ORSA Journal on Computing
Vol. 4, No. 2, Spring 1992

0899-1499/92/0402-0146 $01.25

The Vehicle Routing Problem with Time Windows: Minimizing Route Duration

MARTIN W. P. SAVELSBERGH *Eindhoven University of Technology, P. O. Box 513, 5600 MB Eindhoven, The Netherlands, EMAIL: mwps@bs.win.tue.nl*

(Received: October 1989; final revision received: January 1991; accepted: July 1991)

We investigate the implementation of edge-exchange improvement methods for the vehicle routing problem with time windows with minimization of route duration as the objective. The presence of time windows as well as the chosen objective cause verification of the feasibility and profitability of a single edge-exchange to require an amount of computing time that is linear in the number of vertices. We show how this effort can, on the average, be reduced to a constant.

Over the past 10 years, operations reseachers interested in vehicle routing and scheduling have emphasized the development of algorithms for real-life problems. The size of the problems solved has increased and practical side constraints are no longer ignored. One such constraint is the specification of time windows at customers, i.e., time intervals during which they must be served. These lead to mixed routing and scheduling problems.

Obviously, the introduction of time windows at customers makes it harder to construct and maintain a feasible set of routes. Savelsbergh [3] shows that in the special case of a single vehicle, i.e., the traveling salesman problem (TSP) with time windows, constructing a feasible route is already NP-hard. As a result, most of the research in this area has been directed toward controlling the feasibility of a set of routes.

However, the introduction of time windows at customers also allows the specification of more realistic objective functions, compared to minimizing distance, such as minimizing waiting time, minimizing completion time, and minimizing route duration.

Edge-exchange improvement methods form an important and popular class of algorithms in the context of vehicle routing problems. Recently a number of papers have been published that study efficient implementations of edge-exchange improvement methods for the vehicle routing problem with time windows, such as Savelsbergh[3, 4] and Solomon et al.[6]. However, they concentrate solely on the feasibilty aspect, neglecting the fact that identifying profitable exchanges for realistic objective functions can be as difficult as identifying feasible exchanges.

This paper studies efficient implementations of edge-exchange improvement methods when the objective is to minimize route duration and the departure time of a vehicle at the depot is not fixed, but has to fall within a time window, as is the case in many real-life situations.

We will generalize and extend the techniques presented in Savelsbergh[4] that enable incorporation of time window constraints in edge-exchange improvement methods without increasing their complexity, and show that minimizing route duration, with variable departure times at the depot, can also be handled without increasing the complexity. As in Savelsbergh,[4] the key to achieve this is the use of the lexicographic search strategy and the choice of an appropriate set of global variables to maintain the necessary information. Note that the techniques presented in Solomon et al.[6] are futile when minimization of route duration is the objective.

1. Edge-Exchanges for the Traveling Salesman Problem

In the TSP (Lawler, Lenstra, Rinnooy Kan, Shmoys[1]), 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 objective is to find a route minimizing the sum of the travel times of the edges contained in it. 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

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

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 ways 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. For all possibilities, conditions for improvement 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.

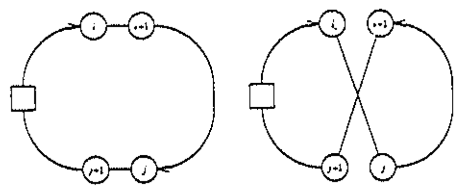

Figure 1. A 2-exchange.

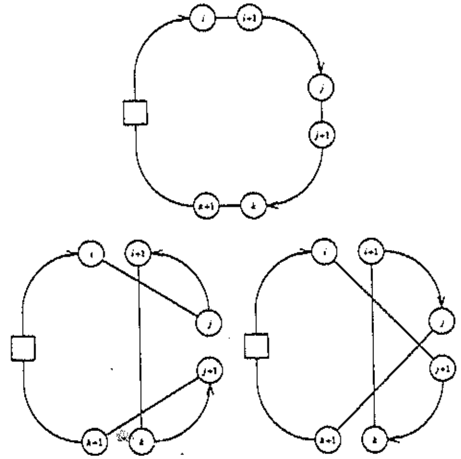

Figure 2. Two ways to perform a 3-exchange.

Because the computational requirement to verify 3-optimality may become prohibitive as the number of vertices increases, proposals have been made to take only a subset of all possible 3-exchanges into account. Or[2] 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.

2. Edge-Exchanges for Constrained TSPs

The main problem with the use of the edge-exchange procedures in the TSP with side constraints is testing the feasibility of an exchange. A 2-exchange, for instance, will reverse the path $(i+1, \ldots, j)$, 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 our 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 $(0, \ldots, i, \ldots, n)$, where i represents the ith 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 substitu-

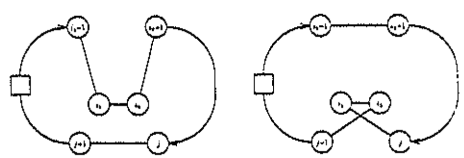

Figure 3. An Or-exchange.

Subject classifications: Subject classification: Transportation: vehicle routing. Networks/graphs: iterative improvement methods for the TSP.

tion 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 $\{i + 2, i + 3\}, \{i + 3, i + 4\}, \ldots, \{n - 1, n\}$ 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 $(i + 1, \ldots, j - 1)$ 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$(i_1, \ldots, i_2)$ in the order of the current route starting with i_1 equal to 2. After the path$(i_1, \ldots, i_2)$ has been fixed, we choose the edge $\{j, j + 1\}$ to be $\{i_1 - 2, i_1 - 1\}, \{i_1 - 3, i_1 - 2\}, \ldots, \{0, 1\}$ 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 $(j + 2, \ldots, i_1 - 1)$ of the previously considered exchange is expanded with the edge $\{j + 1, j + 2\}$.

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

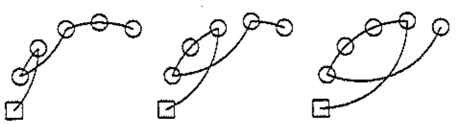

Figure 4. The lexicographic search strategy for 2-exchanges.

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 second, 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.

FRAMEWORK FOR A 2-EXCHANGE PROCEDURE

```
procedure TwoExchange
(*input: a route given as (0, 1, ..., n)*)
(*output: a route that is 2-optimal*)
begin
START:
   for i:= 0 to n - 3 do
   begin
      InitGlobal(i, G)
      for j:= i + 2 to n - 1do
      begin
         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.

3. The TSP 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. It is convenient to assume $A_o = D_o$ and $D_n = A_n$.

The standard objective is to minimize the total travel time, i.e., $\Sigma_{0 \le k < n} t_{k,k+1}$. However, this objective totally ignores possible waiting time at vertices. In this paper, we consider the problem of minimizing route duration, i.e., the difference $D_n - D_0$ between the arrival time at the depot and the departure time at the depot. In case the departure time at the depot is fixed, this corresponds to minimizing the completion time. The basic assumption underlying the current work is that the departure time of a vehicle at the depot is not fixed but has to fall within a given time window, which is the case in many real-life situations. Therefore, the departure time of a vehicle at the depot can be chosen in such a way that the route duration is minimal. Actually, for a given route this corresponds to choosing the departure in such a way that the total waiting time of a route is minimal.

Given a route, choosing the departure time that minimizes route duration is rather easy. However, manipulating a route, for instance relocating a vertex or applying iterative improvement methods to it, becomes more complicated since we have to take into account that some vertices may now be shift backward in time as well as forward in time.

4. Forward Time Slack

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

Taking the departure time at the depot as early as possible, i.e., $D_0 = e_0$, we define for each vertex i a *forward time slack* F_i indicating how far the departure time of this vertex can be shifted forward in time without causing the route to become infeasible as follows:

$$F_i := \min_{i \le k \le n}\{l_k - (D_i + \Sigma_{i \le p < k} t_{p,p+1})\}.$$

More general, we define $F_i^{(i,\ldots,j)}$ to be the forward slack time at vertex i relative to the path $(i, \ldots, j)$ and the departure times $D_i, \ldots, D_j$, which can be formally expressed as:

$$F_i^{(i,\ldots,j)} := \min_{i \le k \le j}\{l_k - (D_i + \Sigma_{i \le p < k} t_{p,p+1})\}.$$

The main tool in controlling feasibility in edge-exchange methods is this forward time slack. However, it turns out that it is also of crucial importance in identifying profitable, with respect to minimizing route duration, edge-exchanges. It is not hard to see that postponing the departure at the depot by the minimum of the forward time slack at the depot and the total waiting time on the route, i.e., $D_0 = e_0 + \min\{F_0, \Sigma_{0 < p < n} W_p\}$, results in a feasible route without unnecessary waiting time. An example is given in Table 1. Consequently, the route duration can be expressed as follows:

$$D_n - (e_0 + \min\{F_0, \Sigma_{0 < p < n} W_p\}).$$

Therefore, to be able to tell whether or not an edge-exchange results in a feasible route, we have to compute the forward time slack at the depot of the resulting route, and to be able to tell whether or not an edge-exchange results in a route with smaller duration, we have to compute the arrival time at the depot and the total waiting time on the route as well.

The following *concatenation theorem* that shows that if two given paths, with associated forward time slacks for the first vertices, are concatenated, the forward time slack for the first vertex of the resulting path can easily be computed, turns out to be very useful.

Theorem. *If two feasible paths* $(i_1, \ldots, j_1)$ *and* $(i_2, \ldots, j_2)$, *with associated forward time slacks* $F_{i_1}^{(i_1,\ldots,j_1)}$ *and* $F_{i_2}^{(i_2,\ldots,j_2)}$ *for the first vertices, are concatenated, the forward time slack for the first vertex of the resulting path is given by:*

$$F_{i_1}^{(i_1,\ldots,j_1,i_2,\ldots,j_2)} = \min\{F_{i_1}^{(i_1,\ldots,j_1)}, F_{i_2}^{(i_2,\ldots,j_2)} + \Sigma_{i_1 < k \le j_1} W_k + D_{i_2} - (D_{j_1} + t_{j_1 i_2})\}$$

Table I

A Feasible Route with and without Unnecessary Waiting Time

i	e_i	l_i	A_i	D_i	W_i	F_i	A_i	D_i	W_i
0	6.00	18.00	—	6.00	—	5.00	—	11.00	—
1	10.00	14.00	6.30	10.00	3.30	1.30	11.30	11.30	—
2	8.00	12.00	10.30	10.30	—	1.30	12.00	12.00	—
3	13.00	17.00	11.00	13.00	2.00	4.00	12.30	13.00	0.30
4	6.00	18.00	13.30	—	—	4.30	13.30	—	—

Proof.

$$F_{i_1}^{(i_1,\ldots,j_1,i_2,\ldots,j_2)}$$
$$= \min\{\min_{i_1 \leqslant k \leqslant j_1}\{l_k - (D_{i_1} + \Sigma_{i_1 \leqslant p < k} t_{p,p+1})\},$$
$$\min_{i_2 \leqslant k \leqslant j_2}\{l_k - (D_{i_1} + \Sigma_{i_1 \leqslant p < j_1} t_{p,p+1} + t_{j_1 i_2} + \Sigma_{i_2 \leqslant p < k} t_{p,p+1})\}\}$$
$$= \min\{F_{i_1}^{(i_1,\ldots,j_1)}, \min_{i_2 \leqslant k \leqslant j_2}\{l_k - (D_{i_2} + \Sigma_{i_2 \leqslant p < k} t_{p,p+1})\} + D_{i_2} - t_{j_1 i_2} - (D_{i_1} + \Sigma_{i_1 \leqslant p < j_1} t_{p,p+1})\}$$
$$= \min\{F_{i_1}^{(i_1,\ldots,j_1)}, \min_{i_2 \leqslant k \leqslant j_2}\{l_k - (D_{i_2} + \Sigma_{i_2 \leqslant p < k} t_{p,p+1})\} + D_{i_2} - t_{j_1 i_2} - (D_{j_1} - \Sigma_{i_1 < p \leqslant j_1} W_p)\}$$
$$= \min\{F_{i_1}^{(i_1,\ldots,j_1)}, F_{i_2}^{(i_2,\ldots,j_2)} + \Sigma_{i_1 < p \leqslant j_1} W_p + D_{i_2} - (D_{j_1} + t_{j_1 i_2})\} \quad \blacksquare$$

For a given route $(0, 1, \ldots, n)$, the above theorem gives us two ways to compute the forward time slack at the depot, i.e., $F_0^{(0,1,\ldots,n)}$. First, through forward recursion as follows:

$$F_0^{(0,\ldots,i,i+1)} = \min\{F_0^{(0,\ldots,i)}, l_{i+1} - D_{i+1} + \Sigma_{0 < p \leqslant i} W_p + W_{i+1}\}.$$

Second, through backward recursion as follows:

$$F_i^{(i,i+1,\ldots,n)} = \min\{l_i - D_i, F_{i+1}^{(i+1,\ldots,n)} + W_{i+1}\}$$

5. Iterative Improvement Methods

The analysis of iterative improvement methods is split in two parts. First, we will show how to test the feasibility of an exchange efficiently, i.e., constant time per exchange. Second, we will show how to test the profitability of an exchange with respect to minimizing route duration efficiently.

5.1. Feasibility

For both the 2-exchanges and the Or-exchanges, we split the resulting route in paths and use the concatenation theorem to show that the forward slack time at the depot can be computed in constant time, i.e., the feasibility can be tested in constant time.

2-exchanges. Consider the 2-exchange that involves the substitution of $\{i, i+1\}$ and $\{j, j+1\}$ with $\{i, j\}$ and $\{i+1, j+1\}$. The resulting route can be divided in the following three paths: $(0, \ldots, i)$, $(j, \ldots, i+1)$, and $(j+1, \ldots, n)$. Now observe that the forward and backward recursion schemes given above show that in $O(n)$ time it is possible to compute $F_0^{(0,\ldots,i)}$ and $F_{j+1}^{(j+1,\ldots,n)}$ for all i and $j+1$ from 0 to n. In addition, $\Sigma_{0<k\leqslant i} W_k$, can also be computed in advance in $O(n)$ time. Furthermore, $F_j^{(j,\ldots,i+1)}$ and $\Sigma_{j<k\leqslant i+1} W_k$ can easily be maintained, i.e., initialized and updated in constant time, in the inner loop. It is easy to see that the above quantities are precisely the ones needed to determine the forward slack time at the depot when the three paths are concatenated. In the terminology of the implementation technique described in Section 2, all the above quantities are maintained as global variables.

Or-exchanges. Consider the backward-Or-exhange where the path $(i_1, \ldots, i_2)$ is relocated between j and $j+1$. The resulting route can be divided in four paths: $(0, \ldots, j)$, $(i_1, \ldots, i_2)$, $(j+1, \ldots, i_1 - 1)$, and $(i_2 + 1, \ldots, n)$. We can compute $F_0^{(0,\ldots,j)}$, $\Sigma_{0<k\leqslant j} W_k$, and $F_{i_2}^{(i_2+1,\ldots,n)}$ in advance in $O(n)$ time, $F_{i_1}^{(i_1,\ldots,i_2)}$ and $\Sigma_{i_1<k\leqslant i_2} W_k$ can be maintained efficiently in the outer loop, and $F_{j+1}^{(j+1,\ldots,i_1-1)}$, and $\Sigma_{j+1<k\leqslant i_1-1} W_k$ can be maintained efficiently in the inner loop. For the forward-Or-exchange a similar argument can be presented.

5.2. Profitability

Our objective is to minimize the route duration, which can be expressed as:

$$D_n - (e_0 + \min\{F_0, \Sigma_{0<k<n} W_k\}).$$

As we have shown above, the lexicographic search strategy in combination with a suitable set of global variables allows the efficient computation, (constant time per edge-exchange) of the forward time slack at the depot. What remains is to show that we can also compute D_n and $\Sigma_{0<k<n} W_k$ efficiently.

The efficiency of the computation of the forward time slack is based on a concatenation theorem that shows that it takes only constant time to compute the forward slack time of a concatenated path using the forward slack times of the original paths. In this section, we present a similar concatenation argument, in this case to compute the waiting time on a concatenated path, to show that D_n and $\Sigma_{0<k<n} W_k$ can also be computed efficiently.

Observe that for a given path $(i, \ldots, j)$ and a given departure time D_i, the departure time D_j can be computed as

$$D_j \leftarrow D_i + \Sigma_{i \leqslant k < j} t_{k,k+1} + \Sigma_{i < k \leqslant j} W_k.$$

The above relation shows that for the computation of D_n it is also of crucial importance to be able to compute the total waiting time on the route.

Consider the concatenation of the paths $(i_1, \ldots, j_1)$ and $(i_2, \ldots, j_2)$. In the following analysis, we show that the sum of the waiting times on the concatenated path can be computed in constant time using the waiting times on the original paths. We distinguish four different cases based on whether or not the change Δ in the departure time at i_2, i.e. $D_{j_1} + t_{j_1,i_2} - D_{i_2}$, is nonnegative and on whether or not the sum of the waiting times on the path $(i_2, \ldots, j_2)$ is zero. The results of the analysis are summarized in Table II.

In case the change in the departure time at i_2 is nonnegative and the sum of the waiting times on the path $(i_2, \ldots, j_2)$ is zero, the waiting on the concatenated path is just the waiting time of the first path.

In case the change in the departure time at i_2 is nonnegative and the sum of the waiting times on the path $(i_2, \ldots, j_2)$ is positive, some or all of the increase in the departure time D_{i_2} will be "absorbed" and used to reduce the waiting time along the path. Therefore, we have

$$\Sigma_{i_1<k\leqslant j_1, i_2\leqslant k\leqslant j_2} W_k = \Sigma_{i_1<k\leqslant j_1} W_k + \max\{0, \Sigma_{i_2<k\leqslant j_2} W_k - \Delta\}.$$

In case the change in the departure time at i_2 is negative and the sum of the waiting times on the path $(i_2, \ldots, j_2)$ is positive, the decrease in departure time D_j only introduces additional waiting time either somewhere along the path or at i_2 itself. Therefore, we have

$$\Sigma_{i_1<k\leqslant j_1, i_2\leqslant k\leqslant j_2} W_k = \Sigma_{i_1<k\leqslant j_1} W_k + \Sigma_{i_2<k\leqslant j_2} W_k + \Delta.$$

In case the change in the departure time at i_2 is negative and the sum of the waiting times on the path $(i_2, \ldots, j_2)$ is zero, some or all of the decrease in the departure time D_i may introduce additional waiting time. To determine how much of the decrease introduces additional waiting time, we define $B_i^{(i,\ldots,j)}$ to be the *backward time slack* at vertex i relative to the path $(i, \ldots, j)$, which indicates how far the departure time of this vertex can be shifted backward in time without introducing waiting time. The backward time slack can easily be computed and expressed as follows:

$$B_i^{(i,\ldots,j)} = \min_{i \leqslant k \leqslant j}\{D_k - e_k.\}$$

Consequently, we have

$$\Sigma_{i_1<k\leqslant j_1, i_2\leqslant k\leqslant j_2} W_k = \Sigma_{i_1<k\leqslant j_1} W_k + \max\{0, -\Delta - B_{i_2}^{(i_2,\ldots,j_2)}\}.$$

2-Exchanges. Consider the 2-exchange that involves the substitution of $\{i, i+1\}$ and $\{j, j+1\}$ with $\{i, j\}$ and $\{i+1, j+1\}$. The resulting route can be divided in the following three paths: $(0, \ldots, i)$, $(j, \ldots, i+1)$, and $(j+1, \ldots, n)$. Observe that D_j does not change, $\Sigma_{0<k\leqslant i} W_k$, $\Sigma_{j+1<k\leqslant n} W_k$ and $B_{j+1}^{(j+1,\ldots,n)}$ can be computed in advance in $O(n)$ time, and $B_j^{(j,\ldots,i+1)}$ and $\Sigma_{j<k\leqslant i+1} W_k$ can be maintained efficiently in the inner loop. It is easy to see that the above quantities allow the determination of D_n and the total waiting on the route in constant time if the paths are concatenated.

Or-exchanges. Consider the backward-Or-exchange where the path $(i_1, \ldots, i_2)$ is relocated between j and $j+1$. The resulting route can be divided in four paths: $(0, \ldots, j)$, $(i_1, \ldots, i_2)$, $(j+1, \ldots, i_1 - 1)$, and $(i_2 + 1, \ldots, n)$. Observe that D_j does not change, $\Sigma_{0<k\leqslant j} W_k$, $\Sigma_{i_2+1<k\leqslant n} W_k$, and $B_{i_2+1}^{(i_2+1,\ldots,n)}$ can be computed in advance in $O(n)$ time, $B_{i_1}^{(i_1,\ldots,i_2)}$ and $\Sigma_{i_1<k\leqslant i_2} W_k$ can be maintained efficiently in the outer loop, and $B_j^{(j,\ldots,i_1-1)}$ and $\Sigma_{j<k\leqslant i_1-1} W_k$ can be maintained efficiently in the inner loop. It is not hard to see that the above quantities allow the determination of both D_n and the total waiting time on the route in constant time if the paths are concatenated. For the forward-Or-exchange a similar argument can be presented.

6. Edge-Exchanges for the Vehicle Routing Problem.

After analyzing iterative improvement methods for the TSP, we now turn to the VRP. We will describe three k-exchange neighborhoods for the VRP, that relocate vertices between two routes. The neighborhoods are chosen such that testing for optimality over the neighborhood requires $O(n^2)$ time. As we are dealing with two routes, we will sometimes refer to the route that currently contains the vertices we want to relocate as the *origin* route and the other as the *destination* route. In addition, for presentational convenience, we will only describe relocations of single vertices. It is straightforward to extend the

Table II
Computation of the Waiting Time on a Concatenated Path

	$\Delta \geqslant 0$	$\Delta < 0$
$W_2 = 0$	W_1	$W_1 + \max\{0, -\Delta - B_2\}$
$W_2 > 0$	$W_1 + \max\{0, W_2 - \Delta\}$	$W_1 + W_2 + \Delta$

:sented techniques to the case where paths are relocated :tead of single vertices.

As the neighborhoods can be completely described in :ms of the substitutions that are considered, we will use : following notation to describe a neighborhood:

:t of links to be removed → {set of links to replace the :om the current routes} removed links}.

:rthermore, a vertex i will always refer to a vertex from : origin route and pre_i and suc_i will denote its prede- :ssor and successor, and a vertex j will always refer to a :rtex from the destination route and pre_j amd suc_j will :note its predecessor and successor.

:elocate: $\{(pre_i, i), (i, suc_i), (j, suc_j)\} \rightarrow \{pre_i, suc_i), \cdot\, i), (i, suc_j)\}$.

:locate tries to insert a vertex from one route into :other. A relocation is pictured in Figure 5.

:xchange: $\{(pre_i, i), (i, suc_i), (pre_j, j), (j, suc_j)\} \rightarrow pre_i, j), (j, suc_i), (pre_j, i), (i, suc_j)\}$.

slight modification of the previously described :locate-neighborhood leads to what we will call the :xchange-neighborhood. Here we look simultaneously at :o vertices from different routes and try to insert them :to the other routes. An exchange is pictured in Figure 6.

:ross: $\{(i, suc_i), (j, suc_j)\} \rightarrow \{(i, suc_j), (j, suc_i)\}$.

:ross tries to remove crossing links and turns out to be :ry powerful. As a special case, if the constraints allow :, it can combine two routes into one. A cross-change is :ictured in Figure 7. Note that if a cross-change is actually :rformed, the last part of either route will become the :st part of the other.

The extension of the lexicographic search strategy to these three neighborhoods is straightforward. We choose i in reverse order of the origin route. More precisely, $i = n-1, \ldots, 1$ for the relocate and exchange neighborhoods and $i = n-1, \ldots, 0$ for the cross neighborhood. After i has been fixed, we choose j in reverse order of the destination route. More precisely, $i = n-1, \ldots, 0$ for the relocate and cross neighborhoods and $i = n - 1, \ldots, 1$ for the exchange neighborhood.

To test the feasibility and profitability of an exchange, we split the resulting routes in paths and use the concatenation ideas to show that the forward slack time at the depot and the total waiting can be computed in constant time.

Relocate. The resulting origin route can be divided in the two paths $(0, \ldots, pre_i)$ and $(suc_i, \ldots, n)$ and is trivially feasible since customers were deleted only. Note that $F_0^{(0,\ldots,pre_i)}$ and $F_{j+1}^{(suc_i,\ldots,n)}$ can be computed in advance for all i from 1 to $n-1$ in $O(n)$ time. The resulting destination route can be divided in the three paths $(0, \ldots, j)$, $(i, \ldots, i)$, and $(suc_j, \ldots, n)$. Note that $F_0^{(0,\ldots,j)}$ and $F_{suc_j}^{(suc_j,\ldots,n)}$, as well as $\Sigma_{0<k\leq j} W_k$, can be computed in advance for all j from 0 to $n-1$ in $O(n)$ time. Furthermore, $F_i^{(i,\ldots,i)}$ can easily be maintained during the lexicographic search. It is easy to see that the above quantities are precisely the ones needed to determine the feasibility and the profitability of an exchange.

Exchange. The resulting origin route can be divided in the following three paths $(0, \ldots, pre_i)$, $(j, \ldots, j)$, and $(suc_i, \ldots, n)$. Note that $F_0^{(0,\ldots,pre_i)}$ and $F_{suc_i}^{(suc_i,\ldots,n)}$, as well as $\Sigma_{0<k<i} W_k$, can be computed in advance for all i from 1 to $n-1$ in $O(n)$ time. Furthermore, $F_j^{(j,\ldots,j)}$ can easily be maintained during the lexicographic search. The resulting destination route can be divided in the following three paths: $(0, \ldots, pre_j)$, $(i, \ldots, i)$, and $(suc_j, \ldots, n)$. Note that $F_0^{(0,\ldots,pre_j)}$ and $F_{suc_j}^{(suc_j,\ldots,n)}$, as well as $\Sigma_{0<k<j} W_k$, can be computed in advance for all j from 1 to $n-1$ in $O(n)$ time. Furthermore, $F_i^{(i,\ldots,i)}$ can easily be maintained during the lexicogragphic search. It is easy to see that the above quantities are precisely the ones needed to determine the feasibility and the profitability of an exchange.

Cross. The resulting origin route can be divided in the two paths $(0, \ldots, i)$ and $(suc_j, \ldots, n)$. Note that $F_0^{(0,\ldots,i)}$, as well as $\Sigma_{0<k\leq i} W_k$, for all i from 0 to n and $F_{suc_j}^{(suc_j,\ldots,n)}$ for all j from 0 to $n-1$ can be computed in advance in $O(n)$ time. The resulting destination route can be divided in the two paths $(0, \ldots, j)$ and $(suc_i, \ldots, n)$. Note that $F_0^{(0,\ldots,j)}$, as well as $\Sigma_{0<k\leq j} W_k$, for all j from 0 to n and $F_{suc_i}^{(suc_i,\ldots,n)}$ for all i from 0 to $n-1$ can be computed in advance in $O(n)$ time. It is easy to see that the above quantities are precisely the ones needed to determine the feasibilty and the profitabilty of an exchange.

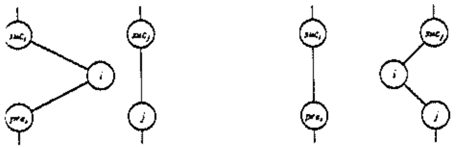

Figure 5. The relocate neighborhood.

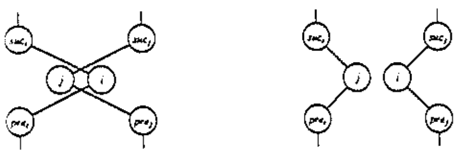

Figure 6. The exchange neighborhood.

Figure 7. The cross neighborhood.

The above described iterative improvement methods can easily be extended to larger neighborhoods by the introduction of paths instead of vertices. The paths have to be checked but that involves only local information. Figure 8 illustrates some possible extensions.

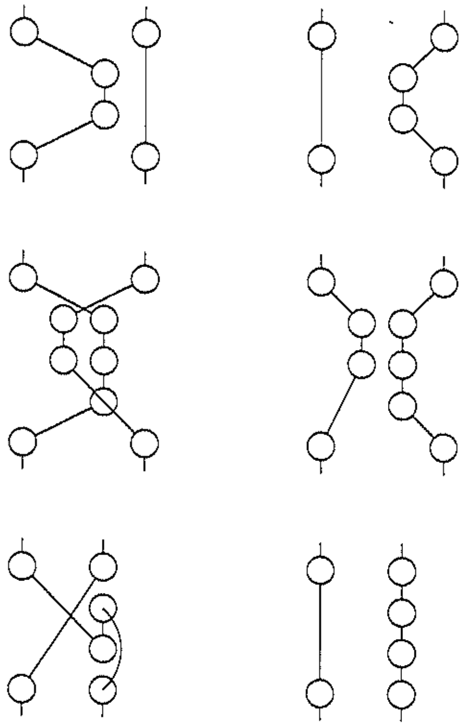

Figure 8. Extensions to the relocate-, exchange-, and cross-neighborhoods.

7. Empirical Analysis

The objective of our empirical analysis is twofold. First, to investigate the effect of different objective functions. Second, to evaluate the efficiency of the proposed methods.

Test problems are randomly generated using the following scheme. There are four control parameters: n, l, w, and p. The locations of the n customers are uniformly distributed in $[1, \ldots, 100] \times [1, \ldots, 100]$; the depot is located at (50, 50). The coordinates of the locations of the customers are used to compute the intercity Euclidean distances. The loads of the customers are uniformly distributed in $[1, \ldots, l]$. Time windows of w minutes are generated for p percent of the customers. The other customers and the depot have the time window [480, 1200], i.e., a period of 10 hours (times will always be given in minutes). There is a homogeneous set of vehicles with capacity 500. Vehicles are assumed to travel at an average speed of 40 km/h.

The iterative improvement methods were embedded in a two phase approximation algorithm for the VRPTW. In the first phase, an initial set of routes is constructed with the parallel insertion heuristic described in Savelsbergh[5]. In the second phase, an improved set of routes is obtained through the use of the exchange procedures. First, the relevant iterative improvement methods are applied to all possible combinations of two routes. Second, the relevant iterative improvement methods are applied to all separate routes. As long as feasible and profitable exchanges have been found, the process is repeated. Although this is clearly an unsophisticated brute-force approach, it is suitable for our purposes.

To investigate the effect of different objective functions, we have compared the solutions obtained with minimizing route duration as objective to those obtained with minimizing travel time and minimizing completion time as objective for a set of 10 randomly generated problem instances ($n = 100$, $l = 50$, $w = 120$, $p = 50$). The results shown in Table III clearly indicate the importance of being able to handle different objective functions.

To evaluate the efficiency of the described methods, we have compared the running times of our proposed implementation of iterative improvement techniques with a straightforward implementation of these techniques, i.e. (temporarily) perform an exchange and test its feasibility and profitability, for various types. The CPU times shown in Table IV, which include the time required to generate the initial set of routes, demonstrate the efficiency of our proposed implementation. As expected the advantage increases with the number of customers per route, but even with 10 to 15 customers per route the overhead is small enough to give a better performance than the straightforward implementation. The fact that the CPU times increase when time windows are present ($w = 120$ and $p = 50$) is a consequence of the fact that more profitable exchanges are identified.

8. Conclusion

The growing importance of side constraints as well as realistic objectives in practical distribution management

Table III
The Effect of Different Objective Functions

	Minimizing Route Duration			Minimizing Travel Time			Minimizing Completion Time		
Problem	Route duration	Travel time	Completion time	Route duration	Travel time	Completion time	Route duration	Travel time	Completion time
1	2758	2695	6236	3263	1857	6324	3480	2978	6216
2	3086	3005	6061	3641	1940	6425	3377	3110	5867
3	2891	2753	7345	4174	1966	7951	4237	3842	7213
4	2401	2383	5462	2911	1923	6027	3275	3009	5722
5	2593	2482	6780	3395	1982	7239	3489	3034	6505
6	3135	3038	6722	3807	1794	6840	3651	3305	6256
7	2560	2446	7166	3731	1833	7675	3624	3439	6769
8	2342	2335	7113	3752	1814	7856	3509	3445	6881
9	3529	2921	7441	4162	1886	7776	3938	3120	7103
10	2622	2612	7584	3287	1931	7796	3499	3403	7077

Table IV
CPU Times (in seconds) for the Two Implementations

Problem Parameters	Straightforward Implementation	Proposed Implementation
(100,50,0,0)	10.60	7.04
(100,50,120,50)	22.61	13.99
(300,25,0,0)	135.97	55.21
(300,25,120,50)	234.05	68.82

and the need for fast implementations of algorithms in the context of interactive planning systems justify the current research. The main contribution of this paper lies in the fact that we have concentrated on more realistic objective functions. The relative ease with which the general framework presented in Savelsbergh [4] could be generalized and extended to accommodate these objective functions exemplifies the robustness of this framework.

REFERENCES

1. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (eds.), 1985. *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, Wiley, Chichester.
2. I. Or 1976. *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Blood Banking*, Ph.D. thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL.
3. M.W.P. Savelsbergh, 1986. Local Search for Routing Problems with Time Windows. *Annals of Operations Research 4*, 285–305.
4. M.W.P. Savelsbergh, 1990. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operations Research 47*, 75–85.
5. M.W.P. Savelsbergh, 1991. A Parallel Insertion Heuristic for the Vehicle Routing Problem. *Statistica Neerlandica 44*, 139–147.
6. 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, in B.L. Golden and A.A. Assad (eds.), *Vehicle Routing: Methods and Studies*, Elsevier, New York, pp. 85–105.

ORSA Journal on Computing
Vol. 4, No. 2, Spring 1992

0899-1499/92/0402-0155 $01.25

An Optimal Algorithm for the Orienteering Tour Problem

R. RAMESH *School of Management, State University of New York at Buffalo, Buffalo, NY 142600, BITNET: mgtrames@ubvms.cc.buffalo.edu*

YONG-SEOK YOON *Bellcore, Bell Communications Research, 331 Newman Springs Road, Red Bank, NJ 07701, EMAIL: athens!yoon@bellcore.com*

MARK H. KARWAN *Department of Industrial Engineering, State University of New York at Buffalo, Buffalo, NY 14260, BITNET: indmark@ubvms.cc.buffalo.edu*

(Received: May 1990; final revision received: October 1991; accepted: October 1991)

Orienteering is a sport in which a competitor selects a path from a start to a destination, visiting control points along the path. Each control point has an associated score, and the travel between control points involves a certain cost. The problem is to select a set of control points to visit, so that the total score is maximized subject to a budget constraint on total cost. Several versions of this problem exist. In the version considered in this research, the start and the destination are the same, and the problem is to construct a subtour of the set of control points. The orienteering problem is a variant of the traveling salesman problem, and arises in vehicle routing and production scheduling situations. This problem has been shown to be NP-hard in the literature. We develop an optimal algorithm to solve this problem, using Lagrangean relaxation within a branch-and-bound framework. The Lagrangean relaxation is solved by a degree-constrained spanning tree procedure. Characteristics of the Lagrangean relaxation are studied, and several implementation features to improve the performance of the algorithm are presented. Detailed computational results for problems having up to 150 control points are presented. The results show that the proposed approach is viable for solving problems of medium to large size.

Score Orienteering is a sport that involves a set of control points, each of which has an associated score. Competitors are required to start from a specified control point and terminate at another specified control point, visiting as many control points as possible such that their total score is maximized within a prescribed amount of time. Competitors arriving at the terminal point after the prescribed time are disqualified, and the eligible competitor with the highest score is declared the winner. If several competitors obtain the same highest score within the prescribed time limit, then all of them are declared winners. Due to the time restriction, competitors usually may not be able to visit all the control points. Hence they must select a subset of points to visit such that the total score is maximized subject to the time restriction. The optimization problem underlying this sport is called the *Orienteering Problem* (OP).

The orienteering event, although an actual sport, has several practical applications. The orienteering problem is in fact, a variant of the well known *Traveling Salesman Problem* (TSP). Tsiligirides[13] refers to this problem as the *Generalized Traveling Salesman Problem*. Golden, Levy and Vohra[6] discuss the applications of the orienteering problem to vehicle routing, such as the routing of oil tankers to service ships at various locations. Other related applications of the orienteering problem that arise in customer/vehicle assignment and inventory/routing problems are discussed in Golden, Assad and Dahl[5] and Golden, Levy and Dahl.[6] An alternate version of Problem (OP) is called the *Prize Collecting Traveling Salesman Problem* (PCTSP), and this has been studied in Balas[1] and Fischetti and Toth.[2] Golden, Levy and Vohra[7] have shown the orienteering problem to be NP-hard. Golden, Levy and Vohra[7] and Golden, Wang and Liu[8] develop heuristics for the euclidean orienteering problem. Ramesh and Brown[12] develop a heuristic for the general cost metric.

In this research, we address the orienteering problem in which the starting and the terminating control points are the same, and the problem is to find the optimal feasible subtour starting from the specified control point. This version is termed the *Orienteering Tour Problem* (OTP). We develop an optimal solution procedure to Problem (OTP) using general cost functions.

The organization of this paper is as follows. Section 1 presents an integer programming formulation of the problem, discusses its structure, and develops a Lagrangean relaxation approach. Section 2 presents the branch-and-bound solution methodology. Section 3 presents the computational results, and Section 4 presents the conclusions.

1. Problem Structure

In this research, we consider Problem OTP with a symmetric cost structure. Let $G = \{V, A\}$ represent an

Subject classifications: Programming: integer, algorithms, branch-and-bound. Orienteering Problem and Lagrangean Relaxation.