European Journal of Operational Research 47 (1990) 75–85
North-Holland

Theory and Methodology

An efficient implementation of local search algorithms for constrained routing problems

M.W.P. SAVELSBERGH
Centre for Mathematics and Computer Science, Amsterdam, and
Erasmus University, Rotterdam, The Netherlands

Abstract: We investigate the implementation of local search algorithms for routing problems with various side constraints such as time windows on vertices and precedence relations between vertices. The algorithms are based on the k-exchange concept. In the case of unconstrained routing problems, a single k-exchange can obviously be processed in constant time. In the presence of side constraints feasibility problems arise. Testing the feasibility of a single solution requires an amount of time that is linear in the number of vertices. We show how this effort can, on the average, still be reduced to a constant.

Keywords: Vehicle routing problem, side constraints, local search, iterative improvement method, edge exchange, efficient implementation

1. Introduction

Croes (1958) and Lin (1965) introduced local search algorithms for the *traveling salesman problem* (TSP) that were based on the notion of k-exchanges. Lin and Kernighan (1973) generalized the approach, and many authors reported on its application to related problems. In the context of vehicle routing, Christofides and Eilon (1969) and Russell (1977) adapted the approach to the basic VRP, and Psaraftis (1983) used it for the single-vehicle dial-a-ride problem.

In this paper we consider routing problems with side constraints. We will show how local search can be efficiently implemented in these more complicated situations. For the description of the approach we will restrict ourselves to the TSP. However, the presented techniques are of a more general nature and can be applied to other types of routing problems as well.

Our main motivation for the development of k-exchange procedures that can handle side constraints efficiently is a quite practical one. We encountered the need for such algorithms during the development of CAR (Computer Aided Routing), an interactive planning system for physical distribution management (Anthonisse, Lenstra and Savelsbergh, 1987; Savelsbergh, 1988). In many real-life situations, routing algorithms must be capable of handling various side constraints. Frequently occurring side constraints are:

- both collections and deliveries at customers;
- precedence relations between customers;
- time windows at customers.

Interactive planning systems rely on a smooth dialogue between man and machine. To guarantee an acceptable response time in such systems, an efficient implementation of the underlying algorithms is at premium.

Received November 1988; revised May 1989

In the TSP, the processing of a single k-exchange takes constant time for any fixed value of k. One only has to test whether the exchange is profitable and does not have to bother about feasibility. In the presence of side constraints, the processing of a k-exchange may take $O(n)$ time. This is because a modification at one point may affect the entire route, so that feasibility questions arise. It will be indicated below that, even in the presence of side constraints, constant time suffices for the processing of a single k-exchange.

Solomon, Baker and Schaffer (1986) have studied several implementations of local search procedures for the TSP with time windows. By applying preprocessing techniques to streamline feasibility testing and, in case of orientation preserving exchanges, tailored updating schemes, they have been able to incorporate time window constraints with an acceptable increase in computation times. Our implementation technique has the advantage that it is conceptually easy, that it does not increase the computation times, and that it can be applied to various side constraints.

The paper is organized as follows. In Section 2, we will briefly review the k-exchange methods for the TSP. In Section 3, we will describe in fairly general terms the modifications needed to efficiently implement k-exchange methods for constrained variants of the TSP. In Sections 4, 5 and 6, we will treat a number of constrained variants of the TSP in more detail as examples of the proposed approach. In Section 7, we will give some concluding remarks.

2. k-Exchanges for the traveling salesman problem

In the TSP (Lawler, Lenstra, Rinnooy Kan and Shmoys, 1985) we are given a complete graph on a set V of vertices and a travel time t_{ij} for each edge $\{i, j\} \in V \times V$. A solution to the TSP is a route, i.e., a cycle which visits each vertex exactly once. The duration of the route is the sum of the travel times of the edges contained in it. The objective is to find a route of minimum duration. Let $n = |V|$ indicate the number of vertices. We assume that a given vertex, say vertex 0, will serve as the first and last vertex of any route (the depot in vehicle routing and scheduling problems) and that the matrix (t_{ij}) is symmetric and satisfies the triangle inequality.

A 2-exchange involves the substitution of two edges, say $\{i, i+1\}$ and $\{j, j+1\}$, with two other edges $\{i, j\}$ and $\{i+1, j+1\}$ (see Figure 1). Note that the orientation of the path $(i+1, \ldots, j)$ is reversed in the new route. Such an exchange results in a local improvement if and only if

$$t_{i,j} + t_{i+1,j+1} < t_{i,i+1} + t_{j,j+1}.$$

Therefore testing improvement involves only local information and can be done in constant time.

In contrast with a 2-exchange, where the two edges $\{i, i+1\}$ and $\{j, j+1\}$ that will be deleted, uniquely identify the two edges $\{i, j\}$ and $\{i+1, j+1\}$ that will replace them, in a 3-exchange, where three edges are deleted, there are several possibilities to construct a new route from the remaining segments. Figure 2 shows two possible 3-exchanges that can be performed by deleting the edges $\{i, i+1\}$, $\{j, j+1\}$ and $\{k, k+1\}$ of a route.

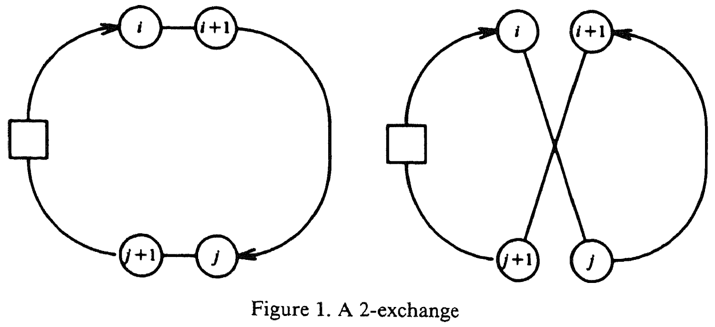

Figure 1. A 2-exchange

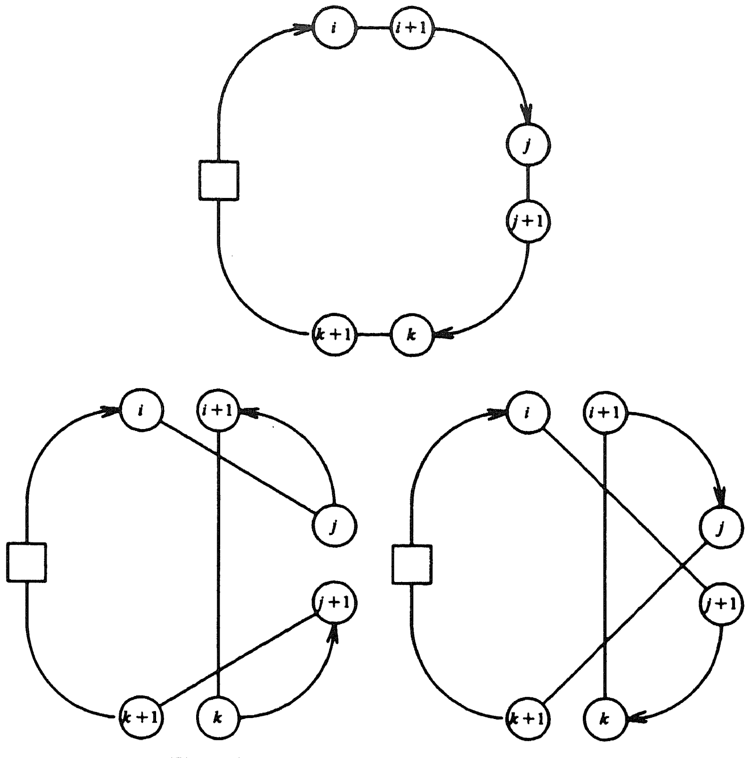

Figure 2. Two ways to perform a 3-exchange

For all possibilities, conditions for improvements are easily derived and again only involve local information and can thus be verified in constant time. There is one important difference between the two 3-exchanges shown above: in the latter the orientation of the paths $(i+1,\ldots,j)$ and $(j+1,\ldots,k)$ is preserved whereas in the former this orientation is reversed.

Because the computational requirement to verify 3-optimality becomes prohibitive if the number of vertices increases, proposals have been made to take only a subset of all possible 3-exchanges into account. Or (1976) proposes to restrict attention to those 3-exchanges in which a string of one, two or three consecutive vertices (a path) is relocated between two others. An Or-exchange is depicted in Figure 3. The path $(i_1,\ldots,i_2)$ is relocated between j and $j+1$. Note that no paths are reversed in this case and that there are only $O(n^2)$ exchanges of this kind.

There are two possibilities for relocating the path $(i_1,\ldots,i_2)$; we can relocate it earlier (backward relocation) or later (forward relocation) in the current route. The cases of backward relocation $(j < i_1)$ and forward relocation $(j > i_2)$ will be handled separately below.

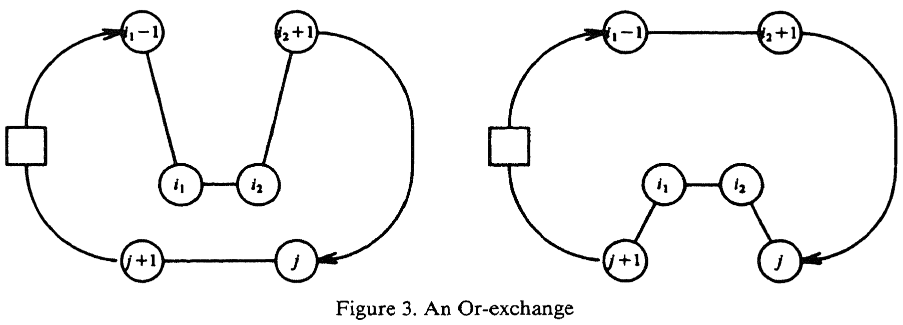

Figure 3. An Or-exchange

3. k-Exchange for constrained traveling salesman problems

The main problem with the use of k-exchange procedures in the TSP with side constraints is testing the feasibility of an exchange. A 2-exchange, for instance, will reverse the path $(i+1,\dots,j)$, which means that one has to check the feasibility of all the vertices on the new path with respect to those constraints. In a straightforward implementation this requires $O(n)$ time for each 2-exchange, which results in a time complexity of $O(n^3)$ for the verification of 2-optimality.

The basic idea of the proposed approach 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.

In the sequel we will assume that the current route is given by a sequence $(0,\dots,i,\dots,n)$, where i represents the i-th vertex of the route and where we have split the vertex that serves as first and last vertex of any route (vertex 0) in an 'origin' (vertex 0) and a 'destination' (vertex n). We also assume that we are always examining the exchange that involves the substitution of edges $\{i,i+1\}$ and $\{j,j+1\}$ with $\{i,j\}$ and $\{i+1,j+1\}$ in case of a 2-exchange, and the substitution of $\{i_1-1,i_1\}$, $\{i_2,i_2+1\}$ and $\{j,j+1\}$ with $\{i_1-1,i_2+1\}$, $\{j,i_1\}$ and $\{i_2,j+1\}$ in case of an Or-exchange.

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 $\{0,1\}$; this will be referred to as the outer loop. After fixing an edge $\{i,i+1\}$, we choose the edge $\{j,j+1\}$ to be $\{i+2,i+3\}$, $\{i+3,i+4\},\dots$, $\{n-1,n\}$ in that order (see Figure 4); this will be referred to as the inner loop.

Now consider all possible exchanges for a fixed edge $\{i,i+1\}$. The ordering of the 2-exchanges given above implies that in the inner loop in each newly examined 2-exchange the path $(i+1,\dots,j-1)$ of the previously considered 2-exchange, i.e., 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 backward Or-exchanges. We choose the path $(i_1,\dots,i_2)$ in the order of the current route starting with i_1 equal to 2. After the path $(i_1,\dots,i_2)$ has been fixed, we choose the edge $\{j,j+1\}$ to be $\{i_1-2,i_1-1\}$, $\{i_1-3,i_1-2\}$, $\dots$, $\{0,1\}$ 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 $(j+2,\dots,i_1-1)$ of the previously considered exchange is expanded with the edge $\{j+1,j+2\}$.

Lexicographic search for forward Or-exchanges. We choose the path $(i_1,\dots,i_2)$ in the order of the current route starting with i_1 equal to 1. After the path $(i_1,\dots,i_2)$ has been fixed, we choose the edge $\{j,j+1\}$ to be $\{i_2+1,i_2+2\}$, $\{i_2+2,i_2+3\}$, $\dots$, $\{n-1,n\}$ in that order. That is, the edge $\{j,j+1\}$ 'walks forward' through the route. Note that in each newly examined exchange the path $(i_2+1,\ \dots,j-1)$ of the previously considered exchange is expanded with the edge $\{j-1,j\}$.

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 path $(i+1,\dots,j)$, and in order to test the feasibility of a single forward (or backward) Or-exchange, we have to check all the vertices on the path $(i_2+1,\dots,j)$ (or $(j+1,\dots,i_1-1)$, respectively). 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.

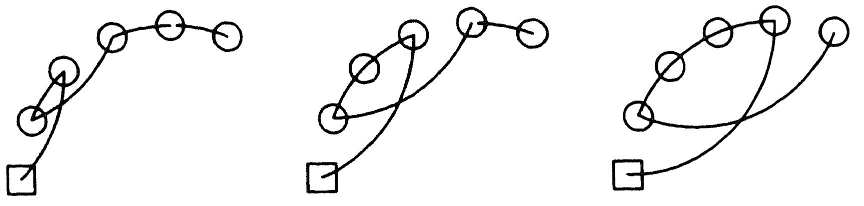
Figure 4. The lexicographic search strategy for 2-exchanges

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

first, the set of global variables makes it possible to test the feasibility of an exchange, i.e., to check the feasibility of all the vertices on the path in question, in constant time, and

secondly, the lexicographic search strategy makes it possible to maintain the correct values for the set of global variables, i.e., update them when we go from one exchange to the next one, 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 Two Exchange
(* input: a route given as $(0,1,\ldots,n)$*)
(* output: a route that is 2-optimal *)
begin
START:
  for $i := 0$ to $n$ do
  begin
    InitGlobal($i,G$)
    for $j := i + 2$ to $n$ do
      if $t_{i,j} + t_{i+1,j+1} < t_{i,i+1} + t_{j,j+1}$ and FeasibleExchange($i,j,G$) then
      begin
        PerformExchange($i,j$)
        goto START
      end
      UpdateGlobal($i,j,G$)
    end
  end
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 InitGlobal( ), FeasibleExchange( ), and UpdateGlobal( ) do what they are supposed to do and take only constant time, is often not so obvious.

If we compare the above framework with the straightforward implementation, we observe the following. Our implementation guarantees that only constant time is spent on a single exchange, which implies an $O(n^2)$ method for verifying 2-optimality, whereas in the straight-forward implementation, the time spent on a single exchange depends on the effort needed to establish either its feasibility or its infeasibility, which only implies an $O(n^3)$ method for verifying 2-optimality. One may argue that the straightforward implementation could be more efficient on the average: in case the exchange is not profitable, no feasibility has to be tested, and no global variables have to be updated either. An in-depth empirical analysis would be required to settle this issue, and the answer would undoubtedly depend on the extend to which the side constraints actually restrict the search for an optimum. However, as the constant time required by our suggested implementation is very small for the applications described in the next sections (not more than ten additions, subtractions, comparisons, or assignments) it does not only produce a nice theoretical result, but is also quite effective in practice.

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 abitrary k), our techniques can be used to implement k-exchange algorithms for the TSP with side constraints for arbitrary k without increasing the time complexity beyond $O(n^k)$.

4. The traveling salesman problem with mixed collections and deliveries

In the TSP with mixed collections and deliveries, we are given in addition to the travel times between vertices, for each vertex i an associated load q_i together with a specification that indicates whether this load should be collected or delivered. The salesman uses a vehicle with fixed capacity Q.

The following quantities will be helpful for the description of the algorithm. Let Γ be the set of customers where the salesman has to make a collection and Δ the set of customers where the salesman has to make a delivery. Now, given a feasible route $(0,1,\ldots,n)$, we define

$$C(r, s) := \sum_{k\in(r,\ldots,s),\, k\in\Gamma} q_k, \qquad D(r, s) := \sum_{k\in(r,\ldots,s),\, k\in\Delta} q_k.$$

A route is feasible if and only if

$$0 \leqslant C(0, k) - D(0, k) \leqslant Q \quad \text{for } k = 0, \ldots, n.$$

2-Exchanges. Consider the 2-exchange where the edges $\{i,i+1\}$ and $\{j,j+1\}$ are replaced by the edges $\{i,j\}$ and $\{i+1,j+1\}$. In the following a quantity with superscript 'new' indicates the value after the exchange has been carried out, and arguments always refer to the ordering of the current route. If the exchange would be carried out, the quantities that determine feasibility can be expressed in terms of the quantities of the current route as follows:

$$\begin{aligned}
C^{\text{new}}(0, k) &= C(0, k) && \text{for } 0 \leqslant k \leqslant k, \quad j+1 \leqslant k \leqslant n,\\
D^{\text{new}}(0, k) &= D(0, k) && \text{for } 0 \leqslant k \leqslant i, \quad j+1 \leqslant k \leqslant n,\\
C^{\text{new}}(0, k) &= C(0, i) + C(k, j) && \text{for } i+1 \leqslant k \leqslant j,\\
D^{\text{new}}(0, k) &= D(0, i) + D(k, j) && \text{for } i+1 \leqslant k \leqslant j.
\end{aligned}$$

The exchange is feasible if an only if

$$0 \leqslant C(0, i) - D(0, i) + \min_{k\in(i+1,\ldots,j)} \{C(k, j) - D(k, j)\},$$

$$C(0, i) - D(0, i) + \max_{k\in(i+1,\ldots,j)} \{C(k, j) - D(k, j)\} \leqslant Q.$$

If we introduce global variables for $C(0,i) - D(0,i)$, $\min_{k\in(i+1,\ldots,j)}\{C(k,j) - D(k,j)\}$ and $\max_{k\in(i+1,\ldots,j)}\{C(k,j) - D(k,j)\}$, it is clear from the above inequalities that testing the feasibility of a single exchange takes constant time. The lexicographic search strategy allows us to maintain the global variables efficiently; $C(0,i) - D(0,i)$ can be updated in constant time in the outer loop, and both $\min_{k\in(i+1,\ldots,j)}\{C(k,j) - D(k,j)\}$ and $\max_{k\in(i+1,\ldots,j)}\{C(k,j) - D(k,j)\}$ can be updated in constant time in the inner loop.

Or-exchanges. Consider the backward Or-exchange, where the path $(i_1,\ldots,i_2)$ is relocated backward between j and $j+1$. We find that

$$\begin{aligned}
C^{\text{new}}(0, k) &= C(0, k) && \text{for } 0 \leqslant k \leqslant j, \quad i_2+1 \leqslant k \leqslant n,\\
D^{\text{new}}(0, k) &= D(0, k) && \text{for } 0 \leqslant k \leqslant j, \quad i_2+1 \leqslant k \leqslant n,\\
C^{\text{new}}(0, k) &= C(0, k) + C(i_1, i_2) && \text{for } j+1 \leqslant k \leqslant i_1 - 1,\\
D^{\text{new}}(0, k) &= D(0, k) + C(i_1, i_2) && \text{for } j+1 \leqslant k \leqslant i_1 - 1,\\
C^{\text{new}}(0, k) &= C(0, k) - C(j+1, i_1-1) && \text{for } i_1 \leqslant k \leqslant i_2,\\
D^{\text{new}}(0, k) &= D(0, k) - D(j+1, i_1-1) && \text{for } i_1 \leqslant k \leqslant i_2.
\end{aligned}$$

The exchange is feasible if and only if

$$0 \leqslant \min_{k \in (j+1, \ldots, i_1 - 1)} \{C(0, k) - D(0, k)\} + C(i_1, i_2) - D(i_2, i_2),$$

$$\max_{k \in (j+1, \ldots, i_1 - 1)} \{C(0, k) - D(0, k)\} + C(i_1, i_2) - D(i_1, i_2) \leqslant Q,$$

$$0 \leqslant \min_{k \in (i_1, \ldots, i_2)} \{C(0, k) - D(0, k)\} - C(j+1, i_1 - 1) + D(j+1, i_1 - 1),$$

$$\max_{k \in (i_1, \ldots, i_2)} \{C(0, k) - D(0, k)\} - C(j+1, i_1 - 1) + D(j+1, i_1 - 1) \leqslant Q.$$

We rewrite this as

$$D(i_1, i_2) - C(i_1, i_2) \leqslant \min_{k \in (j+1, \ldots, i_1 - 1)} \{C(0, k) - D(0, k)\},$$

$$C(i_1, i_2) - D(i_1, i_2) \leqslant Q - \max_{k \in (j+1, \ldots, i_1 - 1)} \{C(0, k) - D(0, k)\},$$

$$C(j+1, i_1 - 1) - D(j+1, i_1 - 1) \leqslant \min_{k \in (i_1, \ldots, i_2)} \{C(0, k) - D(0, k)\},$$

$$D(j+1, i_1 - 1) - C(j+1, i_1 - 1) \leqslant Q - \max_{k \in (i_1, \ldots, i_2)} \{C(0, k) - D(0, k)\}.$$

We have now accomplished our goal: if we introduce global variables for $D(i_1,i_2) - C(i_1,i_2)$, $\min_{k \in (i_1,\ldots,i_2)}\{C(0,k) - D(0,k)\}$, $\max_{k \in (i_1,\ldots,i_2)}\{C(0,k) - D(0,k)\}$, $C(j+1,i_1-1) - D(j+1,i_1-1)$, $\min_{k \in (j+1,\ldots,i_1-1)}\{C(0,k) - D(0,k)\}$, and $\max_{k \in (j+1,\ldots,i_1-1)}\{C(0,k) - D(0,k)\}$, it is clear from the above inequalities that testing feasibility of a single exchange can be done in constant time. In addition, the lexicographic search strategy allows us to maintain the global variables efficiently; $D(i_1,i_2) - C(i_1,i_2)$, $\min_{k \in (i_1,\ldots,i_2)}\{C(0,k) - D(0,k)\}$, and $\max_{k \in (i_1,\ldots,i_2)}\{C(0,k) - D(0,k)\}$ can be updated in constant time in the outer loop, and $C(j+1,i_1-1) - D(j+1,i_1-1)$, $\min_{k \in (j+1,\ldots,i_1-1)}\{C(0,k) - D(0,k)\}$, and $\max_{k \in (j+1,\ldots,i_1-1)}\{C(0,k) - D(0,k)\}$ can be updated in constant time in the inner loop.

Consider the forward Or-exchange, where the path $(i_1,\ldots,i_2)$ is relocated forward between j and $j+1$. Analogously to the backward Or-exchange, we find that

$$\begin{array}{ll}
C^{\text{new}}(0, k) = C(0, k) & \text{for } 0 \leqslant k \leqslant i_1 - 1, \quad j+1 \leqslant k \leqslant n, \\
D^{\text{new}}(0, k) = D(0, k) & \text{for } 0 \leqslant k \leqslant i_1 - 1, \quad j+1 \leqslant k \leqslant n, \\
C^{\text{new}}(0, k) = C(0, k) - C(i_1, i_2) & \text{for } i_2 + 1 \leqslant k \leqslant j, \\
D^{\text{new}}(0, k) = D(0, k) - D(i_1, i_2) & \text{for } i_2 + 1 \leqslant k \leqslant j, \\
C^{\text{new}}(0, k) = C(0, k) + C(i_2 + 1, j) & \text{for } i_1 \leqslant k \leqslant i_2, \\
D^{\text{new}}(0, k) = D(0, k) + D(i_2 + 1, j) & \text{for } i_1 \leqslant k \leqslant i_2.
\end{array}$$

The exchange is feasible if and only if

$$0 \leqslant \min_{k \in (i_2+1, \ldots, j)} \{C(0, k) - D(0, k)\} - C(i_1, i_2) + D(i_1, i_2),$$

$$\max_{k \in (i_2+1, \ldots, j)} \{C(0, k) - D(0, k)\} - C(i_1, i_2) + D(i_1, i_2) \leqslant Q,$$

$$0 \leqslant \min_{k \in (i_1, \ldots, i_2)} \{C(0, k) - D(0, k)\} + C(i_2 + 1, j) - D(i_2 + 1, j),$$

$$\max_{k \in (i_1, \ldots, i_2)} \{C(0, k) - D(0, k)\} + C(i_2 + 1, j) - D(i_2 + 1, j) \leqslant Q.$$

We rewrite this as

$$C(i_1, i_2) - D(i_1, i_2) \leqslant \min_{k \in (i_2+1, \ldots, j)} \{C(0, k) - D(0, k)\},$$
$$D(i_1, i_2) - C(i_1, i_2) \leqslant Q - \max_{k \in (i_2+1, \ldots, j)} \{C(0, k) - D(0, k)\},$$
$$D(i_2+1, j) - C(i_2+1, j) \leqslant \min_{k \in (i_1, \ldots, i_2)} \{C(0, k) - D(0, k)\},$$
$$C(i_2+1, j) - D(i_2+1, j) \leqslant Q - \max_{k \in (i_1, \ldots, i_2)} \{C(0, k) - D(0, k)\}.$$

As in the backward case, we can easily define global variables that allow us to test feasibility and update the global variables in constant time.

5. The traveling salesman problem with precedence constraints

In the TSP with precedence constraints we are given in addition to the travel times, 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. Each customer has a pickup and delivery location and the pickup must precede the delivery. Psaraftis (1983) shows that the k-exchange improvement methods can be modified to handle these restrictions. By a straightforward choice of the set of global variables, our proposed method produces the same result.

To describe the precedence relations, we attach a label to each vertex containing the following information:

$$\text{prec}(v) := \begin{cases} u & \text{if vertex } v \text{ must precede vertex } u, \\ -u & \text{if vertex } u \text{ must precede vertex } v, \\ 0 & \text{if vertex } v \text{ has no precedence relation with other vertices.} \end{cases}$$

Feasibility checking can now be accomplished by a simple marking mechanism based on these labels and an appropriate set of global variables. In the following, when we refer to a 'successor' or 'predecessor' of a vertex, we will always mean its uniquely defined precedence-related successor or precedence-related predecessor, and not a successor or predecessor determined by the current ordering of the route.

2-Exchanges. A 2-exchange is feasible if and only if there is no pair of precedence-related vertices on the path $(i+1, \ldots, j)$:

$$v \in (i+1, \ldots, j) \Rightarrow |\text{prec}(v)| \notin (i+1, \ldots, j).$$

We associate a global variable mark(v) with each vertex $v \in V$ as follows:

$$\text{mark}(v) := \begin{cases} 1 & \text{if prec}(v) > 0 \wedge |\text{prec}(v)| \in (i+1, \ldots, j-1), \\ 0 & \text{otherwise.} \end{cases}$$

With these global variables, the feasibility of exchanges can be tested in constant time. Whenever we try to expand the path $(i+1, \ldots, j-1)$ with the edge $\{j-1, j\}$ and mark(j) = 1, vertex j has a predecessor that is already on the path, which implies that expansion of the path will result in infeasible exchanges. What remains to be shown is that we can maintain these global variables efficiently. Again, the lexicographic search strategy provides a simple way to accomplish this. In the inner loop, whenever we expand the path $(i+1, \ldots, j-1)$ with the edge $\{j-1, j\}$, we test if vertex j has a successor, and if so we set the variable associated with this successor to 1:

– If prec(j) > 0, then mark (prec(j)) ← 1.

Now note that if the inner loop is terminated because a marked vertex is encountered, it is very well possible that there are other marked vertices on the path $(j+1, \ldots, n)$. Fortunately, we do not have to reset all marked vertices on the path $(j+1,\dots,n)$ but just the successor, if any, of vertex i, because this is the only global variable that is no longer valid. This introduces one additional action in the outer loop:

– If $\text{prec}(i) > 0$, then $\text{mark}(\text{prec}(i)) \leftarrow 0$.

Or-exchanges. A backward Or-exchange is feasible if and only if there is no pair of precedence-related vertices with one of them on the path $(i_1,\dots,i_2)$ and the other on the path $(j+1,\dots,i_1-1)$:

$$v \in (i_1, \dots, i_2) \Rightarrow |\text{prec}(v)| \notin (j+1, \dots, i_1-1).$$

We associate a global variable $\text{mark}(v)$ with each vertex $v \in V$:

$$\text{mark}(v) := \begin{cases} 1 & \text{if prec}(v) > 0 \wedge \text{prec}(v) \in (i_1, \dots, i_2), \\ 0 & \text{otherwise.} \end{cases}$$

Whenever we try to expand the path $(j+1,\dots,i_1-1)$ with the edge $\{j, j+1\}$ and $\text{mark}(j) = 1$, its successor is on the path $(i_1,\dots,i_2)$, thus implying that expansion will only result in infeasible exchanges. For the backward Or-exchanges the actual marking and resetting of global variables is performed in the outer loop:

– If $\text{mark}(i_1) < 0$, then $\text{mark}(|\text{prec}(i_1)|) \leftarrow 1$.

– If $\text{mark}(i_2+1) < 0$, then $\text{mark}(|\text{prec}(i_2)|) \leftarrow 0$.

A forward Or-exchange is feasible if and only if there is no pair of precedence-related vertices with one of them on the path $(i_1,\dots,i_2)$ and the other on the path $(i_2+1,\dots,j)$:

$$v \in (i_1, \dots, i_2) \Rightarrow |\text{prec}(v)| \notin (i_2+1, \dots, j).$$

We associate a global variable $\text{mark}(v)$ with each vertex $v \in V$:

$$\text{mark}(v) := \begin{cases} 1 & \text{if prec}(v) < 0 \wedge |\text{prec}(v)| \in (i_1, \dots, i_2), \\ 0 & \text{otherwise.} \end{cases}$$

Whenever we try to expand the path $(i_2+1,\dots,j)$ with the edge $\{j, j+1\}$ and vertex $j+1$ is marked, its predecessor is on the path $(i_1,\dots,i_2)$, thus implying that expansion will only result in infeasible exchanges. The actual marking and resetting of global variables is performed in the outer loop:

– If $\text{mark}(i_2) > 0$, then $\text{mark}(\text{prec}(i_2)) \leftarrow 1$.

– If $\text{mark}(i_1-1) > 0$, then $\text{mark}(\text{prec}(i_1-1)) \leftarrow 0$.

6. The traveling salesman problem with time windows

In the TSP with time windows, we are given in addition to the travel times between vertices, 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 does not lead to infeasibility but introduces waiting time at vertex i. 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.

2-Exchanges. A 2-exchange is feasible and profitable if and only if the following conditions are satisfied:

(a) the reversed part of the route is feasible:

$$D_k^{\text{new}} \leqslant l_k \quad \text{for } i < k \leqslant j;$$

(b) the departure time at $j+1$ is decreased:

$$D_{j+1}^{\text{new}} < D_{j+1};$$

(c) part of the gain at $j+1$ can be carried through to the vertex where the salesman finishes:

$$D_k > e_k \quad \text{for } j+1 \leqslant k \leqslant n.$$

The third condition needs some further consideration. If it is violated the exchange will not alter the completion time of the route. It will only reduce the completion time of the path from 0 to $k-1$ for the smallest k for which violation occurs. Although this condition does not create unsurmountable problems, we will drop it for two reasons. First, keeping it will make the presentation of the ideas unnecessarily complicated. Secondly, introducing slack can be very beneficial for the rest of the procedure.

Recall that in the lexicographic search strategy, after the edge $\{i,i+1\}$ is fixed, the edge $\{j,j+1\}$ is chosen to be equal to $\{i+2,i+3\}$, $\{i+3,i+4\}$, ..., $\{n-1,n\}$. This means that once we have fixed the edge $\{i,i+1\}$, we can completely specify an exchange by the other edge involved. In the following, an edge appearing as superscript will specify the exchange on which the information is based.

We define three global variables. First, S^+ is equal to the possible forward shift in time of the departure time at $j-1$ causing no violation of the time window constraints along the path $(i+1,\dots,j-1)$:

$$S^+ := \min_{i+1 \leqslant k \leqslant j-1} \left(l_k - \left(D_{j-1}^{\{j-1,j\}} + \sum_{k \leqslant p \leqslant j-2} t_{p,p+1} \right) \right).$$

Secondly, W indicates the waiting time on the path $(i+1,\dots,j)$, excluding possible waiting time at j, including possible waiting time at $i+1$:

$$W := \sum_{i+1 \leqslant k \leqslant j-1} W_k^{\{j,j+1\}}.$$

Finally, T is equal to the travel time, excluding the periods of waiting, on the path $(i+1,\dots,j)$:

$$T := \sum_{i+1 \leqslant k \leqslant j-1} t_{k,k+1}.$$

The path $(i+1,\dots,j-1)$ of the previously considered exchange is expanded by the edge $\{j-1,j\}$. This usually results in a change of the departure time at $j-1$ (and thus in the change of the departure time of possibly all the other vertices on the path $(i+1,\dots,j-1)$). We define the local variable S to be this change in the departure time at $j-1$:

$$S := D_j^{\{j,j+1\}} + t_{j,j-1} - D_{j-1}^{\{j-1,j\}}.$$

To prove that the lexicographic search strategy in combination with this set of global variables leads to an overall time complexity of $O(n^2)$ for verifying 2-optimality, we have to establish two facts. First, we have to show that these global variables allow us to test the feasibility of a single exchange in constant time. Secondly, we have to show that we can maintain these global variables in constant time as well.

The following lemma enables us to show that the condition (a) for local improvement can be tested in constant time.

Lemma. *Expanding the path $(j-1,\dots,i+1)$ with the edge $\{j-1,j\}$ is feasible if and only if $S \leqslant S^+$.*

The above lemma implies that a 2-exchange is feasible (condition (a)) and potentially profitable (condition (b)) if and only if $D_j^{\{j,j+1\}} \leqslant l_j$, $S \leqslant S^+$, and $D_{j+1}^{\text{new}} < D_{j+1}$. All three conditions can be tested in constant time. Because the triangle inequality holds, traveling directly from i to j takes no more time than through $i+1$, $i+2,\dots,$ $j-1$, so the first condition is always satisfied. The second is just the comparison of two variables. The third requires the exact departure time at vertex $j+1$, which is equal to

$$\min\left(e_{j+1},\ D_j^{\{j,j+1\}} + T + W + t_{i+1,j+1}\right).$$

This establishes the fact that testing the feasibility of a single 2-exchange takes constant time. What remains to be done is to show that the global variables can also be updated in constant time.

An examination of the definition of S shows that it covers two different cases. In the case that $S<0$, as the triangle inequality guarantees that the new arrival at $j-1$ is never earlier than the old arrival, it must have been the case that the old arrival and old departure did not coincide. This means that the old departure was equal to the opening of the time window. But then $|S|$ is exactly equal to the waiting time at $j-1$. In the case that $S \geqslant 0$, S is exactly equal to the difference between the new arrival time and the old arrival time at $j-1$, that is, the forward shift in time. Therefore the global variables can be updated in constant time as follows:

$$T \leftarrow T + t_{j,\,j-1}, \quad W \leftarrow \max(W - S, 0), \quad S^+ \leftarrow \min\left(l_j - D_j^{\{j,\,j+1\}},\ S^+ - S\right).$$

A more thorough discussion of the application of these techniques to the TSP with time windows, including a proof of the above lemma, a proof of the correctness of the updating formulas, the forward Or-exchange, and the backward Or-exchange can be found in Savelsbergh (1986). Savelsbergh (1988) discusses the extension of these methods to the case in which there are multiple time windows per vertex.

7. Conclusion

We have described an implementation technique that enables us to modify the 2-exchange and Or-exchange algorithms for local improvement in the TSP in such a way that various side constraints can be handled without increasing the time complexity. Although we have only demonstrated its use on a number of constrained variants of the TSP, each time concentrating on one specific side constraint, it is easy to combine these and to implement a local search procedure that handles various side constraints at the same time. The growing importance of side constraints in practical distribution management and the need for fast implementations of algorithms in the context of interactive planning systems are encouragements for fundamental research in this area. This paper is intended as a contribution in this direction.

References

Anthonisse, J.M., Lenstra, J.K., and Savelsbergh, M.W.P. (1987), "Functional description of CAR, an interactive system for computer aided routing", Report OS-R8716, Centre for Mathematics and Computer Science, Amsterdam.

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

Croes, A. (1958), "A method for solving traveling salesman problems", *Operational Research* 5, 791–812.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., and Shmoys, D.B. (eds.) (1985), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

Lin, S. (1965), "Computer solutions to the traveling salesman problem", *Bell System Technical Journal* 44, 2245–2269.

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

Or, I. (1976), "Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking", PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL.

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

Russell, R.A. (1977), "An effective heuristic for the m-tour traveling salesman problem with some side constraints", *Operational Research* 25, 517–524.

Savelsbergh, M.W.P. (1986), "Local search for routing problems with time windows", *Annals of Operations Research* 4, 285–305.

Savelsbergh, M.W.P. (1988), "Computer aided routing", PhD thesis, Centre for Mathematics and Computer Science, Amsterdam.

Solomon, M.M., Baker, E.K., and Schaffer, J.R. (1986), "Vehicle routing and scheduling problems with time window constraints: Efficient implementations of solution improvement procedures", Working paper 87-03, Northeastern University.