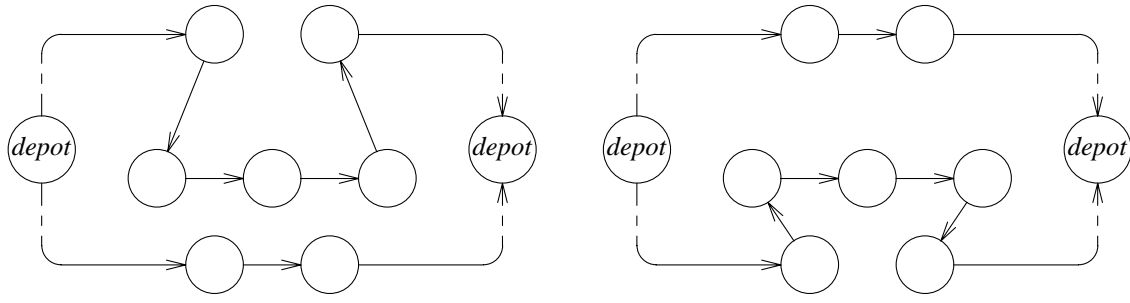
(a) Relocation of a path.
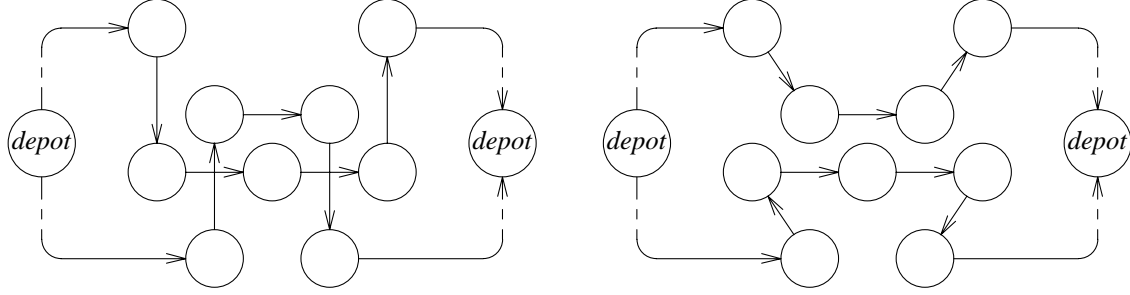
(b) Exchange of two paths.

(c) A crossover plus 2-exchange.

Figure 8 Possible extensions of the neighborhoods.

In the *TSP with precedence constraints*, we are given, in addition to the travel times d_{ij} between each pair of vertices (i, j), precedence constraints specifying that some pairs of vertices have to be visited in a prescribed order. The *single-vehicle dial-a-ride problem*, where a single vehicle has to pick up and deliver n customers, is an example of the TSP with precedence constraints. Given a precedence related pair of vertices $u \rightarrow v$, we will often refer to u as the origin and v as the destination.

In the *TSP with collections and deliveries*, we are given, in addition to the travel times between vertices, for each vertex i an associated load q_i that can be either positive or negative depending on whether the load has to be collected or delivered. The salesman uses a vehicle with fixed capacity Q.

In the *TSP with time windows*, we are given, in addition to the travel times, for each vertex i a time window on the departure time, denoted by $[e_i, l_i]$, where e_i specifies the earliest service time and l_i the latest service time. Arriving earlier than e_i introduces a waiting time at vertex i; arriving after l_i leads to infeasibility. We will use the following notation: A_i will denote the arrival time at vertex i, D_i will denote the departure time at

vertex i, and W_i will denote the waiting time at vertex i.

For notational convenience, we will assume that the current tour is given by the sequence $(1, 2, \ldots, i, \ldots, n, n+1)$, where the origin 1 and the destination $n+1$ denote the same vertex, and i represents the i-th vertex. We also assume that a 2-exchange replaces two edges $\{i, i+1\}$ and $\{j, j+1\}$, with $j > i$, by the edges $\{i, j\}$ and $\{i+1, j+1\}$, and that an Or-exchange involves the substitution of edges $\{i-1, i\}$, $\{i, i+1\}$ and $\{j, j+1\}$ by $\{i-1, i+1\}$, $\{j, i\}$ and $\{i, j+1\}$.

The main difficulty with the use of exchange procedures in the TSP with side constraints is testing the feasibility of an exchange, as opposed to the TSP where one only has to test whether the exchange is profitable and one does not have to bother about feasibility. A 2-exchange, for instance, will reverse the path $(i+1, \ldots, j)$, which means that one has to check the feasibility of at least all the vertices on the new path with respect to those constraints. In a straightforward implementation this requires $O(n^2)$ time in the TSP with precedence constraints, and $O(n)$ time in the TSP with collections and deliveries and the TSP with time windows.

By applying *preprocessing techniques* [Psaraftis, 1983; Solomon, Baker & Shaffer, 1988], *tailored updating mechanisms* [Solomon, Baker & Shaffer, 1988], and *lexicographic search strategies* [Savelsbergh, 1986, 1990, 1992] several researchers have been able to incorporate the various side constraints with an acceptable or even without an increase in computation times.

Before focusing on improvement methods of a given tour, it is worthwhile to mention the influence of the side constraints on the construction of an initial feasible tour. In the TSP with precedence relations, an initial tour can be obtained in polynomial time by visiting the vertices in topological order. In the other two cases, the problem of deciding whether a feasible tour exists is *NP*-complete [Savelsbergh, 1986, 1992]. Constructing a feasible initial tour may, hence, constitute a problem. In the context of physical distribution management, however, the instances of the TSP stem most of the time from some constructive VRP heuristic, which also provides an initial feasible tour.

3.1 Preprocessing

Psaraftis [1983] was the first to study k-exchange procedures for a constrained variant of the TSP. He studied 2-exchanges and Or-exchanges in the context of the single vehicle dial-a-ride problem. The dial-a-ride problem is a TSP with precedence constraints in which each vertex is related to precisely one other vertex.

First, we examine the 2-exchanges. A straightforward test for feasibility in this case requires $O(n^2)$ time. This can be seen as follows. The only way that a 2-exchange can be infeasible is if there is at least one precedence related pair of vertices on the segment of the tour that is reversed. The simplest way to find out whether such a pair of vertices exists is to examine all pairs of vertices in the segment. This requires $O(n^2)$ time and would lead to an overall complexity for verification of 2-optimality of $O(n^4)$.

Psaraftis shows how this can be reduced to $O(n^2)$ by performing a *screening* procedure in the beginning of the algorithm which determines the feasibility of every possible 2-exchange. Information from the screening procedure is stored in a feasibility matrix to be examined during the execution of the actual algorithm.

The screening procedure is based on the following observation: if $firstdes_i$ denotes the first destination of a precedence relation for which both origin and destination lie beyond i, then the exchange of $\{i, i+1\}$ and $\{j, j+1\}$ with $\{i, j\}$ and $\{i+1, j+1\}$ is feasible if and only if $j < firstdes_i$. The screening procedure first computes the values $firstdes_i$ and then constructs the feasibility matrix $feas$, i.e., $feas_{ij} = 1$ if $j < firstdes_i$ and $feas_{ij} = 0$ otherwise. Psaraftis shows that the values $firstdes_i$ can be computed in $O(n^2)$ time, thus proving that verification of 2-optimality can be done in $O(n^2)$ time.

Next, we examine the Or-exchanges. An advantageous feature of Or-exchanges is that they are *direction preserving*, i.e., the segments determined by the deletion of $\{i-1, i\}$, $\{i, i+1\}$, and $\{j, j+1\}$ are traversed in the same direction in the final tour. Therefore, if these segment are feasible in the original tour, they will also be feasible in the final tour. There is only one situation which leads to infeasibility: a precedence related pair of vertices with origin i and destination in the segment $(i+1, \ldots, j)$ for a forward Or-exchange and a precedence related pair of vertices with origin in the segment $(j+1, \ldots, i-1)$ and destination i for a backward Or-exchange, since the order, in which the vertex i and the segment are traversed in the final tour, is reversed.

Similar screening procedures can be developed for both forward and backward Or-exchanges. For forward Or-exchanges, we need the following observation: if $firstdes_i$ denotes the first destination of a precedence relation with origin i, then the exchange of $\{i-1,i\}$, $\{i,i+1\}$ and $\{j,j+1\}$ with $\{i-1,i+1\}$, $\{j,i\}$ and $\{i,j+1\}$ is feasible if and only if $j < firstdes_i$. An analogous observation can be made for backward-Or-exchanges.

The values $firstdes_i$ can be computed in $O(n^2)$ time, thus proving an overall complexity of $O(n^2)$ for verification of Or-optimality.

Solomon, Baker & Schaffer [1988] have adapted the above presented preprocessing scheme to the TSP with time windows. The idea is to identify precedence relations between pairs of vertices based on their time windows. If $e_i + d_{ij} > l_j$, then vertex j has to precede vertex i. The use of this type of preprocessing does not eliminate the need for further checking of feasibility; it may be used as a filter to reduce the number of complete feasibility checks required.

3.2 Updating mechanisms

Solomon, Baker & Schaffer [1988] have carried out an extensive computational study on the efficient implementation of k-exchange procedures for the TSP with time windows. Their implementation incorporated the preprocessing scheme discussed above, as well as tailored updating mechanisms for direction preserving exchanges. These updating mechanisms are based on the observation that if the direction in which we traverse a path is unchanged, we can check the feasibility of each vertex on the path by simply looking at the change in arrival time.

Consider a path $(u, u+1, \ldots, v)$ with associated departure times and suppose that the departure time at the first vertex of the path is decreased. This defines a *push backward*

$$B_u = D_u - D_u^{\text{new}},$$

where D_u and D_u^{new} define the current and the new departure time at vertex u. The push backward at the next vertex on the path can be computed by

$$B_{u+1} = \min\{B_u, D_{u+1} - e_{u+1}\}.$$

Obviously, all vertices on the path remain feasible and the departure times D_k need to be adjusted sequentially for $k = u, \ldots, v$ as long as $B_k > 0$.

Similarly, consider a path $(u, u+1, \ldots, v)$ with associated departure times and suppose that the departure time at the first vertex of the path is increased. This defines a *push forward*

$$F_u = D_u^{\text{new}} - D_u.$$

The push forward at the next vertex on the path can be computed as follows

$$F_{u+1} = \max\{F_u - W_{u+1}, 0\},$$

where W_{u+1} denotes the waiting time at vertex $u+1$. Obviously, if $F_u > 0$ some vertices on the path could become infeasible. The vertices on the path have to be checked sequentially. At a vertex k ($u \le k \le v$), it may happen that $D_k + F_k > l_k$ in which case the path is no longer feasible, or $F_k = 0$ which indicates that the path from vertex k to vertex v has not been changed.

Observe that in the worst case testing the feasibility of an exchange takes $O(n)$ time. However, in practice, the use of a push backward and a push forward leads to a substantial reduction in the number of vertices being examined for time feasibility. Note that these techniques are only useful in direction preserving exchanges.

3.3 Lexicographic search

Savelsbergh [1986, 1990, 1992] introduced an approach that can be used to incorporate all three side constraints in exchange procedures without increasing the time complexity of verifying local optimality.

The basic idea is the use of a specific *search strategy* in combination with a set of *global variables* such that testing the feasibility of a single exchange and maintaining the set global variables requires no more than constant time. Because the search strategy is of crucial importance, we present it first.

Lexicographic search for 2-exchanges. We choose the edges $\{i, i+1\}$ in the order in which they appear in the current route starting with $i = 1$ up to $i = n - 2$; this will be referred to as the outer loop. After fixing an edge $\{i, i+1\}$, we choose the edge $\{j, j+1\}$ successively equal to $\{i+2, i+3\}, \{i+3, i+4\}, \ldots, \{n, n+1\}$ (cf. Figure 9); this will be referred to as the inner loop.
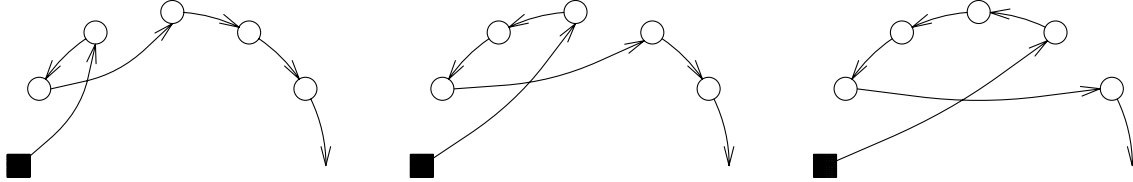

Figure 9 The search strategy for 2-exchanges.

Now consider all possible exchanges for a fixed edge $\{i, i+1\}$. The inspection of the 2-exchanges in the order given above implies that in the inner loop the previously reversed path $(i+1, \ldots, j-1)$, corresponding to the substitution of $\{i, i+1\}$ and $\{j-1, j\}$ with $\{i, j-1\}$ and $\{i+1, j\}$, is expanded by the edge $\{j-1, j\}$.

Lexicographic search for forward Or-exchanges. We choose vertex i in the order of the current route starting with i equal to 2. After fixing i, we choose the edge $\{j, j+1\}$ to be $\{i+1, i+2\}, \{i+2, i+3\}, \ldots, \{n, n+1\}$ consecutively. That is, the edge $\{j, j+1\}$ 'walks forward' through the route. Note that in the inner loop in each newly examined exchange the path $(i+1, \ldots, j-1)$ of the previously considered exchange is expanded with the edge $\{j-1, j\}$.

Lexicographic search for backward Or-exchanges. We choose vertex i in the order of the current route starting with i equal to 2. After fixing i, we choose the edge $\{j, j+1\}$ to be $\{i-2, i-1\}, \{i-3, i-2\}, \ldots, \{1, 2\}$ in that order. That is, the edge $\{j, j+1\}$ 'walks backward' through the route. Note that in the inner loop in each newly examined exchange the path $(j+2, \ldots, i-1)$ of the previously considered exchange is expanded with the edge $\{j+1, j+2\}$.

Now that we have presented the search strategy, let us return to the feasibility question. In order to test the feasibility of a single 2-exchange, we have to check all the vertices on the reversed path $(i+1, \ldots, j)$ and on the path $(j+1, \ldots, n+1)$, and in order to test the feasibility of a single forward (or backward) Or-exchange, we have to check besides vertex i the vertices on the paths $(i+1, \ldots, j)$ and $(j+1, \ldots, n+1)$ (or $(j+1, \ldots, i-1)$ and $(i+1, \ldots, n+1)$). In a straightforward implementation this takes $O(n)$ time for each single exchange. We will present an implementation that requires only constant time per exchange.

The idea is to define an appropriate set of global variables, which will of course depend on the constrained variant of the TSP we are considering, in such a way that

- the set of global variables makes it possible to test the feasibility of an exchange in constant time, i.e., the feasibility of all the vertices on the paths in question can be checked in constant time, and
- the lexicographic search strategy makes it possible to maintain the correct values for the set of global variables in constant time, i.e., the update of the global variables when we go from one exchange to the next one can be done in constant time.

To see how these ideas work out in actual implementations, we show the pseudo-code of a general framework for a 2-exchange procedure.

```
procedure TwoExchange
(* input: a route given as (1, . . . , n + 1) *)
(* output: a route that is 2-optimal *)
REPEAT:
  for i ← 1 to n − 2 do                              (* outer loop: fix edge {i, i + 1} *)
    InitializeGlobalVariables (i, G)                 (* initialize the set of global variables G *)
    for j ← i + 2 to n do                            (* inner loop: fix edge { j, j + 1} *)
      if ProfitableExchange (i, j, G) and            (* test whether the exchange is profitable *)
        FeasibleExchange (i, j, G)                   (* test whether the exchange is feasible *)
      then
        PerformExchange (i, j)        (* replace {i, i + 1} and { j, j + 1} by {i, j} and {i + 1, j + 1} *)
        goto REPEAT
      UpdateGlobalVariables (i, j, G)   (* update the set of global variables G for the next iteration *)
END:
```

Although the above pseudo-code looks rather simple, defining a set of global variables in such a way that, in combination with the lexicographic search strategy, the functions InitializeGlobalVariables, ProfitableExchange, FeasibleExchange, and UpdateGlobalVariables do what they are supposed to do and take only constant time, is often not so obvious.

Reexamining a 2-exchange, a forward Or-exchange and a backward Or-exchange, we see that the algorithms have to be able to handle reversing a path, relocating a path backward and relocating a path forward. As these three types of changes comprise all the possibilities that can occur in a k-exchange (for arbitrary k), the techniques can be used to implement k-exchange algorithms for the TSP with side constraints for arbitrary k without increasing the time complexity beyond $O(n^k)$.

Precedence relations

As a first illustration of the proposed technique we show how precedence relations can be handled. Obviously, we cannot improve the time complexity of $O(n^2)$ of Psaraftis' implementation for the verification of 2-optimality. In fact, we will just present an alternative and simpler implementation of his idea. Attach a label to each vertex u with information on the first vertex on the tour for which a precedence relation $u{\rightarrow}v$ exists: $first_u = \min\{v \mid u{\rightarrow}v\}$.

2-Exchanges. Recall that the exchange of $\{i, i + 1\}$ and $\{ j, j + 1\}$ with $\{i, j\}$ and $\{i + 1, j + 1\}$ is feasible if and only if $j < firstdes_i$, where $firstdes_i$ denotes the first destination of a precedence relation for which both origin and destination lie beyond i. This is equivalent to stating that a 2-exchange is feasible if and only if there is no precedence related pair of vertices on the path that is reversed. Hence, at any stage information concerning this path is sufficient. The lexicographic search strategy makes it possible to gather the required knowledge using a single global variable. We introduce the variable F to denote the first destination of a precedence relation for which the corresponding origin is on the path that is to be reversed. Feasibility of a 2-exchange is now established by verifying that the vertex denoted by the variable F is not on the reversed path.

All we have to do now is to show that we can ensure that the global variable F contains the right information when it is examined. The lexicographic search strategy provides a simple way to accomplish this. In the outer loop, whenever we expand the path $(1, \ldots, i − 1)$ with the edge $\{i − 1, i\}$, we set F equal to $first_{i+1}$. In the inner loop, whenever we expand the path $(i + 1, \ldots, j − 1)$ with the edge $\{ j − 1, j\}$, we set F equal to the minimum of its current value and $first_j$.

Or-exchanges. A forward Or-exchange is feasible if and only if there is no pair of precedence-related vertices with the first being i and the other on the path $(i+1, \ldots, j)$. Whenever we try to expand the path $(i+1, \ldots, j-1)$ with the edge $\{j-1, j\}$ and $i \rightarrow j$, i.e., vertex i is a predecessor of vertex j, the expansion will only result in infeasible exchanges. Similarly, a backward Or-exchange is feasible if and only if there is no pair of precedence-related vertices with the first on the path $(j+1, \ldots, i-1)$ and the other being i. Whenever we try to expand the path $(j+2, \ldots, i-1)$ with the edge $\{j+1, j+2\}$ and $j+1 \rightarrow i$, i.e., vertex $j+1$ is a predecessor of vertex i, the expansion will only result in infeasible exchanges.

Before discussing collections and deliveries and time windows, we take another close look at a k-exchange and the lexicographic search strategy. A k-exchange is the substitution of k links of a tour with k other links. The first step is the deletion of k links, i.e., the tour is broken up into k paths. The second step is the addition of k other links, i.e., the k paths are concatenated in a different order to form a new tour.

More specifically, a 2-exchange deletes the links $\{i, i+1\}$ and $\{j, j+1\}$, with $j > i$, to form the paths $(1, \ldots, i)$, $(i+1, \ldots, j)$, and $(j+1, \ldots, n+1)$, and then adds the links $\{i, j\}$ and $\{i+1, j+1\}$ to obtain the new tour $(1, \ldots, i, j, \ldots, i+1, j+1, \ldots, n+1)$, a forward Or-exchange deletes the links $\{i-1, i\}$, $\{i, i+1\}$ and $\{j, j+1\}$ to form the paths $(1, \ldots, i-1)$, (i), $(i+1, \ldots, j)$, and $(j+1, \ldots, n+1)$, and then adds the links $\{i-1, i+1\}$, $\{j, i\}$ and $\{i, j+1\}$ to obtain the new tour $(1, \ldots, i-1, i+1, \ldots, j, i, j+1, \ldots, n+1)$, and a backward Or-exchange deletes the links $\{i-1, i\}$, $\{i, i+1\}$ and $\{j, j+1\}$ to form the paths $(1, \ldots, j)$, $(j+1, \ldots, i-1)$, (i), and $(i+1, \ldots, n+1)$, and then adds the links $\{i-1, i+1\}$, $\{j, i\}$ and $\{i, j+1\}$ to obtain the new tour $(1, \ldots, j, i, j+1, \ldots, i-1, i+1, \ldots, n+1)$.

The key feature of the lexicographic search strategy is that in consecutive iterations the k paths that result after the deletion of the k links differ by at most a single vertex. The proposed implementation scheme for k-exchange methods associates a set of global variables with each of these paths containing information on its feasibility and its profitability. The global variables are chosen such that initializing the global variables for a single vertex path and computing the values of the global variables for a concatenated path both take constant time.

Collections and deliveries
The following three quantities turn out to be sufficient for the analysis of feasibility for the TSP with collections and deliveries.

- The *maximum load* $L^{\max}_{(u_1, \ldots, u_k)}$ on the path $(u_1, \ldots, u_k)$, assuming the vehicle is empty when it arrives at vertex u_1, i.e.,

$$L^{\max}_{(u_1, \ldots, u_k)} = \max_{1 \leq i \leq k} \Sigma_{1 \leq j \leq i} q_{u_j}.$$

- The *minimum load* $L^{\min}_{(u_1, \ldots, u_k)}$ on the path $(u_1, \ldots, u_k)$, assuming the vehicle is empty when it arrives at vertex u_1, i.e.,

$$L^{\min}_{(u_1, \ldots, u_k)} = \min_{1 \leq i \leq k} \Sigma_{1 \leq j \leq i} q_{u_j}.$$

- The *final load* $L^{\text{final}}_{(u_1, \ldots, u_k)}$ on the path $(u_1, \ldots, u_k)$, assuming the vehicle is empty when it arrives at vertex u_1, i.e.,

$$L^{\text{final}}_{(u_1, \ldots, u_k)} = \Sigma_{1 \leq i \leq k} q_{u_i}.$$

Initializing these quantities for a single vertex path (u) is trivial: $L^{\max}_{(u)} = L^{\min}_{(u)} = L^{\text{final}}_{(u)} = q_u$. The following proposition states that if we concatenate two (vertex-disjoint) paths, we can compute the quantities for the resulting path from the quantities of its constituent paths in constant time.

Proposition. If two (vertex-disjoint) feasible paths $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_l)$, with associated maximal loads $L^{\max}_{(u_1,\ldots,u_k)}$ and $L^{\max}_{(v_1,\ldots,v_l)}$, minimal loads $L^{\min}_{(u_1,\ldots,u_k)}$ and $L^{\min}_{(v_1,\ldots,v_l)}$, and final loads $L^{\text{final}}_{(u_1,\ldots,u_k)}$ and $L^{\text{final}}_{(v_1,\ldots,v_l)}$, are concatenated, the same values for the resulting path $(u_1, \ldots, u_k, v_1, \ldots, v_l)$ are given by:

$$L^{\max}_{(u_1,\ldots,u_k,v_1,\ldots,v_l)} = \max\{L^{\max}_{(u_1,\ldots,u_k)}, L^{\text{final}}_{(u_1,\ldots,u_k)} + L^{\max}_{(v_1,\ldots,v_l)}\},$$
$$L^{\min}_{(u_1,\ldots,u_k,v_1,\ldots,v_l)} = \min\{L^{\min}_{(u_1,\ldots,u_k)}, L^{\text{final}}_{(u_1,\ldots,u_k)} + L^{\min}_{(v_1,\ldots,v_l)}\}, \text{ and}$$
$$L^{\text{final}}_{(u_1,\ldots,u_k,v_1,\ldots,v_l)} = L^{\text{final}}_{(u_1,\ldots,u_k)} + L^{\text{final}}_{(v_1,\ldots,v_l)}.$$

□

From the discussion on k-exchanges and the lexicographic search strategy, it should be clear that the above proposition shows that checking the feasibility of an exchange, i.e., $L^{\max}_{(1,\ldots,n+1)} \leq Q$ and $L^{\min}_{(1,\ldots,n+1)} \geq 0$, as well as updating between consecutive iterations can be done in constant time.

Time windows

Under the assumption that on its way a vehicle always departs at a vertex as early as possible, which is the best choice from a feasibility point of view, a path can be completely specified by giving the sequence in which the vertices are visited and the departure time at the first vertex of the path.

The following quantities turn out to be useful in the analysis of feasibility and profitability of k-exchanges in the TSP with time windows.

- The *total travel time* $T_{(u_1,\ldots,u_k)}$ on the path $(u_1, \ldots, u_k)$, i.e.,

$$T_{(u_1,\ldots,u_k)} = \Sigma_{1 \leq i < k} d_{u_i u_{i+1}}.$$

- The *earliest departure time* $E_{(u_1,\ldots,u_k)}$ at vertex u_k of the path $(u_1, \ldots, u_k)$, assuming vertex u_1 is left at the opening of its time window, i.e.,

$$E_{(u_1,\ldots,u_k)} = \max_{1 \leq i \leq k}\{e_i + T_{(u_i,\ldots,u_k)}\}.$$

- The *latest arrival time* $L_{(u_1,\ldots,u_k)}$ at vertex u_1 of the path $(u_1, \ldots, u_k)$, such that the path remains feasible, i.e.,

$$L_{(u_1,\ldots,u_k)} = \min_{1 \leq i \leq k}\{l_i - T_{(u_1,\ldots,u_i)}\}.$$

Other interesting quantities can be obtained using the above values. So is, for example, the waiting time on the path $(u_1, \ldots, u_k)$ equal to $E_{(u_1,\ldots,u_k)} - e_{u_1} - T_{(u_1,\ldots,u_k)}$.

The following proposition shows that if we concatenate two paths $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_l)$, we can compute the same quantities for the resulting path $(u_1, \ldots, u_k, v_1, \ldots, v_l)$ from the quantities of its constituent paths in constant time.

Proposition. If two (vertex-disjoint) feasible paths $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_l)$, with associated total travel times $T_{(u_1,\ldots,u_k)}$ and $T_{(v_1,\ldots,v_l)}$, earliest departure times $E_{(u_1,\ldots,u_k)}$ and $E_{(v_1,\ldots,v_l)}$, and latest arrival times $L_{(u_1,\ldots,u_k)}$ and $L_{(v_1,\ldots,v_l)}$, are concatenated, the resulting path is feasible if and only if $E_{(u_1,\ldots,u_k)} + d_{u_k,v_1} \leq L_{(v_1,\ldots,v_l)}$ and the same values for the resulting path are given by:

$$T_{(u_1,\ldots,u_k,v_1,\ldots,v_l)} = T_{(u_1,\ldots,u_k)} + d_{u_k,v_1} + T_{(v_1,\ldots,v_l)},$$
$$E_{(u_1,\ldots,u_k,v_1,\ldots,v_l)} = \max\{E_{(u_1,\ldots,u_k)} + d_{u_k,v_1} + T_{(v_1,\ldots,v_l)}, E_{(v_1,\ldots,v_l)}\}, \text{ and}$$
$$L_{(u_1,\ldots,u_k,v_1,\ldots,v_l)} = \min\{L_{(u_1,\ldots,u_k)}, L_{(v_1,\ldots,v_l)} - T_{(u_1,\ldots,u_k)} - d_{u_k,v_1}\}.$$

□

From the discussion on k-exchanges and the lexicographic search strategy, it should be clear that this proposition shows that checking the feasibility of a k-exchange as well as updating between consecutive iterations can be done in constant time.

The presence of time windows also allows for the specification of a variety of objective functions, such as the minimization of the total travel time, i.e., $T_{(1,\dots,n+1)}$, the minimization of the completion time, i.e., $E_{(1,\dots,n+1)}$, and the minimization of the route duration, i.e., $\max\{E_{(1,\dots,n+1)} - L_{(1,\dots,n+1)}, T_{(1,\dots,n+1)}\}$.

Van der Bruggen, Lenstra & Schuur [1993] have developed an efficient variable-depth improvement algorithm for the single-vehicle dial-a-ride problem using the techniques described above. The lexicographic search strategy is applied until a feasible and profitable 2-exchange is found. Now, instead of actually performing the exchange, they continue the search for a 2-exchange on the path that has not been affected by the exchange. They repeat this as long as the combined exchanges are profitable. In this way, in a single step, a series of exchanges is examined from which the best is selected (cf. Figure 10).

One can imagine many variants of this procedure. Any type of exchange, e.g, Or-exchange or 3-exchange, can be used as a basic improvement step in the procedure. Also, the best possible exchange can be chosen at each step, instead of the first feasible and profitable one encountered.

3.4 Computational evidence

To illustrate the effectiveness of the techniques, we now briefly describe the computational experiments carried out by Knops [1993]. These experiments focus on the performance of 2-exchange algorithms for the TSP with time windows.

Knops considers the following three implementations: a straightforward search of the 2-exchange neighborhood, the straightforward search enhanced with the improvements proposed by Solomon, Baker & Shaffer [1988; cf. Section 3.1], and the lexicographic search approach of Savelsbergh [1986, 1990, 1992; cf Section 3.3].

The test problems are generated from Solomon's problem set for the VRPTW [Solomon, 1987]. From each of the problems in Solomon's class R2, instances of the TSPTW are created by taking the first k customers, for $k = 10, 15, 20,$ and 30. Compared to Solomon's other classes, the class R2 is best fit for this purpose, since the location of the customers and the time windows are such that feasible routes exist and allow for a substantial number of improvement steps. For each of these problems an initial route was determined by some simple backtracking procedure based on the nearest neighbor heuristic for the TSP: among the customers not yet visited choose as the next customer the one that is closest to the last visited customer; if a particular choice leads to infeasibility of the route, choose the next closest customer, and so on. Although the problem of deciding whether a feasible tour exists is *NP*-complete, the generated instances are such that the construction of an initial route hardly takes any time.

In the first experiment, the computation times for the three different implementation are compared for various instance sizes, i.e., number of customers. The results are shown in Figure 11. They show that the enhancements proposed by Solomon, Baker & Schaffer, substantially reduce the computation times, compared to the straightforward implementation, but that the techniques proposed by Savelsbergh perform even better and reduce the computation times significantly once more.

Besides the number of customers, an other interesting parameter is the width of the time windows. In the second experiment, the effect of the width of the time window on the performance of the three implementations is tested. The time windows of the customers were tightened by a factor two and four. To keep the initial route found earlier feasible, this was done in the following way. If the initial route arrives at time t at a customer, the original time window $[e, l]$ of this customer is transformed into $[(1 - \gamma)t + \gamma e, (1 - \gamma)t + \gamma l]$ with γ equal to 0.50 or 0.25.

The results are presented in Figures 12 and 13. One would expect that the Solomon, Baker & Schaffer algorithm benefits more from smaller time windows than the lexicographic search approach, since smaller time windows imply more precedence relations. However, this does not seem to be the case. A simple analysis of the techniques based on lexicographic search reveals that they in fact identify and check the precedence relations on-line as a subset of the feasibility tests performed [Knops, 1993].
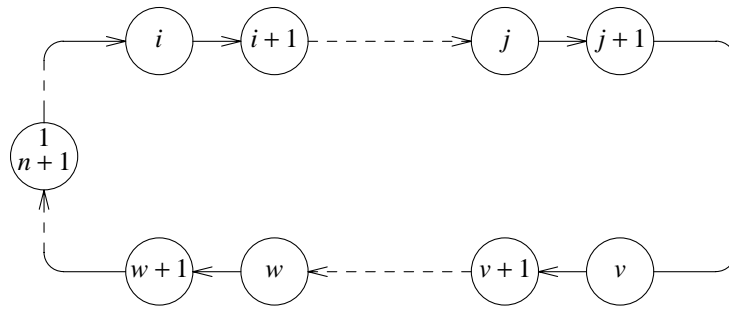
(a) The initial tour.
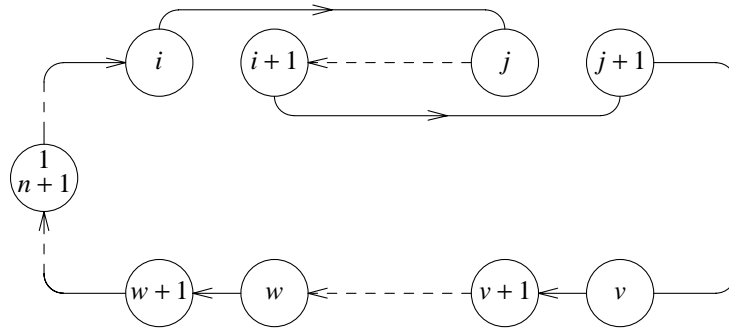
(b) The tour after the first step.
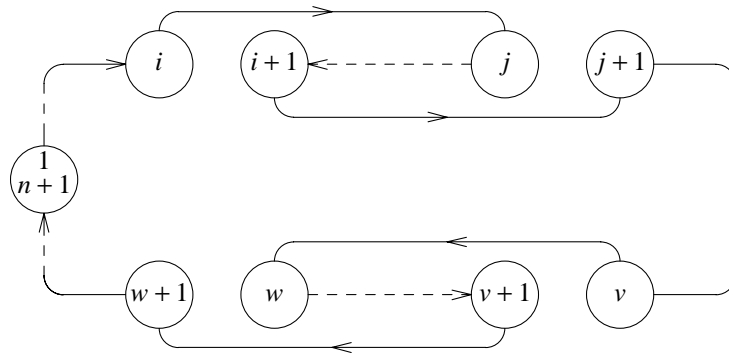
(c) The tour after the second step.

Figure 10 A variable depth exchange.

4 PARALLEL IMPLEMENTATIONS

Local search algorithms for vehicle routing and scheduling problems are often embedded in interactive planning systems for physical distribution management. Any algorithm embedded in an interactive system should have an acceptable response time. Consequently, speed is a primary concern. Nowadays, many computers are able to perform a number of operations in parallel. Such computers have a greater processing power than a serial one, thus making it possible to obtain substantial speedups.
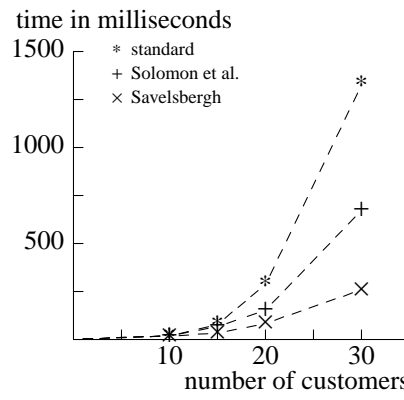
time in milliseconds

Figure 11 Computational results of searching the 2-exchange neighborhood.
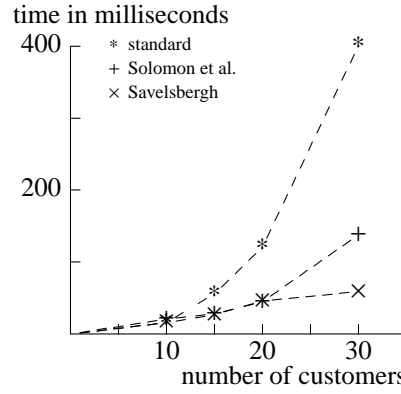
time in milliseconds

Figure 12 The effect of narrowing the time windows, $\gamma = 0.50$.
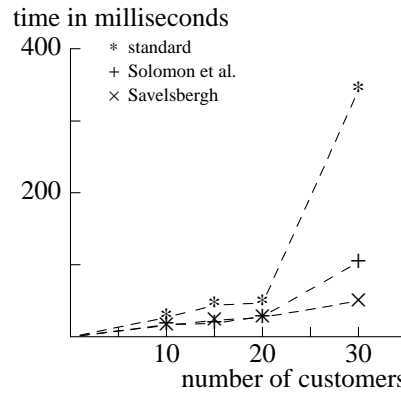
time in milliseconds

Figure 13 The effect of narrowing the time windows, $\gamma = 0.25$.

In this section, we will discuss the verification of local optimality on a parallel random access machine (PRAM), a machine model in which an unbounded number of processors operate in parallel and communicate with each other in constant time through a shared memory. The shared memory allows simultaneous reads from the same location but disallows simultaneous writes into the same location. Although the PRAM

is hardly a realistic computer model, the resulting algorithms can be adequately used for implementation on any 'real-world' machine and the overhead introduced is only minimal (see for example Alt, Hagerup, Mehlhorn & Preparata [1987] and Karlin & Upfal [1988]).

Before addressing the issue of local optimality, we will first consider the *partial sums* problem and describe a basic technique in parallel computing for its solution. Then, we will address the verification of local optimality of a single route for the 2-exchange neighborhood in the presence of time windows. We will not deal with the other cases considered in this paper, but we only mention that parallel algorithms can be obtained in much the same way.

Partial sums

For the sake of simplicity, let $n = 2^m$ and suppose that n numbers are given by $a_n, a_{n+1}, \ldots, a_{2n-1}$. We wish to find the partial sums $b_{n+j} = a_n + \cdots + a_{n+j}$ for $j = 0, \ldots, n-1$. The following procedure is due to Dekel & Sahni [1983]; it consists of two phases:

> **for** $l \leftarrow m - 1$ **downto** 0 **do**
> > **par** $[2^l \leq j \leq 2^{l+1} - 1]$ $a_j \leftarrow a_{2j} + a_{2j+1}$;
>
> **for** $l \leftarrow 0$ **to** m **do**
> > **par** $[2^l \leq j \leq 2^{l+1} - 1]$
> > > $b_j \leftarrow$ **if** $j = 2^l$ **then** a_j **else if** j odd **then** $b_{(j-1)/2}$ **else** $b_{(j-2)/2} + a_j$.

Here, a statement of the form '**par** $[\alpha \leq j \leq \omega]$ s_j' denotes that the statements s_j are executed in parallel for all values of j in the indicated range.
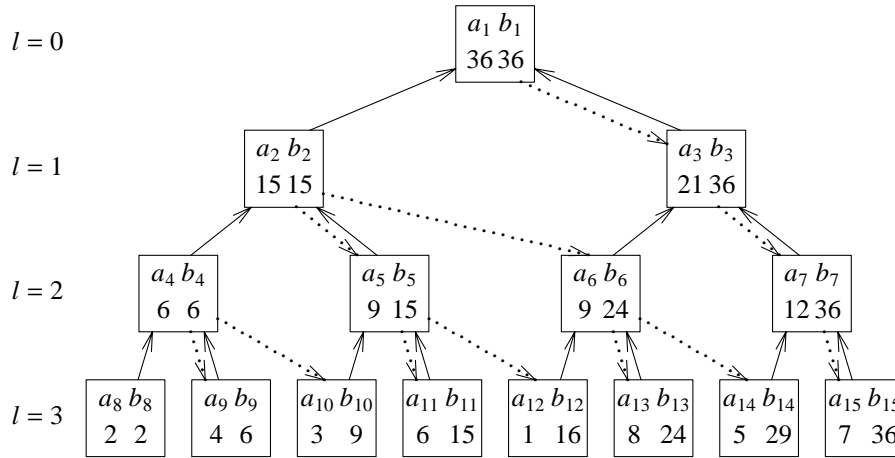

Figure 14 Partial sums: an instance with $n = 8$.

The computation is illustrated in Figure 14. In the first phase, represented by solid arrows, the sum of the a_j's is calculated. The a-value corresponding to a non-leaf node is set equal to the sum of all a-values corresponding to the leaves descending from that node. In the second phase, represented by dotted arrows, the b-value of a certain node is set equal to the sum of all a-values of the nodes of the same generation, except those with a higher index. This implies, in particular, that at the end we have $b_{n+j} = a_n + \cdots + a_{n+j}$ for $j = 0, \ldots, n-1$.

The algorithm requires $O(\log n)$ time and n processors. This can be improved to $O(\log n)$ time and $O(n/\log n)$ processors by a simple device. First, the set of n numbers is partitioned into $n/\log n$ groups of size $\log n$ each, and $n/\log n$ processors determine the sum of each group in the traditional serial way in $\log n$ time. After this aggregation process, the above algorithm computes the partial sums over the groups; this requires $O(n/\log n)$ processors and $O(\log n)$ time. Finally, a disaggregation process is applied with the same

processor and time requirements. The total computational effort is $O(\log n \cdot n/\log n) = O(n)$, as it is in the serial case. This is called a *full processor utilization* or a *perfect speedup*.

The technique described above is widely used in parallel computing. In many situations a kind of partial sums problem has to be solved where the addition is replaced by some other associative binary operator.

Verification of local optimality

We now return to the verification of local optimality. We start by deciding whether or not the tour $(1, 2, \ldots, n, n + 1)$ is 2-optimal:

> **par** $[1 \le i < j \le n]$ $\delta_{ij} \leftarrow d_{ij} + d_{i+1,j+1} - d_{i,i+1} - d_{j,j+1}$; (* compute the effect of all possible 2-exchanges *)
> $\delta_{\min} \leftarrow \min\{\delta_{ij} \mid 1 \le i < j \le n\}$;   (* search for the best 2-exchange *)
> **if** $\delta_{\min} \ge 0$
> **then** $(1, 2, \ldots, n, n + 1)$ is a 2-optimal tour
> **else** let $i *$ and $j *$, with $i* < j *$, be such that $\delta_{i* j*} = \delta_{\min}$,
> $(1, \ldots, i*, j*, j * -1, \ldots, i * +1, j * +1, \ldots, n + 1)$ is a shorter tour.

By adapting the first phase of the partial sums algorithm such that it computes the minimum of a set of numbers and also delivers an index for which the minimum is attained, the above procedure can be implemented to require $O(\log n)$ time and $O(n^2/\log n)$ processors. The total computational effort is $O(\log n \cdot n^2/\log n) = O(n^2)$, as it is in the serial case. Hence, we have obtained a perfect speedup.

Although the serial and parallel implementations seem similar, there is a basic distinction. When the tour under consideration is not 2-optimal, the serial algorithm will detect this after a number of steps that is somewhere in between 1 and $\binom{n}{2}$. In the parallel algorithm, confirmation and negation of 2-optimality or Or-optimality always take the same amount of time.

The presence of time windows complexifies the situation. Consider the tour $(1, 2, \ldots, n, n + 1)$, which is assumed to be feasible. We start by looking at all partial paths along the tour. This enables us to construct the tours that can be obtained by a 2-exchange. For each path $(u_1, \ldots, u_k)$, we define $A_{(u_1, \ldots, u_k)}(t)$ as the earliest possible arrival time at vertex u_k when traveling along the path from vertex u_1 to vertex u_k after arriving at vertex u_1 at time t. Note that $A_{(1, \ldots, n+1)}(e_1)$ is the arrival time at vertex $n + 1$ of the given tour.

Our algorithm has three phases.

(1) First, we compute the functions A for paths consisting of only one edge.

> **par** $[1 \le i \ne j \le n + 1]$ $A_{(i,j)}(t) \leftarrow \begin{cases} \max\{e_i, t\} + d_{ij} & \text{for } t \le l_i, \\ \infty & \text{for } t > l_i. \end{cases}$

Next, we compute the functions A for all paths along the tour in both directions by composition.

> **par** $[1 \le i \le n + 1]$ **par** $[i \le j \le n + 1]$ $A_{(i,i+1,\ldots,j-1,j)}(t) \leftarrow (A_{(j-1,j)} \circ \cdots \circ A_{(i,i+1)})(t)$;
> **par** $[1 \le i \le n + 1]$ **par** $[i \le j \le n + 1]$ $A_{(j,j-1,\ldots,i+1,i)}(t) \leftarrow (A_{(i+1,i)} \circ \cdots \circ A_{(j,j-1)})(t)$.

By considering all possibilities, one can show that each of these functions has one of the three shapes shown in Figure 15. Composing functions is an associative operation. Hence, we can use the partial sums algorithm for obtaining the functions A in parallel. Since a composition of two functions of the type described here can be derived in constant time, we can in fact determine all functions A in $O(\log n)$ time with $O(n^2/\log n)$ processors.

(2) Given the functions A for all paths along the tour, we can compute them for the tours that are obtained after the replacement of the edges $\{i, i + 1\}$ and $\{j, j + 1\}$ by the edges $\{i, j\}$ and $\{i + 1, j + 1\}$:
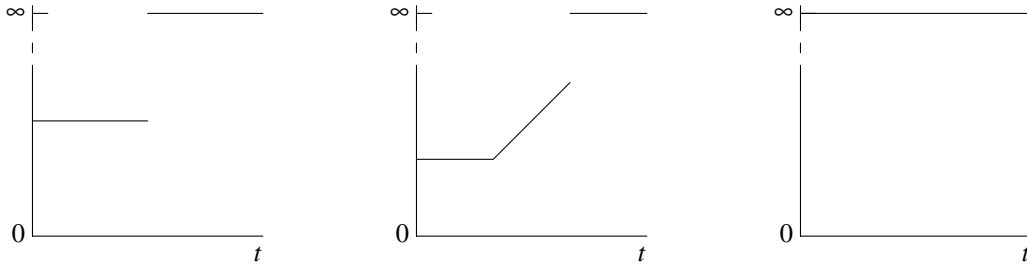
Figure 15 The three possible shapes of the functions A.

$\mathbf{par}\ [1 \le i < j \le n]\ A_{(1,\ldots,i,j,\ldots,i+1,j+1,\ldots,n+1)}(t) \leftarrow (A_{(j+1,\ldots,n+1)} \circ A_{(i+1,j+1)} \circ A_{(j,\ldots,i+1)} \circ A_{(i,j)} \circ A_{(1,\ldots,i)})(t)$.

For this phase we need $O(1)$ time and $O(n^2)$ processors, or $O(\log n)$ time and $O(n^2/\log n)$ processors.

(3) We decide whether or not the given tour is 2-optimal in the same way as in the case without time windows:

$A_{\min} \leftarrow \min\{A_{(1,\ldots,i,j,\ldots,i+1,j+1,\ldots,n+1)}(e_1)\,|\,1 \le i < j \le n\}$;
$\mathbf{if}\ A_{(1,\ldots,n+1)}(e_1) \le A_{\min}$
$\mathbf{then}\ (1, 2, \ldots, n, n + 1)$ is a 2-optimal tour
$\mathbf{else}$ let $i*$ and $j*$, with $i* < j*$, be such that $A_{(1,\ldots,i*,j*,\ldots,i*+1,j*+1,\ldots,n+1)} = A_{\min}$,
$\qquad (1, \ldots, i*, j*, j*-1, \ldots, i*+1, j*+1, \ldots, n+1)$ is a better feasible tour.

For this last phase, the same time and processor bounds as before suffice. So, we end up with an algorithm that runs in $O(\log n)$ time using $O(n^2/\log n)$ processors, which is the same as in the unconstrained case.

For the verification of k-optimality for fixed $k > 2$, we can derive a logarithmic-time algorithm along similar lines. One has to take into account that, given k edges, several k-exchanges are possible. Further, the influence of a k-exchange on a tour is more complex. However, the running time remains $O(\log n)$ using $O(n^k/\log n)$ processors, which is optimal with respect to the number $\Theta(n^k)$ of k-exchanges.

5 SIMULATED ANNEALING AND TABU SEARCH

Not surprisingly, people have started to investigate the potential of recently developed variants of local search, such as simulated annealing [Kirkpatrick, Gelatt & Vecchi, 1983] and tabu search [Glover, 1989, 1990], for the solution of vehicle routing and scheduling problems. For a recent survey of these methods, see Gendreau, Laporte & Potivin [1995]. The majority of the simulated annealing algorithms and the tabu search algorithms for vehicle routing and scheduling problems use one or more of the neighborhoods described in Section 2 and some of the techniques discussed in Section 3. The simulated annealing algorithms [Alfa, Heragu & Chen, 1991; Osman, 1993] mainly differ in their choice of cooling scheme and parameters; the tabu search algorithms [Gendreau, Hertz & Laporte, 1994; Potvin, Kervahut & Rousseau, 1992; Semet & Taillard, 1993] mainly differ in their choice of tabu list structure and parameters. We will not describe any of these algorithms in detail. It suffices to say that the initial computational results are promising and indicate that there is indeed potential for such algorithms.

REFERENCES

A.S. Alfa, S.S. Heragu, M. Chen (1991). A 3-opt based simulated annealing algorithm for vehicle routing problems. *Comput. Indust. Eng. 21*, 635-639.

H. Alt, T. Hagerup, K. Mehlhorn, F.P. Preparata (1987). Deterministic simulation of idealized parallel computers on more realistic ones. *SIAM J. Comput. 16*, 808-835.

D.S. Johnson, L.A. McGeogh (1995). The traveling salesman problem. E.H.L. Aarts, J.K. Lenstra (eds.). *Local Search in Combinatorial Optimization*, Wiley, Chichester, to appear.

L. Bodin, B. Golden, A. Assad, M. Ball (1983). Routing and scheduling of vehicles and crews - the state of

the art. *Comput. Oper. Res. 10*, 63-211.

L.J.J. van der Bruggen, J.K. Lenstra, P.C. Schuur (1993). Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transp. Sci. 27*, 298-311.

M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh, F. Soumis (1988). Vehicle routing with time windows: optimization and approximation. B.L. Golden, A.A. Assad (eds.). *Vehicle Routing: Methods and Studies*, North Holland, Amsterdam, 65-84.

E. Dekel, S. Sahni (1983). Binary trees and parallel scheduling algorithms. *IEEE Trans. Comput. C-32*, 307-315.

M. Gendreau, A. Hertz, G. Laporte (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Oper. Res. 40*, 1086-1094.

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

M. Gendreau, G. Laporte, J.-Y. Potvin (1995). Vehicle routing 1: (subtitel) E.H.L. Aarts, J.K. Lenstra (eds.). *Local Search in Combinatorial Optimization*, Wiley, Chichester, to appear.

F. Glover (1989). Tabu Search, Part I. *ORSA J. Comput. 1*, 190-206.

F. Glover (1990). Tabu Search, Part II. *ORSA J. Comput. 2*, 4-32.

B.L. Golden, A.A. Assad (eds.) (1988). *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam.

A.R. Karlin, E. Upfal (1988). Parallel hashing - an efficient implementation of shared memory. *J. Assoc. Comput. Mach. 35*, 876-892.

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

M. Knops (1993). *Locale Zoekmethoden voor het Handelsreizigersprobleem met Tijdvensters*, MSc. Thesis (in Dutch), Erasmus University, Rotterdam.

G. Laporte, Y. Norbert (1987). Exact algorithms for the vehicle routing problem. *Ann. Discrete Math. 31*, 147-184.

S. Lin, B.W. Kernighan (1973). An effective heuristic algorithm for the traveling salesman problem. *Oper. Res. 21*, 498-516.

I. Or (1976). *Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Blood Banking*, Ph.D. Thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University, Evanston.

I.H. Osman (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Oper. Res. 41*, 421-451.

J.-Y. Potvin, T. Kervahut, J.-M. Rousseau (1992). *A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows*. Working Paper CRT-855, Centre de Recherche sur les Transports, Montreal.

H.N. Psaraftis (1983). *k*-Interchange procedures for local search in a precedence-constrained routing problem. *European J. Oper. Res. 13*, 391-402.

R.A. Russell (1977). An effective heuristic for the *m*-tour traveling salesman problem with some side conditions. *Oper. Res. 25*, 517-524.

M.W.P. Savelsbergh (1986). Local search for routing problems with time windows. *Ann. Oper. Res. 4*, 285-305.

M.W.P. Savelsbergh (1990). Efficient implementation of local search methods for the vehicle routing problems with side constraints. *European J. Oper. Res. 47*, 75-85.

M.W.P. Savelsbergh (1992). The vehicle routing problem with time windows: minimizing route duration. *ORSA J. on Computing 4*, 146-154.

M.W.P. Savelsbergh, M. Sol (1995). The general pickup and delivery problem. *Transp. Sci. 29*, to appear.

F. Semet, E. Taillard (1993). Solving real-life vehicle problems using tabu search. *Ann. Oper. Res. 41*, 469-488.

M.M. Solomon (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res. 35*, 254-265.

M.M. Solomon, E.K. Baker, J.R. Schaffer (1988). Vehicle routing and scheduling problems with time

window constraints: efficient implementations of solution improvement procedures. B.L. Golden, A.A. Assad (eds.). *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam, 85-105.

M.M. Solomon, J. Desrosiers (1988). Time window constrained routing and scheduling problems. *Transp. Sci. 22*, 1-13.

P.M. Thompson, H.N. Psaraftis (1993). Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Oper. Res. 41*, 935-946.

P.M. Thompson, J.B. Orlin (1989). *Theory of Cyclic Transfers*. Working Paper, MIT, Cambridge.