A variable depth approach for the single-vehicle pickup and delivery problem with time windows
Bruggen, van der, L.J.J.; Lenstra, J.K.; Schuur, P.C.

Published: 01/01/1990

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):
COSOR Memorandum 90-48

A variable depth approach for the single-vehicle pickup and delivery problem with time windows

by

L.J.J. van der Bruggen, J.K. Lenstra, P.C. Schuur

December 1990
A variable depth approach for the single-vehicle pickup and delivery problem with time windows

L.J.J. van der Bruggen
J.K. Lenstra
P.C. Schuur

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

Consider a single depot and a set of customers with known demands, each of which must be picked up and delivered at specified locations and has two time windows in which the pickup and delivery must take place. We seek a route and a schedule for a single vehicle with known capacity, which minimizes the route duration, i.e., the difference between the arrival time and the departure time at the depot. In this paper we present a local search method for this problem based on a variable depth approach, similar to the Lin-Kernighan algorithm for the traveling salesman problem. The method consists of two phases. In the first phase a feasible route is constructed. In the second phase this solution is iteratively improved. In both phases we use a variable depth search built up out of seven basic types of arc-exchange procedures. When tested on real-life problems the method is shown to produce near-optimal solutions in a reasonable amount of computation time. Despite this practical evidence, there is the theoretical possibility that the method may end up with a poor or even infeasible solution. As a safeguard against such an emergency, we have developed an alternative algorithm based on simulated annealing. As a rule, it finds high quality solutions in a relatively large computation time.

Keywords: dial-a-ride, pickup and delivery, routing, scheduling, local search, variable depth, simulated annealing.

1 Introduction

In the single-vehicle pickup and delivery problem with time windows (SVPDPTW) we are given a single depot, a vehicle with known capacity, and N customers with known demands. Each customer must be picked up at his origin and delivered at his destination and has two time windows in which the pickup and delivery must take place. The problem is to determine a route and a schedule for the vehicle. A route is an ordering of the pickups and deliveries of the customers; a schedule specifies the times of pickup and delivery. Each route starts and ends at the depot. The departure and the arrival time at the depot also have to fall within given time windows. The objective is to minimize the route duration, i.e., the difference between the arrival time and the departure time at the depot.

The SVPDPTW is a constrained version of the traveling salesman problem (TSP) defined on n + 2 vertices, where n = 2N. Here the depot is represented by two vertices with zero
distance. The constraints are imposed by the time windows, the capacity restriction of the vehicle, and the precedence relations between the origin and the destination of each customer, as well as between the starting and arrival point of the route.

The SVPDPTW has been studied by a number of authors. Psaraftis (1983a) develops a dynamic programming algorithm to solve the SVPDPTW to optimality. The algorithm can handle problems with up to ten customers. Dumas and Desrosiers (1986) develop a set partitioning approach to solve the multi-vehicle case to optimality. The method is applicable to problem instances with up to 50 customers, provided that the constraints restrict the number of feasible solutions considerably.

Among the approximation algorithms, most attention has been paid to insertion algorithms. Starting from the raw data, insertion methods try to construct a feasible solution by inserting customers one by one into partial routes. Sexton and Bodin (1983) present an insertion algorithm for the SVPDPTW. Jaw et al. (1986) consider the multi-vehicle case. In another approximation algorithm, Sexton and Bodin (1985) apply Benders’ decomposition to a mixed binary nonlinear formulation of the SVPDPTW, which separates the routing and scheduling components.

Until now little research has been directed to local search techniques for the SVPDPTW. We recall that a local search algorithm tries to improve a feasible solution iteratively by local modifications. One starts with an initial feasible solution and searches its neighborhood for a better route. A neighborhood of a solution contains solutions that are in some sense ‘close’ to it. A neighborhood search strategy is needed to generate solutions from the neighborhood of a given route. As long as an improved route exists, the neighborhood search is applied to the new route. The algorithm stops at a local optimum, i.e., a solution that cannot be improved by searching its neighborhood for a better route.

Psaraftis (1983b) describes a local search algorithm for the pickup and delivery problem without time constraints. This case is substantially simpler than the SVPDPTW. Savelsbergh (1988) introduces techniques that make it possible to handle the time constraints efficiently.

For a more complete survey of the literature we refer to Desrochers et al. (1988) and Van der Bruggen (1990).

In this paper we present a local search method for the SVPDPTW based on a variable depth approach similar to the well-known technique developed by Lin and Kernighan for the traveling salesman problem. The techniques introduced by Savelsbergh play an important role in the development of this local search algorithm.

A traditional modification mechanism for the TSP is a $k$-exchange, i.e., the replacement of a set of $k$ arcs of a given route by another set of $k$ arcs. A route that cannot be improved by a $k$-exchange is said to be $k$-optimal. The total time requirement to check a given route for $k$-optimality increases rapidly with $k$. Therefore, one often restricts the search to the cases $k = 2$ and $k = 3$. Apart from the 2-exchange, the procedures in this paper also use the Or-exchange, i.e., a 3-exchange in which a string of one, two or three consecutive vertices is relocated between two other vertices [Or (1976)].

As the TSP has no side constraints, the resulting tour after a $k$-exchange is always feasible. In case of the SVPDPTW, one also has to perform feasibility checks in addition to profitability checks, since the tour that results after a $k$-exchange may be infeasible. For the 2-exchange and Or-exchange case, Savelsbergh (1988) introduces an efficient technique to check feasibility and profitability of candidate exchanges.

Lin and Kernighan developed a variable depth exchange procedure for the TSP, i.e., an arc-exchange procedure in which the number of arcs to be replaced is determined dynamically.
Given that it has been decided to exchange $s$ arcs, heuristic rules are applied to determine whether an $s + 1$-exchange should be considered.

This variable depth exchange procedure is not applicable to the SVPDPTW in a straightforward way. Problems concerning feasibility and profitability arise. In this paper we try to overcome these problems. We combine the techniques introduced by Savelsbergh with the ideas of Lin and Kernighan to develop an efficient variable depth procedure for the SVPDPTW.

To obtain a good feasible solution, starting from scratch, we introduce a two-phase method. In the first phase an initial feasible solution is constructed. During the second phase this solution is improved. In both phases we use the variable depth exchange procedure mentioned above. In the first phase we allow infeasibility and try to reduce it using an objective function that penalizes the violation of the restrictions. In the second phase we only consider feasible solutions and take the route duration as our objective function. We have tested this two-phase algorithm on real-life problems as well as on problems that we constructed ourselves. For the real-life problems, the two-phase method produced near-optimal solutions. When tested on the problems constructed by the authors, it sometimes delivered bad solutions, especially in deliberately pathological cases. As an alternative, we have developed an algorithm based on simulated annealing. An advantage of the annealing algorithm is its flexibility, a disadvantage is its large time requirement.

The paper runs as follows. In Section 2 we discuss arc-exchange procedures for the TSP based upon the 2-exchange, the Or-exchange and the variable depth exchange. Furthermore, for the SVPDPTW, we show how these procedures can be used efficiently in order to improve a given feasible route. Focusing on the 2-exchange case, we introduce a lexicographic neighborhood search and a set of global variables. We prove in detail that the global variables associated with the precedence relations can be updated in constant time. Next, we show how the variable depth exchange procedure for the TSP can be modified so as to deal with the constraints imposed by the SVPDPTW. In Section 3 we integrate seven basic types of variable depth exchange procedures into one powerful algorithm. A two-phase method is presented that first constructs a feasible solution and then iteratively improves it. Section 4 gives the computational results. In Section 5, in order to deal with the theoretical possibility of getting stuck using the above approach, we develop an algorithm based on simulated annealing and discuss its performance.

2 Arc-exchange procedures

2.1 Arc-exchange procedures for the traveling salesman problem

Consider the traveling salesman problem (TSP) on $n + 2$ vertices. A solution to the TSP is a tour, i.e., a cycle that visits each vertex exactly once. A tour will be denoted by a sequence $(0, \ldots, i, \ldots, n + 1)$ where $i$ is the $i$th vertex visited on the tour and 0 and $n + 1$ both represent the depot. Since a tour starts and ends at the depot, we split the depot in a ‘departure depot’ 0 and an ‘arrival depot’ $n + 1$. The travel times are given by the matrix $(t_{ij})$, with $t_{0,n+1} = 0$. We assume that the travel times are symmetric, i.e., $t_{ij} = t_{ji}$ for all $i, j$, and also satisfy the triangle inequality, i.e., $t_{ij} + t_{jk} \geq t_{ik}$ for each triple $(i, j, k)$. The objective is to find a tour that minimizes the total travel time $\sum_{0 \leq i \leq n} t_{i,i+1}$.

For the TSP a widely used mechanism to generate a new tour is the $k$-exchange. Figure 2.1 gives an example of a 2-exchange where the arcs $(i, i + 1)$ and $(j, j + 1)$ are replaced by the
arcs \((i, j)\) and \((i + 1, j + 1)\). Note that, in case of a 2-exchange, there is a unique way to replace two arcs such that again a tour is obtained. Throughout we choose the orientation of a tour obtained by a \(k\)-exchange in such a way that the path containing the depot is not reversed in direction. As the TSP has no side constraints, the resulting tour after a \(k\)-exchange is always feasible. To check profitability of an exchange, it suffices to compute the difference in cost between the removed and new arcs. As long as \(k\) is a constant, this takes constant time, since the computational effort does not depend on \(n\). The number of possible \(k\)-exchanges is \(\Theta(n^k)\), and hence the computational requirement to verify \(k\)-optimality may become prohibitive, even for \(k = 3\) if the number of vertices increases. Therefore, in the case \(k = 3\), Or (1976) proposes to restrict the search to a special subset of 3-exchanges. He only considers those 3-exchanges in which a string of one, two or three consecutive vertices is relocated between two other vertices. There are two ways to relocate a string of vertices: it can be relocated earlier (backward Or-exchange) and it can be relocated later (forward
Or-exchange) in the current route. Figure 2.2 gives an example of a backward and a forward Or-exchange. The arcs \((j, j+1), (i_1-1, i_1)\) and \((i_2, i_2+1)\) are replaced by \((j, i_1), (i_2, j+1)\) and \((i_1-1, i_2+1)\), so that the path \((i_1, \ldots, i_2)\) is relocated between the vertices \(j\) and \(j+1\). In case of an Or-exchange, no paths are reversed. The number of possible Or-exchanges is \(\Theta(n^2)\).

For presentational convenience, a forward Or-exchange in which a string of \(k\) (\(k = 1, 2, 3\)) vertices is relocated between two other vertices will be referred to as a \(k\)-Orf exchange; and similarly, a backward Or-exchange as a \(k\)-Orb exchange. To simplify the exposition further, we introduce the term restricted 3-exchange to denote either a 2-exchange or an Or-exchange. By the above there are seven different types of restricted 3-exchanges. In a similar sense we use the term restricted 3-optimality.

It is obvious that the quality of a solution that is \(k\)-optimal improves as \(k\) increases. We have to find a compromise between a reasonable amount of computing time, requiring a small value of \(k\), and the quality of the solution, requiring a large value of \(k\). It is therefore a serious drawback to have to specify the value of \(k\) in advance. A more flexible and very effective arc-exchange procedure for the TSP is the variable depth exchange procedure developed by Lin and Kernighan (1973). In this arc-exchange procedure the number of arcs to be replaced is determined dynamically. Given that it has been decided to exchange \(s\) arcs, heuristic rules are applied to determine whether an \(s+1\)-exchange should be considered. The arcs are chosen in such a way that at every stage of the algorithm a tour can be constructed. In the following, let \(G_s^*\) denote the improvement in cost at stage \(s\) of the procedure, when \(s\) arcs of the original tour have been replaced by \(s\) other arcs and the resulting configuration is closed to yield a new tour. Lin and Kernighan's procedure can roughly be sketched as follows:

1. Choose an initial tour.
2. Set \(G_0^* = 0\). Choose any vertex as starting vertex and consider an arc of the tour that is incident with this vertex. Set \(s \gets 1\). In Figure 2.3a we consider vertex \(i\) and arc \((i, j)\).
3. From the other end of this arc, i.e., vertex \(j\), choose an arc, e.g. \((j, q)\), that is not in the current tour such that the gain \(g_1 = c_{ij} - c_{jq} > 0\), and such that \(g_1\) is maximized. If no such arc exists, try another starting vertex. See Figure 2.3b.
4. Having chosen \((i, j)\) to leave and \((j, q)\) to enter the solution in the previous step, we have uniquely determined the arc that has to leave at stage \(s\), namely the arc with \(q\) that enables us to construct a tour. In this case, remove the arc \((q, p)\) from the current solution. Adding arc \((p, i)\) would yield a tour, so \(G_1^* = g_1 + c_{qp} - c_{pi}\). See Figure 2.3c. Now set \(s \gets s + 1\).
5. We seek to find a vertex, say vertex \(w\), such that \(g_s = c_{wp} - c_{pw}\) is maximized. Compute the gain \(G_s = \sum_{1 \leq k \leq s} g_k\). Choose the arc \((p, w)\) to enter the solution at this stage. If \(w = i\), i.e., the current solution is already a tour, then proceed with step 6. Otherwise, complete step 5 in the following way. Remove the arc \((w, v)\) from the current solution. Adding \((v, i)\) would yield a tour, so \(G_s^* = G_s + c_{wv} - c_{vi}\). See Figure 2.3d. Compute \(G^* = \max\{G_0^*, G_1^*, \ldots, G_s^*\}\). Set \(s \gets s + 1\) and repeat step 5 unless:
   (a) no further feasible swaps exist;
   (b) \(G_s \leq G^*\).

Note that in case (b) the gain associated with the current solution is not larger than the gain associated with the best tour obtained so far. Therefore, (b) can be viewed as a stopping criterion to avoid any fruitless search.
6. Perform the exchange associated with the best of \( \{G_1^*, G_2^*, \ldots, G_s^*\} \). The procedure terminates if no improvement has been found.

### 2.2 Arc-exchange procedures for the SVPDPTW

In this subsection we describe how one can improve a given feasible route for the SVPDPTW by using the arc-exchange procedures introduced for the TSP in an efficient way. As the TSP has no side constraints, we only have to perform profitability checks on candidate improvements to verify \( k \)-optimality of a given tour. In case of the SVPDPTW, we also have to perform feasibility checks. A \( k \)-exchange changes the order in which vertices on the tour are visited, and this might lead to violation of the time windows, precedence constraints or capacity constraints. Moreover, in contrast with the TSP, profitability checks are less trivial. In the SVPDPTW the objective function is the route duration, and since waiting time at customers is to be taken into account, it does not suffice to compute the difference in cost between the new and removed arcs; see Savelsbergh (1990) and Van der Bruggen (1990).

In a straightforward implementation, checking feasibility of a single exchange will take \( O(n) \) time: for each vertex on the tour, starting with the depot, one checks the constraints for feasibility. As there are \( \Theta(n^2) \) possible restricted 3-exchanges, the total time requirement to check a given route for restricted 3-optimality increases to \( O(n^3) \).
Remarkably enough, in the case of the SVPDPTW the total computation time can again be reduced to $O(n^2)$, thereby achieving the same time complexity as for the unconstrained case. The idea is to use a specific neighborhood search strategy that examines all possible 2-exchanges in a lexicographic order. Moreover, a set of global variables is introduced, such that checking feasibility and profitability of a single exchange as well as updating the global variables after each exchange only take constant time. Below we shall illustrate these concepts for the special case of a 2-exchange.

2.2.1 A lexicographic neighborhood search strategy

A neighborhood search strategy that examines all possible exchanges that can result from a given feasible route in a systematic way is the lexicographic neighborhood search strategy, which we describe here for a 2-exchange. Note that a 2-exchange is completely determined by the arcs $(i, i+1)$ and $(j, j+1)$ that are to be removed. We choose $i$ successively equal to 0, 1, ..., $n-2$; this will be referred to as the outer-loop. For a fixed value of $i$, we choose $j$ successively equal to $i + 2, \ldots, n$; this will be referred to as the inner-loop. Note that in the inner-loop the previously reversed path $(j-1, \ldots, i+1)$ is repeatedly expanded with the arc $(j-1, j)$. See Figure 2.4.

![Figure 2.4. Lexicographic neighborhood search strategy for 2-exchanges.](image)

2.2.2 Global variables associated with the precedence constraints

While performing a lexicographic neighborhood search, one needs to keep track of feasibility and profitability of each single exchange. For that purpose, Savelsbergh (1988) introduces a set of quantities containing sufficient information to check feasibility and profitability of each exchange. This so-called set of global variables can be updated in constant time by working incrementally with respect to the lexicographic search strategy. The global variables associated with the precedence and capacity constraints serve to keep track of feasibility. Those associated with the time windows are used both for feasibility and profitability checks. For an extensive treatment of these techniques, we refer to previous publications on this subject, such as Savelsbergh (1988), Desrochers et al. (1988), Savelsbergh (1990) and Van der Bruggen (1990). Since we have slightly modified them for our purposes, let us briefly discuss the global variables associated with the precedence constraints, confining ourselves to the 2-exchange case. In the remainder of this subsection, when using the word 'feasible', we mean feasible with respect to the precedence constraints, i.e., we temporarily disregard the other constraints.
To describe precedence relations between vertices, we attach a label to each vertex $v \in \{0, \ldots, n+1\}$, containing information about its uniquely determined precedence-related successor or predecessor:

$$\text{pred}(v) = \begin{cases} 
-u & \text{if vertex } v \text{ precedes vertex } u; \\
u & \text{if vertex } u \text{ precedes vertex } v.
\end{cases}$$

A 2-exchange involving the arcs $(i, i+1)$ and $(j, j+1)$ only changes the order in which the vertices on the path $(i+1, \ldots, j)$ are visited. In case the current tour satisfies the precedence constraints, a reversal of the path $(i+1, \ldots, j)$ will violate the precedence constraints if and only if both a vertex and its precedence-related predecessor or successor are on this path. Consequently, such an exchange is feasible with respect to the precedence constraints if and only if

1. there is no precedence-related pair on the path $(i+1, \ldots, j)$,

or equivalently

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

In the following let us take a closer look at our feasibility search. We assume that the search process has just ascertained the feasibility of the 2-exchange involving the arcs $(i, i+1)$ and $(j, j+1)$.

Consider first the case $j = n$, i.e., the situation that we have just satisfactorily completed the inner-loop. In this case all subsequent 2-exchanges are feasible since their reversed path is a subpath of the reversed path $(i+1, \ldots, n)$, which has been found to satisfy the precedence constraints. Thus, our feasibility search comes to an end.

As long as the inner-loop has not been completed successfully, we proceed as follows. With each vertex $v \in \{0, \ldots, n+1\}$ we associate a global variable which will be updated during the search process. Bearing in mind that the reversed path $(i+1, \ldots, j)$ has already been examined and found to satisfy the precedence constraints, we now define the global variables in the following way:

$$\text{mark}(v) = \begin{cases} 
1 & \text{if } \text{pred}(v) \in \{i+1, \ldots, j\}; \\
0 & \text{otherwise}.
\end{cases}$$

These variables mark all vertices whose precedence-related predecessor is on the path $(i+1, \ldots, j)$. To prove that the total time requirement to verify feasibility of all possible exchanges that can result from a given tour is $O(n^2)$, we have to show that these global variables can be updated in constant time and that they suffice to check feasibility of the precedence constraints of a single 2-exchange. As to checking feasibility and updating the global variables, we have to distinguish two cases depending on the value of $\text{mark}(j+1)$.

If $\text{mark}(j+1) = 0$, then in our feasibility search we proceed with the inner-loop since expanding the reversed path with the arc $(j, j+1)$ yields a feasible 2-exchange. If $j+1 = n$, we are done. Otherwise, to update, we test if vertex $j+1$ has a successor and if so, we set the variable associated with this successor equal to 1. Thus, the update that has to be performed is

- if $\text{pred}(j+1) < 0$, then $\text{mark}(-\text{pred}(j+1)) \leftarrow 1$. 

8
This update takes constant time. We proceed by examining \( \text{mark}(j + 2) \).

On the other hand, if \( \text{mark}(j + 1) = 1 \), then our feasibility search takes a different course. Since \( \text{pred}(j + 1) \) belongs to the reversed path \((i + 1, \ldots, j)\), expansion of the reversed path with the arc \((j, j + 1)\) would violate the precedence constraints. Therefore there is no need to proceed with the inner-loop since this would only produce infeasible 2-exchanges. Proceeding with the outer-loop, we get some feasibility information for free by noticing that the only feasible 2-exchanges with outer-loop counter \( p = i + 1, \ldots, \text{pred}(j + 1) - 1 \) are those with inner-loop counter \( q = p + 2, \ldots, j \). At this stage we have to distinguish two cases: either \( \text{pred}(j + 1) < j \) or \( \text{pred}(j + 1) = j \).

Let us first consider the case \( \text{pred}(j + 1) < j \). Then, we resume our feasibility search at \( p = \text{pred}(j + 1) \). Note that the 2-exchanges with inner-loop counter \( q = \text{pred}(j + 1) + 2, \ldots, j + 1 \) are feasible. Of course, we terminate our search if \( j + 1 = n \). If \( j + 1 < n \) then, starting from the reversed path \((\text{pred}(j + 1) + 1, \ldots, j + 1)\), we need to update our global variables. The only vertices that have to be reset are the successors, if any, of the vertices \( i + 1, \ldots, \text{pred}(j + 1) \), since the latter are no longer on the reversed path. So we have to perform the following updates:

- for \( p := i + 1 \) to \( \text{pred}(j + 1) \) do
  - if \( \text{pred}(p) < 0 \), then \( \text{mark}(\text{-pred}(p)) \leftarrow 0 \).

Note that the number of updates is not larger than the number of exchanges that were examined in this part of the search process. As a result of this updating we are now in the situation that \( \text{mark}(j + 1) = 0 \). We proceed by examining \( \text{mark}(j + 2) \).

Next, let us consider the case \( \text{pred}(j + 1) = j \). If \( j + 1 = n \), we are at the end of our process since the outer-loop cannot be extended any further. If \( j + 1 < n \), we resume our search at \( p = j \). Then, the 2-exchange with inner-loop counter \( q = j + 2 \) is feasible. The search is terminated if \( j + 2 = n \). If \( j + 2 < n \) then, starting from the reversed path \((j + 1, j + 2)\), we need to perform the following update on our global variables:

- for \( p := i + 1 \) to \( j \) do
  - if \( \text{pred}(p) < 0 \), then \( \text{mark}(\text{-pred}(p)) \leftarrow 0 \);
  - if \( \text{pred}(j + 2) < 0 \) then \( \text{mark}(\text{-pred}(j + 2)) \leftarrow 1 \).

As before, we observe that the number of updates is not larger than the corresponding number of exchanges. Note that we are now in the situation that both \( \text{mark}(j + 1) = 0 \) and \( \text{mark}(j + 2) = 0 \). We proceed by examining \( \text{mark}(j + 3) \).

We conclude that, using the lexicographic neighborhood search strategy and the former global variables, a single 2-exchange can be checked for feasibility with respect to the precedence constraints in constant time. Consequently, all exchanges that can result from a given tour can be verified for feasibility in \( O(n^2) \) time. In case of the Or-exchange neighborhood, precedence constraints can be checked in a similar way.

### 2.3 Variable depth exchange procedure for the SVPDPTW

We now show how the efficient search strategy developed above can be incorporated in a variable depth exchange procedure for the SVPDPTW. We start with a feasible route and seek to improve it maintaining feasibility throughout the procedure. In the next section we shall use a modification of this same variable depth procedure - allowing infeasibility - to construct an initial feasible route.
Lin and Kernighan's variable depth exchange procedure is not applicable to the SVPDPTW in a straightforward way. Two serious problems arise. The first problem concerns the feasibility aspect. In case of the TSP, we swap one arc at a time, and the arcs are chosen in such a way that we can construct a feasible route at every stage of the procedure. Although we are able to construct a closed tour, an exchange will reverse some paths and in case of the SVPDPTW this might lead to violation of time window, precedence or capacity constraints. Therefore, we have to perform additional feasibility checks before accepting an arc that is considered to enter or leave the solution. This will lead to an increase in the total computing time, which is impractical. A second problem concerns the profitability aspect. Lin and Kernighan swap arcs in each step. They compute the difference in cost between arcs and replace an arc by another arc that has lower cost. As we have chosen the route duration as our objective function, more global information is needed to compute the difference in objective value. The costs of the exchanged arcs do not suffice; see Savelsbergh (1990) and Van der Bruggen (1990).

To overcome these problems, we will perform a restricted 3-exchange, instead of swapping one arc at a time. This enables us to maintain feasibility and to compute the actual value of the objective function. To reduce computational effort, we use in our procedure the lexicographic neighborhood search strategy with the associated global variables as presented in the previous subsection.

Focusing our attention on the 2-exchange case, we now present the outline of our variable depth procedure. Let \( G_s \) denote the improvement in objective value at stage \( s \) of the procedure, when \( s \) 2-exchanges have been performed. Then the procedure reads:

![Figure 2.5. Variable depth search applied to the SVPDPTW.](image-url)
1. Construct an initial feasible route. See Figure 2.5a.

2. Set $G_0^* = 0$ and $s \leftarrow 1$. Search for a feasible and profitable 2-exchange satisfying $G_1^* > 0$, using the lexicographic neighborhood search strategy and the global variables. Figure 2.5b shows a 2-exchange involving the arcs $(i, i + 1)$ and $(j, j + 1)$.

3. Set $s \leftarrow s + 1$. The 2-exchange reverses a path, in this case $(i + 1, \ldots, j)$. In the remainder of this iteration, we fix the path $(0, \ldots, i, j, \ldots, i + 1)$. We update the global variables along the path $(j, \ldots, i + 1)$ and search forward along the path $(i + 1, j + 1, \ldots, n + 1)$, using the lexicographic search strategy and global variables, for a feasible 2-exchange satisfying $G_s^* > 0$. See Figure 2.5c. We repeat this step until no further exchanges are found.

4. Perform the exchange associated with the best of $\{G_1^*, G_2^*, \ldots, G_s^*\}$. If no improvement has been found, the procedure terminates.

In each step $s$ we search for a feasible 2-exchange that satisfies $G_s^* > 0$. As there is probably more than one candidate that satisfies this condition, we explicitly have to specify which exchange will be chosen. There are several options. We can choose the best possible exchange that satisfies $G_s^* > 0$, but an alternative is to choose the first feasible 2-exchange that is encountered and satisfies the condition. In practice, a mixed strategy, which restricts the search to a small number of candidate exchanges that satisfy the condition, performs best.

![Figure 2.6. Behavior of $G_s^*$.](image)

At the end of the procedure, we perform the exchange that realizes the maximum gain. Figure 2.6 describes the behavior of $G_s^*$. In this example, we actually perform the exchange associated with $s = 5$. Note that this exchange would have been difficult to find by performing 2-exchanges one at a time. The exchanges for $s = 3$ and $s = 4$ actually lead to a deterioration in cost, i.e., $G_2^* > G_3^* > G_4^*$, and are thus unfavorable. We accept this increase in cost in order to realize a larger gain in the following steps. We perform a favorable $k$-exchange ($k \geq 2$) without the necessity of exhausting all such $k$-exchanges.

In the description of the variable depth exchange procedure we employed the 2-exchange neighborhood. One can derive versions of this procedure based on the Or-exchange neighborhood in a straightforward way. We distinguish between a $k$-Orf and $k$-Orb exchange.
(k = 1, 2, 3) and all cases have to be handled separately. Note that we obtain seven basic types of variable depth exchange procedures for the SVPDPTW in this way.

3 Arc-exchange algorithms for the SVPDPTW

The arc-exchange procedures described in the previous section are the foundation of a local search algorithm for the SVPDPTW. By applying the former variable depth procedures, a local search algorithm iteratively tries to improve an initial feasible solution. The search is repeated as long as an improved solution exists, and only stops when all seven basic types of arc exchange procedures cannot improve the current solution anymore. Note that such a solution is restrictedly 3-optimal.

As to the implementation, there are several ways to embed the arc-exchange procedures into a variable depth exchange algorithm. In practice, the implementation given below performs well.

*** Variable depth exchange algorithm ***

{ input: a feasible solution}
{ output: a solution that is restrictedly 3-optimal}

START:
repeat
2-exchange;
1-Orf exchange;
1-Orb exchange
until (solution cannot be improved);
2-Orf exchange; if (improved solution) then goto START;
2-Orb exchange; if (improved solution) then goto START;
3-Orf exchange; if (improved solution) then goto START;
3-Orb exchange; if (improved solution) then goto START
END.

In this implementation, the arc-exchange procedures have been selected such that the most effective procedures are used most frequently. The effectiveness of our arc-exchange procedures is determined by the depth of search that can be achieved. In each procedure a sequence of exchanges is generated. From the description of the variable depth procedures it is easily seen that this sequence of exchanges can be much longer for the 1-Or case than for the 2-Or and 3-Or case, simply because in the latter cases more vertices are involved in each single exchange. Moreover, the 2-Or and 3-Or exchange cause a larger change in the current route than the other procedures and consequently, the number of feasible exchanges is much smaller. This accomplishes a further reduction of the depth of search. Therefore, we restrict ourselves to the 2-exchange and 1-Or exchange as long as possible, and only use the 2-Or and 3-Or procedures when the former procedures fail in improving the current solution.

This local search algorithm needs an initial feasible solution as input. It may seem easy to construct a feasible solution, but this is not true. The problem of determining whether there exists a feasible solution to the SVPDPTW is NP-complete [Savelsbergh (1988)]. To construct a feasible solution, any existing construction algorithm for the SVPDPTW can be used. To
make the paper self-contained, we indicate in the next subsection how our variable depth exchange procedures can be modified, so that they become suited as construction procedures. In this way we obtain a two-phase method. In the first phase an initial feasible solution is constructed; in the second phase this solution is iteratively improved.

3.1 Construction algorithm for the SVPDPTW

In this subsection we present a construction algorithm for the SVPDPTW which is a modification of the variable depth exchange algorithm presented above. The idea is to start with an infeasible tour and to reduce infeasibility in consecutive steps. In the end we hope to achieve that the total infeasibility equals zero, meaning that a feasible tour has been constructed. To work out this idea one needs a criterion to measure infeasibility, i.e., an objective function that penalizes violation of the constraints. To transform the variable depth exchange algorithm into a construction algorithm, the following modifications have to be performed.

During the improvement phase we only consider solutions that satisfy time window, precedence and capacity constraints. During the construction phase, however, we will also allow solutions that do not satisfy the time constraints, as long as the precedence and capacity constraints remain satisfied. Instead of the route duration, we now employ an objective function that measures the infeasibility of the time constraints. For a given route, the total infeasibility is computed by the total amount of time by which the time windows along the route are exceeded. Unfortunately, it is unlikely that the time violation can be computed in constant time per exchange. Each proposed exchange takes $O(n)$ time and this leads to an increase in the total computation time. It may be possible to reduce computation time by using global variables that yield an appropriate estimation of the time violation. We leave this as a topic for further research.

Although the solution produced at the end of the construction phase is restrictedly 3-optimal with respect to the extended solution space and the new objective function, there is no guarantee that this solution is feasible. In other words, we may end up with an infeasible solution in the first phase.

The variable depth construction algorithm needs as input a solution that satisfies precedence and capacity constraints. Note that such a tour can be generated in a straightforward way. For example, each tour in which all customers are delivered immediately after their pickup satisfies the former constraints. However, such a randomly chosen initial tour can be very bad with respect to the time window constraints, especially when the time windows of the customers are small (compared with the average travel time between the vertices). In practical cases it is possible, by applying a simple sorting technique on the time windows of the vertices, to generate a tour that is nearly feasible, i.e., a tour with only a small time window violation. This can be done as follows.

Each vertex $i \in \{1, \ldots, n\}$ has a time window $[e_i, l_i]$ in which service has to take place. The midpoint $m_i = (e_i + l_i)/2$ of the time window gives a good indication in which part of the day vertex $i$ has to be visited, especially when the time window is small. If we sort all $m_i$ ($i = 1, \ldots, n$) in increasing order, taking into account that the pickup of a customer must precede its delivery, the resulting tour is probably nearly feasible with respect to the time constraints. This will considerably reduce the total computational effort during the construction phase.
4 Computational results

We will now discuss the performance of the two-phase algorithm for the SVPDPTW introduced in the previous section. The performance will be measured by, first, the quality of the local optimum obtained and, second, the running time involved. The quality of a solution can be quantified by the relative difference between the costs of this solution and the global optimum. In case the global optimum is not known, it can only be compared with the best known solution. We coded our algorithm in Turbo-Pascal and implemented it on an Olivetti M240 XT personal computer.

We have tested the two-phase algorithm on real-life problems as well as on problems we constructed ourselves. For all test problems under consideration, we have chosen the vehicle load large enough, so that the capacity constraint plays no role.

First, we have tested the two-phase variable depth exchange algorithm on real-life data stemming from the city of Toronto, Canada. The number of customers varies from 5 to 38 and all customers have small time windows. For the problem instances determined by these data, the optimal solution is known, provided our objective is to minimize the total travel time. This objective neglects possible waiting time at customers and is therefore somewhat easier to handle than the route duration, though the problems differ very little in practical complexity. We can still apply our variable depth algorithm with some of the calculations left out. Practical evidence shows that both objectives require approximately the same computation time.

<table>
<thead>
<tr>
<th>problem instance</th>
<th>number of customers ((N))</th>
<th>construction (phase I)</th>
<th>improvement (phase II)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>relative error (%)</td>
<td>time (sec)</td>
</tr>
<tr>
<td>B038</td>
<td>5</td>
<td>38.9</td>
<td>1</td>
</tr>
<tr>
<td>B028</td>
<td>6</td>
<td>50.4</td>
<td>1</td>
</tr>
<tr>
<td>B090</td>
<td>9</td>
<td>25.6</td>
<td>2</td>
</tr>
<tr>
<td>B042</td>
<td>11</td>
<td>29.5</td>
<td>7</td>
</tr>
<tr>
<td>B003</td>
<td>14</td>
<td>18.6</td>
<td>11</td>
</tr>
<tr>
<td>B001</td>
<td>16</td>
<td>38.6</td>
<td>9</td>
</tr>
<tr>
<td>B118</td>
<td>20</td>
<td>40.9</td>
<td>11</td>
</tr>
<tr>
<td>B009</td>
<td>23</td>
<td>43.4</td>
<td>42</td>
</tr>
<tr>
<td>B112</td>
<td>25</td>
<td>34.7</td>
<td>2</td>
</tr>
<tr>
<td>B004</td>
<td>28</td>
<td>35.9</td>
<td>40</td>
</tr>
<tr>
<td>B064</td>
<td>32</td>
<td>29.2</td>
<td>37</td>
</tr>
<tr>
<td>B080</td>
<td>33</td>
<td>25.5</td>
<td>33</td>
</tr>
<tr>
<td>B073</td>
<td>34</td>
<td>41.4</td>
<td>81</td>
</tr>
<tr>
<td>B113</td>
<td>38</td>
<td>17.5</td>
<td>87</td>
</tr>
</tbody>
</table>

Table 4.1. Computational results for the Toronto data.

Table 4.1 shows the results. In the first phase a feasible solution is constructed. The algorithm never failed in finding an initial feasible solution. Since we only consider time
infeasibility in the first phase and do not take the total travel time into account, the quality of the initial feasible solution varies strongly (see column 2). Column 5 shows that, in the improvement phase, the algorithm produced near-optimal solutions for all test problems. We also note that, on the average, the running time involved in both phases only slightly increases as the number of customers increases. But there are some exceptions, such as problem instance B112, for which it was easy to construct a feasible solution, but it took a relatively large running time to reach a local optimum.

To investigate the influence of the tightness of the time constraints on the behavior of the algorithm, we constructed problem instances ourselves. The basis of our test problems is the EUR100 instance. The EUR100 is a symmetric, Euclidean instance of the traveling salesman problem, defined on 100 European cities. The optimal solution to this problem is known [Aarts et al. (1989)]. To obtain problem instances for the SVPDPTW, each city is given a predecessor or successor and a time window in which service has to take place. This is done in such a way that the known optimal TSP solution satisfies the corresponding constraints. The resulting problem is a SVPDPTW defined on 50 customers. As objective we choose the route duration. Note that this implies that the optimal SVPDPTW solution coincides with the known optimal TSP solution.

The time windows are added in the following way. We partition the total set of vertices into ten blocks, each consisting of ten vertices. All vertices in a specific block are given a time window - in accordance with the known optimal TSP solution -, such that they have to be handled at about the same part of the day. First, the time windows are chosen very large, so that the time constraints play no role. During four subsequent stages, we decrease the width of the time windows in order to diminish the overlap between the time windows of vertices in different blocks. In this way we obtain five different problem instances.

<table>
<thead>
<tr>
<th>problem instance</th>
<th>construction (phase I)</th>
<th>improvement (phase II)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>relative error (%)</td>
<td>time (sec)</td>
</tr>
<tr>
<td>1</td>
<td>529.0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>146.6</td>
<td>89</td>
</tr>
<tr>
<td>3</td>
<td>130.8</td>
<td>76</td>
</tr>
<tr>
<td>4</td>
<td>81.5</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>47.1</td>
<td>51</td>
</tr>
</tbody>
</table>

Table 4.2. The EUR100 results for decreasing width of the time windows.

Table 4.2 gives the results. Problem instances 1 to 5 are chosen in decreasing order of the width of the time windows. We observe that the quality of the initial solution increases and the running time involved decreases, as the time constraints become more stringent. In case the time windows get smaller, the sorting technique on the time windows becomes more effective and this will guide the structure of a feasible solution towards the structure of the known global optimum. Note that in case of problem instance 1, which has no effective time constraints, each solution that satisfies the precedence constraints yields a feasible solution. We chose a random initial solution, which turned out to be of bad quality, and this explains
the large time requirement in the improvement phase. In the second phase we observe that we obtain higher quality solutions as the time windows become smaller. In the latter cases, the number of feasible solutions is enormously reduced, and this makes our algorithm more effective and efficient.

5 Perverse SVPDPTW problem instances

Although our variable depth algorithm performs well on practical test problems, there is still the theoretical possibility of getting stuck in a poor local optimum. If this happens in the construction phase, then we fail to produce a feasible initial solution. Should it occur during the improvement phase, then we may end undesirably far from any optimal solution. Below we shall give some examples that show how bad things can be.

For the sake of simplicity we concentrate on the improvement phase. Let us recall that in this phase the algorithm starts with a feasible solution and tries to improve it through a sequence of restricted 3-exchanges. In the instances of the SVPDPTW that we are about to present, the origins and destinations are points in the Euclidean plane. Furthermore, for each pair of vertices \((i,j)\), the travel time \(t_{ij}\) is directly proportional to the distance \(d_{ij}\) with proportionality constant 1. As in the EUR100 we choose the route duration as objective.

As a first example consider the simple case that there are only two customers. Suppose that customer \(i\) is to be picked up at \(a_i\) and to be delivered at \(b_i\), \(i = 1, 2\), where the points \(a_i\) and \(b_i\) are vertices of the unit square, situated as in Figure 5.1. Let the depot 0 lie inside the square, such that \(d_{0a_1} = d_{0b_1} < d_{0a_2} = d_{0b_2}\). For the time windows we choose \([1, 2]\) for \(a_i\) and \([3, 4]\) for \(b_i\), \(i = 1, 2\). The time windows for the depot are chosen arbitrarily, as long as they do not exclude any otherwise feasible routes, e.g. \([-1, 1]\) for the departure window and \([4, 6]\) for the arrival window.

It is readily verified that the only feasible routes for this problem instance are the routes \((0,a_2,a_1,b_1,b_2,0)\) and \((0,a_1,a_2,b_2,b_1,0)\) depicted in Figure 5.1, where the second route has minimal route duration. Note that each of the two feasible routes has the property that there exists no feasible 2-exchange, nor any feasible Or-exchange. Now suppose our improvement phase starts with the nonoptimal route \((0,a_2,a_1,b_1,b_2,0)\) in Figure 5.1a. Then our improvement phase terminates unsatisfactorily, without even generating a candidate for improvement.

Pathologies of the above kind can also be constructed for problem instances that involve
more customers, as we now show in our second example. Before going into details, let us first sketch the idea of the construction, which is the following. Take the vertex configuration given by Figure 5.1, and replace the vertex 0 by two vertices, a short distance apart. Let the resulting configuration, and any one proportional with it, be called a basic vertex configuration (BVC). Next, extend the configuration by adding BVC's that are translations of the original BVC. Finally, add the depot and choose the time windows such that in a feasible route each separate BVC has to be fully traversed before going to the next one and that the structure of the previous example is maintained in each BVC.

This idea can be worked out as follows. Consider the case that there are $3M$ customers. Suppose that customer $c_i^m$ is to be picked up at $a_i^m$ and to be delivered at $b_i^m$, $i = 0, 1, 2, m = 0, 1, \ldots, M - 1$, where for each $m$ the points $a_i^m$ and $b_i^m$, $i = 1, 2$, are vertices of a square with side 16, located along with the points $a_0^m$ and $b_0^m$ as in Figure 5.2. Here the relevant distances are indicated. In particular, $d_{a_0^m a_2^m} = 10 + \delta$, where $\delta$ is a small positive number. To fix ideas we set $\delta = 3/(M + 2)$.

As suggested by Figure 5.2, the location of each BVC $B^m = \{a_0^m, a_1^m, a_2^m, b_0^m, b_1^m, b_2^m\}$ is determined in the following way. For $m = 0, 1, \ldots, M - 2$ construct $B^{m+1}$ out of $B^m$ by translating it until $d_{a_0^m a_1^m} = 32$. The location of the depot is chosen arbitrarily. Finally, setting $s_m = 6M$, we choose the time windows as follows:

- $a_0^m$:
  $$[6 + s_m, 6 + s_m + 2m\delta],$$
- $a_1^m, a_2^m$:
  $$[16 + s_m, 32 + s_m + (2m + 1)\delta],$$
- $b_1^m, b_2^m$:
  $$[48 + s_m, 64 + s_m + (2m + 1)\delta],$$
- $b_0^m$:
  $$[74 + s_m, 74 + s_m + (2m + 2)\delta].$$

As in the previous example the time windows at the depot are chosen sufficiently large.

Note that the choice of the time windows implies that, in any feasible route, $B^{m+1}$ is traversed immediately after $B^m$. For each $B^m$ there are only two ways to traverse it feasibly: either by the route $(a_0^m, a_1^m, a_2^m, b_1^m, b_2^m, b_0^m)$ or by the route $(a_0^m, a_2^m, a_1^m, b_1^m, b_2^m, b_0^m)$, which is $2\delta$ longer in time. This means that there are precisely $2^M$ feasible routes. Not a single of these feasible routes has a feasible neighbor with respect to the restricted 3-exchange neighborhood. Consequently, each feasible route is also restrictedly 3-optimal. However, there is only one feasible route that is globally optimal, i.e., has minimal route duration, namely the one depicted in Figure 5.2. From this it follows that our algorithm, after having constructed a
feasible solution in the first phase, cannot improve this solution any further in the second phase. The choice of the neighborhood structure is the bottleneck, and the quality of the local optimum obtained merely depends on the quality of the initial feasible solution.

The above examples are, of course, perverse. They nevertheless illustrate the need for an algorithm that is capable of escaping from local optima. For our variable depth algorithm the cause of getting stuck in a poor local optimum is twofold. First, it is an intrinsic property of a deterministic local search process to stop in the first local optimum encountered, with sometimes - e.g., in our variable depth approach - the additional requirement that this local optimum be sufficiently stable. Second, the local optimum obtained may turn out to be poor by the absence of a chain of feasible neighbors connecting the current solution with a better one. Two remedies against the above problems stand out. For the first problem one might allow deteriorations in any iteration. The agony associated with the second problem may be relieved by allowing infeasibility at the expense of penalty costs. Using either one of the above mechanisms on its own will not work in our particular neighborhood setting. To illustrate this, we consider again the first perverse problem. Suppose we start with the feasible route depicted in Figure 5.1a. Then, allowing deteriorations but no infeasibility does not work, since there are no feasible neighbors at all. On the other hand, allowing infeasibility but no deteriorations is fruitless, since the route in Figure 5.1a is still restrictedly 3-optimal by the fact that each neighbor has both a longer route duration as well as a positive penalization cost. A mixed strategy allowing both deteriorations and infeasibility is more likely to succeed. One possibility would be to develop a variable depth algorithm that combines a higher tolerance towards deteriorations - e.g., by relaxing the condition that every first step in an iteration should be an improvement - with the acceptance of infeasibility throughout the process. By increasingly penalizing the infeasibility during the process, one might merge the construction phase and the improvement phase. We leave this as a topic for further research. A second option is to use simulated annealing, which has as its characteristic property the acceptance of deteriorations in a flexible way. From the above reasoning it is clear that we need a penalized version of this algorithm. In the next section we will further pursue this idea.

5.1 Penalized simulated annealing for the SVPDPTW

A stochastic local search algorithm that has the power to escape from local optima and to deal with infeasibility in an elegant way is the simulated annealing algorithm introduced by Kirkpatrick et al. (1983). In fact, simulated annealing is a versatile heuristic optimization technique based on the analogy between simulating the physical annealing process of solids and solving large-scale combinatorial optimization problems. For a detailed explanation of the method we refer to Aarts et al. (1989).

In its usual form, when the search process is limited to feasible solutions, simulated annealing can be summarized as follows. The algorithm starts off from an arbitrary initial configuration. In each iteration, by slightly perturbing the current configuration, a new configuration is generated. The difference in objective value is compared with an acceptance criterion which accepts all improvements but also admits, in a limited way, deteriorations in cost.

The optimization process is governed by a so-called cooling parameter $c > 0$, which controls the acceptance criterion. This cooling parameter decreases slowly to zero during the process. Initially, for high $c$ values, the acceptance criterion is taken such that deteriorations are accepted with a high probability. As the optimization process proceeds, the acceptance
criterion is modified such that the probability for accepting deteriorations decreases. At the end of the process this probability tends to zero. In this way the optimization process may be prevented from getting stuck in a local optimum. The process comes to a halt when during a prescribed number of iterations no further improvement in the best value found so far occurs. When the cooling is done carefully enough, the process ends in a near-optimal solution.

For many combinatorial optimization problems, a local search procedure can be made more flexible by including solutions that violate some of the problem restrictions. To express the badness of a particular infeasible solution, the objective function is augmented with a term that punishes the amount of violation of the restrictions.

The simulated annealing algorithm offers a dynamic way to penalize the violation of the restrictions by letting the penalty term depend on the cooling parameter \( c \). As \( c \) decreases the penalty associated with a fixed infeasible solution gradually increases, expressing a growing dislike of infeasible solutions towards the end of the optimization process. For details on the general behavior of this penalized annealing algorithm and the proof of its asymptotic convergence we refer to Schuur (1989).

Returning to our SVPDPTW, we opt for the following penalized annealing approach, as an alternative in case our variable depth algorithm performs poorly. During the search we consider solutions that satisfy the capacity and precedence constraints, but not necessarily the time window constraints. We choose for a very simple modification mechanism, namely the 1-Or exchange. Now we employ an objective function that simultaneously measures the route duration as well as punishes the time window violation. By punishing the time window violation more severely as the search proceeds, we generally achieve that the optimization process ends at a feasible solution of high quality. Note that in this way we have integrated the construction phase into the improvement phase.

As an example let us consider our second perverse problem instance, taking the route duration as our objective. Though our two-phase algorithm dramatically failed on this example, the annealing alternative performs very well. Table 5.1 shows the results of the annealing algorithm.

\[
\begin{array}{cccccccc}
M & 1 & 2 & 3 & 4 & 5 \\
M' & time & M' & time & M' & time & M' & time & M' & time \\
1 & 1 & 2 & 1 & 1 & 1 & 3 & 1 & 5 & 1 & 1 \\
2 & 2 & 22 & 2 & 14 & 2 & 4 & 2 & 8 & 2 & 13 \\
3 & 3 & 30 & 3 & 29 & 3 & 29 & 3 & 16 & 3 & 30 \\
4 & 4 & 98 & 3 & 70 & 4 & 111 & 4 & 97 & 3 & 58 \\
5 & 5 & 125 & 5 & 195 & 5 & 130 & 4 & 128 & 5 & 124 \\
6 & 6 & 252 & 5 & 179 & 6 & 258 & 6 & 204 & 6 & 278 \\
\end{array}
\]

Table 5.1. Penalized annealing results for perverse problem instances.

\( M \) denotes the number of BVC's. For each fixed value of \( M \) we performed five runs. \( M' \) is the number of BVC's that are traveled in the appropriate way in the best solution obtained by the annealing algorithm in a particular run. Hence, in case \( M' = M \) the algorithm has found the optimal solution. For \( M = 1, 2, 3 \) the algorithm produced the optimal solution in each run; for \( M = 4 \) it produced the optimal solution in 60 percent of the runs and
for $M = 5$ in 80 percent of the runs. We also recorded the time at which this solution was encountered for the first time. The running time needed to find the best solutions strongly varies per run. This clearly illustrates the stochastic character of the annealing algorithm. We also note that the running times rapidly increase as the number of vertices increases.

Acknowledgement

We are indebted to Emile Aarts and Martin Desrochers for providing the EUR100 data and the Toronto data, respectively.

References


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 Science, Northwestern University.


List of COSOR-memoranda - 1990

<table>
<thead>
<tr>
<th>Number</th>
<th>Month</th>
<th>Author</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>M 90-01</td>
<td>January</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>Analysis of the asymmetric shortest queue problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Part I: Theoretical analysis</td>
</tr>
<tr>
<td>M 90-02</td>
<td>January</td>
<td>D.A. Overdijk</td>
<td>Meetkundige aspecten van de productie van kroonwielen</td>
</tr>
<tr>
<td>M 90-03</td>
<td>February</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>Analysis of the asymmetric shortest queue problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Part II: Numerical analysis</td>
</tr>
<tr>
<td>M 90-04</td>
<td>March</td>
<td>P. van der Laan, L.R. Verdooren</td>
<td>Statistical selection procedures for selecting the best variety</td>
</tr>
<tr>
<td>M 90-05</td>
<td>March</td>
<td>W.H.M. Zijm, E.H.L.B. Nelissen</td>
<td>Scheduling a flexible machining centre</td>
</tr>
<tr>
<td>M 90-06</td>
<td>March</td>
<td>G. Schuller, W.H.M. Zijm</td>
<td>The design of mechanizations: reliability, efficiency and flexibility</td>
</tr>
<tr>
<td>M 90-07</td>
<td>March</td>
<td>W.H.M. Zijm</td>
<td>Capacity analysis of automatic transport systems in an assembly factory</td>
</tr>
<tr>
<td>Number</td>
<td>Month</td>
<td>Author</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>M 90-08</td>
<td>March</td>
<td>G.J. v. Houtum, W.H.M. Zijm</td>
<td>Computational procedures for stochastic multi-echelon production systems (Revised version)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M 90-09</td>
<td>March</td>
<td>P.J.M. van Laarhoven, W.H.M. Zijm</td>
<td>Production preparation and numerical control in PCB assembly</td>
</tr>
<tr>
<td>M 90-10</td>
<td>March</td>
<td>F.A.W. Wester, J. Wijngaard, W.H.M. Zijm</td>
<td>A hierarchical planning system versus a schedule oriented planning system</td>
</tr>
<tr>
<td>M 90-11</td>
<td>April</td>
<td>A. Dekkers</td>
<td>Local Area Networks</td>
</tr>
<tr>
<td>M 90-12</td>
<td>April</td>
<td>P. v.d. Laan</td>
<td>On subset selection from Logistic populations</td>
</tr>
<tr>
<td>M 90-13</td>
<td>April</td>
<td>P. v.d. Laan</td>
<td>De Van Dantzig Prijs</td>
</tr>
<tr>
<td>M 90-14</td>
<td>June</td>
<td>P. v.d. Laan</td>
<td>Beslissen met statistische selectiemethoden</td>
</tr>
<tr>
<td>M 90-15</td>
<td>June</td>
<td>F.W. Steutel</td>
<td>Some recent characterizations of the exponential and geometric distributions</td>
</tr>
<tr>
<td>M 90-16</td>
<td>June</td>
<td>J. van Geldrop, C. Withagen</td>
<td>Existence of general equilibria in infinite horizon economies with exhaustible resources. (the continuous time case)</td>
</tr>
<tr>
<td>A 90-17</td>
<td>June</td>
<td>P.C. Schuur</td>
<td>Simulated annealing as a tool to obtain new results in plane geometry</td>
</tr>
<tr>
<td>A 90-18</td>
<td>July</td>
<td>F.W. Steutel</td>
<td>Applications of probability in analysis</td>
</tr>
<tr>
<td>A 90-19</td>
<td>July</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>Analysis of the symmetric shortest queue problem</td>
</tr>
<tr>
<td>A 90-20</td>
<td>July</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>Analysis of the asymmetric shortest queue problem with threshold jockeying</td>
</tr>
<tr>
<td>Number</td>
<td>Month</td>
<td>Author</td>
<td>Title</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>M 90-21</td>
<td>July</td>
<td>K. van Ham, F.W. Steutel</td>
<td>On a characterization of the exponential distribution</td>
</tr>
<tr>
<td>M 90-22</td>
<td>July</td>
<td>A. Dekkers, J. van der Wal</td>
<td>Performance analysis of a volume shadowing model</td>
</tr>
<tr>
<td>M 90-23</td>
<td>July</td>
<td>A. Dekkers, J. van der Wal</td>
<td>Mean value analysis of priority stations without preemption</td>
</tr>
<tr>
<td>M 90-24</td>
<td>July</td>
<td>D.A. Overdijk</td>
<td>Benadering van de kroonwielflank met behulp van regeloppervlakken in kroonwieloverbrengingen met grote overbrengverhouding</td>
</tr>
<tr>
<td>M 90-25</td>
<td>July</td>
<td>J. van Oorschot, A. Dekkers</td>
<td>Cake, a concurrent Make CASE tool</td>
</tr>
<tr>
<td>M 90-26</td>
<td>July</td>
<td>J. van Oorschot, A. Dekkers</td>
<td>Measuring and Simulating an 802.3 CSMA/CD LAN</td>
</tr>
<tr>
<td>M 90-27</td>
<td>August</td>
<td>D.A. Overdijk</td>
<td>Skew-symmetric matrices and the Euler equations of rotational motion for rigid systems</td>
</tr>
<tr>
<td>M 90-28</td>
<td>August</td>
<td>A.W.J. Kolen, J.K. Lenstra</td>
<td>Combinatorics in Operations Research</td>
</tr>
<tr>
<td>M 90-29</td>
<td>August</td>
<td>R. Doornbos</td>
<td>Verdeling en onafhankelijkheid van kwadratevenommen in de variantie-analyse</td>
</tr>
<tr>
<td>M 90-30</td>
<td>August</td>
<td>M.W.I. van Kraaij, W.Z. Venema, J. Wessels</td>
<td>Support for problem solving in manpower planning problems</td>
</tr>
<tr>
<td>M 90-31</td>
<td>August</td>
<td>I. Adan, A. Dekkers</td>
<td>Mean value approximation for closed queueing networks with multi server stations</td>
</tr>
<tr>
<td>M 90-32</td>
<td>August</td>
<td>F.P.A. Coolen, P.R. Mertens, M.J. Newby</td>
<td>A Bayes-Competing Risk Model for the Use of Expert Judgment in Reliability Estimation</td>
</tr>
<tr>
<td>Number</td>
<td>Month</td>
<td>Author</td>
<td>Title</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>M 90-33</td>
<td>September</td>
<td>B. Veitman, B.J. Lageweg, J.K. Lenstra</td>
<td>Multiprocessor Scheduling with Communication Delays</td>
</tr>
<tr>
<td>M 90-34</td>
<td>September</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>Flexible assembly and shortest queue problems</td>
</tr>
<tr>
<td>M 90-35</td>
<td>September</td>
<td>F.P.A. Cooien, M.J. Newby</td>
<td>A note on the use of the product of spacings in Bayesian inference</td>
</tr>
<tr>
<td>M 90-36</td>
<td>September</td>
<td>A.A. Stoovogel</td>
<td>Robust stabilization of systems with multiplicative perturbations</td>
</tr>
<tr>
<td>M 90-37</td>
<td>October</td>
<td>A.A. Stoovogel</td>
<td>The singular minimum entropy $H_\infty$ control problem</td>
</tr>
<tr>
<td>M 90-38</td>
<td>October</td>
<td>Jan H. van Geldrop, Cees A.A.M. Withagen</td>
<td>General equilibrium and international trade with natural exhaustible resources</td>
</tr>
<tr>
<td>M 90-39</td>
<td>October</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>Analysis of the shortest queue problem (Revised version)</td>
</tr>
<tr>
<td>M 90-40</td>
<td>October</td>
<td>M.W.P. Savelsbergh, M. Goetschalckx</td>
<td>An Algorithm for the Vehicle Routing Problem with Stochastic Demands</td>
</tr>
<tr>
<td>M 90-41</td>
<td>November</td>
<td>Gerard Kindervater, Jan Karel Lensra, Martin Savelsbergh</td>
<td>Sequential and parallel local search for the time-constrained traveling salesman problem</td>
</tr>
<tr>
<td>M 90-42</td>
<td>November</td>
<td>F.W. Steutel</td>
<td>The set of geometrically infinitely divisible distributions</td>
</tr>
<tr>
<td>M 90-43</td>
<td>November</td>
<td>A.A. Stoovogel</td>
<td>The singular linear quadratic Gaussian control problem</td>
</tr>
<tr>
<td>M 90-44</td>
<td>November</td>
<td>H.L. Trentelman, J.C. Willems</td>
<td>The dissipation inequality and the algebraic Riccati equation</td>
</tr>
<tr>
<td>M 90-45</td>
<td>November</td>
<td>A.C.M. Ran, H.L. Trentelman</td>
<td>Linear quadratic problems with indefinite cost for discrete time systems</td>
</tr>
<tr>
<td>M 90-46</td>
<td>November</td>
<td>A.A. Stoovogel, J.W. van der Woude</td>
<td>The disturbance decoupling problem with measurement feedback and stability for systems with direct feedback and stability for systems with direct feed-through matrices</td>
</tr>
<tr>
<td>Number</td>
<td>Month</td>
<td>Author</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>M 90-47</td>
<td>December</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>An error on &quot;A generating-function analysis of multiprogramming queues&quot;.</td>
</tr>
</tbody>
</table>