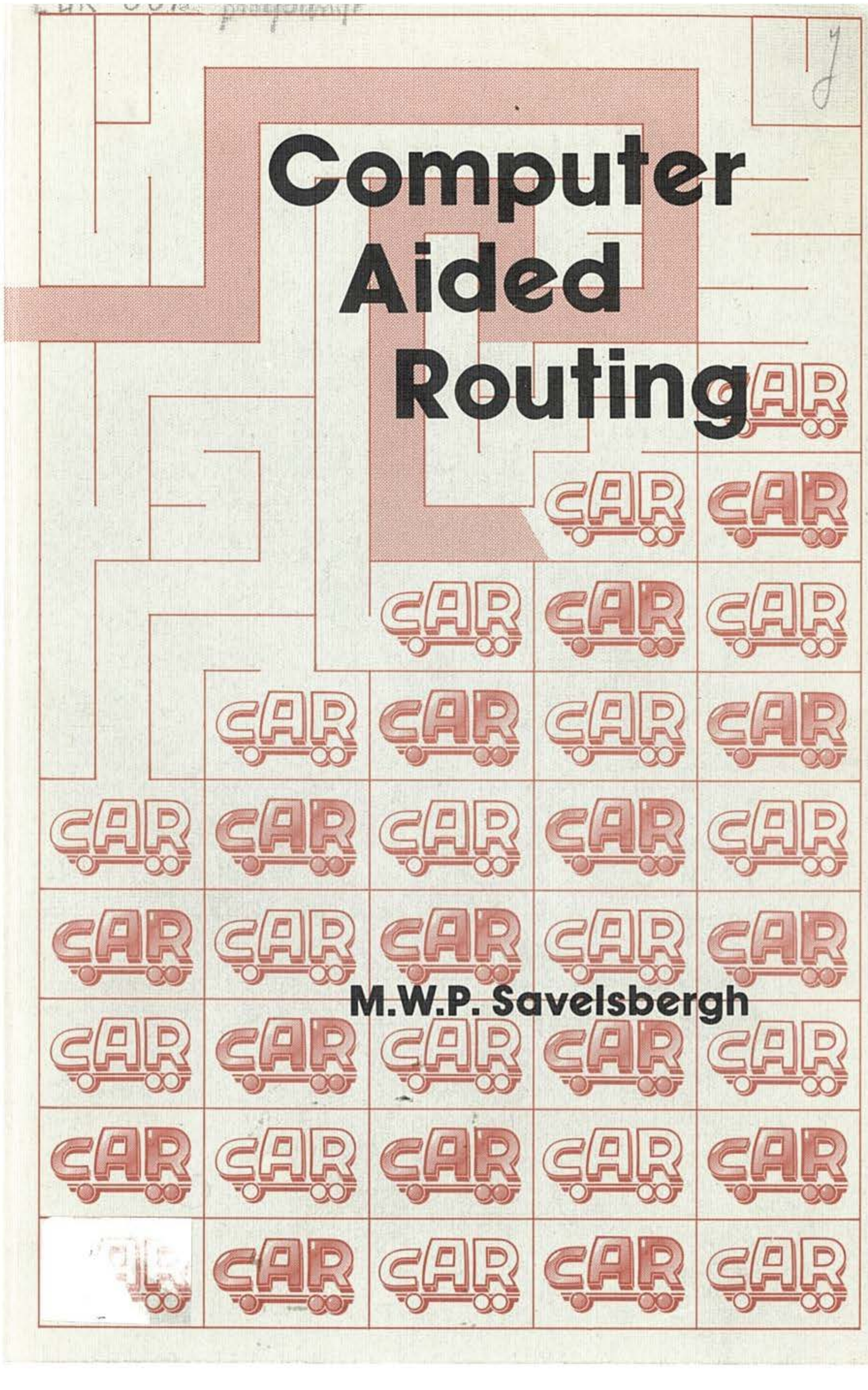
Computer Aided Routing



M.W.P. Savelsbergh

COMPUTER AIDED ROUTING

PROEFSCHRIFT
ter verkrijging van de graad van doctor
aan de Erasmus Universiteit Rotterdam
op gezag van de Rector Magnificus
Prof.dr. A.H.G. Rinnooy Kan
en volgens besluit van het College van Dekanen.
De openbare verdediging zal plaatsvinden op
donderdag 31 maart 1988 om 16.00 uur
door

MATHIEU WILLEM PAUL SAVELSBERGH

geboren te Amsterdam

1988
Centrum voor Wiskunde en Informatica, Amsterdam

Promotiecommissie

Promotor:	Prof.dr. J.K. Lenstra
Overige leden:	Prof.dr. K.M. van Hee
	Prof.dr.ir. J.A.E.E. van Nunen
	Prof.dr. A.H.G. Rinnooy Kan

Preface

This thesis would never have become what it is right now, if it were not for the interest and support of several people. It is only fair to start by expressing my gratitude.

First of all, I would like to thank those that aroused my interest and curiosity in the fields that, in one way or another, form the basis of this thesis. I am grateful to Peter van Emde Boas, Gerard Laman, Gijs de Leve and Lex Schrijver for introducing me to complexity theory, graph theory, mathematical programming and combinatorial optimization respectively.

The major part of this thesis is devoted to CAR, an interactive system for computer aided routing. CAR is a result of the research project 'Design and implementation of vehicle routing algorithms' that was financed by the Stichting voor de Technische Wetenschappen and the Stichting Mathematisch Centrum. I thank both organizations for their support, and A.D. Brink, K.M. van Hee and A.H.G. Rinnooy Kan for being members of the users' committee of the project. In the course of developing CAR, I have benefited from the help of many people. It is a pleasure to thank Willie van Sonderen-Huisman, Bob Vis and Cas van der Baan from Logion bv for stimulating discussions. I am greatly indebted to Pierre Timmermans, Jan Wullink, Arie Kok, Frans Geldman and Jan de Bruyn from Fashion Transport Holland. Pierre developed the necessary data entry and report generation facilities, Jan, Arie and Frans indefatigably acted as guinea-pigs for the prototype versions of the system, and Jan de Bruyn provided the managerial support. On the more technical side, I benefited enormously from the knowledge of Bèhr de Ruiter, Fons Kuijk and Kees Blom from the CWI Department of Interactive Systems. They taught me the basics of computer graphics and the Graphical Kernel System, and helped me out whenever I had problems in that area. It is also a pleasure to acknowledge the help of a number of people at the CWI Department of Operations Research and System Theory. Ko Anthonisse, Gerard Kindervater and Ben Lageweg were always there to discuss any problem I might have.

The last part of the thesis results from a series of discussions I had with Martin Desrochers and Leen Stougie. I hereby express my appreciation to both of them. Inspiring discussions with Chris Jones on user interfaces and modeling systems also proved to be extremely helpful. I am grateful to Art Geoffrion for providing the structured modeling schema for the vehicle routing problem that can be found in Appendix B. I also thank Tobias Baanders for designing and drawing the cover.

There is one other person that largely influenced both the research and this thesis. However, he is not just one among the others, as his support throughout the past four years has been invaluable. It has been a great pleasure and rewarding experience to work with Jan Karel Lenstra as a thesis advisor.

Finally, I thank Simpha for putting up with me all the days and nights when all I could think of was this thesis.

M.W.P. Savelsbergh

Contents

COMPUTER AIDED ROUTING

0. Introduction

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

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

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

Although already useful and profitable, many of the currently available software packages for physical distribution management have two important shortcomings. First, the implemented solution methods are often incapable of handling the various side constraints encountered in real-life problems. Secondly, they are often inadequate in their interface with the user. We elaborate on both these points in more detail.

In practical distribution problems, difficulties arise due to the size of the planning situation as well as the number and complexity of the side constraints. We will mention some of the features encountered in realistic environments.

Problems may involve both deliveries and collections. In addition, it may be possible to mix deliveries and collections on a single route, or alternatively, it may be required to first perform all deliveries before performing all collections. This latter case is often referred to as backhauling.

On many occasions, commodities have several dimensions, such as weight, volume and time. For example, in air freight both weight and volume may play an important role in determining what gets loaded on a given trip.

When the requirement of a single customer is large relative to vehicle capacity, it may be economical to split a customer among several vehicles. When splitting is possible, it may be important to take lumpiness of the cargo into account. That is, the cargo is measured in certain units such that only an integral amount may be assigned to vehicles involved in the split.

Service is often restricted to fall within one of a given number of working time windows. In the dial-a-ride problem for instance, each customer has a desired time window in which he would like to be served (either picked up at his origin or delivered at his destination). Routes and schedules have to be devised such that the required service is performed during these time windows.

Service requirements can be periodic, in the sense that a 'customer' has to be served a specific number of times within a given period such as a week. Typical problems of this type are coin collection from parking meters and garbage collection.

The standard objective function is to minimize the total distance traveled over all routes selected. In reality, there may be other objectives. Frequently, driver overtime is allowed at a certain cost and one may be able to reduce the number of vehicles required by making trips longer and incurring overtime. More complex objectives have been utilized in various problem settings to capture the flavor of constraints that are difficult to quantify.

The second point, inadequacy of the system, is a constant source of difficulties. Nothing can discourage the user more than the inability to obtain and manipulate information that is supposedly 'in the computer'. This lack of emphasis on the data handling capabilities of routing and scheduling systems has led to the demise of many such systems. This may be traced in part to the tendency in certain developers to concentrate on the algorithmic aspects of the system. The user's feeling of loss of control over the underlying physical system is one of the main impediments to user acceptance of computer systems. Many computer-generated solutions are rejected based on relatively minor issues that could be corrected if certain controls over the computer system were given to the user.

Only during the last decade, researchers started to emphasize the development of methods to solve *real-life* distribution problems. Before that, most effort has been put in methods that solve the basic theoretical models, especially for pure routing problems. To obtain methods that are able to solve practical problems we take the following two steps.

In Part I of this thesis, we strengthen the links between theory and practice by the introduction of various side constraints into the theoretical models and the development of algorithms to solve these extended models. In doing so we will concentrate on three important classes of side constraints: time windows at customers, combination of pickups and deliveries, and precedence relations between customers.

An important consideration in the formulation and solution of vehicle routing and scheduling problems is the required computational effort associated with various solution techniques. The computational effort required to solve a given problem clearly increases with the problem size. The nature of this growth in computation time as a function of problem size is an issue of both theoretical and practical interest. Computational complexity theory distinguishes two classes of problems: the well-solvable problems, for which there exist optimization algorithms whose running times are bounded by a polynomial function of the problem size, and the *NP*-hard problems, for which strong evidence exists that any optimization algorithm has, in the worst case, a running time that is a superpolynomial function of the problem size. Virtually all vehicle routing and scheduling problems belong to the class of *NP*-hard problems [Lenstra and Rinnooy Kan 1981]. This indicates that it is difficult to solve even small instances of a problem to optimality with a reasonable computational effort. As a consequence, when we have to solve large-scale real-life problems, we should not insist on finding an optimal solution, but instead try to find an acceptable solution within an acceptable amount of computation time. To accomplish this we have to resort to approximation algorithms.

In Part II of this thesis, we discuss the application of a new tool provided by computer engineering called *interaction*. We investigate solution approaches that are not purely algorithmic but that integrate algorithms and human problem solving capabilities via man-machine interaction.

Interaction is desirable because planning problems tend to be both hard and soft. To conquer their complexity, it is often prudent to use a variety of models under user control. Each of these is a picture of the actual situation, but different aspects are emphasized or ignored. To deal with the impreciseness of the notions of feasibility and optimality in real-life problems, it is often beneficial to allow the user to adjust problem parameters.

Interaction has a threefold advantage in that it adds to effectivity, efficiency and acceptability. First, the cooperation between man and machine leads to better solutions. The machine can not be beaten in solving well defined detailed problems. The human planner is superior in judging fuzzy situations, in recognizing global patterns, and in observing ad hoc constraints which do not form part of the underlying models. Secondly, these better solutions are obtained faster, because interaction allows for flexibility in manipulating data and in selecting solutions. Finally, an interactive system is more readily accepted. The human planner is not replaced by a black box but gets a versatile tool.

A very important part of an interactive system is the user interface, the part of the program that determines how the user and the computer communicate. The user interface should be easy to use and consistent in order to help the user assimilate. We believe that a *graphical* user interface is a necessity in case of a system for physical distribution management.

Computer graphics is a topic of rapidly growing importance. It has always been one of the most visually spectacular branches of computer technology, producing images whose appearance and motion make them quite unlike any other form of computer output. Computer graphics can also be an extremely effective medium for communication between man and computer. The principal usefulness of computer graphics is the ability to provide different, and perhaps more insightful, representations of the same data. Now that the cost of computer graphics technology is dropping, interactive computer graphics is becoming available to more and more people.

As an example of an interactive planning system for physical distribution management we will describe CAR (Computer Aided Routing). CAR incorporates approximation algorithms able to handle various side constraints and is equipped with a friendly user interface based on color graphics. CAR acts as an assistant and advisor to the planner. CAR has a supporting function; it is the user who is in charge and who is responsible for making all the decisions.

In Part III of this thesis we will report on our efforts to design a model and algorithm management system for vehicle routing and scheduling problems. The great variety of physical distribution problems and the large number of existing algorithms make it difficult for an unexperienced distribution manager, and even for an experienced one, to select a method that is well suited for his specific situation. The model and algorithm management system will provide support in modeling real-life problem situations and in suggesting algorithms that might be applicable to the resulting models. The research discussed in this part is still in its initial stage and by no means completed. Therefore, Part III is mainly a presentation of ideas and thoughts.

The thesis draws heavily on material found in the following papers: Desrochers, Lenstra, Savelsbergh and Soumis [1987], Savelsbergh [1986], Savelsbergh [1987], Anthonisse, Lenstra and Savelsbergh [1987a], Anthonisse, Lenstra and Savelsbergh [1987b], and Desrochers, Lenstra and Savelsbergh [1987].

PART I. ALGORITHMS

1. Introduction

Over the past ten years, operations researchers 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.

Most of the existing algorithms have been designed to solve pure routing problems and hence only deal with spatial aspects. They are not capable to handle all kinds of features that frequently occur in practice. One such feature 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 and ask for algorithms that also take temporal aspects into account. Another is the combination of deliveries and collections. In that case, it is not only the total load assigned to a vehicle that determines feasibility with respect to the capacity constraints, but also the order in which the customers are visited.

In Part I, we consider three types of problems. One is the *vehicle routing problem with time windows* (VRPTW), which is defined as follows. A number of vehicles is located at a single depot and must serve a number of geographically dispersed customers. Each vehicle has a given capacity. Each customer has a given demand and must be served within a specified time window. The objective is to minimize the total cost of travel.

The special case in which the vehicle capacities are infinite is called the *multiple traveling salesman problem with time windows* (m-TSPTW). It arises in school bus routing problems. The problem here is to determine routes that start at a single depot and cover a set of trips, each of which starts within a time window. Trips are considered as customers. There are no capacity constraints, since each trip satisfies those by definition and vehicles moving between trips are empty.

The second problem type is an extension of the VRPTW, in which demands are no longer restricted to be only deliveries or only collections. It involves the delivery of commodities from the depot to customers as well as the collection of commodities from customers to the depot. The algorithms relevant to the solution of this problem type will be treated in some detail, as they make up part of the interactive distribution planning system CAR, that was developed to solve this particular problem type and will be discussed in Part II of this thesis.

The third problem type is the *pickup and delivery problem with time windows* (PDPTW). Again, there is a single depot, a number of vehicles with given capacities, and a number of customers with given demands. Each customer must now be picked up at his origin during a specified time window, and delivered to destination during another specified time window. The objective is to minimize total travel cost.

The special case in which all customer demands are equal is called the *dial-a-ride problem* (DARP). It arises in transportation systems for the handicapped and the elderly. In these situations, the temporal constraints imposed by the customers strongly restrict the total vehicle load at any point in time, and the capacity constraints are of secondary importance. The cost of a route is a combination of travel time and customer dissatisfaction.

We will denote the time window of an address i (whether it be a customer in the VRPTW or an origin or destination in the PDPTW) by $[e_i, l_i]$, the time of arrival at i by A_i, and the time of departure at i by D_i. It is assumed that the service time at i is included in the travel time t_{ij} from address i to address j. Since service must take place within the time windows, we require that $e_i \leqslant D_i \leqslant l_i$ for all i. If $A_i < e_i$, then a waiting time $W_i = e_i - A_i$ occurs before the opening of the window at i.

There are several ways to define the tightness of the time windows. One could say that the windows are tight when the underlying network with addresses as vertices contains no time-feasible cycles. This guarantees that all feasible routes are elementary paths. However, this condition is difficult to verify, and we do not get much information if it does not hold. The following two definitions may be more useful:

$$T_1 := \frac{\overline{(l_i - e_i)}}{\overline{t_{ij}}}, \quad T_2 := \frac{\overline{(l_i - e_i)}}{\max_i\{l_i\} - \min_i\{e_i\}}.$$

T_1 is the ratio between the average window width and the average travel time. If T_1 is at its minimum value 0, we have a pure scheduling problem. If T_1 is in between 0 and 2, we can expect that there are not many time-feasible cycles, and the temporal aspects are likely to dominate the spatial aspects. If T_1 is large, we have almost a pure routing problem. These are, of course, only rough indications.

T_2 is the ratio between the average window width and the time horizon. The value of T_2 is between 0 and 1, with 0 indicating a pure scheduling problem and 1 a problem with identical time windows.

In the following, VRP denotes the VRPTW without time windows. TSPTW is the m-TSPTW with a single salesman, and TSP is the TSPTW without time windows. Since the TSP is already NP-hard, one has to obtain solutions to the VRPTW and PDPTW by fast approximation or enumerative optimization. In

Chapter 2, we present mathematical programming formulations for these problems and some of their extensions. In Chapter 3, we survey optimization algorithms based on dynamic programming and set partitioning. In Chapters 4, 5 and 6, we review various types of approximation algorithms. Chapter 4 presents results on the complexity of finding feasible solutions and surveys approximation algorithms based on construction. Chapter 5 is devoted to approximation by incomplete optimization. We adapt Fisher and Jaikumar's [1981] approach to the VRP so as to incorporate time windows and mixed collections and deliveries. Among the detailed implementations given is one of the route construction methods reviewed in Chapter 4. Chapter 6 deals with approximation by iterative improvement. The adaptation of this principle so as to handle various side constraints without increasing its time complexity poses some nontrivial algorithmic problems, which are solved in this chapter.

There are more time-constrained routing problems and more solution approaches than we can cover. The interested reader is referred to a recent collection of papers on this topic [Golden and Assad 1986].

2. Formulation

In this chapter, the VRPTW and the PDPTW are defined and formulated as mathematical programs. We concentrate on the basic problems, with a single depot and a single vehicle type. We indicate generalizations involving multiple depots, multiple vehicle types, and constraints on the travel time of the vehicles.

2.1. *The vehicle routing problem with time windows*

Given is a graph $G = (V,A)$ with a set V of vertices and a set A of arcs. We have $V = \{0\} \cup N$, where 0 indicates the depot and $N = \{1,...,n\}$ is the set of customers, and $A = (\{0\} \times N) \cup I \cup (N \times \{0\})$, where $I \subset N \times N$ is the set of arcs connecting the customers, $\{0\} \times N$ contains the arcs from the depot to the customers, and $N \times \{0\}$ contains the arcs from the customers back to the depot. For each customer $i \in N$, there is a demand q_i and a time window $[e_i, l_i]$. For each arc $(i,j) \in A$, there is a cost c_{ij} and a travel time t_{ij}. Finally, there is a set M of vehicles, each with capacity Q. We note that an arc $(i,j) \in I$ may be eliminated by temporal constraints ($e_i + t_{ij} > l_j$), by capacity constraints ($q_i + q_j > Q$), or by other considerations. The objective is to minimize the total travel costs.

The mathematical programming formulation has three types of variables: x_{ij} ($(i,j) \in A$), equal to 1 if arc (i,j) is used by a vehicle and 0 otherwise; D_i ($i \in N$), specifying the departure time at customer i; and y_i ($i \in N$), specifying the load of the vehicle arriving at i. The problem is now to minimize

$$\sum\nolimits_{(i,j) \in A} c_{ij} x_{ij} \tag{1}$$

subject to

$$\sum\nolimits_{j \in N} x_{ij} = 1 \qquad \text{for } i \in N, \tag{2}$$

$$\sum\nolimits_{j \in N} x_{ij} - \sum\nolimits_{j \in N} x_{ji} = 0 \qquad \text{for } i \in N, \tag{3}$$

$$x_{ij} = 1 \Rightarrow D_i + t_{ij} \leqslant D_j \qquad \text{for } (i,j) \in I, \tag{4}$$

$$e_i \leqslant D_i \leqslant l_i \qquad \text{for } i \in N, \qquad (5)$$
$$x_{ij} = 1 \Rightarrow y_i + q_i \leqslant y_j \qquad \text{for } (i,j) \in I, \qquad (6)$$
$$0 \leqslant y_i \leqslant Q \qquad \text{for } i \in N, \qquad (7)$$
$$x_{ij} \in \{0,1\} \qquad \text{for } (i,j) \in A. \qquad (8)$$

The objective function (1) represents the total travel costs; it is possible to include the fixed charge of using a vehicle by adding it to all c_{0j}. Minimizing (1) subject to (2), (3) and (8) is a minimum cost flow problem, which has an integral solution. Constraints (4) and (5) ensure feasibility of the schedule, and constraints (6) and (7) guarantee feasibility of the loads. We note that the number of vehicles is unbounded in the present formulation.

This VRPTW formulation is more compact than the VRP formulation due to Bodin and Golden [1981]. The latter formulation has $O(n^3)$ variables and an exponential number of subtour elimination constraints. The above formulation has $O(n^2)$ variables, while the subtours are eliminated by (4), as well as by (6). These constraints can be rewritten as follows, where C is a large constant:

$$D_i + t_{ij} - D_j \leqslant (1-x_{ij})C \qquad \text{for } (i,j) \in I, \qquad (4a)$$
$$y_i + q_i - y_j \leqslant (1-x_{ij})C \qquad \text{for } (i,j) \in I. \qquad (6a)$$

In their TSP formulation, Miller, Tucker and Zemlin [1960] propose the following subtour elimination constraints:

$$D_i - D_j + nx_{ij} \leqslant n - 1 \qquad \text{for } (i,j) \in I.$$

These appear as a special case of (4a) when all $t_{ij} = 1$ and $C = n$, and as a special case of (6a) when all $q_i = 1$ and $C = n$.

The above single-depot formulation is based on a single-commodity flow. There is no explicit flow conservation constraint for the depot, as this is implied by the flow conservation constraints (3) for the customers. Let us now consider the multi-depot case. The single depot 0 is replaced by a set L of depots. In the graph $G = (V,A)$, we now have $V = L \cup N$ and $A = (L \times N) \cup I \cup (N \times L)$, where N and I are as before. There are two variants. In case each vehicle must return to its home depot, we need a multi-commodity flow formulation with a separate commodity for each depot. Each variable x_{ij} is replaced by variables x_{ij}^l ($l \in L$), where $x_{ij}^l = 1$ if arc (i,j) is used by a vehicle from depot l, and 0 otherwise. In case vehicles do not have to return to their points of origin, all we have to do is to add a flow conservation constraint for each depot.

The case of multiple vehicle types is modeled with fictitious depots. For each type of vehicle at a given depot, we create a fictitious depot with a separate commodity to ensure that the number of vehicles of each type at each depot is balanced. The case that the vehicles have upper bounds on their total travel time is handled by the specification of a time window for the depot. The case that the vehicles have different periods of availability is obviously dealt with by the introduction of fictitious depots with time windows.

Fisher and Jaikumar [1978] propose a slightly different formulation for the VRPTW. We present it here because it is relevant to the solution approach taken by CAR. Their formulation has three types of variables: y_{ik} $(i \in V, k \in M)$, equal to 1 if address i is assigned to vehicle k and 0 otherwise; x_{ij}^k $((i,j) \in A, k \in M)$, equal to 1 if arc (i,j) is used by vehicle k and 0 otherwise; and D_i $(i \in N)$, specifying the departure time at customer i. The problem is then to minimize

$$\sum_{(i,j)\in A} c_{ij} \sum_{k\in M} x_{ij}^k \tag{9}$$

subject to

$$\sum_{k\in M} y_{ik} = \begin{cases} |M| & \text{for } i=0, \\ 1 & \text{for } i \in N, \end{cases} \tag{10}$$

$$\sum_{i\in N} q_i y_{ik} \leq Q \qquad \text{for } k \in M, \tag{11}$$

$$\sum_{j\in V} x_{ji}^k = \sum_{j\in V} x_{ij}^k = y_{ik} \qquad \text{for } i \in V,\ k \in M, \tag{12}$$

$$\sum_{k\in M} x_{ij}^k = 1 \Rightarrow D_i + t_{ij} \leq D_j \qquad \text{for } (i,j) \in A, \tag{13}$$

$$e_i \leq D_i \leq l_i \qquad \text{for } i \in V, \tag{14}$$

$$y_{ik} \in \{0,1\} \qquad \text{for } i \in V,\ k \in M, \tag{15}$$

$$x_{ij}^k \in \{0,1\} \qquad \text{for } (i,j) \in A,\ k \in M. \tag{16}$$

Two well known combinatorial optimization problems are embedded within this formulation. Constraints (10) and (11) are the constraints of a *generalized assignment problem* (GAP) and ensure that the depot is part of each route, that every address is served by some vehicle, and that the load assigned to a vehicle does not exceed its capacity. If the y_{ik} are fixed to satisfy (10) and (11), then for each k, constraints (12)-(14) define a TSPTW over the addresses assigned to vehicle k.

2.2. *The pickup and delivery problem with time windows*

As in the previous section, there is a set N of customers. In the current situation, however, each customer $i \in N$ requests the transportation from an origin i^+ to a destination i^-. We write $N^+ = \{i^+ \mid i \in N\}$ for the set of origins and $N^- = \{i^- \mid i \in N\}$ for the set of destinations. The graph $G = (V,A)$ is now defined as follows. The vertex set is given by $V = \{0\} \cup N^+ \cup N^-$, where 0 denotes the depot. The arc set is given by $A = (\{0\} \times N^+) \cup I \cup (N^- \times \{0\})$, where $I \subset (N^+ \cup N^-) \times (N^+ \cup N^-)$ is the set of arcs corresponding to feasible trips between origins and destinations. For each customer $i \in N$, there are a demand q_i and two time windows $[e_{i^+}, l_{i^+}]$ and $[e_{i^-}, l_{i^-}]$. For each arc $(i,j) \in A$, there is a cost c_{ij} and a travel time t_{ij}. Finally, there is a set M of vehicles, each with capacity Q. The objective is to minimize the total travel costs.

The mathematical programming formulation has the same three types of variables as in the case of the VRPTW: x_{ij}^k $((i,j) \in A,\ k \in M)$, equal to 1 if arc (i,j) is used by vehicle k and 0 otherwise; D_i $(i \in N^+ \cup N^-)$, specifying the departure time at vertex i; and y_i $(i \in N^+ \cup N^-)$, specifying the load of the vehicle arriving at i. We note that the flow variables have now a third index in order to ensure that the pickup at i^+ and delivery to i^- are done by the same vehicle. The problem is to minimize

$$\sum_{(i,j)\in A, k\in M} c_{ij} x_{ij}^k \tag{17}$$

subject to

$$\begin{array}{lll}
\sum_{k\in M}\sum_{j\in V} x_{ij}^{k} = 1 & \text{for } i \in N^{+}, & (18)\\
\sum_{j\in V} x_{ij}^{k} - \sum_{j\in V} x_{ji}^{k} = 0 & \text{for } i \in N^{+} \cup N^{-},\ k \in M, & (19)\\
\sum_{j\in V} x_{i^{+}j}^{k} - \sum_{j\in V} x_{ji^{-}}^{k} = 0 & \text{for } i \in N,\ k \in M, & (20)\\
D_{i^{+}} + t_{i^{+}i^{-}} \leqslant D_{i^{-}} & \text{for } i \in N, & (21)\\
x_{ij}^{k} = 1 \Rightarrow D_i + t_{ij} \leqslant D_j & \text{for } (i,j) \in I,\ k \in M, & (22)\\
e_i \leqslant D_i \leqslant l_i & \text{for } i \in N^{+} \cup N^{-}, & (23)\\
x_{ij}^{k} = 1 \Rightarrow y_i + q_i \leqslant y_j & \text{for } (i,j) \in I,\ k \in M, & (24)\\
0 \leqslant y_i \leqslant Q & \text{for } i \in N^{+}, & (25)\\
x_{ij}^{k} \in \{0,1\} & \text{for } (i,j) \in A,\ k \in M. & (26)
\end{array}$$

Minimizing (17) subject to (18), (19) and (26) is a multi-commodity minimum cost flow problem of a more complex structure than in case of the VRPTW. Constraints (20) ensure that each i^{+} and i^{-} are visited by the same vehicle. Constraints (21) represent the precedence relation between pickup and delivery points. Constraints (22) and (23) ensure feasibility of the schedule, and constraints (24) and (25) guarantee feasibility of the loads; we note that capacity constraints are only specified for origins because a vehicle reaches its maximum load after a pickup. We also note that all model extensions presented for the VRPTW can be applied to the PDPTW.

3. Optimization

Optimization algorithms for routing problems with time windows employ the two standard principles of implicit enumeration: dynamic programming and branch and bound. Among the branch and bound methods, two approaches stand out. One is the set partitioning approach, which uses column generation to solve a continuous relaxation of the problem and branch and bound to obtain integrality. The other approach uses state space relaxation to compute lower bounds. Dynamic programming is mainly applied to solve single-vehicle problems. Those problems arise in the context of column generation and state space relaxation, so that dynamic programming algorithms appear as subroutines in branch and bound methods.

In Section 3.1, we collect the applications of dynamic programming, including state space relaxation. In Section 3.2, we discuss the set partitioning approach. A variety of other methods is reviewed below.

Baker [1983] presents a branch and bound method for the TSPTW, in which bounds are derived from longest path problems. He solves small problems with this method.

The most widely studied routing problem with time windows is the school bus routing problem [Orloff 1976], which is essentially an m-TSPTW. Two objectives are distinguished: minimizing fleet size and minimizing a weighted combination of fleet size and total travel time.

As to the first objective, Swersey and Ballard [1984] discretize the time windows and solve the linear programming relaxation of the resulting integer programming problem. For most instances, the solution is integral; otherwise, they are able to modify the solution so as to obtain integrality without increasing the fleet size. Desrosiers, Sauvé and Soumis [1985] study the Lagrangean relaxation which is obtained by relaxing constraints (2). As one visit to each customer is no longer required, the Lagrangean problem is a shortest path problem with time windows. Although the lower bound is often equal to the optimal fleet size, this dual method does not necessarily produce a feasible solution, in which case branch and bound has to be applied.

For the m-TSPTW with the second objective function, Desrosiers, Soumis, Desrochers and Sauvé [1985] study the network relaxation which is obtained by removing the scheduling constraints (4) and (5). If $e_i = l_i$ for all $i \in N$, then this relaxation produces an optimal solution in view of the definition of I. The quality of the bounds deteriorates with an increasing number of customers and an increasing width of the time windows. Two branching rules are proposed: branching on the flow variables and branching by splitting time windows. In the case of very tight time windows, Soumis, Desrosiers and Desrochers [1985] apply the first rule to solve problems with up to 150 customers; as the time windows become wider, the tree grows rapidly in size. The second branching rule can handle wider time windows, but it is concluded that the network relaxation is inferior to the set partitioning relaxation considered in Section 3.2.

Sörensen [1986] suggests the use of Lagrangean decomposition [Guignard 1984; Jörnsten, Nasberg and Smeds 1985] for the VRPTW. The two resulting subproblems are the shortest path problem with time windows and the generalized assignment problem. No computational results have been reported.

The VRPTW formulation of Fisher and Jaikumar [1978] consists of two interrelated components: a GAP and a TSPTW. To bring out this structure, the formulation is rewritten as a nonlinear generalized assignment problem: minimize

$$\textstyle\sum_k f_k(y_{1k},\ldots,y_{nk})$$

subject to

$$\textstyle\sum_k y_{ik} = \begin{cases} |M| & \text{for } i=0, \\ 1 & \text{for } i \in N, \end{cases} \qquad (10)$$

$$\textstyle\sum_i q_i y_{ik} \leqslant Q_k \qquad \text{for } k \in M, \qquad (11)$$

$$y_{ik} \in \{0,1\} \qquad \text{for } i \in V,\ k \in M, \qquad (15)$$

where $f_k(y_{1k},\ldots,y_{nk})$ is the cost of an optimal solution to the TSPTW defined by the address set $\{i \mid y_{ik} = 1\}$ and the depot, for each k. This value is given by

$$f_k(y_{1k},\ldots,y_{nk}) = \min \textstyle\sum_{ij} c_{ij} x_{ij}^k$$

subject to

$$\textstyle\sum_j x_{ji}^k = \sum_j x_{ij}^k = y_{ik} \qquad \text{for } i \in V, \qquad (12)$$

$$\textstyle\sum_k x_{ij}^k = 1 \Rightarrow D_i + t_{ij} \leqslant D_j \qquad \text{for } (i,j) \in A, \qquad (13)$$

$$e_i \leqslant D_i \leqslant l_i \qquad \text{for } i \in V, \qquad (14)$$

$$x_{ij}^k \in \{0,1\} \qquad \text{for } (i,j) \in A,\ k \in M. \tag{16}$$

Obviously, $f_k(y_{1k},\dots,y_{nk})$ is a very complicated function that cannot be written down explicitly. Fisher and Jaikumar [1978] suggest to iteratively construct a piecewise linear approximation of $f_k(y_{1k},\dots,y_{nk})$ by applying Benders decomposition. Each time the GAP, with some approximation for $f_k(y_{1k},\dots,y_{nk})$, is solved to obtain $(y_{1k},\dots,y_{nk})$, a lower linear support for $f_k(y_{1k},\dots,y_{nk})$ is constructed. This support is derived by solving the m independent TSPTW's for the given $(y_{1k},\dots,y_{nk})$ and using the dual variables thus obtained. The Benders inequalities describing this lower linear support are then added to constraints (10) and (11) to form an extended GAP. This problem is now resolved to obtain a new improved $(y_{1k},\dots,y_{nk})$, which in turn leads to new TSPTW's, whose solution provides further Benders inequalities, and so on.

3.1. *Dynamic programming*

Dynamic programming is a traditional solution method for constrained shortest path problems. The constituents of a dynamic programming algorithm are states, transitions between states, and recurrence equations that determine the value of the objective function at each state. Let us consider the standard shortest path problem on a graph $G = (V,A)$ with vertex set V, arc set A, a source $0 \in V$, and a travel time t_{ij} for each $(i,j) \in A$. Each vertex represents a state, each arc represents a transition between two states, and the value $d(j)$ associated with state j is the shortest path duration from the source 0 to vertex j. The recurrence equations to compute these values are:

$$d(0) = 0,$$

$$d(j) = \min_{(i,j)\in A}\{d(i)+t_{ij}\} \text{ for } j \in V \setminus \{0\}.$$

This algorithm has a running time that is polynomially bounded in the size of G.

Constraints are treated by expansion of the state space and modification of the recurrence equations. Such a dynamic programming approach can be useful for several NP-hard routing problems. However, the cardinality of the state space is usually exponential in the problem size. The practical use of dynamic programming in this context is restricted to state spaces of at most pseudopolynomial size and relatively small problem instances.

3.1.1. *Single-vehicle problems with time windows*

We will consider four problems in this section: the traveling salesman problem with time windows, the single-vehicle dial-a-ride problem, and two constrained shortest path problems.

The TSPTW can be viewed as the problem of finding a shortest path from an origin 0 to a destination $n+1$ that visits all vertices in the set N and respects the time window of each vertex. Christofides, Mingozzi and Toth [1981c] propose the following dynamic programming algorithm. There are states of the form (S,j) with $S \subset N$ and $j \in S$, and $d(S,j)$ denotes the shortest duration of a feasible path starting at 0, visiting all vertices in S, and finishing at j. The optimal solution value $d(N \cup \{n+1\}, n+1)$ is determined by the following recurrence equations:

$$d(\{0\},0) = e_0,$$

$$d(S,j) = \min_{i \in S - \{j\},(i,j) \in A} \{d(S - \{j\},i) + t_{ij}\} \text{ for } j \in N \cup \{n+1\},$$

where we redefine $d(S,j) = e_j$ in case $d(S,j) < e_j$ and $d(S,j) = \infty$ in case $d(S,j) > l_j$.

Psaraftis [1983a] uses dynamic programming to solve the single-vehicle DARP. The states are of the form $(j,k_1,...,k_n)$, where j is the vertex presently visited and each k_i can assume three values that denote the status of customer i: not yet picked up, picked up but not yet delivered, and delivered. It is now straightforward to define the feasible transitions between states. The algorithm has $2n$ stages, each of which extends the paths constructed so far with one arc. The total time requirement is $O(n^2 3^n)$. Psaraftis estimates that this approach is able to solve problems with up to ten customers.

Desrosiers, Dumas and Soumis [1986b] give a similar $2n$-stage algorithm for the capacitated single-vehicle PDPTW. They propose a number of state elimination rules to reduce the computational effort. In addition to Psaraftis' feasibility tests which eliminate states on the basis of information about customers picked up so far, they also have feasibility tests which use information about customers not yet delivered. The algorithm can solve real-life problems with up to 40 customers.

Two types of constrained shortest path problems have been considered: the shortest path problem with time windows (SPPTW) and the capacitated shortest path problem with pickups, deliveries and time windows (SPPPDTW). The main difference between these problems and the single-vehicle DARP is that the path is no longer required to visit all customers. For the SPPTW, which is defined by (1), (3)-(5) and (8), Desrosiers, Pelletier and Soumis [1984] propose a label correcting method. Desrochers and Soumis [1985a, 1985b] give two pseudopolynomial algorithms. One is a label setting method, the other a primal-dual method. Desrochers [1986] generalizes the latter algorithm to the case of multidimensional time windows. For the SPPPDTW ((17) and (19)-(26)), Dumas [1985] and Dumas and Desrosiers [1986] present an algorithm which is similar to the one for the capacitated single-vehicle PDPTW.

As we have mentioned before, dynamic programming algorithms are mostly used as subroutines in other solution methods. This is because the problems considered in this section occur as subproblems in multi-vehicle problems. The TSPTW and the single-vehicle DARP arise in the second phase of cluster-first route-second approaches, where the first phase allocates customers to vehicles and the second phase asks for single-vehicle routes. The SPPTW occurs as a subproblem in the set partitioning algorithm for the m-TSPTW due to Desrosiers, Soumis and Desrochers [1984], in the Lagrangean relaxation algorithm for the fleet size problem due to Desrosiers, Sauvé and Soumis [1985], and in the Lagrangean decomposition algorithm for the VRPTW due to Sörensen [1986]. The SPPPDTW is a subproblem in the set partitioning algorithm for the PDPTW due to Desrosiers, Dumas and Soumis [1987].

3.1.2. *State space relaxation*

For a number of problems, Christofides, Mingozzi and Toth [1981a, 1981b, 1981c] have developed branch and bound algorithms that obtain lower bounds by dynamic programming on a relaxed state space. They take a dynamic programming algorithm for the problem under consideration as a starting point and replace its state space by a smaller space in such a way that the recursion over the new state space requires only polynomial time and yields a lower bound on the optimal solution value of the original problem.

State space relaxation is based on a mapping g from the original state space to a space of smaller cardinality. If there is a transition from S_1 to S_2 in the original state space, then there must be a transition from $g(S_1)$ to $g(S_2)$ in the new state space. We illustrate this idea on the TSPTW [Christofides, Mingozzi and Toth 1981c].

With each vertex i, an arbitrary integer β_i is associated, with $\beta_0 = \beta_{n+1} = 0$. The mapping is defined by $g(S,j) = (k,\beta,j)$, where $k = |S|$ and $\beta = \Sigma_{i \in S}\beta_i$. The new recurrence equations are:

$$d(0,\beta,0) = \begin{cases} 0 & \text{if } \beta = 0, \\ \infty & \text{if } \beta \neq 0, \end{cases}$$

$$d(k,\beta,j) = \min_{i \neq j,(i,j) \in A}\{d(k-1,\beta-\beta_j,i)+t_{ij}\} \text{ for } j \in N \cup \{n+1\},$$

where we redefine $d(k,\beta,j) = e_j$ in case $d(k,\beta,j) < e_j$ and $d(k,\beta,j) = \infty$ in case $d(k,\beta,j) > l_j$. The lower bound is now given by

$$\min_{j \in N,(j,n+1) \in A}\{d(n, \textstyle\sum_{i \in N}\beta_i,j)+t_{j,n+1}\}.$$

This lower bound can be improved by the use of vertex penalties and state space modifications. Vertex penalties serve to decrease the travel times of arcs incident to undercovered vertices and to increase the travel times of arcs incident to overcovered vertices; these penalties are adjusted by subgradient optimization. Similarly, the weights β_i can be modified by subgradient optimization. The resulting branch and bound method is able to solve problems with up to 50 vertices.

Kolen, Rinnooy Kan and Trienekens [1987] extend this approach to the VRPTW. They use a two-level state space relaxation. At the first level, a lower bound on the costs of a time-constrained path from the depot to vertex j with load q is computed. This is done with an adaptation of the above method for the TSPTW. The states are of the form (t,q,j), where q is the load of a shortest path arriving at vertex j no later than time t. We have $0 \leqslant t \leqslant T$ where T is the scheduling horizon, $0 \leqslant q \leqslant Q$ where Q is the vehicle capacity, and $j \in N$. At the second level, a lower bound on the costs of m routes with total load $\Sigma_{i \in N}q_i$ and different destination vertices is computed. The states are now of the form (k,q,j), where q is the total load of the first k routes and j is the destination vertex of route k. Vertex penalties are used to improve the lower bounds. Problems with up to fifteen customers are solved.

3.2. *Set partitioning*

Vehicle routing problems and in particular the VRPTW and the PDPTW can be reformulated as set partitioning problems, with variables (columns) corresponding to feasible routes.

Let R be the set of feasible routes of the problem under consideration. For each route $r \in R$, we define γ_r as the sum of the costs of its arcs and δ_{ri} ($i \in N$) as a binary constant, equal to 1 if route r visits customer i and 0 otherwise. If x_r ($r \in R$) is equal to 1 if route r is used and 0 otherwise, the set partitioning problem is to minimize

$$\textstyle\sum_{r \in R} \gamma_r x_r \qquad (27)$$

subject to

$$\textstyle\sum_{r \in R} \delta_{ri} x_r = 1 \qquad \text{for } i \in N, \qquad (28)$$

$$x_r \in \{0,1\} \qquad \text{for } r \in R. \qquad (29)$$

Although problems (17)-(26) and (27)-(29) are equivalent, their continuous relaxations are not. This is because the variables in the latter problem are restricted to feasible paths in which each customer is included or not. Any solution to the relaxed version of (27)-(29) is a feasible solution to the relaxation of (17)-(26), but not vice versa. We can therefore expect to obtain better lower bounds on the basis of the set partitioning formulation.

Because of the cardinality of R, the relaxed set partitioning problem cannot be solved directly and column generation is used. That is, a new column of minimum marginal cost is generated by solving an appropriate subproblem. If its marginal cost is negative, then it is added to the linear program, the problem is reoptimized and column generation is applied again; otherwise, the current solution to the linear program is optimal. Before discussing results for specific vehicle routing problems, we first describe some general aspects of this approach.

3.2.1. *The subproblem*

The objective function of the subproblem has coefficients that depend on the values of the dual variables π_i ($i \in N$) of the continuous relaxation of the set partitioning problem. The constraints define a path subject to side constraints but not necessarily visiting all customers. They include (3)-(8) for the VRPTW, (3)-(5) and (8) for the m-TSPTW, and (19)-(26) for the PDPTW.

As we have seen in Section 3.1.1, dynamic programming is a suitable method to solve these subproblems to optimality, because the state spaces are relatively small.

3.2.2. *The master problem*

The continuous relaxation of the set partitioning problem is solved by the simplex algorithm. This method produces the dual values π_i that are needed for column generation and enables easy reoptimization each time new columns are generated.

To obtain an integral solution to the master problem, we add cutting planes or we use branch and bound. Each time a new constraint is added, another round of column generation is applied in order to solve the modified master problem. We must restrict ourselves to types of constraints that are compatible with the column generation method. For any cutting plane, the method must be able to compute its coefficients in order to evaluate the marginal cost of new columns. For any branching rule, the method must be able to exclude the columns that have become infeasible by branching

In case the c_{ij} are integral, a compatible type of cut is the one that rounds the objective up to the next integer. In the particular case that we minimize fleet size, this cut has the same coefficient 1 in each column; if it has a dual value π, a new column is generated by minimizing the reduced cost

$$\sum_{(i,j)\in A}(c_{ij}-\pi_i)x_{ij} - \pi.$$

However, we cannot use Gomory cuts or other types of cuts whose coefficients are not known before the new column is generated.

As to branching, the usual rule to fix a fractional variable x_r to 0 or 1 is not compatible. We can fix $x_r = 1$ by simply deleting the customers on route r from the subproblem. But we cannot fix $x_r = 0$: there is no way to prevent route r from being generated again. Four types of compatible branching rules have been proposed: branching on the flow variables of route r; branching on the position of a customer in route r; branching by splitting time windows; and branching on the number of vehicles of a given type in problems with multiple vehicle types. These rules have been listed here in order of increasing effectiveness.

3.2.3. *Acceleration techniques*

There are various ways to improve the performance of the set partitioning approach.

First of all, the set partitioning problems that arise in the context of vehicle routing are highly degenerate. It is an obvious idea to improve the convergence of the simplex method by a perturbation strategy.

Secondly, the solution of the relaxed master problem can be accelerated by the simultaneous generation of columns. The solution of a subproblem by dynamic programming produces not only a column of minimum reduced cost, but also many other columns of negative reduced cost. Several of these can be added.

In the third place, the solution of most of the subproblems can be greatly sped up by the heuristic elimination of vertices, arcs, and states. The first columns are generated in subnetworks, which only consist of customers with large dual values and inexpensive arcs; in addition, less promising states are ruled out during the recursion. At later stages, the elimination rules are gradually relaxed, until at the final stage the full network and state space are used in order to prove optimality.

3.2.4. *The multi-salesman and vehicle routing problem with time windows*

Desrosiers, Soumis and Desrochers [1984] propose a set partitioning approach to the m-TSPTW. The column generation problem is the SPPTW, which was reviewed in Section 3.1.1. In their algorithm, two cuts are added to the master problem: one to round up the number of vehicles and one to round up the total costs. After that, branching on flow variables is applied. With this rule, it is time consuming to achieve optimality, even if the integrality gap is small. They solve problems with up to 151 customers; the solution time on a CDC Cyber 173 ranges from 100 to 1000 seconds, depending on the width of the time windows. A recent improvement of the algorithm is able to solve problems with 223 customers within 600 seconds. A branching rule based on time window splitting is under development.

Desrosiers, Dumas and Soumis [1986a] extend this algorithm to the case of multiple vehicle types. Several SPPTW's are now to be solved, one for each type of vehicle. Branching is first done on the number of vehicles of a given type; when this number is integral for each type, the usual branching on flow variables is applied.

No set partitioning algorithm for the VRPTW has been proposed so far. However, Desrochers [1986] presents a dynamic programming algorithm for the shortest path problem with a variety of constraints. This method is suitable for solving the subproblems that occur in this context.

3.2.5. *The pickup and delivery problem with time windows*

Dumas [1985] develops a set partitioning approach for the PDPTW. He solves problems with 30 customers (60 vertices) within 100 seconds on a CDC Cyber 173. These problems have tight capacity constraints ($q_i \geqslant Q/3$) and loose time window constraints. Narrowing the time windows significantly decreases the cardinality of the state space and thereby the computation time.

The subproblem in this case is the SPPPDTW, which was reviewed in Section 3.1.1. The algorithm of Dumas [1985] first branches on the number of vehicles per type and then on flow variables. Desrosiers, Dumas and Soumis [1987] replace the latter branching rule by branching on the position of customers in routes and obtain improved results.

4. Approximation: construction

In spite of the recent success of optimization algorithms for vehicle routing with time windows, it is unlikely that they will be able to solve large-scale problems. In many situations one has to settle for algorithms that run fast but may produce suboptimal solutions. In this chapter, we review three types of approximation algorithms. *Construction* methods try to build a feasible solution starting from the raw data. *Incomplete optimization* methods use a combination of enumeration of the solution space and heuristic rules to truncate the search. *Iterative improvement* methods start from a feasible solution and seek to improve it through a sequence of local modifications. These types of methods have been widely applied to unconstrained routing problems. Their extension to constrained problems has only recently become a subject of investigation. In presenting this work, we will concentrate on feasibility rather than optimality aspects. As already indicated in Chapter 1, we consider construction in the present chapter, incomplete optimization in Chapter 5, and iterative improvement in Chapter 6.

Side constraints of interest to us are: *single time windows* at customers, *multiple time windows* at customers, both *deliveries to and collections from* customers, and *precedence constraints* between customers. For presentational convenience, we will often consider the side constraints one at a time. Furthermore, when describing iterative improvement methods, we will restrict ourselves to the TSP with side constraints and at the end indicate how the presented techniques can be extended to the VRP with side constraints. We note that it is possible to model the PDPTW as a VRPTW with both collections and deliveries and precedence constraints. For each customer in the PDPTW there are two customers in the VRPTW; one corresponding to the origin, where load will be collected, and the other corresponding to the destination, where load will be delivered. In addition, there is a precedence constraint specifying that the customer where load will be collected should precede the customer where load will be delivered. Therefore, the described iterative improvement methods can also be applied to the PDPTW.

As in Section 3.1.1, we split the depot (vertex 0) in an 'origin' (vertex 0) and a 'destination' (vertex $n+1$). In the sequel, when we refer to a route, we assume that it is given by $(0,1,...,i,...,n,n+1)$, where i is the ith customer visited by the vehicle. There are two quantities associated with a subpath $(h,...,k)$ that play a dominant role in the algorithms below. The *possible forward shift* S^+_{hk} is the largest increase in the departure time D_h at h which causes no violation of the time windows along the path $(h,...,k)$:

$$S^+_{hk} := \min_{h \leqslant j \leqslant k}\{l_j - (D_h + \textstyle\sum_{h \leqslant i < j} t_{i,i+1})\}.$$

The *possible backward shift* S^-_{hk} is the largest decrease in the departure time D_h at h which causes no waiting time along the path $(h,...,k)$:

$$S^-_{hk} := \min_{h \leqslant j \leqslant k}\{D_j - e_j\}.$$

These quantities express the flexibility we have when we want to push customers forward or backward in time.

In the design of construction methods, there are two key questions:

(1) Selection criterion: *which* customer is selected next to be inserted into the current solution?

(2) Insertion criterion: *where* will it be inserted?

While such decisions may be made at the same time, several of the algorithms to be discussed in Sections 4.2 and 4.3 employ different criteria for selection and insertion. Before we start our discussion of construction methods, we present some results on the complexity of finding initial feasible solutions.

4.1. *Complexity*

Although constructing an initial solution may seem easier than finding an optimal solution, we will show that in the presence of side constraints this is not always true.

The traveling salesman problem. In the TSP [Lawler, Lenstra, Rinnooy Kan and Shmoys 1985], we are given a complete graph on a set $V = \{0,1,...,n,n+1\}$ of vertices, and a travel time t_{ij} for each arc $(i,j) \in V \times V$. A solution to the TSP is a path of minimum duration from origin 0 to destination $n+1$ that visits each other vertex exactly once. The duration of a path is the sum of the travel times of the arcs contained in it. We assume that the matrix (t_{ij}) is symmetric and satisfies the triangle inequality, i.e.,

$$t_{ij} = t_{ji} \qquad \text{for } i,j \in V,$$

$$t_{ik} \leqslant t_{ij} + t_{jk} \qquad \text{for } i,j,k \in V.$$

Constructing an initial feasible tour is trivial because any permutation of the vertices constitutes a feasible tour. Note that the above definition stipulates the existence of a complete graph, contrary to our earlier definition in Chapter 2. In case of an arbitrary graph, the problem of finding an initial solution is equivalent to the problem of finding a Hamiltonian cycle in a graph, which is known to be NP-complete in the strong sense.

The traveling salesman problem with time windows. In the TSPTW, we are given in addition to the travel times between vertices, for each vertex i a time window, denoted by $[e_i, l_i]$, where e_i specifies the earliest service time and l_i the latest service time. The problem of determining whether there exists a feasible tour for the TSPTW is NP-complete in the strong sense. Our proof starts from the recognition version of the TSP, which is known to be NP-complete in the strong sense [Garey and Johnson 1979]:

TSPDECISION
Instance: A set $V = \{0, \ldots, n+1\}$ of vertices, a travel time $t_{ij} \in \mathbb{Z}^+$ for each $(i,j) \in V \times V$, and a bound $B \in \mathbb{Z}^+$.
Question: Does there exist a path from origin 0 to destination $n+1$ of duration not larger than B that visits each vertex exactly once?

Given an instance of TSPDECISION, we construct the corresponding instance of TSPTW by giving each city a time window $[0, B]$. This implies that there exists a feasible tour if and only if TSPDECISION has a solution.

In addition, the problem of determining whether there exists a feasible solution to the TSPTW belongs to NP: a non-deterministic algorithm need only guess an ordering of the vertices and test in polynomial time whether it is feasible.

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

In the special case where all load to be delivered has to be collected at vertex 0 and all load to be collected has to be delivered to vertex $n+1$, there exits a feasible tour if and only if it is feasible to visit all delivery vertices before all collection vertices. This strategy is known in vehicle routing problems as *back-hauling*. In the general case, the existence problem is more difficult. The problem of determining whether there exists a feasible tour for the TSP with mixed collections and deliveries is NP-complete in the strong sense. Our proof starts from the following problem, which is known to be NP-complete in the strong sense [Garey and Johnson 1979]:

3-PARTITION
Instance: A finite set A of $3m$ elements, a bound $B \in \mathbb{Z}^+$ and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$, with $B/4 < s(a) < B/2$ and

$$\textstyle\sum_{a\in A} s(a) = mB.$$

Question: Can A be partitioned into m mutually disjoint sets $S_1, S_2, \cdots, S_m$ such that, for $1 \leqslant i \leqslant m$,

$$\textstyle\sum_{a\in S_i} s(a) = B?$$

(Notice that the above constraints on the item size imply that every such S_i must contain exactly three elements from A.)

Given an instance of 3-PARTITION, we construct the following instance of the TSP with mixed collections and deliveries. There are $3m$ delivery vertices a with load equal to $s(a) (a \in A)$ and m collection vertices with load equal to B. The salesman has a vehicle with capacity B. Note that in a feasible solution a collection can only be made if the vehicle is empty. This implies that a feasible solution will consist of m subsequences, each consisting of a collection followed by three deliveries. But such a solution exists if and only if 3-PARTITION has a solution.

In addition, the problem of determining whether there exists a feasible solution to the TSP with mixed collections and deliveries is a member of NP: a non-deterministic algorithm need only guess an ordering of the vertices and test in polynomial time whether it is feasible.

The traveling salesman problem with precedence constraints. In the TSP with precedence constraints, we are given in addition to the travel times, precedence constraints specifying that some pairs of vertices have to be visited in a prescribed order. The problem of finding an initial feasible solution is trivial because all we have to do is to visit the vertices in topological order.

The vehicle routing problem. The problem of determining whether there exists a feasible set of routes for the VRP is NP-complete in the strong sense. Our proof starts again from 3-PARTITION [Garey and Johnson 1979].

Given an instance of 3-PARTITION, we construct the following instance of the VRP. There will be $3m$ customers a with load equal to $s(a) (a \in A)$ and m vehicles with capacity equal to B. Note that the total load to be delivered is equal to the total vehicle capacity. This implies that a feasible set of routes corresponds to a division of the set of customers into m subsets, such that the sum of the loads over the members of a subset is exactly equal to the vehicle capacity. But such a set of routes exists if and only if 3-PARTITION has a solution.

In addition, the problem of determining whether there exists a feasible set of routes for the VRP belongs to NP: a non-deterministic algorithm need only guess a division of the set of vertices into subsets, guess an ordering for the vertices in each subset, and test in polynomial time whether it is feasible.

4.2. *The vehicle routing problem with time windows*

Solomon [1983] was one of the first who attempted to adapt the existing approximation algorithms for the VRP to the VRPTW. Part of the material in this section is based on his work.

Savings. The savings method of Clarke and Wright [1964] is probably the first and certainly the best known heuristic proposed to solve the VRP. It is a sequential procedure. Initially, each customer has its own route. At each iteration, an arc is selected so as to combine two routes into one, on the basis of some measure of cost savings and subject to vehicle capacity constraints. Note that in this case the selection criterion applies to arcs rather than customers and that the insertion question does not occur.

In order to adapt this procedure for the VRPTW, we must be able to test the time feasibility of an arc. While in pure routing problems the direction in which a route is traversed is usually immaterial, this is not the case anymore in the presence of time windows. Hence, we only consider arcs from the last customer on one route to the first customer on another.

If two routes are combined, the departure times on the first route do not change. As to the second route, one necessary condition for feasibility is that the departure time at the first customer is no more than his latest service time, but that is not all. The other departure times on the route could be pushed forward, and one of them could become infeasible. This is where the possible forward shift enters the picture. For any path $(1,...,n+1)$, a change in the departure time at 1 is feasible if and only if it is no more than $S^+_{1,n+1}$.

By selecting of a cost effective and time feasible arc, the modified heuristic could link two customers whose windows are far apart in time. This suggests a further modification which selects arcs on the basis of both spatial and temporal closeness of customers, e.g., by adding a waiting time penalty to the cost savings.

Nearest neighbor. Initially, a route consists of the depot only. At each iteration, an unvisited customer who is closest to the current end point of the route is selected and added to the route to become its new end point. The selection is restricted to those customers whose addition is feasible with respect to capacity and time window constraints. A new route is started any time the search fails, unless there are no more customers to schedule.

The measure of closeness should include spatial as well as temporal aspects. Solomon [1983] proposes the following:

$$\alpha_1 t_{ij} + \alpha_2(\max\{e_j, D_i + t_{ij}\} - D_i) + \alpha_3(l_j - (D_i + t_{ij})), \text{ with } \alpha_1 + \alpha_2 + \alpha_3 = 1.$$

This measures the travel time between customers i and j, the difference between their respective delivery times, and the urgency of a delivery at j.

Insertion. Insertion methods treat the selection and insertion decisions separately. We distinguish sequential and parallel insertion rules. The former construct the routes one by one, whereas the latter build them up simultaneously. All methods considered here are of the sequential type.

The general scheme of an insertion method is simple. Let U be the set of unrouted customers. For each customer $u \in U$, we first determine the best feasible point i_u after which it could be inserted into the emerging route:

$$\iota(u,i_u) = \min_{0 \leqslant i \leqslant n}\{\iota(u,i)\} \text{ for } u \in U.$$

We next select the customer u^* to be inserted into the route:

$$\sigma(u^*,i_{u^*}) = \min_{u\in U}\{\sigma(u,i_u)\}.$$

The insertion criterion ι and the selection criterion σ are still to be specified; we refer to Solomon [1983] and Savelsbergh [1985] for a number of possible definitions which take both spatial and temporal aspects into account. When no more customers can be inserted, a new route is started, unless all customers have been routed.

A detailed implementation of an insertion algorithm, given an assignment of customers to vehicles, is given in Section 5.3.

Cluster first-route second. The best known example of the cluster first-route second approach is the sweep method of Gillett and Miller [1974]. Their idea is to sweep a ray with the depot as pivot and a randomly selected 'seed' customer, clockwise or counterclockwise in the plane, and add customers to a cluster as they are encountered, until the vehicle capacity is exceeded. A route for the customers in this sector is then constructed using an insertion method. This sweeping process is repeated until all customers are routed.

In the presence of time windows, some of the customers in a sector may remain unscheduled. Therefore, in a time oriented sweep method, extra attention is paid to the selection of the seed customers. Solomon [1983] suggests to bisect the previously constructed sector, and let the customer that gives rise to the smallest positive angle formed by the ray from the depot through that customer and the bisector be the seed for the next cluster to be formed. The intuition behind this partitioning of the unrouted customers in the sector into two subsets is that, assuming a counterclockwise sweep, the customers in the right half sector will be relatively far away from the new cluster. Hence, a better schedule may be obtained by scheduling them at a later stage.

Chapter 5 discusses a cluster first-route second approach based on the iterative optimization algorithm suggested by Fisher and Jaikumar [1978] and described in Chapter 3.

Solomon [1983] concludes on the basis of extensive computational experiments that insertion methods outperform other types of construction methods.

4.3. *The pickup and delivery problem with time windows*

Jaw, Odoni, Psaraftis and Wilson [1986] consider a variant of the DARP. Their approach seems to be applicable to the proper DARP as well.

The customers that are to be picked up and delivered have the following types of service constraints. Each customer i specifies either a desired pickup time $\overline{D}_{i^+}$ or a desired delivery time $\overline{A}_{i^-}$, and a maximum travel time $\overline{T}_i$; in addition, there is a tolerance $\overline{U}$. If customer i has specified a desired pickup time, the actual pickup time D_{i^+} should fall within the time window $[\overline{D}_{i^+}, \overline{D}_{i^+} + \overline{U}]$; if he has specified a desired delivery time, the actual delivery time A_{i^-} should fall within the window $[\overline{A}_{i^-} - \overline{U}, \overline{A}_{i^-}]$. Moreover, his actual travel time should not exceed his maximum travel time: $A_{i^-} - D_{i^+} \leqslant \overline{T}_i$. Note that this information suffices to determine two time windows $[e_{i^+}, l_{i^+}]$ and $[e_{i^-}, l_{i^-}]$ for each customer i. Finally, waiting time is not allowed when the vehicle is carrying passengers.

The selection criterion is simple: customers are selected for insertion in order of increasing e_{i^+}. The insertion criterion is as follows: among all feasible points of insertion of the customer into the vehicle schedules, choose the cheapest; if no feasible point exists, introduce an additional vehicle.

For the identification of feasible insertions, the notion of an *active period* is introduced. This is a period of time a vehicle is active between two successive periods of slack time. For convenience, we drop the superscript indicating pickup or delivery. For each visit to a customer i during an active period, we define the following variants of possible backward and forward shifts:

$$\Sigma_i^- = \min\{\min_{j \leqslant i}\{A_i - e_i\}, \Lambda\},$$
$$\Sigma_i^+ = \min_{j \leqslant i}\{l_i - A_i\},$$
$$S_i^- = \min_{j \geqslant i}\{A_i - e_i\},$$
$$S_i^+ = \min\{\min_{j \geqslant i}\{l_i - A_i\}, L\},$$

where Λ and L are the durations of the slack periods immediately preceding and following the active period in question. Σ_i^- (Σ_i^+) denotes the maximum amount of time by which every stop preceding but not including i can be advanced (delayed) without violating the time windows, and S_i^- (S_i^+) denotes the maximum amount of time by which every stop following but not including i can be advanced (delayed). These quantities thus indicate how much each segment of an active period can be displaced to accommodate an additional customer. Once it is established that some way of inserting the pickup and delivery of customer i satisfies the time window constraints, it must be ascertained that it satisfies the maximum travel time constraints.

The cost measure that is used to choose among feasible insertions is a weighted combination of customer dissatisfaction and resource usage.

Sexton and Bodin [1985a, 1985b] consider a variant of the single-vehicle DARP in which only deadlines for the deliveries are specified. Their solution approach is to apply Benders decomposition to a mixed 0-1 nonlinear programming formulation, which separates the routing and scheduling component.

5. Approximation: incomplete optimization

Fast approximation algorithms can also be derived from optimization algorithms. The principal idea is to use heuristic rules to truncate the search of the solution space.

In case of the set partitioning algorithms the two main techniques are the heuristic generation of columns and the partial exploration of the branch and bound tree.

Heuristic generation of columns is based on the third type of acceleration technique mentioned in Section 3.2.3. While solving the relaxed master problem, we eliminate vertices, arcs and states in a heuristic fashion. The elimination rules are not relaxed, so that an approximate solution to the linear program is obtained.

Partial exploration of the search tree can take place in several ways. One is to obtain an integral solution by depth-first search and then to explore the tree for the remaining available time. Another way is to use an invalid branching rule, i.e., to eliminate branches on heuristic grounds.

A combination of these techniques has been used to obtain feasible integral solutions within two percent from the optimum with highly reduced running times.

The iterative optimization algorithm suggested by Fisher and Jaikumar [1978], which has the potential of obtaining an optimal solution, can be turned into an effective approximation algorithm if terminated early. For the VRP (that is, the case without time windows) Fisher and Jaikumar [1981] propose a cluster first-route second method by only considering the first iteration of the Benders decomposition. In the first phase, an assignment of customers to vehicles is obtained by solving a GAP with an objective function that approximates the cost of the traveling salesman tours of the vehicles through the customers. In the second phase, once the assignment has been made, a routing of each vehicle through its set of customers is obtained by solving a TSP. The objective function that approximates delivery cost is obtained by *seed routes* and the cost of inserting customers into these seed routes. A seed route is an artificial route consisting initially of the depot and a *seed point* (or *seed* for short), which indicates an area that is expected to be visited by one vehicle.

In the process of developing the interactive distribution planning system CAR, that is the subject of Part II, we have extended this cluster first-route second approach to handle various side constraints: time windows, mixed deliveries and collections, and different vehicle types.

In the present chapter, we describe the clustering phase and the route construction phase. The resulting solution can be subjected to the iterative improvement procedures presented in Chapter 6; these procedures serve primarily to improve individual routes (Sections 6.1-7), but may also modify the clustering (Section 6.8).

To be able to handle mixed deliveries and collections we make use of the following observation. As long as all load to be delivered is collected at the depot and thus all load collected is to be delivered at the depot, a necessary and sufficient condition for a load feasible route to exist is that neither the total amount of deliveries nor the total amount of collections exceeds the vehicle capacity. In that case, a route can be constructed using the backhauling strategy which specifies that all deliveries have to be made before any collection. In the cluster phase, we therefore have to solve two GAP's: one for the deliveries and one for the collections, both based on the same set of seeds.

5.1. *The generalized assignment problem*

Given are a set N of customers and a set M of vehicles. For each customer $i \in N$ there is a load q_i. For each vehicle $k \in M$ there is a capacity Q_k. Finally, there is a cost c_{ik} associated with the assignment of customer i to vehicle k. The objective is to assign all customers to a vehicle at minimal cost. The mathematical programming formulation for the GAP has one type of variable: y_{ik} $(i \in N, k \in M)$ is equal to 1 if customer i is assigned to vehicle k and 0 otherwise. The problem is then to minimize

$$\sum_{i \in N, k \in M} c_{ik} y_{ik}$$

subject to

$$\begin{aligned} &\sum_{k \in M} y_{ik} = 1 && \text{for } i \in N, \\ &\sum_{i \in N} q_i y_{ik} \leqslant Q_k && \text{for } k \in M, \\ &y_{ik} \in \{0,1\} && \text{for } i \in N,\ k \in M. \end{aligned}$$

This is, in fact, a special case of the GAP because the coefficients in the constraint matrix do not depend on the rows. (The demand of an customer is independent of the vehicle that is going to serve the customer.) A number of optimization algorithms for the GAP have been developed [Ross and Soland 1975; Fisher, Jaikumar and Van Wassenhoven 1984; Martello and Toth 1981], but these are only able to solve small problem instances. For the solution of very large GAP's, an approximation algorithm is required.

The basis of our approximation algorithm is a very simple but effective scheme [Martello and Toth 1981]. For its description we introduce the following notation:

F:	the set of unassigned customers, initially all customers;
A_k:	the set of customers assigned to vehicle k, initially empty;
$c(i,k)$:	the cost of assigning customer i to vehicle k;
$f(i,k,S)$:	a boolean function indicating whether or not customer i can be assigned to vehicle k given that a set S of customers has already been assigned to vehicle k;
$v(i)$:	the vehicle to which customer i is assigned (only introduced for notational convenience).

The algorithm iteratively considers all unassigned customers and determines the customer which has the maximum difference between the smallest and second smallest cost of a feasible assignment. This customer is then assigned to the vehicle for which the minimum cost was attained. More formally:

```
$WHILE\ (F \neq \varnothing)\ \{$
  $bound \leftarrow \infty$
  $FOR\ (i \in F)\ \{$
    $c_1(i,k_1) \leftarrow \min_{k \in V}\{c(i,k) \mid f(i,k,A_k) = TRUE\}$
    [if $\forall k : f(i,k,A_k) = FALSE$, then $F \leftarrow F \setminus \{i\}$]
    $c_2(i,k_2) \leftarrow \min_{k \in V \setminus \{k_1\}}\{c(i,k) \mid f(i,k,A_k) = TRUE\}$
    [if $\forall k \neq k_1 : f(i,k,A_k) = FALSE$, then $c_2(i,k_2) \leftarrow -\infty$]
    $diff \leftarrow c_2(i,k_2) - c_1(i,k_1)$
```

```
        IF ( $diff < bound$ ) {
          $bound \leftarrow diff$
          $i^* \leftarrow i$
          $k^* \leftarrow k_1$
        }
      }
      $F \leftarrow F \setminus \{i^*\}$
      $A_{k^*} \leftarrow A_{k^*} \cup \{i^*\}$
    }
```

Note that there is no guarantee that each customer will be assigned to some vehicle. Especially when $\Sigma_{i \in N} q_i$ is almost equal to $\Sigma_{k \in M} Q_k$, it is possible that some of the customers will be left unassigned. In order to prevent customers with large loads to be treated in the end we split the set of customers based on their demands and process the set with the larger demands first. A natural choice for the threshold to divide the set of customers would be half the vehicle capacity because we know in advance that loads larger than half the vehicle capacity should end up in different vehicles.

Once the initial assignment of customers to vehicles is obtained we apply two local improvement procedures. The first one tries to reassign a customer to another vehicle, the second one tries to swap two customers between their associated vehicles.

Improvement procedure 1:

```
    FOR ( $i \in V$ ) {
      $c(i,k_1) \leftarrow \min_{k \in V \setminus \{v(i)\}} \{c(i,k) \mid f(i,k,A_k) = TRUE\}$
      IF ( $c(i,k_1) < c(i,v(i))$ ) {
        $A_{v(i)} \leftarrow A_{v(i)} \setminus \{i\}$
        $A_{k_1} \leftarrow A_{k_1} \cup \{i\}$
      }
    }
```

Improvement procedure 2:

```
    FOR ( $i \in V$ ) {
      FOR ( $j \in V \wedge j \neq i \wedge v(j) \neq v(i)$ ) {
        IF ( $f(j,v(i),A_{v(i)} \setminus \{i\}) = TRUE \wedge$
          $f(i,v(j),A_{v(j)} \setminus \{j\}) = TRUE \wedge$
```

```
            $c(j,v(i)) + c(i,v(j)) < c(i,v(i)) + c(j,v(j))$ ) {
                $A_{v(i)} \leftarrow A_{v(i)} \setminus \{i\} \cup \{j\}$
                $A_{v(j)} \leftarrow A_{v(j)} \setminus \{j\} \cup \{i\}$
            }
        }
    }
```

To complete the description of the algorithm we have to specify two functions: $c(i,k)$, which indicates the costs associated with the assignment of customer i to vehicle k, and $f(i,k,S)$, which indicates whether or not it is possible to assign customer i to vehicle k given that a set S of customers has already been assigned to vehicle k.

The function c(i,k). The cost of assigning customer i to vehicle k should reflect the knowledge that in the routing phase a traveling salesman problem has to be solved for each of the sets of customers assigned to the vehicles. Our function $c(i,k)$ is based on *seed routes* and the cost of inserting a customer into a seed route. A seed route is an artificial route consisting initially of the depot and a *seed point*, where the seed point indicates an area that is expected to be visited by one vehicle. This results in the following cost function

$$c(i,k) = \min\{t_{0,i} + \alpha t_{i,s_k} - t_{0,s_k}, \alpha t_{s_k,i} + t_{i,0} - t_{s_k,0}\},$$

where α is the route shape parameter introduced by Gaskell [19xx]. The metric defined by this cost function will sometimes be referred to as the *extra mileage* metric.

The function f(i,k,S). The boolean function $f(i,k,S)$ establishes whether or not it is feasible to assign customer i to seed k given that a set S of customers has already been assigned to seed k. This feasibility function is our only means to prevent assignments that turn out to be bad in the routing phase. The primary component is of course concerned with vehicle capacity:

$$f(i,k,S) \leftarrow FALSE \text{ if } q_i > Q_k - \sum_{j \in S} q_j$$

We can add several types of heuristic feasibility tests:

- $f(i,k,S) \leftarrow FALSE$ if the vehicle associated with a seed k is not allowed to perform the service at customer i (for instance because it does not have the appropriate loading equipment);
- $f(i,k,S) \leftarrow FALSE$ if the number of customers assigned to seed k exceeds a given bound C, i.e., if $|S| \geqslant C$;
- $f(i,k,S) \leftarrow FALSE$ if the total time spent on unloading exceeds a given bound U, i.e., $u_i > U - \Sigma_{j \in S} u_j$, where u_j denotes the unloading time at customer j;
- $f(i,k,S) \leftarrow FALSE$ if there is no time feasible path from the depot to seed k via customer i, and no time feasible path from seed k to the depot via customer i, i.e., $\max\{e_i, e_0 + t_{0,i}\} + t_{i,s_k} > l_{s_k} \wedge \max\{e_i, e_{s_k} + t_{s_k,i}\} + t_{i,0} > l_0$.

It would even be possible to construct partial routes on a subset of customers of S

and check the time feasible insertion in this partial route. However, we have to keep in mind that the function $f(i,k,S)$ is called many times and should therefore require a moderate amount of computing time.

5.2. *Seed selection*

As the cost of assigning customers to vehicles is based on seed routes, two questions remain to be answered before we can actually apply the algorithm presented in the previous section. We have to decide where the seeds will be located and which vehicle type will be allocated to each seed. Although these questions form essentially one problem and should ideally be treated simultaneously, we propose to treat them heuristically in the order specified above.

Location of seeds. In order to simplify the computations, seeds are located at customers. First, a set of candidate seeds is constructed based on the difficulty degree of customers. This set is then gradually refined to end up with a set containing exactly the requested number of seeds. The difficulty degree of a customer is a weighted combination of several of its characteristics:

$$d_i := \alpha_1 t_{0,i} + \alpha_2 q_i - \alpha_3(l_i - e_i).$$

This means that customers far from the depot, customers with large loads, and customers with small time windows are considered to be difficult. The parameters α_1, α_2 and α_3 are used to tune the algorithm to specific problem instances. The initial set S^3 of candidate seeds will have a cardinality of tree times the number of requested seeds and will contain the customers with the largest difficulty degrees. This set is refined by choosing among its members a subset S^2 that will have a cardinality of twice the number of requested seeds and will contain candidate seeds that are geographically dispersed. This is achieved by the following procedure:

```
WHILE ( $|S^2| <$ twice the requested number of seeds ) {
    $k^* \leftarrow argmax_{k \in S^3} \{\beta t_{0k} + \sum_{l \in S^2} t_{kl}\}$
    $S^3 \leftarrow S^3 \setminus \{k^*\}$
    $S^2 \leftarrow S^2 \cup \{k^*\}$
}
```

Again, the parameter β is used to tune the algorithm. If β is taken greater than one, the procedure favours addresses far away from the depot. Next, we associate with each seed $k \in S^2$ a load $L(k)$ as follows:

```
FOR ( $i \in N$ ) {
    $k^* \leftarrow argmin_{k \in S^2} \{\min\{t_{0i} + \alpha t_{ik} - t_{k0}, \alpha t_{ki} + t_{i0} - t_{k0}\}\}$
    $L(k^*) \leftarrow L(k^*) + q_i$
}
```

Note that the associated load is equal to the load that would be assigned to this seed by the GAP if it did not have a capacity constraint. The final set S of seeds that will have a cardinality of exactly the requested number of seeds will contain the seeds with largest associated loads.

Allocation of vehicles. Given the locations of the seeds we have to decide which vehicle types, and thus how much capacity, will be allocated to each seed. To accomplish this we use the following approximation algorithm. Define:

S:	the set of seeds;
N:	the set of customers;
T:	the set of vehicle types (capacities), $T_1 < T_2 < \cdots < T_m$;
$D_1(k)$:	the set of customers that have k as closest seed with respect to the extra mileage metric;
$D_2(k)$:	the set of customers that have k as second closest seed with respect to the extra mileage metric;
$t(k)$:	index of the vehicle type allocated to seed k;
$L(k)$:	the load associated with seed k.

The objective for the allocation can now be stated as

$$\sum_{j \in N} q_j \leqslant \sum_{k \in S} T_{t(k)} \leqslant (1+\gamma) \sum_{j \in N} q_j.$$

The goal is thus to find an allocation that has sufficient capacity to accommodate all the load but in addition one that does not have to much spare capacity. A first step in achieving this goal is to associate a load with each seed as follows:

$$L(k) := (1 - \delta) \sum_{j \in D_1(k)} q_j + \delta \sum_{j \in D_2(k)} q_j.$$

In case we take δ equal to zero, we associate with seed k all the customers that have k as their closest seed. The parameter δ is introduced to take account of the fact that it might not be possible to find a vehicle allocation in which all customers are allocated to the closest seed. In that case some customers have to be allocated to another seed.

To prepare the initial allocation we define for each vehicle type a region of attraction:

$$\begin{array}{ll} [0, T_1 + \epsilon(T_2 - T_1)) & \text{for type } 1, \\ [T_l - (1-\epsilon)(T_l - T_{l-1}), T_l + \epsilon(T_{l+1} - T_l)) & \text{for type } l \ (l = 2,\ldots,m-1), \\ [T_m - (1-\epsilon)(T_m - T_{m-1}), \infty) & \text{for type } m. \end{array}$$

This region of attraction is introduced to prevent the system from deciding too soon to allocate large vehicles to seeds. Next, we allocate to each seed the vehicle type with the region of attraction that contains its associated load. Unless we take ϵ equal to zero, we are not sure whether the current allocation has sufficient capacity to accommodate all the load. Two mechanisms are applied to manipulate the current vehicle allocation.

In case $\Sigma_{k \in S} T_{t(k)} < \Sigma_{j \in N} q_j$, we try to identify a seed for which we will increase the capacity as follows. We determine the seeds that are not yet at maximum capacity and that have insufficient capacity to accommodate their associated load. Among these we select the one that results in the minimum additional spare capacity if we switch to the next larger vehicle type:

```
$k^* \leftarrow argmin_{k \in S}\{T_{t(k)+1} - L(k) \mid L(k) > T_{t(k)} \wedge t(k) < m\}$
$IF\ (\ k^*\ EXISTS\ )$
    $t(k^*) \leftarrow t(k^*) + 1$
```

If this fails, there is at least one seed at maximum capacity with an associated load that exceeds this maximum capacity. In that case, we increase the capacity of the seed for which this is possible and that is closest in distance:

```
$l^* \leftarrow argmax_{l \in S}\{L(l) - T_{t(l)} \mid L(l) > T_{t(l)} \wedge t(l) = m\}$
$k^* \leftarrow \min_{k \in S}\{d(k,l^*) \mid t(k) < m\}$
$t(k^*) \leftarrow t(k^*) + 1$
```

In case $\Sigma_{k \in S} T_{t(k)} > \gamma \Sigma_{j \in N} q_j$, we try to decrease the capacity of one of the seeds that are not yet at maximum capacity in such a way that the load that has to be redistributed is minimum and the total allocated capacity is still sufficient:

```
$k^* \leftarrow argmin_{k \in S}\{L(k) - T_{t(k)-1}$
                 $\mid t(k) > 1 \wedge \sum_{l \in S} T_{t(l)} - (T_{t(k)} - T_{t(k)-1}) > \sum_{j \in N} q_j\}$
$t(k^*) \leftarrow t(k^*) - 1$
```

5.3. *Route construction*

When the clusters have been formed, a route has to be constructed for each of them. This amounts to solving a traveling salesman problem with side constraints. An insertion algorithm, as described Section 4.2, is used to accomplish this task.

At this point, it is appropriate to analyze, in some detail, the insertion of a yet unrouted or *free* customer into a route. This basic action contains all the ingredients needed later in more sophisticated functions. The analysis is split in two parts: the first deals with the *feasibility* of an insertion, the second deals with its *profitability*. Let $(0,1,...,n+1)$ be the considered route, u the unrouted customer to be inserted, and i and $i+1$ the customers between which u is being inserted. Figure 1 illustrates this insertion.

To establish the feasibility of an insertion, we have to test the side constraints. We will consider time window and capacity constraints. (Note that even if after the insertion both the total load to be delivered and the total load to be collected do not exceed the vehicle capacity, it is still possible that the ordering of the customers leads to a violation of the capacity constraints.) The insertion of u between i and $i+1$ has generally two effects. First, it can affect all the arrival times at vertices $i+1, i+2, ..., n+1$, which may result in an infeasible tour. Secondly, it affects either the vehicle load when visiting the vertices $0,1,...,i$, in case u is a delivery, or the vehicle load when visiting the vertices $i+1, i+2, ..., n+1$, in case u is a collection.

To test the feasibility of an insertion with respect to the *time window constraints* efficiently, we use the quantity $S^+_{i+1,n+1}$ which expresses the *possible forward shift* in time of the departure time at $i+1$ causing no violation of the time window

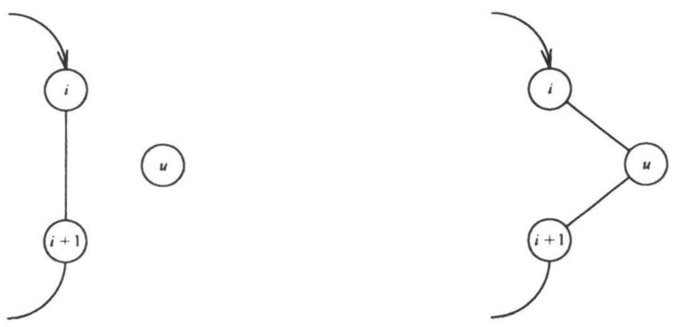


Figure 1. The insertion of u between i and $i+1$.

constraints along the path $(i+1,...,n+1)$. The feasibility test of an insertion then amounts to

$$\max\{D_i + t_{i,u}, e_u\} + t_{u,i+1} - D_{i+1} \leqslant S^+_{i+1,n+1}.$$

The following backward recursion will compute $S^+_{k,n+1}$ for all customers k in $O(n)$ time:

$$S^+_{n+1,n+1} \leftarrow l_{n+1} - D_{n+1};$$

$$S^+_{k,n+1} \leftarrow \min\{S^+_{k+1,n+1}, l_k - D_k\} + W_{k+1} \qquad \text{for } k = n,..., 1.$$

To test the feasibility of an insertion with respect to the *capacity constraints* efficiently, we introduce the following quantities. Let Γ be the set of customers where the salesman has to make a collection and Δ the set of customers where the salesman has to make a delivery. We define

$$C_k := Q - \textstyle\sum_{j>k,\, j\in\Delta} q_j - \sum_{j\leqslant k,\, j\in\Gamma} q_j,$$

which is the remaining capacity in the vehicle at the departure at vertex k,

$$L^-_k := \min_{j\leqslant k}\{C_j\},$$

which is the *maximum delivery increase* the vehicle can accommodate on the path $(0,...,k)$, and

$$L^+_k := \min_{j\geqslant k}\{C_j\},$$

which is the *maximum collection increase* the vehicle can accommodate on the path $(k,...,n+1)$. Note that the formula given for the remaining capacity is based on the fact that in the VRPTW all deliveries have to be collected at the depot and all collections have to be delivered at the depot. The feasibility of an insertion can now be tested by

$$q_u \leqslant L^-_i$$

in case u is a delivery and

$$q_u \leqslant L^+_{i+1}$$

in case u is a collection. The values of C_k, L_k^-, L_k^+ can be computed for all customers k in $O(n^2)$ time as follows:

$$C_0 \leftarrow Q - \textstyle\sum_{j \in \Delta} q_j,$$

$$C_k \leftarrow \begin{cases} C_{k-1} - q_k \text{ if } k \in \Gamma \\ C_{k-1} + q_k \text{ if } k \in \Delta \end{cases} \qquad \text{for } k = 1,\ldots,n+1,$$

$$L_0^- \leftarrow Q - \textstyle\sum_{j \in \Delta} q_j,$$

$$L_k^- \leftarrow \min\{C_k, L_{k-1}^-\} \qquad \text{for } k = 1,\ldots,n+1,$$

$$L_{n+1}^+ \leftarrow C_n,$$

$$L_k^+ \leftarrow \min\{C_k, L_{k+1}^+\} \qquad \text{for } k = n,\ldots,0.$$

To establish the profitability of an insertion, we have to compute the insertion criterion ι and the selection criterion σ. The insertion criterion ι determines the place where a customer will be inserted in the emerging route and the selection criterion σ serves as a guideline to choose between the vertices available for insertion. Therefore, in trying to achieve our primary goal, creating a feasible tour, we have to rely on the first criterion.

Which criteria to use strongly depends on the tightness of the time windows involved. If the time window are relatively wide, the spatial aspect is more important, but if the time windows are quite tight, the temporal aspect becomes dominant. Therefore, we introduce two phases: first the vertices with tight time windows are inserted, and next the vertices with wide time windows. (The definition of tight and wide can be set according to the user's preferences.)

Let the *extra travel time* of customer u with respect to the link $(i,i+1)$ be defined by:

$$E(u,i) := \max\{D_i + t_{i,u}, e_u\} + t_{u,i+1} - A_{i+1}.$$

In phase 1, where the customers with tight time windows are routed, the temporal aspect is dominant. The criteria to be used are

$$\iota(u,i) = \min\{l_u - \max\{D_i + t_{i,u}, e_u\}, S_{i+1,n+1}^+ - E(u,i)\},$$

$$\sigma(u,i_u) = E(u,i_u).$$

The first criterion is guided by the remaining flexibility of the route under construction with respect to the time windows whereas the second criterion searches for the vertex whose inclusion will lead to the smallest increase in travel time of the tour. In phase 2, where the vertices with large time windows are routed, feasibility problems play a minor role and we can concentrate on the spatial aspect. Therefore the criteria are interchanged to obtain:

$$\iota(u,i) = E(u,i),$$

$$\sigma(u,i_u) = \min\{l_u - \max\{D_u + t_{i_u,u}, e_u\}, S_{i_u,n+1}^+ - E(u,i_u)\}.$$

As ι and σ can be computed in constant time, the complexity of the insertion scheme is $O(n^2)$, even in the presence of time window and capacity constraints. There are n insertions, each taking $O(n)$ time, because the $O(n)$ possible insertion places can be tested in constant time and an actual insertion gives rise to an $O(n)$ update.

6. Approximation: iterative improvement

Iterative improvement procedures are based on what is perhaps the oldest optimization principle: neighborhood search. It is a simple and natural idea, which has proven to be surprisingly successful on a variety of problems. The general iterative improvement procedure proceeds as follows. We start at some initial feasible solution and search in its neighborhood for a better (cheaper) one. As long as an improved solution exists, we adopt it and repeat the neighborhood search from the new solution. Finally, we will reach a local optimum and stop.

To apply this approach to a particular problem, we have to make a number of choices. We have to decide how to obtain an initial feasible solution, we have to define a neighborhood for the problem at hand, and we have to choose a method for searching it.

As to obtaining an initial feasible solution in vehicle routing problems, we refer the reader to the previous section on construction methods. The most often used neighborhood for vehicle routing problems is the k-exchange neighborhood. A k-exchange is a substitution of k arcs of a route with k others. A route is said to be k-optimal if it is impossible to obtain a shorter route by replacing k of its arcs by another set of k arcs. The number of possible k-exchanges in a given route is $O(n^k)$. The computational requirement of k-exchange procedures thus increases rapidly with k, and one usually only considers the cases $k = 2$ and $k = 3$. The choice of an appropriate search strategy for this neighborhood will turn out to be of crucial importance.

Croes [1958] and Lin [1965] introduced the notion of k-exchanges to improve solutions to the TSP. Lin and Kernighan [1973] generalized the approach, and many authors reported on its application to related problems. In the context of vehicle routing, Christofides and Eilon [1969] and Russell [1977] adapted the approach to the basic VRP, and Psaraftis [1983] used it for the DARP.

In this section, we will show how various side constraints can be handled in the k-exchange methods without increasing the time complexity.

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

6.1. *The traveling salesman problem*

A 2-exchange replaces two arcs $(i,i+1)$ and $(j,j+1)$ by (i,j) and $(i+1,j+1)$, thereby reversing the path $(i+1,...,j)$ (see Figure 2). Such an exchange results in a local improvement if and only if

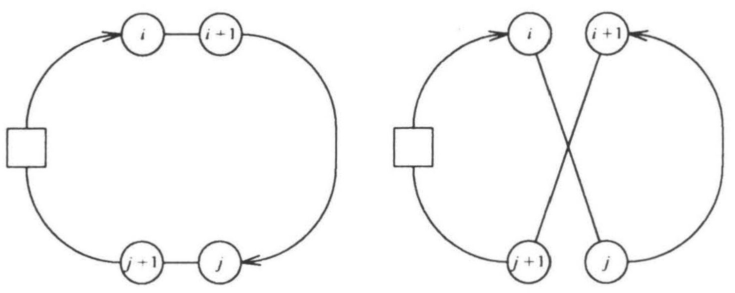


Figure 2. A 2-exchange.

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

In a 3-exchange, three arcs are deleted and there are seven possibilities to construct a new route from the remaining segments. Figure 3 shows two possible 3-exchanges that can be performed by deleting the arcs $(i,i+1)$, $(j,j+1)$ and $(k,k+1)$ of a route.

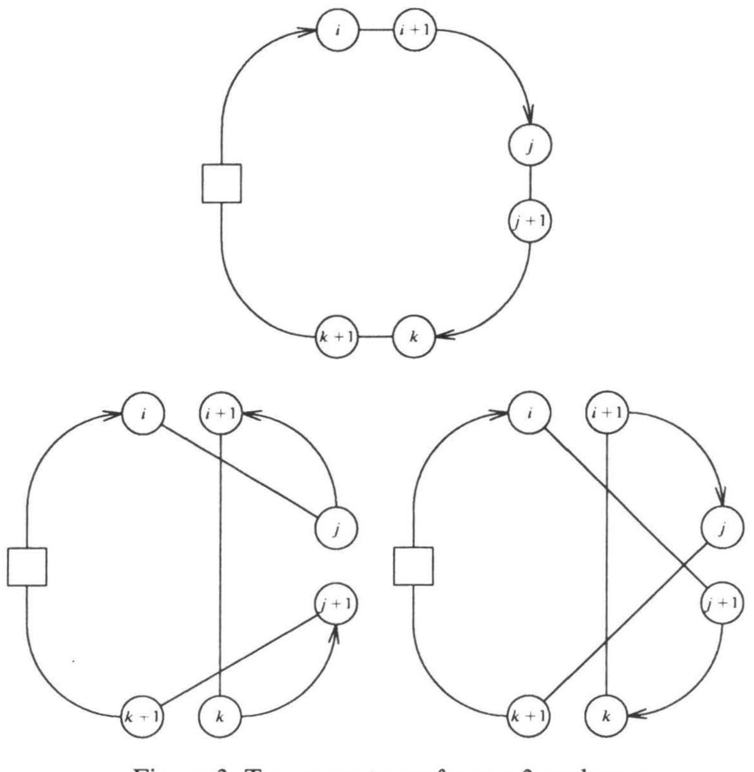


Figure 3. Two ways to perform a 3-exchange.

For all cases conditions similar to the one given for the case $k = 2$ can be given to obtain local improvement. There is one important difference between the two 3-exchanges shown above: in the latter the orientation of the paths $(i+1,...,j)$ and $(j+1,...,k)$ is preserved whereas in the former this orientation is reversed.

Because the computational requirement to verify 3-optimality becomes prohibitive if the number of vertices increases, proposals have been made to take only a subset of all possible 3-exchanges into account. Or [1976] proposes to restrict attention to those 3-exchanges in which a string of one, two or three consecutive vertices is relocated between two others. To see how the *Or-opt* procedure works, the reader is referred to Figure 4. In this route the path $(i_1,...,i_2)$ 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.

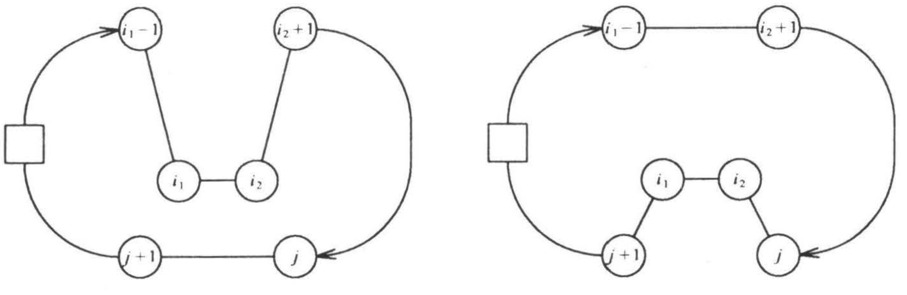


Figure 4. An Or-exchange.

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 $(j>i_2)$ are handled separately.

6.2. *A lexicographic search strategy*

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

The basic idea of the proposed approach is the use of a search strategy and of a number of global variables such that, for each considered exchange, checking its feasibility and updating the global variables require 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, for which we want to prove optimality, is given by a sequence $(0,...,i,...,n+1)$, where i represents the ith vertex of the route, and that we are always examining the exchange that involves the substitution of arcs $(i,i+1)$ and $(j,j+1)$ with (i,j) and $(i+1,j+1)$ in case of a 2-exchange, and the substitution of (i_1-1,i_1), (i_2,i_2+1) and $(j,j+1)$ with (i_1-1,i_2+1), (j,i_1) and $(i_2,j+1)$ in case of an Or-exchange.

Lexicographic search for 2-exchanges. We choose the arcs $(i,i+1)$ 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 a arc $(i,i+1)$, we choose the arc $(j,j+1)$ to be $(i+2,i+3), (i+3,i+4),..., (n,n+1)$ in that order (see Figure 5); this will be referred to as the inner loop.

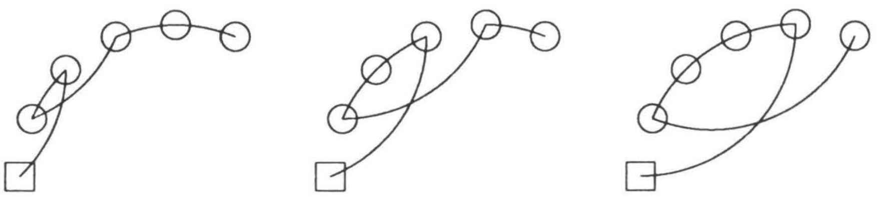


Figure 5. The lexicographic search strategy for 2-exchanges.

Now consider all possible exchanges for a fixed arc $(i,i+1)$. The ordering of the 2-exchanges given above implies that in the inner loop in each newly examined 2-exchange the path $(i+1,...,j-1)$ of the previously considered 2-exchange is expanded by the arc $(j-1,j)$. This observation, together with an appropriate set of global variables, makes it is possible to maintain information on the feasibility and duration of this path efficiently, i.e., to check its feasibility and to update the global variables in constant time.

Lexicographic search for backward Or-exchanges. We choose the path $(i_1,...,i_2)$ in the order of the current route starting with i_1 equal to 2. After the path $(i_1,...,i_2)$ is fixed, we choose the arc $(j,j+1)$ to be $(i_1-2,i_1-1), (i_1-3,i_1-2),..., (0,1)$ in that order. That is, the arc $(j,j+1)$ 'walks backward' through the route. Note that in the inner loop in each newly examined exchange the path $(j+2,...,i_1-1)$ of the previously considered exchange is expanded with the arc $(j+1,j+2)$.

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

6.3. *The traveling salesman problem with time windows*

2-Exchanges. In the following, a quantity with superscript 'new' indicates the value after the exchange has been carried out, and subscripts always refer to the ordering of the current tour. A 2-exchange is feasible and profitable if and only if the following conditions are satisfied:

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

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

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

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

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

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

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

Recall that in the lexicographic search strategy after the arc $(i,i+1)$ is fixed, the arc $(j,j+1)$ is chosen to be equal to $(i+1,i+2), (i+2,i+3),..., (n,n+1)$. This means that once we have fixed the arc $(i,i+1)$, we can completely specify an exchange by the other arc involved. In the following, an arc appearing as superscript will specify the exchange on which the information is based. To be able to check feasibility, we define three global variables:

S^+: possible forward shift in time of the departure time at $j-1$ causing no violation of the time window constraints along the path $(i+1,...,j-1)$:

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

W: waiting time on the path $(i+1,...,j)$, excluding possible waiting time at j, including possible waiting time at $i+1$:

$$W := \textstyle\sum_{i+1 \leqslant k \leqslant j-1} W_k^{(j,j+1)};$$

T: travel time, excluding the periods of waiting, on the path $(i+1,...,j)$:

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

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

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

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

LEMMA. *Expanding the path $(j-1,...,i+1)$ with the arc $(j-1,j)$ is feasible if and only if $S \leqslant S^+$.*

PROOF. If the tour that results if the exchange is carried out is feasible, we know that

$$D_j^{(j,j+1)} + \textstyle\sum_{k \leqslant p \leqslant j-1} t_{p,p+1} \leqslant l_k \quad \text{for } i+1 \leqslant k < j,$$

which implies that

$$D_j^{(j,j+1)} + t_{j,j-1} \leqslant l_k - \sum_{k \leqslant p \leqslant j-2} t_{p,p+1} \quad \text{for } i+1 \leqslant k < j,$$

which just says that $S \leqslant S^+$.

To prove the converse, note that $D_k^{(j,j+1)} \geqslant D_k^{(j-1,j)}$ for $i+1 \leqslant k \leqslant j-1$. The only vertices for which infeasibility can occur are those for which $D_k^{(j,j+1)} \neq D_k^{(j-1,j)}$. A necessary condition for this to occur is that there is no waiting time on the path $(j,...,k)$ after the exchange is carried out. Suppose now that $S \leqslant S^+$. This implies that

$$D_j^{(j,j+1)} + t_{j,j-1} - D_{j-1}^{(j-1,j)} \leqslant l_k - (D_{j-1}^{(j-1,j)} + \sum_{k \leqslant p \leqslant j-2} t_{p,p+1}),$$

so that

$$D_j^{(j,j+1)} + \sum_{k \leqslant p \leqslant j-1} t_{p,p+1} \leqslant l_k$$

and (since we may assume that there is no waiting time)

$$D_j^{(j,j+1)} + \sum_{k \leqslant p \leqslant j-1} t_{p,p+1} + \sum_{k \leqslant p \leqslant j-1} W_k^{(j,j+1)} \leqslant l_k,$$

which is equivalent to

$$A_k^{(j,j+1)} \leqslant l_k. \ \square$$

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

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

Now that we have shown that testing a single 2-exchange takes constant time, what remains to be done is to show that the global variables can also be updated in constant time.

An examination of the definition of S shows that it covers two different cases (Figure 6). In the case that $S < 0$, the triangle inequality guarantees that the new arrival at $j-1$ is never earlier than the old arrival, so it must have been the case that the old arrival and old departure did not coincide. This means that the old departure was equal to the opening of the time window. But then $|S|$ is exactly equal to the waiting time at $j-1$. In the case that $S \geqslant 0$, S is exactly equal to the difference between the new arrival time and the old arrival time at $j-1$, that is, the forward shift in time.

We now assert that the global variables can be updated in constant time as follows:

$$T \leftarrow T + t_{j,j-1};$$

$$W \leftarrow \max\{W - S, 0\};$$

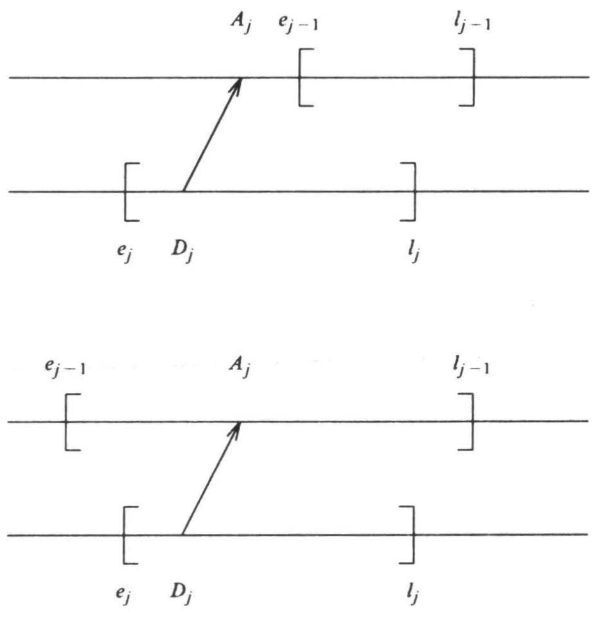


Figure 6. Schematic presentation of the possible shifts.

$$S^+ \leftarrow \min\{l_j - D_j^{(j,j+1)}, S^+ - S\}.$$

It is easily verified that the transformations for W and T are correct. The correctness of the transformation for S^+ can be proved as follows. Define F_k as the maximal forward shift in time of the departure time at j causing no violation of the time window constraints at k:

$$F_k^{(j,j+1)} := l_k - (D_j^{(j,j+1)} + \textstyle\sum_{k \leqslant p \leqslant j-1} t_{p,p+1}).$$

We have

$$\begin{aligned} F_k^{(j,j+1)} &= l_k - (D_{j-1}^{(j,j+1)} + \textstyle\sum_{k \leqslant p \leqslant j-1} t_{p,p+1}) \\ &= l_k - (D_j^{(j-1,j)} + \textstyle\sum_{k \leqslant p \leqslant j-2} t_{p,p+1}) - D_j^{(j,j+1)} - t_{j,j-1} + D_{j-1}^{(j-1,j)} \\ &= F_k^{(j-1,j)} - S. \end{aligned}$$

But this implies

$$\begin{aligned} S^+ &= \min\nolimits_{i+1 \leqslant k \leqslant j}\{F_k^{(j,j+1)}\} \\ &= \min\{l_j - D_j^{(j,j+1)}, \min\nolimits_{i+1 \leqslant k \leqslant j-1}\{F_k^{(j,j+1)}\}\} \\ &= \min\{l_j - D_j^{(j,j+1)}, \min\nolimits_{i+1 \leqslant k \leqslant j-1}\{F_k^{(j-1,j)} - S\}\} \\ &= \min\{l_j - D_j^{(j,j+1)}, S^+ - S\}. \end{aligned}$$

It is easy to see that the complexity for each individual 2-exchange is reduced to constant time because the necessary tests for feasibility and local improvement plus the updating of all quantities involved require constant time. This gives an overall time complexity of $O(n^2)$ for the verification of 2-optimality.

Or-exchanges. For presentational convenience, we will present our method only for those Or-exchanges in which a single customer i is relocated. This implies that $i_1 = i_2 = i$. Because the concepts presented in this part differ only slightly from those described for the 2-exchanges we will take a more intuitive and informal approach. Note that the orientation of the path $(j+1,...,i-1)$ is preserved, which makes it easier to handle the feasibility checks. The global variables we need are:

S^+: possible forward shift, which is equal to $S^+_{j+1,i-1}$ as defined earlier;

S^-: possible backward shift, which is equal to $S^-_{i+1,j-1}$ as defined earlier;

G: gain made by going directly from $i-1$ to $i+1$:

$$G := A_{i+1} - (D_{i-1} + t_{i-1,i+1});$$

L: loss incurred by going from j through i to $j+1$:

$$L := \max\{D_j + t_{ji}, e_i\} + t_{i,j+1} - A_{j+1};$$

W: waiting time on the path $(j+1,...,i-1)$:

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

During the backward search, an exchange is feasible if $D_k^{\text{new}} \leqslant l_k$ for $k = j+1,...,i-1$, and potentially profitable if $D_{i+1}^{\text{new}} < D_{i+1}$. In terms of global variables, feasibility and potential profitability are equivalent to

$$L < \min\{S^+, G + W\}.$$

The global variables are updated by

$$S^+ \leftarrow \min\{l_{j+1} - D_{j+1}, S^+\} + W_{j+1};$$

$$W \leftarrow W + W_{j+1}.$$

During the forward search, an exchange feasible if $D_i^{\text{new}} \leqslant l_i$ and potentially profitable if $D_{j+1}^{\text{new}} < D_{j+1}$. This is equivalent to

$$L < \min\{S^-, G\}.$$

The only update is

$$S^- \leftarrow \min\{D_j - e_j, S^-\}.$$

It follows that a single exchange of this type can be handled in constant time. The adaptation to the relocation of a string of vertices instead of a single vertex is conceptually similar but technically a little more complicated.

6.4. *The traveling salesman problem with multiple time windows*

In the previous section, we have shown that k-exchange procedures can be adapted to handle a single time window at each vertex of a route without increasing the time complexity. A natural extension is to look at iterative improvement algorithms for problems where each vertex can have more than one time window. For instance, in practical distribution problems it often occurs that shops can only be served either in the morning or in the afternoon, but not during the lunch hours.

Although the techniques discussed in this section can be applied to the general case where the number of time windows at a vertex is bounded by a constant, we restrict ourselves, for presentational convenience, to the case where each vertex k has two time windows, denoted by $[e_k^1, l_k^1]$ and $[e_k^2, l_k^2]$.

In the presence of multiple time windows, we are no longer able to maintain the $O(n^2)$ time complexity for verifying *2-optimality*. However, the time complexity we achieve, $O(n^2 \log n)$, is still better than the straightforward implementation that requires $O(n^3)$ time. For the *Or-exchanges* we do better. The computational requirement to verify Or-optimality remains $O(n^2)$.

To illustrate certain aspects of the proposed algorithms, we will often resort to pictorial representations. Because of their importance, they will be explained in detail here. A pictorial representation (see Figure 7) will contain information on two consecutive vertices. For each vertex there will be a time axis with the two time windows, the arrival time, and the departure time. A time window is represented by a pair of square brackets, the first indicating the earliest service time, the second indicating the latest service time. Arrival and departure times are represented by a dot. In case the arrival and departure time coincide (we assumed there is no actual service time), there is only one dot. Otherwise, the dot associated with the departure time will coincide with the earliest service time within one of the two time windows, and the difference between the two dots represents the waiting time. In addition, an arrow will represent the travel time between the two vertices. We will draw the time axis associated with the vertex with the earliest departure time at the bottom.

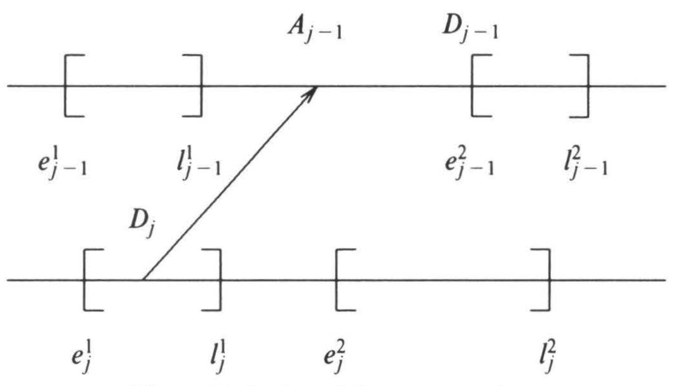


Figure 7. A pictorial representation.

2-Exchanges. Let us briefly review the variables needed in the single window case:

S^+: possible forward shift in time of the departure at $j-1$ causing no violation of the time window constraints along the path $(i+1,\ldots,j-1)$;
W: waiting time on the path $(i+1,\ldots,j)$ (excluding possible waiting time at j, including possible waiting time at $i+1$);
T: travel time, excluding periods of waiting, on the path $(i+1,\ldots,j)$;
S: change in departure time at $j-1$ when the arc $(j-1,j)$ is added to the path $(i+1,\ldots,j-1)$.

In each iteration the global variables were updated using the following formulas:

$$T \leftarrow T + t_{i,j},$$
$$S^+ \leftarrow \min\{S^+ - S, l_j - D_j\},$$
$$W \leftarrow \max\{0, W - S\}.$$

It is obvious that in the multiple window case, the first is still valid, but the other two might no longer be valid.

Let us consider the possible forward shift. As infeasibility only occurs when departure is later than the closing of the last time window, a first idea might be to control the feasibility by only considering the latest service time of the last time window and thus replacing l_j by l_j^2. To show that this does not suffice, suppose that at some point

$$D_j < l_j^1$$

and

$$l_j^1 - D_j < S^+ - S < e_j^2 - D_j.$$

In that case, an update would set S^+ to $S^+ - S$ because $S^+ - S < l_j^2 - D_j$, whereas it should be equal to $l_j^1 - D_j$. It is obvious that this can easily be fixed by using a slightly more sophisticated updating mechanism:

$$S^+ \leftarrow \begin{cases} l_j^1 - D_j & \text{if } l_j^1 - D_j < S^+ - S < e_j^2 - D_j, \\ \min\{S^+ - S, l_j^2 - D_j\} & \text{otherwise.} \end{cases}$$

Let us consider the waiting time. In the single window case, we were able to keep track of the waiting time on the path $(i+1,\ldots,j-1)$ using only local information obtained when an arc $(j-1,j)$ was added to the path $(i+1,\ldots,j-1)$. To show that this no longer suffices in the multiple window case, suppose that at some point and for some k on the path $(i+1,\ldots,j-2)$

$$l_k^1 < D_k + S - \sum_{k \leqslant p \leqslant j-1} W_p < e_k^2.$$

The quantity $S - \Sigma_{k \leqslant p \leqslant j-1} W_p$ is the shift in departure time at k, i.e., the shift in departure time at $j-1$ minus the consumed waiting time along the path $(j-1,\ldots,k)$. It is not hard to see that in this case S induces a waiting time at k equal to

$$e_k^2 - (D_k + S - \sum_{k \leqslant p \leqslant j-1} W_p).$$

Therefore it is impossible, in the multiple window case, to restrict attention to waiting time that occurs at vertex $j-1$ when the path $(i+1,...,j-1)$ of the previously considered exchange is extended with the arc $(j-1,j)$. The waiting time that might occur anywhere along the path $(i+1,...,j-2)$, has to be taken into account as well. The global nature of the waiting time makes it very hard to control. However, the lexicographic search strategy enables us to maintain a set of intervals that can be used to calculate any waiting time along the path $(i+1,...,j-2)$ based on the value of S.

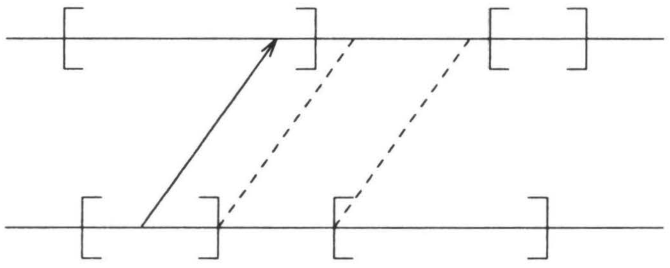
The set of intervals, denoted by $\{(i_1^l,i_1^u),...,(i_m^l,i_m^u)\}$, will have two properties:
- they are all disjoint;
- if $i_k^l < S < i_k^u$ for some k, then the waiting time on the path $(i+1,...,j-2)$ induced by this value of S is equal to $i_k^u - S + c_k$, where c_k is a constant associated with the interval.

What happens is basically the following. When the path $(i+1,...,j-1)$ is extended with the arc $(j-1,j)$ and $D_j < l_j^1$, the interval $(l_j^1 - D_j,\ e_j^2 - D_j)$ is added to the current set of intervals. The logic behind this is that when the departure time at vertex j is, at some time, shifted with an amount that falls inside this interval it will induce waiting time.

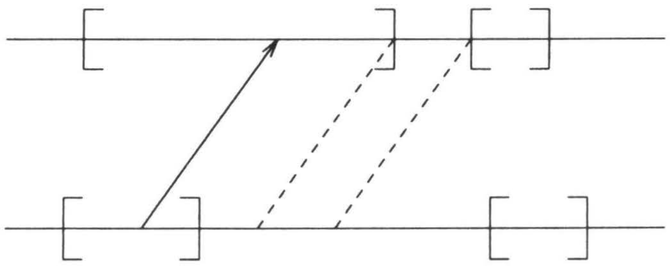
There are five basic cases that have to be considered when an interval $(i_{\text{new}}^l i_{\text{new}}^u)$ is added to the current set of intervals. Composite cases can all be handled as a sequence of the five basic ones.

Case 1. $k:(i_{\text{new}}^l,i_{\text{new}}^u) \cap (i_k^l,i_k^u) = \varnothing$. This is the simplest case. The new interval is added with $c_{\text{new}} = 0$.

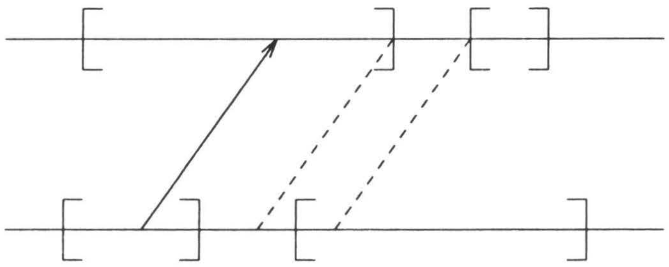
Case 2. $k:(i_{\text{new}}^l,i_{\text{new}}^u) \cap (i_k^l,i_k^u) = (i_{\text{new}}^l,i_{\text{new}}^u)$. The waiting time induced when S falls inside the new interval is completely dominated by the waiting time induced by the interval (i_k^l,i_k^u), because $i_k^u - S$ is larger than $i_{\text{new}}^u - S$. Therefore, the set of intervals is not changed.



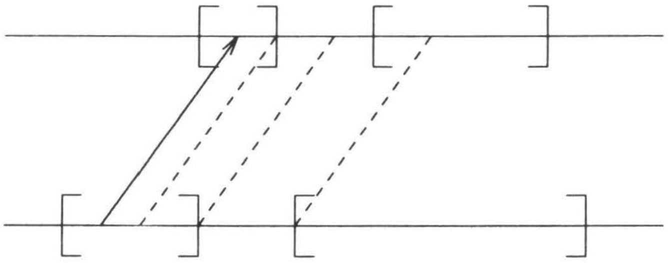
Case 3. $k:(i_{\text{new}}^l,i_{\text{new}}^u) \cap (i_k^l,i_k^u) = (i_k^l,i_k^u)$. Here, the situation is opposite to the previous case. The waiting time induced when S falls inside the new interval completely dominates the waiting time induced by (i_k^l,i_k^u). Therefore, the interval (i_k^l,i_k^u) is replaced by $(i_{\text{new}}^l,i_{\text{new}}^u)$ with $c_{\text{new}} = 0$.



Case 4. $k : (i^l_{\text{new}}, i^u_{\text{new}}) \cap (i^l_k, i^u_k) = (i^l_k, i^u_{\text{new}})$. Here, the situation is a bit more complicated. At first glance, there is only partial dominance. In fact, a kind of chaining occurs. The waiting time induced when S falls inside the new interval is $i^u_k - S$. Therefore, the interval (i^l_k, i^u_k) is replaced by $(i^l_{\text{new}}, i^u_k)$ with the associated constant equal to zero.



Case 5. $k : (i^l_{\text{new}}, i^u_{\text{new}}) \cap (i^l_k, i^u_k) = (i^l_{\text{new}}, i^u_k)$. Here, there really is partial dominance. When $i^l_k \leqslant S \leqslant i^l_{\text{new}}$, it will still induce a waiting time equal to $i^u_k - S$, but when $i^l_{\text{new}} \leqslant S \leqslant i^u_k$, it will induce a waiting time equal to $i^u_{\text{new}} - S$ instead of $i^u_k - S$. Therefore, the interval (i^l_k, i^u_k) is replaced by $(i^l_k, i^l_{\text{new}})$ with the associated constant equal to $i^u_k - i^l_{\text{new}}$, and a new interval $(i^l_{\text{new}}, i^u_{\text{new}})$ is added with associated constant equal to zero.



The battle is now nearly won. One small problem remains to be solved. An interval $(l^1_j - D_j, e^1_j - D_j)$ is created on the basis of the departure time. As soon as waiting time occurs at vertex $j-1$, it has to be absorbed in all the intervals. This means that both the lower and the upper ends of the intervals have to be increased with an amount equal to this waiting time.

To analyze the complexity of the 2-exchange procedure for the TSP with multiple time windows, let us drop the assumption that there are at most two time windows at each vertex. Let us assume instead that there are at most k time windows at each vertex, for a fixed k. Now, when the path $(i+1,...,j-1)$ is expanded with the arc $(j-1,j)$, there are at most $k-1$ intervals that have to be compared with the current set of intervals. The worst that can happen is that each interval leads to the creation of a new interval (Case 1 or Case 5), and the cardinality of the current set of intervals increases by exactly $k-1$. Overall this leads to a worst case of $O(kn)$ intervals. With data structures like balanced trees, it is possible to determine the waiting time and perform an update of the current set of intervals in $O(\log kn)$ time. This leads to an overall worst case time complexity for testing 2-optimality of $O((kn)^2 \log kn)$.

Or-exchanges. The introduction of multiple windows at vertices does not lead to a worse time complexity for the Or-exchanges. We encounter the same type of problems, but controlling them is easier because there are no path reversals.

Let us, for the sake of completeness, review the global variables used in the single window case:

S^+: possible forward shift in time of the departure time at $j+1$ causing no violations of the time window constraints on the path $(j+1,...,i-1)$;

S^-: possible backward shift in time at vertex $i+1$ causing no additional waiting time on the path $(i+1,...,j)$;

G: gain made by going directly from $i-1$ to $i+1$:

$$G := A_{i+1} - (D_{i-1} + t_{i-1,i+1});$$

L: loss L incurred by going from j through i to $j+1$:

$$L := \max\{D_j + t_{ji}, e_i\} + t_{i,j+1} - A_{j+1};$$

W: waiting time on the path $(j+1,...,i-1)$:

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

During the backward search (in the single window case) an exchange is feasible and profitable if

$$L < \min\{S^+, G+W\},$$

and the variables are updated by

$$S^+ \leftarrow \min\{l_{j+1} - D_{j+1}, S^+\} + W_{j+1};$$
$$W \leftarrow W + W_{j+1}.$$

As in the case of the 2-exchanges, we have to modify the updates for the possible forward shift and the waiting time. In the 2-exchanges, the waiting time on the path $(i+1,...,j-2)$ created a problem because of its global nature. Here, because the ordering on the path $(j+1,...,i-1)$ remains the same, the waiting time can be controlled using only local information.

To see this, let us first take a closer look at the updates that have to be performed when the path $(j+1,...,i-1)$ is expanded with the link $(j+1,j+2)$. The test for feasibility and profitability is

$$L < \min\{S^+, G + W\}.$$

Now, the same test in the next iteration, when expressed in the current quantities, looks as follows:

$$L < \min\{(\min\{l_{j+1} - D_{j+1}, S^+\} + W_{j+1}), G + (W + W_{j+1})\}.$$

We can rewrite this as

$$L < \min\{\min\{l_{j+1} - D_{j+1}, S^+\}, G + W\} + W_{j+1}$$

or

$$L < \min\{l_{j+1} - D_{j+1}, \min\{S^+, G + W\}\} + W_{j+1}.$$

This reveals the fact that in actual implementations we will not use the two global variable S^+ and W, which is conceptually simpler, but just one. We call this one global variable the *possible profit* P, which is equal to $\min\{S^+, G + W\}$. Whereas up to now, initialization has been trivial, here it is different. P is initialized by

$$P \leftarrow \min\{l_{i-1} - D_{i-1}, A_{i+1} - (D_{i-1} + t_{i-1,i+1})\} + W_{i-1},$$

and updated by

$$P \leftarrow \min\{l_{j+1} - D_{j+1}, P\} + W_{j+1}.$$

Similarly to what we have seen with the 2-exchanges, we have to modify this slightly in the presence of multiple windows to

$$P \leftarrow \begin{cases} l^1_{j+1} - D_{j+1} & \text{if } l^1_{j+1} - D_{j+1} < P < e^2_j - D_j, \\ \min\{P, l^2_{j+1} - D_{j+1}\} & \text{otherwise.} \end{cases}$$

During the forward search (in the single window case) an exchange is feasible and profitable if

$$L < \min\{S^-, G\}$$

and the possible backward shift is updated by

$$S^- \leftarrow \min\{D_j - e_j, S^-\}.$$

In the multiple window case we have to be more careful. When $e^2_j < D_j < l^2_j$, the two intervals $(D_j - e^1_j, D_j - l^1_j)$ and $(D_j - e^2_j, D_j)$ will not lead to waiting time if L falls in one of them, whereas the interval $(D_j - l^1_j, D_j - e^2_j)$ will lead to waiting time if L falls in it. Fortunately, we do not have to maintain a set of intervals, like we had to do for the 2-exchanges, because we can use the gain G to choose the appropriate interval at once. If $D_j - G > l^1_j$, the update of S^- is given by $D_j - e^2_j$. If $D_j - G \leqslant l^1_j$, then it is given by $D_j - e^1_j$.

6.5. *The traveling salesman problem with mixed collections and deliveries*

The following quantities will be helpful for the description of the algorithm. Given a feasible tour $(0, 1, ..., n, n+1)$, we define

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

A tour is feasible if and only if

$$0 \leqslant C(0,k) - D(0,k) \leqslant Q \qquad \text{for } k = 0,\dots,n+1.$$

An important variant of the TSP with mixed collections and deliveries arises when it is required that all load to be delivered has to be collected at vertex 0, and all load to be collected has to be delivered at vertex $n+1$. In that case a route is feasible if and only if

$$C(0,k) - D(0,k) \leqslant Q \qquad \text{for } k = 0,\dots,n+1.$$

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

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

The exchange is feasible if and only if

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

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

If we introduce global variables for $C(0,i) - D(0,i)$, $\min_{k \in \{i+1,\dots,j\}} \{C(k,j) - D(k,j)\}$ and $\max_{k \in \{i+1,\dots,j\}} \{C(k,j) - D(k,j)\}$, checking the feasibility of an exchange reduces to two additions and two comparisons, which take constant time. The lexicographic search strategy allows us to maintain the global variables efficiently, i.e., to update them for each new value of j in constant time.

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

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

The exchange is feasible if and only if

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

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

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

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

We rewrite this as

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

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

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

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

We have now accomplished our goal: we can introduce global variables for $D(i_1,i_2) - C(i_1,i_2)$, $\min_{k \in \{i_1,\ldots,i_2\}}\{C(0,k) - D(0,k)\}$ and $\max_{k \in \{i_1,\ldots,i_2\}}\{C(0,k) - D(0,k)\}$ that can be maintained efficiently in the outer loop and global variables for $D(j+1,i_1-1) - C(j+1,i_1-1)$, $\min_{k \in \{j+1,\ldots,i_1-1\}}\{C(0,k) - D(0,k)\}$ and $\max_{k \in \{j+1,\ldots,i_1-1\}}\{C(0,k) - D(0,k)\}$ that can be maintained efficiently in the inner loop.

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

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

The exchange is feasible if and only if

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

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

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

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

We rewrite this as

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

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

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

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

As in the backward case, we can now easily define global variables that can be efficiently maintained in both loops.

6.6. *The traveling salesman problem with precedence constraints*

The *single-vehicle dial-a-ride problem*, where a single vehicle has to pickup and deliver n customers, is an example of the TSP with precedence constraints. Each customer has a pickup and delivery location and the pickup must precede the delivery. Psaraftis [1983] shows that the k-exchange improvement methods can be modified to handle these restrictions. By a straightforward choice of the set of global variables, the lexicographic search strategy produces the same result.

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

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

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

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

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

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

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

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

- If $prec(j) > 0$, then $mark(prec(j)) \leftarrow 1$.

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

- If $prec(i) > 0$, then $mark(prec(i)) \leftarrow 0$.

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

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

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

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

Whenever we try to expand the path $(j+1,...,i_1-1)$ with the arc $(j,j+1)$ and *mark* $(j)=1$, its successor is on the path $(i_1,...,i_2)$, thus implying that expansion will only result in infeasible exchanges. For the backward Or-exchanges the actual marking and resetting can both be controlled in the outer loop:

- If $mark(i_1)<0$, then $mark(|prec(i_1)|) \leftarrow 1$.
- If $mark(i_2+1)<0$, then $mark(|prec(i_2+1)|) \leftarrow 0$.

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

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

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

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

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

- If $mark(i_2)>0$, then $mark(prec(i_2)) \leftarrow 1$.
- If $mark(i_1-1)>0$, then $mark(prec(i_1-1)) \leftarrow 0$.

6.7. *The traveling salesman problem with fixed paths*

In many applications of the *k*-exchange improvement algorithms in vehicle routing, and especially in interactive planning situations, it is useful to be able to specify parts of the tour that may not be separated. One way of doing this is attaching a label to each vertex in the following way:

$$link(v) := \begin{cases} 1 \text{ if vertex } v \text{ may not be separated from its predecessor,} \\ 0 \text{ otherwise.} \end{cases}$$

If we try to modify the *k*-exchange methods to handle the fixed path restrictions, we find ourselves in the unique situation where we do not need global variables at all. Feasibility can be checked as follows.

2-Exchange. A 2-exchange is feasible if and only if

$$link(i+1) = 0 \wedge link(j+1) = 0.$$

Or-exchange. An Or-exchange, backward and forward, is feasible if and only if

$$link(i_1) = 0 \wedge link(i_2+1) = 0 \wedge link(j+1) = 0.$$

6.8. *The vehicle routing problem*

After showing how various side constraints can be handled in iterative improvement methods for the TSP, we now turn to the VRP. We will describe three k-exchange neighborhoods for the VRP, that relocate customers 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 customers 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 customers. It is straightforward to extend the presented techniques to the case where paths are relocated instead of single customers. Also, we will only describe modifications that are necessary to be able to test time window and capacity constraints, because handling precedence constraints is trivial.

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

{set of links to be removed from the current routes} → {set of links to replace the removed links}.

Furthermore, a vertex i will always refer to a vertex from the origin route and pre_i and suc_i will denote its predecessor and successor, and a vertex j will always refer to a vertex from the destination route and pre_j and suc_j will denote its predecessor and successor.

The discussion below will focus on the feasibility aspects. With respect to profitability, all procedures apply the same scheme: consider all feasible exchanges and carry out the most profitable one, if any.

Relocate: $\{(pre_i,i),(i,suc_i),(j,suc_j)\} \rightarrow \{(pre_i,suc_i),(j,i),(i,suc_j)\}$
Relocate tries to insert a vertex from one route into another. A relocation is pictured in Figure 8.

It is obvious that infeasibility can only occur at the destination route. The feasibility tests that have to be performed are similar to those described for the insertion method to construct an initial feasible tour.

Exchange: $\{(pre_i,i),(i,suc_i),(pre_j,j),(j,suc_j)\} \rightarrow \{(pre_i,j),(j,suc_i),(pre_j,i),(i,suc_j)\}$
A slight modification of the previously described relocate-neighborhood leads to what we will call the exchange-neighborhood. Here we look simultaneously at two customers from different routes and try to insert them into the other routes. An exchange is pictured in Figure 9.

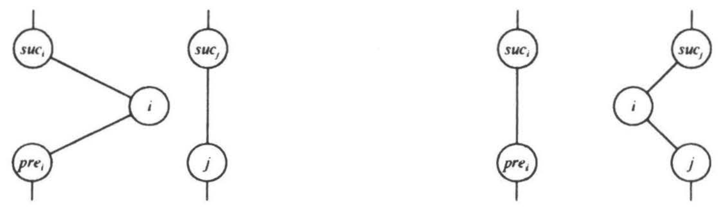


Figure 8. The relocate neighborhood.

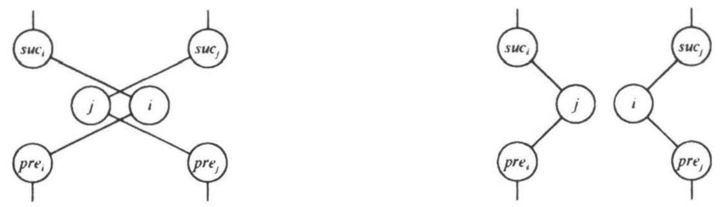


Figure 9. The exchange neigborhood.

The feasibility tests for the time window constraints are the same as for the relocation, but the feasibility tests for the capacity constraints have to be modified to take into account that load has been, or will be removed, from the route where an insertion will be made. If the customer that is removed is a delivery it only affects the part of the route before this customer, if it is a collection it only affects the part of the route after this customer. This results in the following feasibility tests:
- if $i,j \in \Delta$, then $q_j \leqslant L^-_{pre_i} + q_i$ and $q_i \leqslant L^-_{pre_j} + q_j$;
- if $i \in \Delta \wedge j \in \Gamma$, then $q_j \leqslant L^+_{suc_i}$ and $q_i \leqslant L^-_{pre_j}$;
- if $i \in \Gamma \wedge j \in \Delta$, then $q_j \leqslant L^-_{pre_i}$ and $q_i \leqslant L^+_{suc_j}$;
- if $i,j \in \Gamma$, then $q_j \leqslant L^+_{pre_i} + q_i$ and $q_i \leqslant L^+_{pre_j} + q_j$.

Cross: $\{(i,suc_i),(j,suc_j)\} \rightarrow \{(i,suc_j),(j,suc_i)\}$
Cross tries to remove crossing links and turns out to be very powerful. As a special case, if the constraints allow it, it can combine two routes into one. A cross-change is pictured in Figure 10.

After fixing a link (i,suc_i) in route r_1, the algorithm will check every link of route r_2 for a feasible and profitable cross-change. For a cross-change the feasibility checks are slightly more complicated than for the other neighborhoods we discussed. Notice that if a cross-change is actually performed, the last part of either route will become the last part of the other. Checking the time window constraints is rather easy and involves the two tests:

$$D_i + t_{i,suc_j} \leqslant S^+_{suc_j,n+1};$$

$$D_j + t_{j,suc_i} \leqslant S^+_{suc_i,n+1}.$$

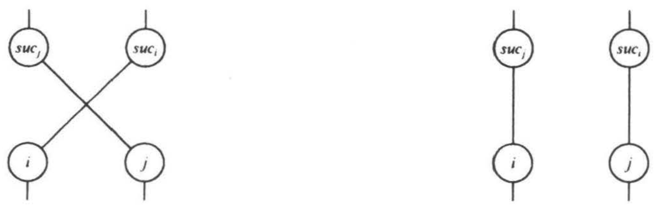


Figure 10. The cross neighborhood.

As to capacity constraints, let us restate the criterion for a route to be feasible with respect to these constraints:

$$\max_{0 \leqslant k \leqslant n+1}\{\textstyle\sum_{k+1 \leqslant l \leqslant n+1,\, l \in \Delta} q_l + \sum_{0 \leqslant l \leqslant k,\, l \in \Gamma} q_l\} \leqslant Q_r.$$

The first summation is the amount of load in the vehicle at the departure at vertex i that is still to be delivered. The second summation is the amount of load in the vehicle at the departure at i that has already been collected. An analysis of the routes that result if a cross-exchange is performed with respect to the feasibility condition stated above reveals how we can test feasibility efficiently. Because of the symmetry, we will consider only one of the resulting routes: $(0,...,i,suc_j,...,n+1)$. In addition it is useful to split the analysis in two parts.

Part 1: $(0,...,i)$

$$Q_{r_1} - \max_{0 \leqslant k \leqslant i}\{\textstyle\sum_{l \in (k+1,...,i,suc_j,...,n+1),\, l \in \Delta} q_l + \sum_{l \in (0,...,k),\, l \in \Gamma} q_l\} \geqslant 0 \Rightarrow$$

$$Q_{r_1} - \max_{0 \leqslant k \leqslant i}\{\textstyle\sum_{l \in (k+1,...,i,suc_i,...,n+1),\, l \in \Delta} q_l + \sum_{l \in (0,...,k),\, l \in \Gamma} q_l\} +$$

$$\textstyle\sum_{l \in (suc_j,...,n+1),\, l \in \Delta} q_l - \sum_{l \in (suc_i,...,n+1),\, l \in \Delta} q_l \geqslant 0 \Rightarrow$$

$$L_i^- \geqslant \textstyle\sum_{l \in (suc_i,...,n+1),\, l \in \Delta} q_l - \sum_{l \in (suc_j,...,n+1),\, l \in \Delta} q_l$$

Part 2: $(suc_j,...,n+1)$

$$Q_{r_1} - \max_{suc_j \leqslant k \leqslant n+1}\{\textstyle\sum_{l \in (k+1,...,n+1),\, l \in \Delta} q_l + \sum_{l \in (0,...,i,suc_j,...,k),\, l \in \Gamma} q_l\} \geqslant 0 \Rightarrow$$

$$Q_{r_2} - \max_{suc_j \leqslant k \leqslant n+1}\{\textstyle\sum_{l \in (k+1,...,n+1),\, l \in \Delta} q_l + \sum_{l \in (0,...,j,suc_j,...,k),\, l \in \Gamma} q_l\} +$$

$$\textstyle\sum_{l \in (0,...,i),\, l \in \Gamma} q_l - \sum_{l \in (0,...,j),\, l \in \Gamma} q_l + Q_{r_1} - Q_{r_2} \geqslant 0 \Rightarrow$$

$$L_{suc_j}^+ + Q_{r_1} - Q_{r_2} \geqslant \textstyle\sum_{l \in (0,...,j),\, l \in \Gamma} q_l - \sum_{l \in (0,...,i),\, l \in \Gamma} q_l$$

If the values of L_v^- and L_v^+ are stored for each $v \in V$ and a lexicographic search strategy is used, feasibility testing requires in constant time because the right-hand side differences can be updated by one addition or subtraction at each iteration.

The above described local search methods can easily be extended to larger

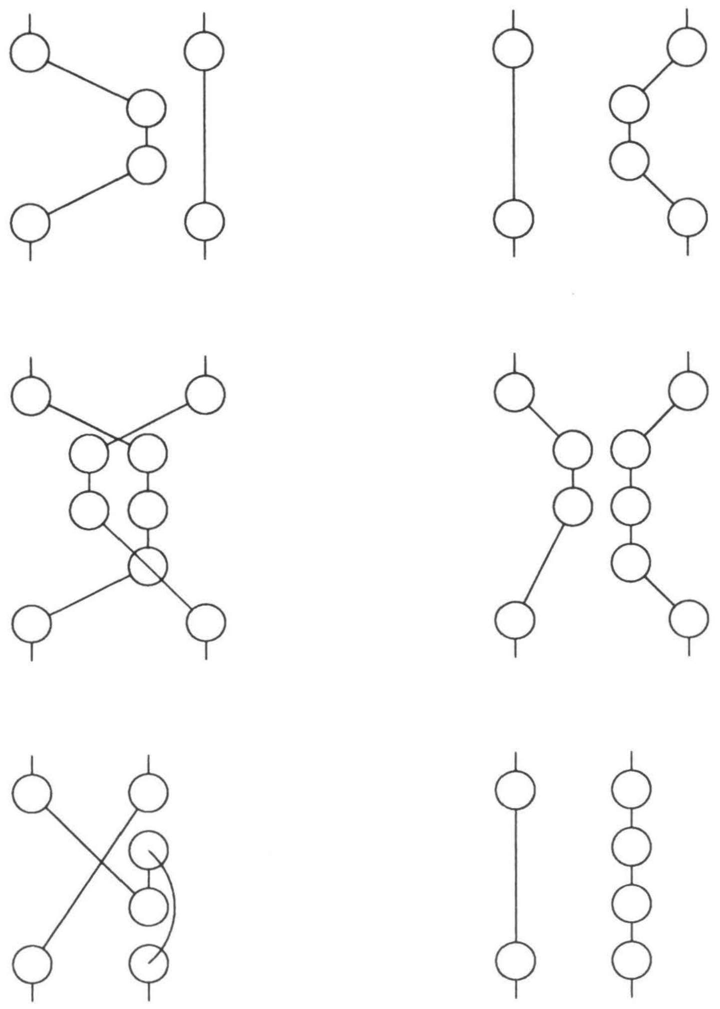


Figure 11. Extensions to the various neighborhoods.

neighborhoods by the introduction of paths instead of vertices. The paths have to be checked but that involves only local information. Figure 11 illustrates some possible extensions.

7. Conclusion

If one conclusion emerges from Part I, it is that the development of efficient methods that are capable of solving large-scale routing problems subject to real-life constraints is no longer a neglected research area.

A striking example is the set partitioning approach, which appears to be particularly efficient for strongly constrained problems. The continuous relaxation of the set partitioning formulation can be solved by the use of a column generation scheme and provides for better bounds than the relaxation of other formulations. Dynamic programming turns out to be a powerful tool to generate columns. This family of algorithms is well designed to produce approximate solutions to large problems. Optimization algorithms of this type are being used for school bus scheduling.

The construction and iterative improvement algorithms that have received so much attention in the context of the TSP and the VRP have now been extended to incorporate time windows and other constraints, such as precedence constraints and mixed collections and deliveries. These types of algorithms are all familiar, but their modification to handle practical problems is nontrivial. Although the worst-case performance of these methods is very bad [Solomon 1986], they have been successfully incorporated in distribution management software, as we will see in Part II.

PART II: INTERACTION

8. Introduction

This part of the thesis will be devoted to *decision support systems*. We will concentrate on systems that are designed to support decision making in practical planning situations through man-machine interaction. Hence, we will often use the more specific term *interactive planning systems*. In Keen's terminology [Keen, 1986], they would probably be named *extended* decision support systems, as the use of quantitative techniques is as vital as the role of human insight.

For us, DSS represents a novel approach towards the practice of operations research, which has been made possible by advances in information technology. While the mathematics of operations research is a *normative* occupation which intends to develop a theory of models and algorithms, practical operations research is an *empirical* activity in which formal tools are applied to actual problem situations in a heuristic fashion. This is in particular true for DSS.

Part II is organized as follows. Chapter 9 discusses some general concepts and addresses the questions 'What should should an interactive planning system look like and how should it behave?' and, perhaps even more importantly, 'Why should it do so?'. Chapters 10, 11 and 12 describe CAR, an interactive system for computer aided routing. This will illustrate the concepts discussed in Chapter 9.

9. Interactive planning systems

We will use the term *interactive planning system* (IPS) to indicate a system that provides support with planning activities by the integration of human perception and mechanical algorithmics in an interactive environment. The purpose of the system is to improve the quality of decision making in terms of effectivity and efficiency.

In what follows, we first review the process of planning, the role of models, and the need for interaction. We next specify a number of desirable functional requirements of an IPS. We then elaborate on the concept of man-machine interaction, and finally discuss its realization in the form of a graphical user interface.

9.1. *Planning*

Depending upon the tasks of a unit and its level within an organization, it will have different sets of short-term and long-term goals. Depending upon its size, an organization will have more or less formalized planning procedures to define these goals in the best interest of the organization as a whole and to translate these into a plan for the activities of each unit. Planning is a never ending activity. A plan is usually a revised and extended version of a previous plan. The final stage of a planning process is the decision to adopt a certain plan. In the preceding stages, many plans may have been generated, evaluated, compared and rejected. It is a challenging task to develop and implement systems that support this process.

Before starting our discussion of interactive planning systems, we must consider the characteristics of the user we have in mind and the nature of the problems he has to solve. We assume that the user is a trained professional, knowledgeable about his subject area but not necessarily familiar with the techniques of operations research and computer science. The planning situation he is facing is complex in at least two respects. First, the objectives and constraints are numerous and difficult to quantify. That is, it is impossible to construct a model that precisely captures the real-life situation. Secondly, the process required to achieve an acceptable plan cannot be completely specified in advance. Even after the plan has been developed, it may be difficult to say which of the steps taken were directly relevant to the construction of the final plan.

Each generation of users is confronted with a variety of approaches that claim to facilitate their task, each with its own acronym. Before DSS and IPS, we had - and we still have - MIS and OR.

The aim of *management information systems* is to improve the quality of information in terms of accuracy and timeliness. The emphasis is on *registration* of data in the broad sense of the word: their collection, storage, retrieval, and presentation.

The aim of *operations research* is to improve the quality of decisions. The emphasis is on *planning* on the basis of models of decision situations and algorithms that evaluate tentative decisions and generate reasonable decisions.

9.2. *Models*

A model is an abstract description of a decision situation which relates possible decisions to their quality. In a model, decisions and their quality are specified in terms of variables and relations between them. It is illuminating to distinguish two classes of models.

In the first class, the model is designed to *evaluate* decisions. Thus, a tentative decision is input and its quality is output. Simulation models are examples of this approach. Such models are usually defined as computer programs. The user fully governs the search for a good decision, and several decisions may be tried before one is adopted.

In the second class, the model is designed to *generate* decisions. Thus, a desired quality is input and a decision is output. Linear programming is the prime example of this approach. Quality is here a multidimensional notion, stipulating feasibility on a number of dimensions and optimality on another one. In case the linear programming paradigm does not suffice and one of its many extensions - integer, nonlinear or stochastic programming - is called upon, optimization may be too time consuming and approximation algorithms are used.

With evaluative models, different kinds of 'what if' questions can be answered. First, the situation is fixed and the consequences of different decisions are studied. Secondly, the decision is fixed and its consequences in different situations are studied. With generative models, decisions for a variety of decision situations and quality requirements can be obtained and analyzed.

9.3. *Decision making vs. decision support*

The prototypical OR approach is oriented towards *decision making*: 'Give me the problem, then I will give the optimal solution.' This simplistic attitude does not match the complexity of many planning situations. If a single model is chosen to represent such a situation, its solution - mathematically correct or not - may be unusable in practice. This is because no model, no matter how elaborate, can ever be a perfect representation of reality.

It is often prudent to use a variety of models. Each of these is a picture of the actual situation, but different aspects are emphasized or ignored. Moreover, it is not always known a priori what constitutes a good decision, because the decision maker does not fully specify his tolerances and priorities.

Quantitative techniques cannot substitute the human decision maker, but the reverse of this statement is also true. Instead of lamenting the limitations of either, one should profit from combining the strong points of both: the insight and experience of the planner, and the power and precision of the algorithms. This is what IPS is all about. An IPS aims at *decision support* rather than decision making. It focuses on helping users prepare decisions.

This point of view has some consequences for the realization of an IPS. At the algorithmic side, it must be equipped with evaluative as well as generative models to enable the planner to produce and judge alternative decisions. As to the interaction, it must be able to manipulate massive amounts of data in real time. It is in this sense that IPS merges OR and MIS.

In accordance with the above, we define the following *design goals* for the development of an IPS:

(1) combine the use of operations research models and methods with advanced data access and retrieval functions;

(2) focus on features which make the system easy to use, such as interactivity, computer graphics, and error prevention;

(3) strive for flexibility and adaptability in order to accommodate changes in the decision situation, the interactive environment, and the planning approach.

9.4. *Functional requirements*
These design goals lead in turn to a number of functional requirements for an IPS.

1. *Functional flexibility*. On the one hand, the system should enable the planner to define and modify a plan. It is then used as an *automatic scratch pad*, which supports the traditional manual planning in a modern way. It provides facilities for the storage, retrieval and display of data of problem situations and decisions; in this respect, it resembles an MIS. It is also able to evaluate the quality of a given plan. The system acts as an *assistant* to the planner.

On the other hand, the system should be able to construct a complete plan and to modify an existing plan by itself. It has now the role of an *automatic pilot*. In addition to the registrative and evaluative facilities, it provides the means to generate a plan of a given quality. The system acts as an *advisor* to the planner.

The roles of assistant and advisor are the extremes of a broad spectrum and there is much inbetween. When the user constructs a plan by hand, he may do so on the basis of suggestions provided by the system at various points. When he completes a plan in this way, he may ask the system for possible improvements. Alternatively, he may construct a partial plan and leave it to the system to complete it; the result can then serve as the starting point for manual modifications. The number of possibilities is virtually unlimited, and it depends on the entire context which style of planning is employed most frequently. Even if the system does not go beyond the role of assistant, it is already a useful tool for planners. It is always the user who is in charge, even if the system functions as advisor.

2. *Ease of use*. If the system is easy to use or 'user friendly', the planner can concentrate on solving the problem at hand. This is a hard job under any circumstances, and a system perceived as difficult to operate may go unused even though of potential value. Features that contribute to ease of use are the following:

(a) *Simplicity*. Features that are not simple to understand will not be used. It is often difficult for the software engineer to detect troublesome aspects of his design. These aspects do become apparent, however, when the functional description is written. They can be avoided by completing this document before implementing the system or at least by having feedback from specification to implementation. Anything that is difficult to explain will almost certainly be difficult to use.

(b) *Consistency*. A consistent system is one that behaves in a generally predictable manner. Function names and calling sequences, graphical representations and colors, all these should follow simple and similar patterns without exceptions. The user is then able to build a conceptual model of how the system reacts; in new situations, he can apply his knowledge with a good chance that it will work. Again, inconsistencies often show up when the functional description is written.

(c) *Completeness and conciseness*. The system must contain a complete and concise set of functions that allow the user to handle his problem effectively. There should be no irritating omissions or redundancies. The strength of the system lies in the coherence of the functions, not in their number.

3. *Robustness*. Users are capable of an extraordinary misuse of the system, either through misunderstanding or for enjoyment. The system should accept such treatment with a minimum of complaint. When the user does something unexpected, the system reports the error in the most helpful manner possible. Only in extreme circumstances errors cause termination of execution, as this generally results in the loss of valuable information.

9.5. *Interaction*

Until now, we have discussed the issue of man-machine interaction in fairly broad terms. We will be more specific in this section.

In the last decade we have witnessed extraordinary advances in information technology, which have resulted in enormous increases in processing power and graphics capabilities. There is now an alternative to batch processing and centralized operations. Due to the practicality to perform intricate computations in real time and to display data and results in an informative way, it is a feasible idea to involve humans throughout the planning process.

Interaction is possible, but why is it desirable? The brief answer is that planning problems tend to be both *hard* and *soft*.

Most practical planning problems are, in any reasonable abstraction, *NP*-hard. This implies that these problem types are probably inherently intractable in a well defined sense [Garey & Johnson 1979]. For practical purposes, it indicates that the solution of realistic problem instances to optimality may require an inordinate computational effort. We have to resort to approximation algorithms, that deliver acceptable solutions within an acceptable amount of time. It is just one step further to embed such algorithms in a heuristic setting. The solution is then found by means of a trial-and-error procedure, in which man and machine divide the tasks in accordance with their respective capabilities. In *interactive optimization* [Fisher 1986], the user controls the solution process by setting initial parameters, selecting algorithms, and adjusting solutions. Jones [1987] introduces the term *grey box* for this type of optimization: the traditional single black box is replaced by a network of black boxes with user intervention required whenever one of them completes execution. In this way, the human planner guides the computer towards promising parts of the solution space.

Another aspect of real-life problem solving is that the notions of *feasibility* and *optimality* are not as precise as in mathematics. Most planning problems contain subjective elements that are difficult to quantify. Feasibility requirements may be soft rather than strict, and tradeoffs between optimality criteria are often not explicitly known but carried implicitly in the value judgement of the decision maker. Interaction is one way of coping with this aspect [Fisher 1986]. While the planner constructs (or modifies, or extends) a plan, he may override constraints; the system should warn him as soon as violation occurs, but it is the planner who determines feasibility. Similarly, the planner decides about the comparative evaluation of the objectives. He has full control and responsibility.

As a consequence, interaction adds to effectivity, efficiency, and acceptability. First, the cooperation between man and machine leads to better solutions. The machine cannot be beaten in solving well-defined detailed problems. The human planner is superior in guiding the overall solution process, in recognizing global patterns, and in observing all kinds of *ad hoc* constraints. Secondly, these better solutions are obtained faster, because interaction allows for flexibility in manipulating data and in selecting alternatives. Finally, an interactive system is more readily accepted. The human planner is not replaced but gets a versatile tool.

9.6. *User interface*

Now that we have indicated *why* interaction is a desirable feature of the planning process, we discuss in general terms *how* interaction has to take place.

The user interface is the part of the system that provides the means for communication between man and machine. It essentially consists of two languages [Bennett 1983]. The first one is the *presentation* language, which is employed by the machine and understood by the user; it expresses what the user sees or senses as context for interaction. The second one is the *action* language, employed by the user and understood by the machine; it expresses what the user can do in order to change the context in a way which will help him to meet his goals. By 'language' we mean the collection of patterns of signs and symbols which one participant in the interaction (man or machine) is allowed to use in presenting information to the other participant.

An IPS should be able to present problem instances and solutions in a meaningful way, i.e., one that permits a quick assessment and analysis of the data being presented. The principal usefulness of computer graphics is the possibility to provide different, and perhaps more insightful, representations of the same data. The use of *iconic* as well as *representational* graphics is clearly relevant here. Iconic graphics display part of the real world, such as a road network or a facility layout, while representational graphics display data summaries, such as bar charts and pie charts. Noteworthy research in this area is Tufte's [1983] work on the visual display of quantitative information and Jones' [1988] attempts to develop novel representations of machine schedules. The benefit of computer graphics to decision making, however, is still a topic of debate. DeSanctis [1984] summarizes the literature on this subject and arrives at several propositions based on persistent trends.

The effect of a graphical user interface can even be stronger if *color* graphics are used. Colors provide an easy way to distinguish between various objects. One should color with taste, however; an excessive use of colors may confuse the picture.

As to the action language, we have already mentioned ease of use as a major functional requirement of an IPS. Simplicity, consistency, completeness and conciseness are, of course, worthy goals in the design of any computerized system. For an IPS they are especially important, and the action language is the prime feature of the system that will reveal whether these goals have been achieved.

All in all, an IPS should provide an interface which the user can interpret easily and control effectively. The design of the user interface is a principal component of the overall design process. Many guidelines have been proposed for this purpose; the recent book by Shneiderman [1987] reviews the subject area. At the risk of repeating ourselves, we emphasize the two central issues: focus on a limited number of well-chosen *representations* and operations on them, and provide *uniformity of structure* so that the user can take the interface for granted as he concentrates on the problem he is solving.

10. CAR: An interactive planning system for computer aided routing

CAR is an interactive software package which has been developed as a tool to support operational distribution management. CAR enables the user to construct economical vehicle routes and schedules in a simple way. CAR acts as an assistant and advisor to the planner. CAR has a supporting function; it is the user who is in charge and who is responsible for making all the decisions. The use of CAR can lead to savings in physical distribution costs, a faster and simpler planning process, a more constant level of service towards the customers, a lesser dependence on the quality of the human planner, and better management information facilities.

The planning module CAR should form part of larger system. Data entry and report generation do not belong to CAR. Managing the permanent and temporary data bases of vehicles, addresses, distances and travel times, processing incoming orders and generating printed schedules are tasks of the environment.

CAR was developed at the Centre for Mathematics and Computer Science during the period 1983-1987. Its development was financially supported by the *Stichting voor de Technische Wetenschappen* and the *Stichting Mathematisch Centrum.*

10.1. *Problem type*

CAR is suited for distribution problems with the following characteristics.

- There is a single depot where several vehicles, possibly with different capacities, are stationed.
- The commodity to be transported is homogeneous in the sense that the allocation of commodities to vehicles is restricted only by the vehicle capacities.
- A vehicle can make several trips a day.
- A vehicle has a time window that specifies its availability. (This allows one to impose a maximum route duration.)
- There may be both collections and deliveries. Vehicles depart from the depot with the commodities to be delivered and eventually return to the depot with the collected commodities. Anything collected on the way is transported to the depot. Collections and deliveries may occur in any sequence on the same trip.
- An address may impose restrictions on the capabilities of the vehicle visiting it. For example, it may require special loading equipment.
- An address has one or more time windows within which service must take place.
- An address can have a priority, indicating that it must be visited.
- An address is to be visited by at most one vehicle.

10.2. *Solution method*

The solution method used is based on the 'cluster first-route second' principle. It is a two-phase approach. The first phase clusters the addresses into groups, one for each vehicle; the second phase routes each vehicle through the addresses assigned to it. It is not a purely algorithmic approach, but a heuristic one, proceeding along the lines indicated in Chapter 9.

10.3. *User interface and screen*

As argued in Chapter 9, the user interface is an important component of an interactive planning system. In developing CAR, we have chosen for a graphical user interface. Data that are traditionally presented in alphanumerical form can now be used to create a graphical representation. CAR provides facilities for three types of graphical representations of the problem, oriented towards distance, time and load. Colors are used to distinguish between clusters and routes. For small problems this has a minor advantage, for large problems it is a necessity. Next to these graphical representations, the user is also able to consult the relevant part of the huge amounts of alphanumerical data.

In order to obtain a steady view of the entire screen, we have divided it into three fixed regions, which contain a *problem representation*, *commands*, and *alphanumerical information*, respectively.

The interaction is menu driven. This has the advantage that at any moment all the feasible commands are visible, and it prevents the user from giving infeasible commands. The commands are divided over three primary and three secondary menus. The first primary menu (COMMUNICATION) contains commands that allow the user to change the problem instance and to write partial solutions on an output file. The second one (CLUSTERING) contains commands that constitute the clustering phase, and the third one (ROUTING) those that form the routing phase. The secondary menus provide the supporting functions. The first secondary menu (INFORMATION) contains commands to show alphanumerical information. The second one (SCREEN) contains commands to define the contents of the problem representation region. The user can ask for four different representations: the three graphical ones mentioned above and a purely alphanumerical one. In the spatial representation, he can work with separate parts of the solution by zooming in and making trips invisible. The third secondary menu (STORAGE) contains commands that enable the user to temporarily store the current solution. The clustering of commands into groups leads to an efficient use of the screen, because exactly one primary menu and at most one secondary menu can be active at any time.

10.4. *Graphical Kernel System (GKS)*

GKS was used for the implementation of the user interface. It is a graphical library for applications that generate two-dimensional pictures on vector or raster graphic terminals.

10.5. *Input*

The input is read from three files *address.CAR*, *vehicle.CAR* and *table.CAR*, the first two of which are specified in more detail in Table 1.

Addresses
CAR requires the following data for each address:
- collection or delivery;
- identification: postal code, description (optional);
- limitations on vehicle types;
- time windows;
- address dependent (un)loading time; summing this and the order dependent (un)loading time results in the total (un)loading time;
- number of orders;

per order:
- order number;
- priority, indicating if it must be included in the plan;
- size;
- (un)loading time.

Vehicles
The following data are required for each vehicle:
- identification;
- capabilities;
- capacity;
- availability.

Remarks
- The notions 'size of an order' and 'capacity of a vehicle' have to be further defined in consultation with the user. They have to be expressed in the same units.
- The 'capabilities of a vehicle' have to be further defined in consultation with the user. They relate to address-vehicle restrictions.
- The 'availability of a vehicle' is expressed in terms of a time window. The window also reflects the opening hours of the depot and the working period of the driver.

Distances, travel times, and coordinates
CAR uses distances, travel times, and coordinates. This information does not belong to CAR and has to be provided by the user in the form of separate tables. CAR uses the postal code as a key to these tables. Whether these tables are compiled on the basis of a road network or in any other way is immaterial.

10.6. *Output*
The output is written on a file *plan.CAR*, which is specified in more detail in Table 1.

Trips
CAR supplies the following data for each trip:
- identification of the vehicle allocated to the trip;
- load;

file				#characters
address.CAR	per address[1]	collection/delivery		1
		identification		40
		limitations		8
		time windows		4/4/4/4
		(un)loading time		4
		#orders		3
		per order	order number	10
			priority	1
			size	5/5
			(un)loading time	4
vehicle.CAR	per vehicle[2]	identification		9
		capabilities		8
		capacity		6/6
		availability		4/4
plan.CAR	per trip	vehicle identification		9
		load		6/6
		load factor		3/3
		length		6
		travel time		4
		waiting time		4
		#addresses		3
		per address[3]	identification	16
			arrival time	4
			waiting time	4
			departure time	4
	#trips			2
	average load factor			3/3
	total length			6
	total travel time			6
	#included addresses			4
	#free addresses			4
	per free address	identification		16
	modifications			

1. First all delivery addresses, then all collection addresses.
2. In order of decreasing capacities.
3. In the order in which they are visited, with the depot as the first and last address.

Table 1. Detailed specification of input and output files.

- load factor;
- length;
- travel time;
- waiting time;
- number of addresses;

per address (where the depot is included as the first and the last address in the trip):
- identification;
- arrival time;
- waiting time;

- departure time.

Overall summary
This contains the following data:
- number of trips;
- average load factor of the used vehicles;
- total length;
- total travel time;
- identifications of the addresses that have not been included in the trips.

Input modifications
All the changes in the input data that have been made during the planning session are listed here.

Remark
All information concerning loads consists of a delivery and a collection component.

10.7. *Current implementations*
CAR has been written in the C programming language and implemented on three configurations. More implementations will become available in due course.

IBM 6150 (PC/RT) & IBM 5080 (Graphics Display)
The operating system on the IBM 6150 is AIX, IBM's UNIX variant. This is a state-of-the-art configuration with a powerful processor and an advanced screen. We use the C implementation of GKS developed at CWI.

IBM PC/AT & IBM PGA (Professional Graphics Adapter)
The operating system on the IBM PC/AT is MS-DOS 3.2. A 20Mb hard disk, 640Kb internal memory, and a mathematical co-processor are required. We use the C binding of GKS developed by IBM for PC's.

IBM PC/AT & IBM EGA (Enhanced Graphics Adapter)
The specifications are the same as for the previous configuration.

10.8. *Modifications and extensions*
CAR has a modular structure, which makes it easy to change existing functions and to add new ones.

The precise output and the alphanumerical information displayed on the screen are defined in consultation with the user. This document describes a possible specification.

Extensions that could be considered are:
- multiple depots;
- collection and delivery of the same commodity during a single trip (as in dial-a-ride systems);
- heterogeneous commodities;

- division of an order over several vehicles.

11. CAR: FUNCTIONAL DESCRIPTION

At the start of a planning session, CAR searches the environment for the files *address.CAR*, *vehicle.CAR*, and *table.CAR*, which should contain all the required input data, and processes them. If new orders arrive during a planning session, they can be appended to the file *address.CAR*. At the end of a planning session, CAR leaves a file *plan.CAR* in the environment, which contains data on the trips created, an overall summary, and input modifications, if any.

CAR distinguishes two types of commands. The distinction is visualized on the screen by the presence or absence of an exclamation point in front of the command. Commands preceded by an exclamation point call functions that correspond to algorithms, the results of which are generally not predictable by the user. These are the functions that generate or modify a plan automatically. Commands without an exclamation point have an easily predictable result. These are the functions that allow the user to generate or modify a plan manually. Figure 12 lists all available commands.

One enters all commands by choosing from menus or specifying addresses with the mouse, light pen, or joystick, depending on the configuration. An address is specified by indicating the associated point on the screen. If the postal area corresponding to this point contains just this address, then one is done; otherwise, CAR lists all the addresses in the area in a menu and the user indicates the desired address.

Some commands ask for the identification of a cluster or a trip. In the spatial representation, a cluster or a trip is specified by indicating an address that belongs to it. In the three other representations, a color palette is used in which each cluster or trip has its own color.

When CAR is started, the user sees a spatial representation of the problem and the CAR 'supermenu'. This contains five commands: **COMMUNICATION**, **CLUSTERING**, **ROUTING**, **RESTART**, and **STOP**. It serves to select one of the three primary menus, to restart the planning (without having to process the files *address.CAR* and *vehicle.CAR* again), and to end the planning session.

11.1. COMMUNICATION

This primary menu contains the commands that enable the user to modify the problem instance and to write partial solutions to the file *plan.CAR*. CAR maintains a record of all modifications of the original data and writes these on the file *plan.CAR* at the end of the planning session.

ADD

effect:	Data of addresses that have been added to the file *address.CAR* are processed.

DELETE

input:	Address.
effect:	The specified address is deleted from the set of addresses.

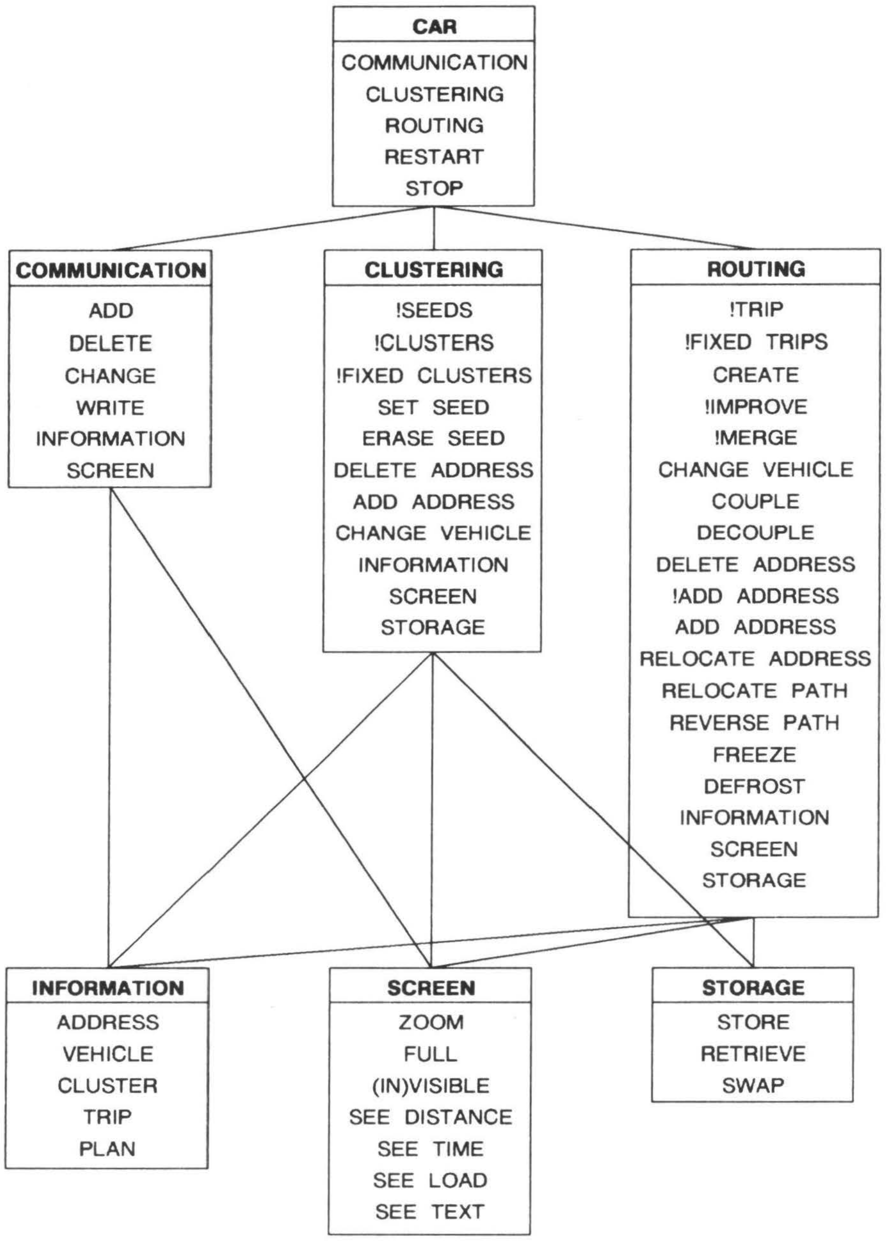


Figure 12. Relations between menus.

CHANGE

input:	Address.
effect:	The input data of the specified address can be changed.

WRITE

input:	Trip.
effect:	The specified trip is written on *plan.CAR*. The addresses of this trip and the vehicle allocated to it are no longer taken into consideration.

INFORMATION

effect:	The secondary menu **INFORMATION** is activated.

SCREEN

effect:	The secondary menu **SCREEN** is activated.

11.2. **CLUSTERING**

This primary menu contains the commands to generate clusters in the first phase of the planning process. A cluster is a collection of addresses with a vehicle that will visit them. The user can form and modify clusters manually or ask the system to construct them automatically, as he sees fit. One can also reenter the clustering phase from the routing phase. In that case, the existing trips define the clusters, but the orderings that define the trips disappear.

For each cluster, there is an address which serves as a seed point around which the cluster is grown. A seed point is marked with a circle around the address in question, in the color of the cluster.

An address that does not belong to any cluster is called *free*.

!SEEDS

input:	Integer m.
effect:	A set of m new seed points is created on the basis of distances, order sizes, address-vehicle restrictions, time windows, and existing seed points. A vehicle is allocated to each new seed point, which defines an upper bound on the total order size in the corresponding cluster.
algorithm:	See Section 5.2.

!CLUSTERS

effect:	The free addresses are assigned to a seed point, as far as the restrictions permit it.
algorithm:	See Section 5.1.

!FIXED CLUSTERS

effect:	When this function is called, it is assumed that no seed points exist. CAR searches the environment for the file *fixed.CAR*. This file contains a number of trips that have been earlier defined by the user; these are now interpreted as clusters. For each of these clusters, its intersection with the collection of addresses is determined. In case the

intersection is nonempty, it constitutes a new cluster; the system generates a seed point and allocates a vehicle.

SET SEED

input:	Free address; vehicle.
effect:	The specified address will serve as a seed point. The specified vehicle is allocated to the seed point.

ERASE SEED

input:	Seed point.
effect:	All assignments of addresses to the specified seed point are canceled. The seed point is erased.

CHANGE VEHICLE

input:	Seed point; vehicle.
effect:	The specified vehicle is allocated to the specified seed point. This cancels the previous allocation.

DELETE ADDRESS

input:	Address.
effect:	The assignment of the specified address is canceled.

ADD ADDRESS

input:	Free address; seed point.
effect:	The specified address is assigned to the specified seed point.

INFORMATION

effect:	The secondary menu **INFORMATION** is activated.

SCREEN

effect:	The secondary menu **SCREEN** is activated.

STORAGE

effect:	The secondary menu **STORAGE** is activated.

11.3. **ROUTING**

This primary menu contains the commands to generate routes in the second phase of the planning process. A route consists of one or more trips. A trip is a collection of addresses that have been put in the order in which they have to be visited, starting and finishing at the depot. The user can form and modify trips manually or ask the system to do so automatically, as he sees fit.

A trip is represented on the screen by linking each pair of successive addresses by a line segment. The depot is the only address that can occur in more than one trip; it occurs in each trip. To avoid congestion around the depot on the screen, we have decided to delete the line segments of which the depot is an end point. The first address on a trip is marked with a circle around it, in the color of the trip.

An address that does not belong to any cluster or trip is called *free*.

There are commands for trip optimization and trip manipulation. An important aspect of trip manipulation is the presentation: the user first sees the effect of the action he proposes, and then decides to accept it or not. Even if the proposed action results in an infeasible trip, acceptance is allowed.

Next to the spatial representation, the user can switch to three other representations of a single trip: a bar chart indicating the schedule in time, a bar chart indicating the load of the vehicle, and a complete alphanumerical survey. These representations are secondary to the spatial one because they relate to just a single trip. In all representations, trip optimization and manipulation are possible.

!TRIP

input:	Cluster.
effect:	For the specified cluster, a short feasible trip is generated. If it appears to be impossible to include all addresses of the cluster in a feasible trip, the assignment of some addresses is canceled.
note:	If the user indicates the depot, this function is carried out for all trips. This is consistent with the fact that the depot is an address that belongs to all trips.
algorithm:	See Section 5.3.

!FIXED TRIPS

effect:	When this function is called, it is assumed that no clusters exist. CAR searches the environment for the file *fixed.CAR*, which contains a number of trips that have been earlier defined by the user. For each of these trips, its intersection with the collection of addresses is determined. In case the intersection is nonempty, a new trip is defined by visiting the addresses in the intersection in the same order as in the fixed trip; the system allocates a vehicle.

CREATE

input:	Number of addresses; (vehicle).
effect:	The user creates a trip in a stepwise fashion. If the address that is specified first is free, then a new cluster will be defined: a vehicle has to be allocated to it, and only free addresses may be assigned. If the first address does belong to a cluster, then a vehicle has already been allocated, and only addresses belonging to this cluster or free addresses may be assigned. The user specifies addresses in the order in which he wants them to be visited. He removes the last address of the partial trip by specifying the next to last address. He completes the trip by specifying the depot, or by specifying the last address twice in a row. During this process the user is informed about the feasibility of an extension and of the length, travel time, waiting time and load of the trip created thus far.

!IMPROVE

input:	Trip.
effect:	Optimization techniques are applied to shorten the specified trip. Information about changes in length, travel time and waiting time of the trip is provided.
algorithm:	See Sections 6.1, 6.2, 6.3, 6.4, 6.5, and 6.7.
note:	If the user indicates the depot, this function is carried out for all trips.

!MERGE

input:	Two trips from different routes.
effect:	Optimization techniques are applied to shorten the specified trips by the exchange of addresses. It is possible that both trips are merged into one. Information about changes in length, travel time, waiting time and load of the trip is provided.
algorithm:	See Section 6.8.

CHANGE VEHICLE

input:	Trip; vehicle.
effect:	The specified vehicle is allocated to the specified trip. This cancels the previous allocation.

COUPLE

input:	Two trips.
effect:	The specified trips are concatenated. In contrast to the function !MERGE, this function does not change the ordering of each trip. The largest of the vehicles allocated to the original trips is allocated to the newly created route. Information about the new departure and arrival times at the depot for each trip is provided.

DECOUPLE

input:	Trip.
effect:	The specified trip is decoupled from possibly preceding and succeeding trips. It becomes a separate route again.

DELETE ADDRESS

input:	Address.
effect:	The specified address is deleted from the cluster to which it belongs. It becomes a free address. In case the specified address belonged to a trip, information about changes in length, travel time and load of the trip in question is provided.

!ADD ADDRESS

input:	Free address.
effect:	The specified address is inserted at the best feasible point in one of the existing trips. Information about changes in length, travel time and load of the trip in question is provided. In case each insertion is

infeasible, the address remains free.

ADD ADDRESS
input: Free address; two addresses that occur successively in a trip.
effect: The specified free address is inserted between the two successive addresses. Information about the feasibility of the insertion and about changes in length, travel time and load of the trip in question is provided.

RELOCATE ADDRESS
input: Address belonging to a trip; two addresses that occur successively in a trip.
effect: The first specified address is relocated between the other two. Note that the trips from which these addresses are taken may be the same. Information about the feasibility of the relocation and about changes in length, travel time and load of the trips in question is provided.

RELOCATE PATH
input: Two addresses belonging to the same trip; two addresses that occur successively in a trip.
effect: The path which is specified by the first two addresses is relocated between the other two. Note that the trips from which these addresses are taken may be the same. Information about the feasibility of the relocation and about changes in length, travel time and load of the trips in question is provided.

REVERSE PATH
input: Two addresses belonging to the same trip.
effect: The path specified by the two addresses is reversed. Information about the feasibility of the reversal and about changes in length and travel time of the trip in question is provided.

FREEZE
input: Two addresses belonging to the same trip.
effect: The path specified by the two addresses is fixed. This means that the path will not be changed by algorithmic commands (i.e., those preceded by an exclamation point).

DEFROST
input: Two addresses belonging to the same trip.
effect: The fixation of the path specified by the two addresses, caused by a FREEZE command, is canceled.

INFORMATION
effect: The secondary menu **INFORMATION** is activated.

SCREEN

effect: The secondary menu **SCREEN** is activated.

STORAGE

effect: The secondary menu **STORAGE** is activated.

11.4. **INFORMATION**

This secondary menu can be activated from the primary menus **COMMUNICATION**, **CLUSTERING** and **ROUTING**. It contains commands that enable the user to examine information about addresses, vehicles, clusters, trips, and the overall plan. These functions will be further defined in consultation with the user.

ADDRESS

input: Address.

effect: The following information about the specified address is provided:
- identification;
- limitations on vehicle types;
- time windows;
- (un)loading time;
- priority;
- size;
- arrival time;
- waiting time;
- departure time.

VEHICLE

input: Vehicle.

effect: The following information about the specified vehicle is provided:
- corresponding cluster or trip;
- identification;
- capabilities;
- capacity;
- availability.

CLUSTER

input: Cluster.

effect: The following information about the specified cluster is provided:
- allocated vehicle;
- load;
- load factor;
- number of addresses.

TRIP

input: Trip.

effect: The following information about the specified trip is provided:
- allocated vehicle;

- load;
- load factor;
- length;
- travel time;
- waiting time;
- number of addresses.

PLAN

effect: An overall summary is given:
- number of trips;
- average load factor;
- total length;
- total waiting time;
- total travel time;
- number of free addresses.

Next, a summary of each trip is given:
- load factor;
- length;
- travel time;
- waiting time.

11.5. **SCREEN**

This secondary menu can be activated from the primary menus **COMMUNICATION**, **CLUSTERING** and **ROUTING**. It contains commands that define the problem representation region.

When CAR is started, the user sees a spatial representation of the problem. In this representation, he can work with separate parts of the problem by zooming in and by making trips invisible. Next to the spatial representation, the user can switch to three other representations of a single trip: a bar chart indicating the schedule in time, a bar chart indicating the load of the vehicle, and a complete alphanumerical survey.

ZOOM

input: Rectangle, specified by its lower left and upper right corners.

effect: The contents of the rectangle form the new contents of the problem representation region.

FULL

effect: In the problem representation region, all trips are made visible in their original sizes.

(IN)VISIBLE

input: Trip, or the depot.

effect: If a trip is specified, it is made invisible. If the depot is specified, all trips are made visible.

SEE DISTANCE

effect:	Transition from the current problem representation to the one oriented towards distance.

SEE TIME

input:	Trip.
effect:	Transition from the current problem representation of the specified trip to the one oriented towards time.

SEE LOAD

input:	Trip.
effect:	Transition from the current problem representation of the specified trip to the one oriented towards load.

SEE TEXT

input:	Trip.
effect:	Transition from the current problem representation of the specified trip to an alphanumerical survey.

11.6. **STORAGE**

This secondary menu can be activated from the primary menus **CLUSTERING** and **ROUTING**. It contains commands that enable the user to temporarily store the current solution.

STORE

effect:	The current set of clusters or trips is stored. A set that may have been stored before is erased.

RETRIEVE

effect:	The current set of clusters of trips is erased. The set that has been stored before is retrieved.

SWAP

effect:	The current set of clusters or trips is exchanged with the one that has been stored.

12. CAR: USER INTERFACE

As already indicated in Section 10.4, we used the *Graphical Kernel System* (GKS) for the implementation of the color graphics user interface of CAR. GKS is a basic graphics system for applications that produce computer generated two-dimensional pictures. It supports man-machine interaction by supplying basic functions for graphical input and picture segmentation. In order to enable the reader to understand the implementation issues relating to the user interface, we start with a brief description of the principal facilities and more important functions of GKS [Hopgood, Duce, Gallop and Sutcliffe 1983; Sproull, Sutherland and Ullner 1985].

12.1 *Graphical Kernel System*

Coordinate systems and transformations. In order to specify the geometry of a graphical primitive, GKS measures its location relative to a cartesian coordinate system. GKS uses three two-dimensional coordinate systems at various stages in the control of graphical input and output:

(1) The *world coordinate system* (WC) is used by an application program to specify the location and size of the graphical object to be drawn; it may be freely chosen by the application program.

(2) The *normalized device coordinate system* (NDC) acts like a device independent display surface. The size of this region is limited; only objects lying in the region $0 \leqslant x \leqslant 1$ and $0 \leqslant y \leqslant 1$ can be displayed.

(3) The *device coordinate system* (DC) measures physical locations on a workstation display surface.

There are always two active coordinate transformations establishing mappings between these systems:

(1) One of several *normalization transformations*, which convert world coordinates to normalized device coordinates, is applied when an application calls GKS to output a graphical object. The principal use of normalization transformations is to allow the application program's coordinate system to be independent of NDC space, whose properties are fixed. Normalization transformations allow scaling and translation.

(2) A *workstation transformation*, which maps NDC coordinates into the coordinate system used by the output device. The purpose of this transformation is to achieve device independence, so that users of most GKS functions can think of drawing on an NDC surface and need not be concerned with the details of the device coordinate system.

GKS maintains a list of several normalization transformations, each described by a window in the world coordinate system and a viewport in NDC space. The application program may set the window and viewport limits of each transformation. It may also select one of them to be the current active one, which is applied by GKS to world coordinates supplied in calls to graphical primitives. The viewport serves another role besides being part of a normalization transformation: it defines a *clipping rectangle*. If clipping is enabled, GKS will clip all graphical primitives to the viewport limits, so that no lines, text, or other object will be shown outside the viewport. Normalization transformations thus serve two purposes: they define a transformation and a clipping rectangle.

Graphical output. At the heart of GKS are the functions that display primitive geometric objects. Pictures are considered to be constructed from a number of basic building blocks, or *output primitives*. The four basic output primitives in GKS are:

(1) *polyline*, which draws a sequence of line segments;

(2) *polymarker*, which marks a sequence of points with the same symbol;

(3) *fill area*, which displays a specified region;

(4) *text*, which displays a string of characters.

The various properties of graphical primitives are collectively called *attributes*.

Attributes govern the color of a line, whether it is solid or dashed, and its thickness. Similar attributes apply to other primitives.

Segments and their attributes. Segments are structures held within GKS that allow an application program to represent and manipulate pictures or portions of pictures. Loosely speaking, a segment is a collection of graphical primitives specified by the functions for graphical output described above. An application program may create as many segments as storage permits; for identification, it assigns each one a *segment name* of its choice. Segments are manipulated by changing the *segment attributes*. They allow the application program to modify the appearance of the segment. The following attributes are associated with a segment:

(1) The segment *visibility* attribute determines whether a segment will be displayed or not.

(2) The segment *highlighting* attribute specifies whether a segment is displayed normally or highlighted.

(3) The segment *priority* attribute is a number between 0 and 1 that determines how overlapping segments are displayed. When one segment overlaps another, primitives in the segment with higher priority may obscure primitives in the segment with lower priority.

(4) The segment *transformation* is a geometric transformation that is applied whenever a segment is drawn on the screen to translate, rotate, or scale the primitives of the segment before they are displayed.

(5) The segment *detectability* attribute determines whether primitives in the segment can be identified by the *pick* input device (see below).

Graphical input devices. Graphical input is obtained from one or more *logical input devices* associated with a workstation. A logical input device obtains user actions from the physical device, but also provides feedback on the screen to help the user operate the physical device. The different classes of logical input devices provide different kinds of feedback, designed for different kinds of graphical input. The logical input device is best thought of as an *interaction technique* rather than a device per se. The six classes of logical input devices provided by GKS and the values they report to the application program are:

(1) *locator*: The user identifies a location on the screen; the application program obtains the world coordinates of the location and the identity of the normalization transformation that was used to map NDC to WC.

(2) *stroke*: The user traces a path; an array of world coordinates on the path is returned.

(3) *pick*: The user points to an object on the screen; the name of the segment that contains the object is returned, together with the objects *pick identifier*. The pick identifier is an output primitive attribute that can be associated with graphical primitives so as to distinguish picked objects within a segment.

(4) *valuator*: The user indicates a numeric value in some range; the value is returned.

(5) *choice*: The user selects one of a fixed number of alternatives; the index of the selected alternative is returned.

(6) *string*: The user enters a text string; a character string is returned.

When a logical device is used to obtain input, it goes through the following stages:

(1) A prompt may be displayed. For example, a device may prompt with an initial string.

(2) The user manipulates the input device, and an echo appears on the screen that allows the user to see what he is doing. For example, a rubber band.

(3) The user triggers the completion of the interaction. For example, by striking a key on a mouse.

The interaction process, involving a logical input device, can be considered as taking place between two processes. One is the application program; the other is the input process, which looks after the input device and delivers data from the device to the application program. The relationship between the two processes can vary and, in doing so, will produce different styles of interaction which will effect the way that the user sees the system.

The three operating modes for logical input devices that GKS knows specify who (the user or the application program) has the initiative: SAMPLE input is acquired directly by the application program; REQUEST input is produced by the user in direct response to the application program; EVENT input is generated asynchronously by the user. They work as follows:

(1) SAMPLE. The application program samples the state of a particular device. GKS does not wait for a trigger, but instead returns the device's value at the time of the call.

(2) REQUEST. The application program requests input from a particular logical device. The logical device is started, using any beginning values specified in its initialization. As it is operated, the prompt/echo type determines what is shown on the screen. Finally, when the user triggers the end of the input, the results are returned to the application program.

(3) EVENT. The application program examines an event queue, which describes completed input events obtained from any input device that are in event mode. An event is entered in the queue when the user triggers its completion.

As we already mentioned in Section 9.6, the user interface consists of two parts: the presentation language and the action language. We will discuss some of the design considerations and implementation techniques used to build the user interface of CAR. In doing so we will concentrate on the presentation language and the action language.

12.2. *Presentation language*

The basic question we have to solve when designing the user interface is how to present information on the screen in the most effective manner, i.e. the manner that promotes the most effective interaction between user and computer. Problems in information display generally relate either to the *representation of objects and data*, i.e. the graphical representation of each of the items that appear on the screen, or to the *overall layout* of the information on the screen.

Before addressing the question of representing problem instances and solutions, we first take a closer look at the data we want to represent. In vehicle routing and scheduling all data relate to addresses, vehicles or the underlying network. The system maintains a large data base with all these data. For presentational convenience we assume that we are using a relational data base. When discussing a relational data base we will use the terminology found in Date [1981]. Let us introduce the relation 'address' with a number of attributes which will then be used for all our examples:

```
RELATION ADDRESS (
     IDENTIFICATION
     COORDINATES
     TYPE
     DEMAND
     TIME WINDOW
     ARRIVAL TIME
     WAITING TIME
     DEPARTURE TIME
     SUCCESSOR
     ASSOCIATED VEHICLE )
```

Let us stress again that this is intended only as a conceptual tool and has little to do with the actual implementation. The interesting point of using the relational data base model is that all of the questions on representations can be viewed as part of the query language. Consider for instance the following query:

```
SELECT  *
FROM    ADDRESS
WHERE   IDENTIFICATION = identification
```

Simple qualified retrieval queries like this are embedded for instance in the functions **INFO ADDRESS** and **INFO VEHICLE**, that show the requested data alphanumerically on the screen. This type of information is of course of limited use when we actually want to solve routing and scheduling problems. A somewhat more complicated query might look like:

```
SELECT  COUNT(*)
FROM    ADDRESS
WHERE   TYPE = DELIVERY
AND ASSOCIATED VEHICLE = identification
```

or

```
SELECT  SUM(DEMAND)
FROM    ADDRESS
WHERE   TYPE = DELIVERY
AND ASSOCIATED VEHICLE = identification
```

These queries, that use built-in functions in the 'select' clause, are embedded in functions like INFO CLUSTER and INFO ROUTE.

The four representations that CAR supplies are based on queries that we feel are very important. For instance, the spatial representation in the cluster phase is based on the following query:

```
SELECT  COORDINATES, TYPE, ASSOCIATED VEHICLE
FROM    ADDRESS
```

We could of course present these data alphanumerically in tabular form, but because we have a graphical display we can show data with graphics rather that with characters. We represent the items of the ADDRESS TYPE domain set, **depot**, **delivery** and **collection**, as follows: an address of type **depot** by a large star, an address of type **delivery** by a small square, and an address of type **collection** by a small diamond. In addition, we color the addresses according to their associated vehicle, where every vehicle has its own color, and use the coordinates to plot them on the screen. The spatial representation in the routing phase is based on the following query:

```
SELECT  COORDINATES, TYPE, ASSOCIATED VEHICLE, SUCCESSOR
FROM    ADDRESS
```

The only difference with the query that led to the primary display in the cluster phase is the addition of the **successor** attribute. The **successor** attribute is represented by a line segment from the coordinates of the address to the coordinates of the successor address with the exception that a line segment is not shown when one of the addresses has type **depot**.

The examples given above show that the issue of representing data, or a plan, boils down to two questions:

(1)What are the interesting queries?

(2)How do we represent the attributes graphically?

In addition to its primary (spatial) representation CAR knows three secondary representations. The temporal representation of a route is based on the following query:

```
SELECT  IDENTIFICATION, TIME WINDOW, ARRIVAL TIME,
    WAITING TIME, DEPARTURE TIME
FROM    ADDRESS
WHERE   ASSOCIATED VEHICLE = identification
ORDER   BY ARRIVAL TIME
```

For this query we chose to use a combination of an alphanumerical table and some graphical representations. Each row in the table will show the selected attributes. The identification and arrival time are represented alphanumerically, the time window as a yellow bar, waiting time, if any, as a green bar, the difference between departure time and latest service time, if greater than zero, as a red bar and the arrival time as a blue dot. Note that the arrival time attribute is represented twice! The load representation is based on the query:

```
SELECT   IDENTIFICATION, TYPE, DEMAND
FROM     ADDRESS
WHERE    ASSOCIATED VEHICLE = identification
ORDER    BY ARRIVAL TIME,
```

and the text representation on the query:

```
SELECT   IDENTIFICATION, SUM(DEMAND), ARRIVAL TIME,
     WAITING TIME, DEPARTURE TIME, SUM(DISTANCE),
FROM     ADDRESS
WHERE    ASSOCIATED VEHICLE = identification
ORDER    BY ARRIVAL TIME.
```

As we have seen above we have used two ways to distinguish different objects graphically. These are both applications of general techniques: different styles to display output primitives (square, star and diamond), and different colors. Both techniques have a decreasing effect if the objects to which they are applied become smaller.

Once we have decided on the representations we will use, we have to deal with the overall layout of the screen. This amounts to deciding how to use the limited screen area. The scarcity of the screen space is often exacerbated by the need to accommodate menus, prompts and other control objects on the screen, that are there to assist the control of the program. CAR divides the screen in three regions, which contain a *problem representation*, *commands*, and *alphanumerical information*, respectively. This division into regions makes it possible for the user to view the result of two queries simultaneously, one in the problem representation region and one in the alphanumerical information region.

As we have already mentioned above, one of the most vexing problems in information display is that of dealing with the limited capacity of the display. If problem instances get larger, reducing the overall size of the picture to make it fit in the graphical representation region is usually not effective; the screen clutter can make it hard or impossible for the user to find the information he needs. The system should therefore provide commands that allow the user to enlarge or reduce the picture size, so that he can see an overall view or a detailed view as needed. Commands to pan around a detailed view will also be useful. CAR offers the user three screen handling utilities: **ZOOM**, **FULL**, and **(IN)VISIBLE**.

These functions can be easily implemented by using the segment attributes and normalization transformations. As each trip is stored in a separate segment (see below), changing the visibility of a trip amounts to changing the visibility attribute of the associated segment. Zooming is only slightly more complicated. We have to take care of both output primitives that still have to be created and output primitives that already exist. Zooming is basically selecting part of the current visible picture and enlarging it. This can be easily accomplished by defining the appropriate normalization transformation. Let W_1 be the current window on our picture and V be the associated viewport. If we want to zoom in on part W_2, the appropriate normalization transformation is defined precisely by W_2 and V (see Figure 13).

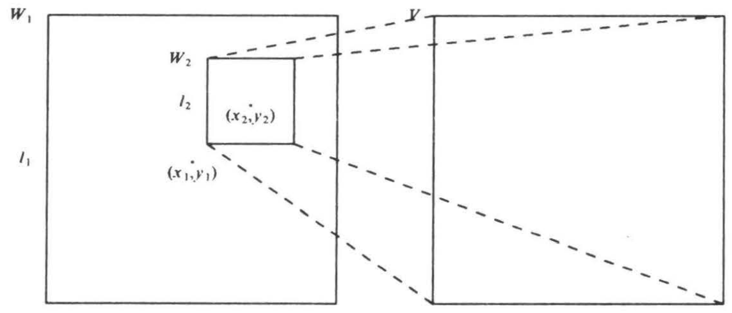


Figure 13. Normalization transformation for ZOOM.

For already existing output primitives, stored in segments, we have to adjust the segment transformation accordingly. The transformation is given by:

$$\begin{pmatrix} \frac{l_1}{l_2} & 0 & x_1 - \frac{l_1}{l_2}x_2 \\ 0 & \frac{l_1}{l_2} & y_1 - \frac{l_1}{l_2}y_2 \\ 0 & 0 & 1 \end{pmatrix}$$

In addition we enable clipping in order to make sure that the graphical representations are only visible inside the graphical information area.

12.3. *Action language*

The action language consists of all the user actions that control the application program and the syntactic and semantic rules that allow command sequences and disallow others, and that assign meaning to some command sequences and not to others. Several considerations apply to the design of the action language:

(a) *Consistency*. If all commands adhere to a common structure, the user will find them easier to remember and to invoke without error.

(b) *Command abort*. An action language should provide a graceful way for a user to change his mind. He may select a command, start to enter its arguments, and then decide to do something else instead.

(c) *Error handling*. Mistakes in entering commands must be treated carefully by the command interpreter. You will usually want to inform the user of an error and allow selection of a new command.

We have chosen for a menu driven interaction mode. Menu driven interaction has the advantage that the full range of options available to the user at any stage is plainly displayed and it prevents the user from making selections outside this range, and hence solves the problem of erroneous commands. The division of commands into subsets enables us to reduce the space of the screen needed to display the commands as at any stage at most two of the menus are displayed together. Menus are implemented as segments with a separate pick identifier for each option. The visibility attribute of the segment allows us to make a menu pop-up and disappear.

As we have seen above, the availability of a graphical display allows us to represent the result of a query to the data base in various ways. The query itself is also different from the ones found in standard query languages in data base environments. Most of the queries presented require as argument either an address or a vehicle identification (shown in lower case in the queries). A user can enter an identification by identifying the corresponding address on the screen. In CAR this is implemented by use of the pick logical input device, which returns the name of the segment and the pick identifier of the output primitive picked. This second level of naming is provided in GKS to reduce the segment overhead for applications where a great number of picture parts need to be distinguished for input but the need for manipulation is less important. In the primary representation this is exactly the situation. There is a large number of addresses that need to be distinguished for input, divided into a relatively small number of trips. Therefore the natural thing to do is to create a segment for each trip and give each of the addresses that make up the trip a different pick identifier. As an immediate consequence, a call of the request pick function uniquely determines both the address and the trip of which it is a member. (In addition, all the free addresses are also in a separate segment.)

We only have input in REQUEST mode.

Another feature of the command input is the fact that every command has been implemented as an 'infinite loop', that is, we assume that the user wants to perform the same command over and over again unless he explicitly states otherwise, by issuing another command. This has the effect that if the user wants information on several addresses he only has to give the **INFO ADDRESS** command once.

13. Conclusion

To summarize our views, we find that interaction can play a vital role in complex planning situations by integrating human insight and formal models. Many planning problems are too hard and at the same time too soft to be amenable to solution by purely algorithmic techniques. A variety of evaluative and generative models, meaningful representations of problem instances and solutions, and a uniform set of actions to manipulate all these are the main constituents of an IPS which realizes functional flexibility, ease of use and robustness.

If we contrast traditional OR with the above presented concept of an IPS, we find that on the practical level decision making is replaced by decision support, and on the technical level algorithms are no longer as prominent as they were. The most visible part of an IPS is the user interface. Its only purpose, however, is to create the opportunity to manipulate information in a convenient way. Whether information and manipulation make sense depends on the context, which consists of the practical planning situation on the one hand and the formal models and methods on the other. One might say that the role of information technology pertains to the form, while practice and its abstractions provide the substance. For the OR researcher, an IPS is like the wooden horse of Troy: it enables him to disguise his weapons in an attractive fashion and to bring them closer to practice.

PART III: EXPERTISE

14. INTRODUCTION

As we have seen in Part I, man centuries have been devoted to the development of optimization and approximation algorithms for vehicle routing and scheduling problems. This interest is due to the practical importance of effective and efficient methods for handling physical distribution situations as well as to the intriguing nature of the underlying combinatorial optimization models. The great variety of vehicle routing and scheduling problems in practice and the large number of existing algorithms make it difficult for an unexperienced distribution manager, and even for an experienced one, to select a method that is well suited for his specific situation. In order to facilitate this decision process, we propose to develop a model and algorithm management system that provides support in modeling problem situations and in suggesting algorithms that might be applicable to the resulting models.

The system will represent and manipulate information at three different levels. At the first level, there is the *real-life problem situation*. It may contain many aspects that are not relevant for the selection of a solution method. At the second level, there is the *abstract problem type*. It is obtained from the real-life problem situation by determining and modeling the relevant entities that describe it in terms of decisions, objectives and constraints. At the third level, there are the *algorithms*. One that appears to be suitable in the situation at hand is selected or constructed.

The knowledge and expertise that must be built into the system concern two different issues. On the one hand, there is the knowledge and expertise that is applied to obtain an abstract representation of the problem situation. On the other hand, there is the knowledge and expertise that is applied to choose from among the multitude of available algorithms one that is appropriate for this model.

There is a vast literature on vehicle routing and scheduling that contains the knowledge and expertise of either type. This is one interesting aspect of the project: the knowledge exists, and the question is how to formalize its use. In order to meet this challenge, we will have to create a vocabulary for representing the knowledge and to design inference algorithms for manipulating it.

In Chapter 15, we propose a language to define abstract problem types in vehicle routing and scheduling and illustrate its use on a number of examples. In Chapter 16, we outline the components of the system that select or construct a suitable algorithm for a problem type.

15. Classification

It is often difficult to keep track of all the available information on a problem class, even if the class is well structured and the information is of very elementary kind. Lageweg, Lenstra, Lawler, and Rinnooy Kan [1981, 1982] built a specialized inference engine in order to be able to keep track of the complexity results for a class of 4536 deterministic machine scheduling problems. Their main purpose was to determine the complexity status of each of these problems: solvable in polynomial time, *NP*-hard, or open. The resulting MSPCLASS system, using simple inference rules and straightforward knowledge on problem transformations, is able to deduce listings of essential results: maximal easy problems, minimal and maximal open problems, and minimal hard problems. To construct these listings by hand would be a very tedious task.

While the system we propose to develop calls for the synthesis of a massive amount of knowledge on vehicle routing and scheduling, achieving such a synthesis is already a worthwhile purpose in itself. We hope that the classification scheme which is presented in this chapter is a step in this direction.

There already exist classification schemes for other problem areas in operations research. Conway, Maxwell, and Miller [1967] introduced a four-parameter notation to classify deterministic and stochastic scheduling problems. Graham, Lawler, Lenstra, and Rinnooy Kan [1979] extended and modified this system for the class of deterministic machine scheduling problems; their scheme formed the basis for the specialized inference engine mentioned above. Handler and Mirchandani [1979] classified a limited class of location problems. Bodin and Golden [1981] outlined a classification scheme for vehicle routing and scheduling problems, and Ronen [1987] lists a number of practical problem characteristics in this area.

15.1. *The definition language*

A number of vehicles, stationed at one or more depots, have to serve a collection of customers in such a way that given constraints are respected and a given objective function is optimized. To define one such problem type in a formal way, our language uses four fields. The first field describes the characteristics and constraints that are relevant only to single addresses (customers and depots). We prefer the term 'address' to 'customer' because of the great variety of customer types: apart from the usual single-address customer, there is also the customer corresponding to an origin-destination pair or to all the addresses located on a street segment. The second field specifies the characteristics relevant only to single vehicles. The third field contains all problem characteristics that cannot be identified with single addresses or vehicles. The fourth field defines one or more objective functions.

A fifth field may be added to describe additional information about a specific class of problem instances. Although such information does not belong to the model as defined in our four fields, it might still be useful for the selection of a suitable algorithm. For example, it might be helpful to know the average number of addresses that are to be assigned to one vehicle. The specification of this field has been postponed until the development of the system is at a more advanced stage.

All the elements in our problem definition are in principle unidimensional. However, superscripts can be added to indicate multidimensional constraints. For example, cap_i indicates that vehicle load is bounded from above in one dimension, which may be, e.g., volume or weight; cap_i^2 indicates a two-dimensional capacity constraint, which may mean that upper bounds on both volume and weight are to be taken into account.

The classification language consists of a set of rules that define allowable structures. Each rule defines a nonterminal symbol in terms of other nonterminal symbols (fields, subfields, and elements) and terminal symbols (values of elements, or 'tokens'); the symbol $\vee$ is used to represent an exclusive or. Each nonterminal symbol is enclosed in angular brackets. Each token is followed by a comment on its interpretation between square brackets. The token $\circ$ indicates the empty symbol; it is used to indicate a default value, which is usually either the simplest or the most frequently occurring value.

Each problem type in the class under consideration is defined by a number of tokens, some of which may be equal to $\circ$. For notational convenience, two successive tokens are separated by a vertical bar if they belong to different fields and by a comma if they belong to the same field and are both not equal to $\circ$.

Note that we have chosen for a brief verbal interpretation of each token rather than for a complete definition in mathematical terms. A formal approach, while possible and useful in itself, would distract our attention from the main purpose of this paper.

```
<classification> ::=

        <addresses>
        <vehicles>
        <problem characteristics>
        <objectives>
```

15.1.1. *Addresses*

The first field defines the characteristics that can be associated with single addresses. All the addresses will be located on a network $G = (V,E)$ with a set V of nodes and a set E of (undirected) edges and (directed) arcs. There are four subfields.

The first subfield specifies the *number of depots*. There are single-depot problems and problems where the number of depots is given as part of the problem instance.

The second subfield specifies the *type of demand*. There are three parts. First the location of the demand: ∘ indicates that the customers are located on the nodes, EDGE indicates that the customers are located on the edges (arcs) of the network, MIXED indicates that the customers are located on both the nodes and edges (arcs), and TASK indicates the case that each customer corresponds to an origin-destination pair; the load is picked up at the origin address and delivered to the destination address. The second part of the subfield specifies if all the demands are of the same type (all deliveries or all collections) or not (mixed deliveries and collections). The third part specifies the nature of the demand: deterministic or stochastic.

The third subfield specifies the *address scheduling constraints*, i.e., the temporal aspect of the demand. Either there are no temporal constraints, or the departure time is fixed (fixed schedule), or the departure time is restricted to a single interval (single time windows) or to a set of intervals (multiple time windows).

The final subfield specifies the *address selection constraints*. There is a basic distinction between two problem classes. In the first class, a single plan is to be made for the given collections of addresses and vehicles. There are three subclasses: all addresses must be visited; a given subset of addresses must be visited and the others may be visited if it is profitable; or the addresses are partitioned into subsets and at least one address in each subset must be visited. In the second class, a number of plans is to be made over a certain time period, during which the addresses must be visited with given priorities or at given frequencies. This requirement is open to various specifications. It may lead to problems with a longer time horizon, in which a weekly allocation problem is to be solved before the daily routing problem is defined.

<addresses> ::=

<number of depots>
<type of demand>
<address scheduling constraints>
<priority constraints>

<number of depots> ::= $1 \vee l$

1	[one depot]
l	[specified as part of the problem instance]

<type of demand> ::= $<\alpha_1> <\alpha_2> <\alpha_3>$

$<\alpha_1>$ ::= $\circ \vee$ EDGE $\vee$ MIXED $\vee$ TASK

∘	[node routing]

EDGE	[edge routing]
MIXED	[mixed routing (nodes and edges)]
TASK	[task routing]

$<\alpha_2> ::= \circ \vee \pm$

$\circ$	[either all deliveries or all collections]
$\pm$	[mixed deliveries and collections]

$<\alpha_3> ::= \circ \vee \sim$

$\circ$	[deterministic demand]
$\sim$	[stochastic demand]

<address scheduling constraints> $::= \circ \vee fs_j \vee tw_j \vee mw_j$

$\circ$	[no scheduling constraints]
fs_j	[fixed schedule]
tw_j	[single time windows]
mw_j	[multiple time windows]

<address selection constraints> $::= \circ \vee$ *subset* $\vee$ *choice* $\vee$ *period*

$\circ$	[single plan; all addresses must be visited]
subset	[single plan; a given subset of addresses must be visited]
choice	[single plan; at least one address in each subset of a given partition must be visited]
period	[a number of plans over a given time period is to be made]

15.1.2. *Vehicles*

The second field defines the characteristics of the vehicles and their routes. There are three types of information in this field: the number of vehicles, physical characteristics of the vehicles, and temporal constraints on a route.

The first subfield specifies the *number of vehicles*: the number of vehicles is a constant, specified as part of the problem type, or a variable, specified as part of the problem instance. The symbol '=' can be used to indicate that all vehicles must be used.

The second and third subfields specify the *physical characteristics* of the vehicles: the capacity and the presence of compartments. The fleet can be *homogeneous* (all vehicles have the same capacity) or *heterogeneous*. There are two types of compartmentalized vehicles. Some vehicles have *interchangeable* compartments. These can be used to separate incompatible commodities such as chickens and foxes. Other vehicles have *dedicated* compartments, each used to store one type of good; e.g., frozen meals and fresh vegetables must be kept in separate dedicated compartments.

The fourth and fifth subfields specify *temporal constraints*. There can be availability intervals for the vehicles and upper bounds on the duration of the routes.

<vehicles> ::=

<number of vehicles>
<capacity constraints>
<commodity constraints>
<vehicle scheduling constraints>
<route duration constraints>

<number of vehicles> ::= $<\beta_1> <\beta_2>$

$<\beta_1>$ ::= $\circ \vee =$

$\circ$	[at most β_2 vehicles can be used]
$=$	[all β_2 vehicles must be used]

$<\beta_2>$ ::= $c \vee m$

c ($c \in \mathbb{N}$)	[c vehicles]
m	[specified as part of the problem instance]

<capacity constraints> ::= $\circ \vee cap \vee cap_i$

$\circ$	[no capacity constraints]
cap	[vehicles with identical capacities]
cap_i	[vehicles with different capacities]

<commodity constraints> ::= $\circ \vee sep \vee ded$

$\circ$	[no compartments]
sep	[vehicles have interchangeable compartments]
ded	[vehicles have dedicated compartments]

<vehicle scheduling constraints> ::= $\circ \vee tw \vee tw_i$

$\circ$	[no scheduling constraints]
tw	[identical time windows for vehicles]
tw_i	[different time windows for vehicles]

<route duration constraints> ::= $\circ \vee dur \vee dur_i$

$\circ$	[no route duration constraints]

dur	[identical upper bounds on route duration]
dur_i	[different upper bounds on route duration]

15.1.3. *Problem characteristics*

The third field defines the network underlying the problem, the service strategy, and constraints on the relations between addresses and vehicles.

The first subfield specifies the properties of the *network* (directed, undirected or mixed) and of the *travel time matrix* (satisfying the triangle inequality or not).

The second subfield specifies the *service strategy* chosen by the user. There are four types of strategic decisions.

(1) The first type allows or disallows splitting of the customer demand. A priori splitting of demand occurs when it is decided at the outset that the demand may be satisfied by more than one visit to the customer. A posteriori splitting of the demand occurs in the case of stochastic demand when, once on the customer premises, the driver discovers that the demand is larger than foreseen and decides not to satisfy the demand completely during that visit.

(2) In the case of node routing with mixed deliveries and collections, the user can choose for *backhauling*, i.e., delivering first to empty the vehicle and then collecting loads on the way back to the depot. In the case of task routing, the user can choose for *full load* routing, i.e., only one load can be in the vehicle at any time.

(3) In most cases a vehicle performs at most one route per period, but the user can allow more than one route per vehicle.

(4) Usually vehicles are restricted to start and finish at the same depot, but this can be relaxed and the user can allow multi-depot routes.

The other subfields specifies the possible *relations* between two addresses, between an address and a vehicle, or between two vehicles. Such relations are caused by a number of very different factors, and enumerating these here would not be feasible. Instead of describing the underlying factors, we have chosen to specify the restrictions caused by them.

The best known of these relations is the *precedence constraint* between two customers: the vehicle must visit one customer before visiting the other. Note that these precedence constraints have nothing to do with the implicit precedence constraints in the origin-destination pairs in TASK routing, because we view such an origin-destination pair as a single customer.

Most of the other relations are *inclusion* and *exclusion restrictions*. It may be that an address must be served from a given depot, must be allocated to the same route as another address, or must be visited by a given vehicle. For example, an address-vehicle inclusion restriction occurs if the vehicle must be equipped with an unloading device because the customer has no delivery dock. It may also be that an address should *not* be served from a given depot, should *not* be allocated to the same route as another address, or should *not* be visited by a given vehicle.

The last type of restriction is *vehicle synchronization*, occurring when two or more vehicles must exchange loads or assist each other.

<problem characteristics> ::=

<type of network>
<type of strategy>
<address-address restrictions>
<address-vehicle restrictions>
<vehicle-vehicle restrictions>

<type of network> ::= $<\gamma_1>$ $<\gamma_2>$

$<\gamma_1>$ ::= $\circ \vee \Delta$

$\circ$	[general costs]
Δ	[the costs satisfy the triangle inequality]

$<\gamma_2>$ ::= $\circ \vee$ *dir* $\vee$ *mix*

$\circ$	[undirected network]
dir	[directed network]
mix	[mixed network]

<type of strategy> ::= $<\delta_1>$ $<\delta_2>$ $<\delta_3>$ $<\delta_4>$

$<\delta_1>$ ::= $\circ \vee / \vee \div$

$\circ$	[splitting of demand not allowed]
/	[a priori splitting of demand allowed]
$\div$	[a posteriori splitting of demand allowed]

$<\delta_2>$ ::= $\circ \vee$ *back* $\vee$ *full*

$\circ$	[no backhauling or full loads required]
back	[backhauling, in case of node routing]
full	[full loads, in case of task routing]

$<\delta_3>$ ::= $\circ \vee \geqslant 1R/V$

$\circ$	[at most one route per vehicle]
$\geqslant 1R/V$	[more than one route per vehicle allowed]

$<\delta_4>$ ::= $\circ \vee \geqslant 1D/R$

$\circ$	[a route starts and finishes at the same depot]
$\geqslant 1D/R$	[multi-depot routes allowed]

<address-address restrictions> ::= $<\epsilon_1>$ $<\epsilon_2>$ $<\epsilon_3>$

$<\epsilon_1> ::= \circ \vee prec$

$\circ$	[no precedence constraints]
prec	[precedence constraints]

$<\epsilon_2> ::= \circ \vee DA$

$\circ$	[no depot-address restrictions]
DA	[depot-address restrictions]

$<\epsilon_3> ::= \circ \vee AA$

$\circ$	[no address-address restrictions]
AA	[address-address restrictions]

<address-vehicle restrictions> :: = $<\zeta_1> <\zeta_2>$

$<\zeta_1> ::= \circ \vee DV$

$\circ$	[no depot-vehicle restrictions]
DV	[depot-vehicle restrictions]

$<\zeta_2> ::= \circ \vee AV$

$\circ$	[no address-vehicle restrictions]
AV	[address-vehicle restrictions]

<vehicle-vehicle restrictions> $::= \circ \vee VV$

$\circ$	[no vehicle-vehicle restrictions]
VV	[synchronization between vehicles needed]

15.1.4. *Objectives*

The fourth field defines the objective functions. To specify an objective function we introduce five quantities.

The travel and service time of vehicle i, i.e., the tour duration of its route, will be denoted by T_i. With this quantity we can express the standard objectives found in the vehicle routing and scheduling literature: minimization of the total travel and service time, and minimization of the span of a solution.

To be able to express more realistic objective functions, we introduce a vehicle cost function C_i, an address cost function c_j, a vehicle penalty function P_i, and an address penalty function p_j. A vehicle cost function can be used to model situations where, in addition to routing and scheduling, it is also required to determine the fleet size and mix. An address cost function allows the modeling of costs incurred due to deviation of a preferred service level. The penalty functions enable the modeling of costs incurred due to the violation of constraints. Sometimes it is allowed to violate constraints at a certain cost, if it is profitable; driver overtime is an example.

In practice, the problems often have a composite objective function. The user can specify this by listing the components of the objective function in order of decreasing importance. At the other end, our schemes also leaves the possibility open that no objective is specified, so that the problem reduces to a feasibility question.

As an addendum to the rules below, we note that, in the case of a single vehicle, the operator sum or max and the subscript i are dropped in the objectives related to routes and vehicles.

<objectives> ::= <objective> $\vee$ <objectives> <objective>

<objective> ::= $\circ$ $\vee$ <operator> <function>

<operator> ::= sum $\vee$ max

sum	[minimize the sum of the cost function values]
max	[minimize the maximum cost function value]

<function> ::= $T_i \vee C_i \vee P_i \vee c_j \vee p_j$

T_i	[route duration]
C_i	[vehicle costs]
P_i	[vehicle penalty]
c_j	[address costs]
p_j	[address penalty]

15.2. *Examples*

In this section, fourteen problems taken from the vehicle routing and scheduling literature are classified using the scheme given in the previous section. This presentation has a twofold purpose. First, the examples illustrate the use of the classification scheme. Secondly, they show that a broad class of problems, including very practical ones, can be handled.

The examples are all presented in the same format. The difference in style of the various problem descriptions is due to the fact that we have quoted the source texts throughout.

Example 1. $1|1|\,|T$ [Jaques 1859]

'In this new Game (invented by Sir William Rowan Hamilton, LL.D., &c., of Dublin, and by him named *Icosian*, from a Greek word signifying "twenty") a player is to place the whole or part of a set of twenty numbered pieces or men upon the points or in the holes of a board ... in such a manner as always to proceed *along the lines* of the figure'. [The board is a planar representation of the pentagonal dodecahedron].

Example 2. $1\,|\,1\,|\,\Delta\,|\,T$ [Menger 1930]
'Wir bezeichnen als *Botenproblem* (weil diese Frage in der Praxis von jedem Postboten, übrigens auch von vielen Reisenden zur lösen ist) die Aufgabe, für endlichviele Punkte, deren paarweise Abstände bekannt sind, den kürzesten die Punkte verbindenden Weg zu finden.' [We call this the *messenger problem* (because in practice the problem has to be solved by every postman, and also by many travelers): finding the shortest path joining all of a finite set of points, whose pairwise distances are known.]

Example 3. $1,\mathrm{EDGE}\,|\,1\,|\,|\,T$ [Guan 1962]
'When the author was plotting a diagram for a mailman's route, he discovered the following problem: "A mailman has to cover his assigned segment before returning to the post office. The problem is to find the shortest walking distance for the mailman." This problem can be reduced to the following: "Given a connected graph in the plane, we are to draw a continuous graph (repetition permitted) from a given point and back minimizing the number of repeated arcs."'

Example 4. $1\,|\,m,cap\,|\,|\,\mathrm{sum}T_i$ [Dantzig and Ramser 1959]
'The Traveling Salesman Problem may ... be generalized by imposing the condition that specified deliveries q_i be made at every point P_i (excepting the terminal point). If the capacity of the carrier $C \geqslant \Sigma_i q_i$, the problem is formally identical with the Traveling-Salesman Problem in its original form ... the Truck Dispatching Problem is characterized by the relation $C \ll \Sigma_i q_i$. ... For simplicity of presentation it will be assumed that only one product is to be delivered and that all trucks have the same capacity C.'

Example 5. $1,tw_j\,|\,1\,|\,\Delta\,|\,T$ [Savelsbergh 1986]
'In the TSPTW [traveling salesman problem with time windows] we are given in addition to the travel time $t_{i,j}$ for each pair of vertices $i,j \in V$, for each vertex i a *time window*, denoted by $[e_i, l_i]$, where e_i specifies the earliest service time and l_i the latest service time. The latter bound is strict in the sense that departing later than l_i is not allowed and causes the tour to become infeasible, whereas arriving earlier than e_i does not lead to infeasibility but merely introduces waiting time at vertex i. ... Minimize the completion time of the tour'.

Example 6. $1,choice\,|\,1\,|\,|\,T$ [Laporte and Nobert 1983]
'... we consider a generalization of the TSP in which each city is replaced by a set of cities. More precisely, we consider a city (city 0) used as the trip starting and ending point, and also n sets of cities $(S_1, S_2, \ldots, S_n)$. The problem ... consists of finding the shortest route through city 0 and at least one city taken from each S_k. As in the TSP each city may be visited only once.'

Example 7. $1,\pm\,|\,m,cap\,|\,back\,|\,\mathrm{sum}T_i$ [Goetschalckx and Jacobs]
'The linehaul-backhaul problem is an extension of the VRP involving both delivery and pick-up points. Linehaul (delivery) points are sites which are to receive a quantity of goods from the single central DC [distribution center].

Backhaul (pick-up) points are sites which send a quantity of goods back to the DC. The quantities to be delivered and picked up are fixed and known in advance. There exists a homogeneous fleet of vehicles each of which is assumed to have a fixed capacity of some weight or volume. The crucial assumption is that all deliveries must be made before any pick-ups can be made. This is caused by the fact that the vehicles are rear-loaded and the rearrangement of the loads on the truck at delivery points is not deemed feasible. Hence, a feasible solution to the problem consists of a set of routes where all deliveries for each route are completed before any pick-ups are made and the vehicle capacity is not violated either by the linehaul or backhaul points assigned to the route. The objective is to find such a set of routes which minimizes the total distance traveled.'

Example 8. 1,*period* | m,cap_i,dur_i | | sumT_i [Christofides and Beasley 1984]
'In the period vehicle routing problem ... the problem is to design a set of routes for each day of a given (p-day) period. Each customer may require a number of visits by a vehicle during this period. If a customer requires k (say) visits during the period, then the visits may only occur in one of a given number of allowable k-day *combinations*.'

Example 9. 1,EDGE,*subset* | m,*dur* | / | sumC_i,sumT_i [Stern and Dror 1979]
'This paper is addressed to the problem of collecting data on household consumption of electricity for billing purposes ... Each reader has a maximum ... workshift time limit of 5 hr established by union regulations. ... Figure 2 [not reproduced here] presents a graph that corresponds to the network of streets. The heavy lined edges represent those streets that contain meters and must be covered by the meter readers while moving from house to house. Dotted edges represent streets that contain no meters but may be traversed as connecting streets if required. There are no oneway streets ... as readers proceed by foot and walking can be done on sidewalks in any direction. ... working tours may begin and end at intermediate locations of an edge. ... The primary objective is to find the minimum number of working tours needed to cover the required edges in the graph. A secondary objective, given the minimum number of tours, is to find the routes of each tour such that the total length traversed is minimal.'

Example 10. l,tw_j | m,cap_i,*sep*,tw_i | $\geqslant 1R/V,AV$ | sumT_i,sumP_i,sump_j [Brown and Graves 1981]
'The dispatchers, located at a central national order processing facility, must each handle several bulk terminals ... Drivers are domiciled with company-owned vehicles at the terminal ...

Delivery vehicles possess a wide variety of features relevant to their use in the dispatch. A model truck and rig ... is equipped with multiple, isolated compartments. Each compartment has a volumetric capacity specific to the density of the product contained. ...

Vehicle operating costs are specified for each proprietary truck on a customer-by-customer basis as a function of mileage and standard delivery time. Nonproprietary truck costs may also be simple functions of actual delivery time and mileage, or may be fixed point-to-point charges for each trip depending upon operating region and contract terms and duration. Each vehicle is assigned a sequence of loads for a shift with the duration of each shift determined by driver availability, vehicle availability, and contract terms. Overextension of vehicle shifts leads to overtime labor costs ...

Each order typically includes three products, usually grades of gasoline, jointly constituting a complete truck load ... and additional data regarding special equipment requirements (such as special couplings, pumps, an unmarked truck, and so forth).'

Example 11. 1,TASK | 1 | | T [Psaraftis 1983]
'In the DARP [dial-a-ride problem]'s generic version, a vehicle, initially located at point A, is called to service N customers, each of whom wishes to travel from a distinct origin to a distinct destination, and then return to A so that the total length of the route is minimized.'

Example 12. $1,\pm,mw_j,subset \mid m,cap_i,tw_i \mid \geqslant 1R/V,AV \mid \max T_i,\mathrm{sum}T_i$ [Anthonisse, Lenstra, and Savelsbergh 1987]
'CAR is an interactive software package ... for distribution problems with the following characteristics.
- There is a single depot where several vehicles, possibly with different capacities, are stationed.
- The commodity to be transported is homogeneous in the sense that the allocation of commodities to vehicles is restricted only by the vehicle capacities.
- A vehicle can make several trips a day.
- A vehicle has a time window that specifies its availability.
- There may be both collections and deliveries. Vehicles depart from the depot with the commodities to be delivered and eventually return to the depot with the collected commodities. Anything collected on the way is transported to the depot. Collections and deliveries may occur in any sequence on the same trip.
- An address may impose restrictions on the capabilities of the vehicle visiting it. For example, it may require special loading equipment.
- An address has one or more time windows within which service must take place.
- An address can have a priority, indicating that it must be visited.
- An address is to be visited by at most one vehicle.'

Example 13. $1,\sim,mw_j,period \mid m,cap_i,tw_i \mid /,\geqslant 1R/V,AV \mid \mathrm{sum}c_j,\mathrm{sum}T_i,\mathrm{sum}C_i$ [Bell, Dalberto, Fisher, Greenfield, Jaikumar, Kedia, Mack, and Prutzman 1983]
'The degree of freedom available to distribution management at Air Products is greater than in any other industry. They decide when to supply a customer based on the inventory level in the customer tank, how much to deliver, how to combine the different loads on a truck and how to route the truck. Thus inventory management at customer locations is integrated with vehicle scheduling and dispatching. ...

Because of the uncertainty in customer demand, ... inventory must be maintained at a specified safety-stock level. Customers are not open for delivery on every day of the week or during every hour of the day and trucks must make their deliveries within certain prescribed time windows which can vary among customers. The trucks in the fleet differ in characteristics such as capacity and operating costs. ... Finally, some trucks are incapable of serving certain customers because they are too big, require an external power source for an electric pump, and so forth. The availability of trucks, drivers and product is limited. ...

The costs that must be considered in scheduling include driver pay, tolls, and vehicle-related costs such as depreciation, fuel, and maintenance. ...

The scheduling module is used daily at each depot to produce a detailed schedule for a two- to five-day horizon, with the first day's schedule being the most important one. ... The object of the model is to maximize the value of the product delivered less the costs incurred in making these deliveries.'

Example 14. 1,TASK,tw_j | m | *dir,full* | sumC_i,sumT_i [Desrosiers, Soumis, and Desrochers 1984]

'A *trip* is a productive journey which may be carried out by a vehicle. The trip i is characterized by a place of origin, a destination, a duration, a cost and a time interval $[a_i,b_i]$ during which the trip must begin. An *intertrip arc* is an unproductive (i.e., empty) journey carried out by a vehicle. The intertrip arc (i,j) goes from the end of trip i to the beginning of trip j. Its duration t_{ij} and its cost c_{ij} may include respectively the duration and cost of the trip i. A *route* is a sequence of trips and intertrip arcs carried by the same vehicle. The problem is to determine routes and schedules for all the trips so as to minimize the number of vehicles and travel costs while respecting network and scheduling constraints.'

16. Model and algorithm management

Model management and model management systems (MMS) are relatively new concepts which have emerged from the recent interest in decision support systems, expert systems, and artificial intelligence. The primary objective of an MMS can be viewed as the counterpart of that of a database management system. It provides an environment for storing, retrieving, and manipulating models. The MMS serves as a bridge linking the decision maker's environment with the appropriate models. The current design paradigm for these systems stresses the need for expert knowledge in the system along with associated knowledge-handling facilities.

The ultimate goal of the model and algorithm management system for vehicle routing and scheduling is to provide a user with a suitable algorithm for the problem situation he is facing. In order to achieve this goal, the system maintains models and algorithms, and contains inference mechanisms to manipulate them. Also, because the system is being used as a consultant, it is able to provide explanations of its line of reasoning, i.e., explain why it is asking particular questions and how it reached a conclusion, and it provides means to perform sensitivity analysis, for instance by allowing the user to attach confidence factors to his responses to the system's queries.

In the first phase, the system will establish a problem type, the characteristics of the expected problem instances and the algorithm requirements through a set of questions in a man-machine dialogue. The classification scheme discussed in the previous section is used to represent problem types. The characteristics of the expected problem instances, such as the average number of customers in a route and the average load factor of the vehicles, supplement the information embodied in the problem type. Algorithm requirements reflect the type of algorithm the user wants. For example, a user might be interested only in very fast algorithms, or in algorithms that produce an optimal solution. The characteristics of the expected problem instances and the algorithm requirements have a large impact on the inference mechanisms the system employs in the second phase. Although an important part of the system, the first phase will not be treated in detail.

In the second phase, the system will try to select or construct a suitable algorithm based on the knowledge residing in the system. The system distinguishes two classes of algorithms. The first class contains algorithms that are based on either a mathematical programming formulation or a set of recursion equations; the second class contains all the others. The reason for this differentiation is the fact that formulations in terms of mathematical programming or recursion equations often reveal information about the structure of the problem that can be used in determining which algorithm should be applied.

The system's distinction between algorithms is reflected in the organization of the knowledge. It has four different knowledge bases: a problem knowledge base (PKB), a formulation knowledge base (FKB), an algorithm knowledge base (AKB), and a general knowledge base (GKB). These knowledge bases contain formulations and algorithms for well known and well investigated problem types (such as the traveling salesman problem and the vehicle routing problem), and knowledge on formulation and algorithm construction. Together they represent the knowledge of researchers in the field and piles of literature.

Note that the problems as represented by the classifications of Chapter 15 must be viewed as abstractions that belong to the PKB, although the mathematical formalism in which they are defined is not made explicit. The FKB, however, contains representations of the problems based on a specific mathematical formalism.

The process of selecting or constructing a suitable algorithm for a given problem type proceeds as follows:

(1) If the problem is already present in the PKB, we are done. In that case, there are also associated algorithms in the AKB.

(2) If the problem is not present in the PKB, we try to identify problems in the PKB that have a 'similar' structure and use their associated formulations, if any, and associated algorithms to piece together one or more promising algorithms for the problem at hand; this is called 'analogical reasoning' in the artificial intelligence literature.

(3) If the problem is not present in the PKB and there are no problems in the PKB with a similar structure, we cannot handle the problem.

Of course, step (2) is by far the most intriguing and difficult one. There are a number of issues that need further consideration. We indicate them briefly here, and will be more elaborate in the next subsections.

Similarly structured problems. If a problem is not present in the PKB, the PKB is searched for problems with a similar structure, which will then be used to construct an algorithm for the problem at hand. In order to relate problem types to each other, we will define a metric on the problem types represented by the classifications.

Saturation and refinement. 'Almost all traditional problem-solving control structures are susceptible to saturation, the situation in which so many applicable knowledge sources are retrieved that it is unrealistic to consider exhaustive, unguided invocation. ... One useful approach to controlling saturation is by refining the set of knowledge sources retrieved, i.e., prune and reorder this set.' [Davis 1980]

In the model and algorithm management system we encounter a similar problem. Depending on our definition of 'similarity in structure', we might end up with a large set of problems that have a similar structure. And that is not where the story ends. Often, a problem does not uniquely identify one formulation and one algorithm, but instead identifies a set of associated formulations, each of which in turn identifies a set of associated algorithms. It might very well be infeasible to explore them all and we have to do pruning and reordering.

Model integration and validation. Model integration, i.e., building composite models and decomposing models into their constituent parts, is used to create a mathematical programming formulation for a problem that is not explicitly in the PKB. Some form of validation has to be performed on the obtained formulation to ensure that it deals with all the constraints of the problem.

Algorithm integration and validation. Alongside model integration, there is algorithm integration, i.e., building composite algorithms and decomposing algorithms into their constituent parts, to construct an algorithm for a problem not present in the PKB. Validation, in this case, should also test whether the algorithm obtained meets the specified algorithm requirements.

The remaining part of this chapter is divided into two sections, one on representation and one on manipulation. Both contain only basic ideas and examples and are far from being a complete blueprint of the system.

16.1. *Representation*

A major question pertaining to the system is that of representation, i.e., how to represent knowledge, formulations, and algorithms. This is a crucial issue since all manipulations, i.e., the kinds of inferences that can be made, depend directly upon this structure.

Knowledge. There are several approaches to knowledge representation. An overview of them can be found in Mylopoulos and Levesque [1984]. With regard to modeling and decision support, most knowledge representation approaches can be divided into three categories: logical, network, and frame. Our brief description of these three is based on a paper by Mylopoulos [1980].

Logical representation schemes employ the notions of constant, variable, function, predicate, logical connective, and quantifier to represent facts as logical formulas in some logic. A knowledge base, according to this view, is a collection of logical formulas which provides a partial description of reality. Modifications to the knowledge base occur with the introduction and deletion of logical formulas. So logical formulas are the atomic units for knowledge base manipulation in such schemes.

Network representation schemes, often called *semantic networks*, attempt to describe reality in terms of objects (nodes) and binary associations (labelled edges), the former denoting individuals and the latter binary relationships of the reality being modeled. According to the network representational view, a knowledge base is a collection of objects and associations, or a directed labelled graph, and modifications to the knowledge base occur through the insertion and deletion of objects and the manipulation of associations.

Frame-based representation schemes view a knowledge base as a frame system. A frame is a complex data structure for representing a stereotypical situation. The frame has slots for the objects that play a role in the stereotypical situation as well as for the relations between these objects. Attached to each frame are different kinds of information, such as how to use the frame and default values for its slots. A further feature of the frame approach is the concept of a frame system which is a collection of frames linked together by an information retrieval network. The main function of the frame system is to provide a retrieval capability for matching frames with 'reality' or some part thereof.

Frames were conceived originally for visual applications [Minsky 1975], but have been used for scheduling [Goldstein and Roberts 1979] and as major component of the 'model abstraction' concept of Dolk and Konsynski [1984].

The model and algorithm management system will probably use a hybrid scheme, in the sense that all of the above mentioned representation schemes will be present in some form.

Formulations. Most of the formulations for vehicle routing and scheduling problems are mixed integer programming models. Therefore, the system should be able to represent a mixed integer programming formulation in a manageable form. The research done on *modeling languages* for mathematical programming might prove useful in that context. Practical large-scale mathematical programming involves more than just the application of an algorithm to minimize or maximize an objective function subject to constraints. Before any optimization routine can be invoked, considerable effort must be expended to formulate the underlying model and to generate the requisite computational data structures. Modeling languages are designed to make these steps easier and less error-prone. Examples of modeling languages are GAMS [Bisschop and Meerhaus 1980] and AMPL [Fourer, Gay and Kernighan 1987]. In the model and algorithm management system an extension of AMPL could be used to represent a mixed integer programming formulation. Extensions are needed to enable model integration. Comments will no longer be comments, but sources of information for the inference engine. In case of a constraint, they will at least provide information as to which characteristic of the problem it models. Appendix A shows an AMPL model of the vehicle routing problem with time windows, $1, tw_j \mid m, cap \mid \mid \text{sum}T_i$.

An alternative to this might be *structured modeling* as proposed by Geoffrion [1986]. The overall objectives of structured modeling are to provide a formal mathematical framework, language, and computer environment for conceiving, representing, and manipulating a wide variety of models. Structured modeling has benefited significantly from ideas from modeling languages, spreadsheet modeling, and database theory. It might be more widely applicable than a modeling language because it does not restrict itself to mathematical programming models. Appendix B shows a structured modeling schema of the vehicle routing problem, $1 \mid m, cap \mid \mid \text{sum}T_i$.

Algorithms. An important notion in the description of algorithms is that of a *technique.* A technique is a building block that can be used to construct algorithms, such as Langrangean relaxation and branch and bound.

The model and algorithm management system will use *templates* to represent techniques and algorithms. Templates are based on ideas borrowed from frames and concepts [Brachman 1979], and are packets of knowledge that provide descriptions of objects and relationships of the domain being modeled. A template consists of a set of *slots* and a set of *structural descriptions.* Slots represent the conceptual parts of an object, while structural descriptions account for the relations between them. The fundamental operation performed on templates is *instantiation*, i.e., creating a specific instance of a template by filling the slots. As a consequence we can distinguish two types of templates: generic and instantiated. Generic templates represent classes of objects by describing the characteristics of the prototypical member of the class, i.e., by providing a skeleton for describing any possible instance of the class. Instantiated templates represent specific objects by instantiation of more general templates.

The model and algorithm management system will have one generic algorithm template and several generic technique templates.

An algorithm template has to provide three types of information. First, there should be general information on the algorithm: the problem it is solving and a step by step description. Secondly, there should be information that can be used by the inference mechanisms: if based on a formulation, a reference to that formulation, what parts of the algorithm deal with which constraints, whether it is an optimization or an approximation algorithm, whether there are known generalizations or special cases, and a performance analysis. Finally, there should be information that will be useful when the algorithm is suggested to the user: an English description and references to related papers in the literature.

The most important information in a technique template is how the technique specific slots interact. This can best be explained by means of an example. Consider the technique template for branch and bound. It has four technique specific slots: upper bound, lower bound, branching rule, and selection rule. The interaction between them is described in the explain structural description.

```
EXPLAIN
    ActiveSet ← OriginalProblem
    UpperBoundValue & UpperBoundSolution ← UPPERBOUND()
    while ( ActiveSet not empty ) do
    begin
        Node ← SELECTIONRULE()
        delete Node from ActiveSet
        NewNodes ← BRANCHINGRULE()
        for Node in NewNodes do
        begin
            LowerBoundValue & LowerBoundSolution ← LOWERBOUND()
            if ( LowerBoundValue ≥ UpperBoundValue )
            then
                discard Node
            else
                if ( LowerBoundSolution is feasible )
                then
                    UpperBoundValue ← LowerBoundValue
                    UpperBoundSolution ← LowerBoundSolution
                else
                    add Node to ActiveSet
        end
    end
```

Upon instantiation the technique specific slots are filled with a value, a formula, or an algorithm. Next to the technique specific slots, there are slots that provide information on the complexity and performance of the technique.

As we have seen above, upon instantiation the slots of a generic template are filled. A *slot facet* is used to specify the types of entities that can fill a slot. Possible slot facets are: *ALGORITHM, TECHNIQUE, FORMULA, VALUE, TIME, MEMORY, EMPIRICAL, WORSTCASE, AMORTIZED, PROBABILISTIC* and *EMPTY*.

Both slots and structural desriptions can have attached comments. These are used to refer to relevant papers in the literature and, in case of the technique specific slots, to indicate which problem characteristics are dealt with.

More specifically, an algorithm template will have the following slots: self slot, problem slot, discuss slot, type slot, formulation slot, explain slot, several technique or algorithm slots, complexity slot, analysis slot, special case slot, and generalization slot. The names of the slots are chosen to be almost self-explanatory. We will discuss some of them briefly. The self slot lists the steps of the algorithm. The problem slot gives the classification that describes the problem the algorithm is solving. The discuss slot contains a short English description of the problem. The type slot indicates whether it is an optimization or an approximation algorithm. In case the algorithm is based on a mathematical programming model, the model slot records where this model can be found. The complexity slot contains information on the computational requirements of the algorithm in terms of running time and memory. The analysis slot contains known worst case results, an amortized or probabilistic analysis, and empirical behavior. The generalization slot contains information on possible generalizations to other problem types together with the modifications needed to the algorithm. The special case slot contains information on special cases together with the modifications needed to the algorithm.

A technique template will have the following slots: one or more technique specific slots, complexity slot, and analysis slot, and the following structural descriptions (SD): name SD, self SD, explain SD, and discuss SD. The self SD is a list of the technique specific slots. The explain SD contains a procedural description of the interaction between the technique specific slots. The discuss SD contains an English description of the technique the template represents. Appendix C shows an algorithm and the associated techniques for the vehicle routing problem, $1 \mid m,cap \mid \mid \text{sum}T_i$.

16.2. *Manipulation*

Similaritity in structure. A very important notion in the model and algorithm management system is that of *similarity in structure.* The system will try to identify problems that have a similar structure by comparing their respective problem representations. The basic idea is that similarity in structure can be measured by looking at the differences between problem representations. Recall that the representation defined in Chapter 15 uses 26 subfields to describe a problem type and that each subfield can have a limited number of values. We introduce the weight w_{ij}^k so as to reflect the change in structure that results when the value of subfield k is changed from i to j. The weight w_{ij}^k can de defined in several ways. The simplest is to allow only two weights: *small* and *large*, which would be interpreted as follows. If two problem representations differ in only one subfield and the weight associated with the corresponding change is small, the problems are considered to have a similar structure and it is likely that formulations and algorithms for one of them can be modified in order to be of use to the other; otherwise the problems are considered to have a different structure and it is unlikely that formulations and algorithms for one of them can be modified to be of use for the other. Of course, it is possible to define more complex weight functions. For example, it is possible and maybe even necessary to define conditional weight functions, where the value of the weight w_{ij}^k depends on the value of another subfield.

To get a little better acquainted with the concept of subfield changes and the corresponding interpretation, let us consider a couple of examples.

$1 \mid 1 \mid \mid T \rightarrow 1, tw_j \mid 1 \mid \mid T$. This corresponds to adding time windows to the nodes in a traveling salesman problem, which makes the problem more difficult.

$1, TASK \mid m,cap \mid \mid \text{sum}T_i \rightarrow 1, TASK \mid m,cap \mid full \mid \text{sum}T_i$. This corresponds to allowing only full truck loads in a pickup and delivery problem. This is a simplification because it can now be modeled as a multiple traveling salesman problem.

$1 \mid m, cap \mid \mid \text{sum}T_i \rightarrow 1, \sim \mid m, cap \mid \mid \text{sum}T_i$. This corresponds to changing from deterministic demands to stochastic demands in a vehicle routing problem, which completely changes its structure.

Using the weights introduced above we are now able to compare two problem types and to conclude whether they have a similar structure. Again, there are several ways to do so; similarity can be measured by $\max_{k \in subfields} w_{ij}^k$ or $\Sigma_{k \in subfields} w_{ij}^k$. Note that the definitions of the weight values represent part of the expertise that is built into the system.

Saturation and refinement. In the field of artificial intelligence there have been two complementary responses to the problem of saturation: development of new accessing, indexing and knowledge organization schemes, and acceptance of its existence and provision of a mechanism for guiding the system in spite of it. Meta-level knowledge is suggested [Davis 1980] as a means for trying to accomplish the latter. It concerns the issue of having both (object-level) information about a task domain and (meta-level) information about that information. Davis [1980] treats issues of knowledge organization and knowledge representation with respect to the process of knowledge source invocation. This process can be viewed as occurring in three steps: retrieval, refinement, and execution. In retrieval, some knowledge source property is used to select from the knowledge base a subset of knowledge sources. During the refinement phase, this set may be pruned or possibly (re)ordered to provide a finer degree of control over the knowledge source use. The final phase is execution, in which one or more of the knowledge sources in the revised set are applied. Useful gains can result from adding to a system a store of (meta-level) knowledge that indicates which chunk of object-knowledge to invoke next. Davis [1980] argues that it is very important to assemble a body of knowledge and heuristics about guiding invocation and that the same mechanism should be used to reason about object-level tasks and about meta-level tasks.

Saturation problems are encountered at various levels in the model and algorithm management system. For instance, when a problem is not present in the PKB, the system retrieves all problems in the PKB that have a similar structure in order to use their associated formulations, if any, and associated algorithms to piece together an algorithm for the problem at hand. It is very important to refine the set of retrieved formulations and algorithms and work only with the most promising ones to achieve an acceptable performance in terms of running time. Also during algorithm integration, when we want to apply techiques to an obtained formulation, the system should incorporate meta-level knowledge that guides the search for promising techniques.

Model integration. The creation of a formulation for a problem not explicitly present in the PKB is another interesting part of the system. The system views a formulation as composed of two parts: the variable definitions and the constraint definitions. A variable definition will include information on the type of decision that is being modeled. For vehicle routing and scheduling models, there is only a limited number of decision types (Table 2). A constraint definition will include information on the problem characteristic that is being modeled. The objective function is considered to be a constraint.

x_i^k	Assignment variable; node i is visited by vehicle k or not.
x_{ij}^k	Assignment variable; arc (i,j) is used by vehicle k or not.
y_{ij}^k	Flow variable; the amount of commodity k (such as type of good or type of vehicle) on the arc (i,j).
D_i	Resource utilization variable; utilization of a scarce resource (such as time) at node i.
S_i^k	Node set; the nodes left to visit by vehicle k when departing at node i.

Table 2. Decision variables.

There are two ways to come up with a formulation for a problem type P not in the PKB. The first is to find a problem P' in the PKB of which P is either a generalization or a special case. This can be done by checking the comments for all the formulations in the FKB for *generalizations* and *special cases*. If problem P is found as either a generalization or a special case, the comment will also contain the information needed to modify the model to obtain a valid model for P.

The second is to carefully merge two formulations associated with problems similar to P. In this case, we really construct a new formulation. Let P' and P'' be two problems similar in structure to P. If the union of their characteristics completely covers the characteristics of P, and their formulations have compatible variable and constraint definitions, we can attempt to merge the two formulations. First, the new variable set is defined as the union of the two variable sets. Secondly, complementary constraint sets are extracted from the formulations, expressed in the new variables, and merged to form the new formulation. Finally, the new formulation is validated, i.e., it is checked whether the formulation is syntactically correct and deals with all the constraints of problem P. Knowledge on the syntactic structure of a mathematical programming formulation could be in the form of rules concerning the relationships between coefficient, variable and right-hand-side indices and how the indices are used in summations.

As an example consider a PKB that contains, among others, the problems $1,tw_j \,|\, 1 \,|\,|\, T$ and $1 \,|\, m,cap \,|\,|\, \text{sum}T_i$, and a FKB that contains the associated formulations

$$\text{minimize } \sum c_{ij} x_{ij}$$

subject to

$$\begin{array}{ll}
\sum_j x_{ij} = \sum_j x_{ji} = 1 & \text{for } i \in N, \\
D_i + t_{ij} - D_j \leqslant C(1 - x_{ij}) & \text{for } (i,j) \in A, \\
e_i \leqslant D_i \leqslant l_i & \text{for } i \in N, \\
x_{ij} \in \{0,1\} & \text{for } (i,j) \in A,
\end{array}$$

and

$$\text{minimize } \sum_{i,j} c_{ij} \sum_k x_{ij}^k$$

subject to

$$\begin{array}{ll}
\sum_k y_{ik} = \begin{cases} |M| \\ 1 \end{cases} & \begin{array}{l} \text{for } i = 0, \\ \text{for } i \in N, \end{array} \\
\sum_i q_i y_{ik} \leqslant Q & \text{for } k \in M, \\
\sum_j x_{ij} = \sum_j x_{ji} = y_{ik} & \text{for } i \in N, \\
\sum_{i \in S, j \notin S} x_{ij} \geqslant 1 & \text{for } \varnothing \neq S \subseteq N, \\
y_{ik} \in \{0,1\} & \text{for } i \in N, k \in M, \\
x_{ij} \in \{0,1\} & \text{for } (i,j) \in A.
\end{array}$$

Now, suppose we are interested in a formulation for the problem $1, tw_j \mid m, cap \mid \mid \text{sum}T_i$ not in the PKB. The system recognizes that the characteristics of this problem are completely covered by the union of the characteristics of the two problems mentioned above and that their associated formulations have compatible variable and constraint definitions, so it merges the two formulations to obtain

$$\text{minimize } \sum_{i,j} c_{ij} \sum_k x_{ij}^k$$

subject to

$$\begin{array}{ll}
\sum_k y_{ik} = \begin{cases} |M| \\ 1 \end{cases} & \begin{array}{l} \text{for } i = 0, \\ \text{for } i \in N, \end{array} \\
\sum_i q_i y_{ik} \leqslant Q & \text{for } k \in M, \\
\sum_j x_{ij} = \sum_j x_{ji} = y_{ik} & \text{for } i \in N, \\
D_i + t_{ij} - D_j \leqslant C(1 - x_{ij}) & \text{for } (i,j) \in A, \\
e_i \leqslant D_i \leqslant l_i & \text{for } i \in N, \\
y_{ik} \in \{0,1\} & \text{for } i \in N, k \in M, \\
x_{ij} \in \{0,1\} & \text{for } (i,j) \in A.
\end{array}$$

Algorithm integration. An algorithm for a problem type P not explicitly in the PKB can be constructed in several ways.

First, the generalization and special case slots of the algorithms in the AKB should be scanned to see if P occurs there. If so, the slot will also contain information indicating how the algorithm should be modified to obtain a valid algorithm for P.

Secondly, two algorithms associated with problems that are similar to P might be merged. This closely resembles the merging of two formulations as described above. However, there is a distinction between merging algorithms based on a formulation and merging algorithms not based on a formulation. In the first case, there exists a formulation F which is obtained by merging two formulations F' and F'' associated with problem types P' and P'' similar in structure to P. Let A'

and A'' be two algorithms associated with the formulations F' and F'' respectively. The system tries to adapt one of them using parts of the other. Suppose the system tries to adapt A'. To start, the system identifies the characteristics or constraints of problem P that are not dealt with by A'. Then, it establishes how these characteristics or constraints are dealt with by A'' and, if possible, modifies A' according to the techniques used in A''. Knowledge about the structure of the associated formulations might guide this process. In the second case, the system performs the same steps without the additional knowledge from the formulations.

Finally, techniques might be applied to construct an algorithm. Suppose the system obtains a formulation for some problem by merging formulations for problems that have a similar structure. Instead of trying to merge the associated algorithms, it might try to apply one or more techniques to this formulation. For instance, it could try to apply Langrangean relaxation. Consider the following formulation:

$$\text{minimize } \textstyle\sum_{i,j} c_{ij}x_{ij}$$

subject to

$$\begin{array}{lll}\sum_j x_{ij} = 1 & \text{for } i \in N, & (1)\\ \sum_j x_{ji} = 1 & \text{for } i \in N, & (2)\\ \sum_{i \in S, j \notin S} x_{ij} \geqslant 1 & \text{for } \emptyset \neq S \subseteq N, & (3)\\ x_{ij} \in \{0,1\} & \text{for } (i,j) \in A. & (4)\end{array}$$

The system could successively try to move constraints sets (1)-(3) into the objective function and evaluate the resulting formulations. Note that in order to be able to perform this evaluation, the system has to be able to recognize the resulting subproblems as being the minimal spanning 1-tree problem, in case constraint sets (1) or (2) are moved into the objective function, and the assignment problem, in case constraint set (3) is moved into the objective function. Therefore, formulations should be stored such that these structural properties are included. Lee [1986] addresses the question of how to manipulate and store formulations in such a way that structural information is included.

It is obvious that the research that is being done in the area of solving general mixed integer programming problems, such as Van Roy and Wolsey's [1985] work on automatic reformulation and Glover and Klingman's [1987] work on exploitable structure in linear and integer programs, is relevant here.

17. Conclusion

This final part of the thesis reported on our efforts to design a model and algorithm management system for vehicle routing and scheduling problems. It only contains basic ideas and is far from being a complete blueprint of the system. This is especially true for the algorithm selection and construction phase, as can be seen from the frequent use of words like appropriate, adequate, and suitable. The main challenge of the current research is to investigate the questions how the knowledge should be organized and what types of inferences should be made so as to achieve our goals.

In view of the close relation of these questions to the first two parts of the thesis and the growing interest from management scientists and operations researchers in artificial intelligence techniques, we felt it worthwile to present our ideas. There still remain a lot of research questions to be settled. We will mention a few.

We have been looking at problem types rather than problem instances. An interesting question is whether it is possible to do parameter setting and algorithm tuning on the basis of the analysis of typical problem instances. Another area for further research is that of structure identification in mathematical programming models. If really successful, it might be combined with automatic decomposition and reformulation. Similarly, it is interesting to investigate whether we can automate the determination of the level of aggregation best suited for a certain problem type. Also, model building based on a set of predefined constraint types, available in some knowledge base, in combination with automatic index matching seems to be an interesting subject for further research.

Appendix A. An AMPL representation of the VRPTW

```
### sets ###
set vert                              # vertices
set arcs within vert cross vert       # arcs
### parameters ###
param Q > 0 integer                   # vehicle capacity
param c {arcs}                        # cost
param t {arcs}                        # travel time
param e {vert}                        # earliest service time
param l {vert}                        # latest service time
param q {vert} integer                # demand
### variables ###
var x {arcs} integer
      check {(i,j) in arcs} 0 <= x[i,j] <= 1
                                 # x[i,j] is equal to 1 if arc (i,j) is chosen
                                 # to be in the solution and 0 otherwise
var l {vert} integer
                                 # l[i] is equal to the vehicle load when
                                 # arriving at a vertex
var d {vert}
                                 # d[i] is equal to the departure time at a vertex
### objective ###
minimize totalcost:
      sum {(i,j) in arcs} c[i,j] * x[i,j]
### constraints ###
subject to visit
      {i in vert} sum {j in vert} x[i,j] = 1
                                 # each customer has to be visited exactly once
subject to flowconservation
      {i in vert} sum {j in vert} x[i,j] - sum {j in vert} x[j,i] = 0
                                 # arriving at a customer should imply departing
                                 # as well
subject to timefeasible
      {(i,j) in arcs} d[i] + t[i,j] - d[j] <= CONST * ( 1 - x[i,j] )
                                 # time feasibilty
subject to esfeasible
      {i in vert} d[i] >= e[i]
subject to lsfeasible
      {i in vetr} d[i] <= l[i]
                                 # departure should fall with the specified
                                 # time window
subject to loadfeasible
      {(i,j) in arcs} l[i] + q[i] - l[j] <= CONST * ( 1 - x[i,j] )
                                 # load feasibility
```

Appendix B. A structured modeling schema for the VRP

&VEHI *The single depot, capacitated vehicle routing problem.*

&LOCATIONS *There are some LOCATION DATA*

LOCi,j/pe/ *There is a list of LOCATIONS. The first of these is the depot and the rest are customers.*

CUST_LOC(LOCi)/ce/filter(i>1){LOC}. *All but the first locations are CUSTOMER LOCATIONS.*

DEMAND(CUST_LOCi)/a/{CUST_LOC}:R+ *Each CUSTOMER LOCATION has a certain amount of DEMAND.*

DIST(LOCi,LOCj)/a/{LOC}X{LOC} where irreflexive: R+ *There is a DISTANCE between each pair of non-identical LOCATIONS.*

&VEHICLES *There are some VEHICLE DATA.*

VEHIv/pe/ *There is a list of VEHICLES.*

CAPA(VEHIv)/a/{VEHI}:R+ *Each VEHICLE has a maximimum CAPACITY.*

CUST_SUB(CUST_LOCi,VEHIv)/ce/Select {CUST_LOC}X{VEHI} where i covers {CUST_LOC}, i unique, v covers {VEHI} *Every VEHICLE is assigned a CUSTOMER SUBSET of its own. CUSTOMER SUBSETS do not overlap.*

AUG_CUST_SUB(LOCi,CUST_SUB.v)/ce/{CUST_SUB} Union ((Filer(i=1){LOC})X{VEHI}) *Every VEHICLE is assigned an AUGMENTED CUSTOMER SUBSET consisting of its CUSTOMER SUBSET plus the depot.*

ROUTE(AUG_CUST_SUBiv,AUG_CUST_SUBjv)/ce/Select {AUG_CUST_SUB}X{AUG_CUST_SUB} where (i,v) covers {AUG_CUST_SUB}, (i,v) is unique, (j,v) is unique. *For every VEHICLE there is a ROUTE specified as (from,to) pairs. Each ROUTE makes a tour through the depot and the VEHICLE's CUSTOMER SUBSET.*

&CAP_CALC *There are some CAPACITY CALCULATIONS*

VEHI_DEM(CUST_SUB.v,DEMAND)/f/{VEHI}; SUMi(CUST_SUBiv*DEMANDi) *For each VEHICLE there is a*

TOTAL DEMAND.

T:VEHI_CAP(VEHI_DEMv,CAPAv)/t/{VEHI};
VEHI_DEMv<=CAPAv *For each VEHICLE its TOTAL DEMAND does not exceed its capacity (CAPACITY TEST).*

&DISTANCE_RESULTS

VEHI_DIST(ROUTE.v.,DIST)/f/{VEHI};
SUMi_SUMj(ROUTEivj*DISTij) *For each VEHICLE, there is a TOTAL DISTANCE for its ROUTE.*

GRAND_TOT_DIST(VEHI_DIST)/f/1;SUMv(VEHI_DISTv)
There is a GRAND TOTAL DISTANCE over all VEHICLES.

TECHNIQUE

NAME
SEQUENTIAL INSERTION

SELF
INITIALADDRESS(),
INSERTIONCRITERION(),
SELECTIONCRITERION()

EXPLAIN

```
FreeAddresses ← set of free addresses
while ( FreeAddresses not empty ) do
begin
    r ← INITIALADDRESS( FreeAddresses )
    repeat
        for ( u in FreeAddresses ) do
            i(u) ← INSERTIONCRITERION( r, u )
        u* ← SELECTIONCRITERION( r, u, i(u) )
        if ( u* found ) then
        begin
            insert u* after i(u*) in r
            FreeAddresses ← FreeAddresses - u*
        end
    until ( u* not found )
end
```

DISCUSS
Build routes sequentially by the following iterative procedure:
(1) Find an address to form an initial tour.
(2) For all free addresses determine the best feasible insertion point after which it can be inserted in the emerging route.
(3) Among all free addresses select one to be actually inserted.

INITIALADDRESS
FORMULA $\max_{u \in U}\{d_{0,u}\}$

INSERTIONCRITERION
FORMULA $\min_{0 \leqslant p \leqslant n}\{d_{p,u} + d_{u,p+1} - d_{p,p+1} \mid \sum_{0 \leqslant p \leqslant n} q_p + q_u < Q\}$
[*cap*]

SELECTIONCRITERION
FORMULA $\min_{u \in U}\{d_{i(u),u} + d_{u,i(u)+1} - d_{i(u),i(u)+1}\}$

COMPLEXITY

TIME $O(n^2)$

ANALYSIS

EMPIRICAL good

TECHNIQUE

NAME

2-EXCHANGE

[A. CROES (1958). A method for solving the traveling salesman problem. *Oper. Res. 12*, 568-581.]

SELF

EVALUATE()

EXPLAIN

```
CurrentSolution ← InitialSolution
CurrentQuality ← EVALUATE( CurrentSolution )
while ( $\{(i,i+1),(j,j+1)\} \rightarrow \{(i,j),(i+1,j+1)\}$
        not examined in CurrentSolution ) do
begin
    ExchangeSolution ← perform $\{(i,i+1),(j,j+1)\} \rightarrow \{(i,j),(i+1,j+1)\}$
    ExchangeQuality ← EVALUATE( ExchangeSolution )
    if ( ExchangeQuality > CurrentQuality ) then
    begin
        CurrentSolution ← ExchangeSolution
        CurrentQuality ← ExchangeQuality
    end
end
```

DISCUSS

Starting from an initial tour, all possible exchanges of two arcs in the tour with two arcs not in the tour are evaluated. If an improved tour is found it is adopted and the process is repeated.

EVALUATE

FORMULA $\sum_{0 \leqslant p \leqslant n} \{d_{p,p+1}\}$

COMPLEXITY

TIME verification of two optimality requires $O(n^2)$ time

ANALYSIS

EMPIRICAL good

[S. Lin (1965). Computer solutions to the traveling salesman problem. *Bell System Tech. J. 44*, 2245-2269.]

ALGORITHM

SELF

SEQUENTIAL INSERTION,
2-EXCHANGE

PROBLEM

$1 \mid m, cap \mid \mid \sum T_i$

DISCUSS

A set of vehicles based at a central depot is required to fulfill customers demands. Each customer i has a demand q_i and the vehicles have capacity Q. The objective is to minimize total travel time.

TYPE

APPROXIMATION

FORMULATION

EMPTY

EXPLAIN

```
Routes ← SEQUENTIAL INSERTION
for ( r in Routes ) do
begin
    apply 2-EXCHANGE( r )
end
```

INITIALADDRESS

FORMULA $\max_{u \in U}\{d_{0,u}\}$

INSERTIONCRITERION

FORMULA $\min_{0 \leq p \leq n}\{d_{p,u} + d_{u,p+1} - d_{p,p+1} \mid \sum_{0 \leq p \leq n} q(p) + q(u) < Q\}$

SELECTIONCRITERION

FORMULA $\min_{u \in U}\{d_{i(u),u} + d_{u,i(u)+1} - d_{i(u),i(u)+1}\}$

EVALUATE

FORMULA $\sum_{0 \leq p \leq n}\{d_{p,p+1}\}$

COMPLEXITY

EMPTY

ANALYSIS
EMPIRICAL reasonable

SPECIAL CASE
$1|1||T$
$[q_i = 0, Q = \infty]$

GENERALIZATION
$1|m, cap_i||\text{sum}T_i$

```
[(1) AvailableVehicle ← set of available vehicles
    ...
    $r$ ← INITIALADDRESS( FreeAddresses )
    $v$ ← SELECTVEHICLE( AvailableVehicles )
    AvailableVehicle ← AvailableVehicles - $v$
    ...
 (2) INSERTIONCRITERION:
        FORMULA $\min_{0 \leqslant p \leqslant n}\{d_{p,u} + d_{u,p+1} - d_{p,p+1} \mid \sum_{0 \leqslant p \leqslant n} q_p + q_u < Q_v\}$]
```

REFERENCES

J.M. ANTHONISSE, J.K. LENSTRA, M.W.P. SAVELSBERGH (1987). *Functional Description of CAR, an Interactive System for Computer Aided Routing*, Report OS-R8716, Centre for Mathematics and Computer Science, Amsterdam.

J.M. ANTHONISSE, J.K. LENSTRA, M.W.P. SAVELSBERGH (1988). Behind the screen: DSS from an OR point of view. *Decision Support Systems*, to appear.

E.K. BAKER (1983). An exact algorithm for the time-constrained traveling salesman problem. *Oper. Res. 31*, 65-73.

W.J. BELL, L.M. DALBERTO, M.L. FISHER, A.J. GREENFIELD, R. JAIKUMAR, P. KEDIA, R.G. MACK, P.J. PRUTZMAN (1983). Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *Interfaces 13*, 4-23.

J.L. BENNETT (1983). Analysis and design of the user interface for decision support systems. J.L. BENNETT (ed.). *Building Decision Support Systems*, Addison-Wesley, Reading, MA, 41-64.

J. BISSCHOP, A. MEERHAUS (1982). On the development of a General Algebraic Modeling System in a strategic planning environment. *Math. Programming Study, 20*, 1-29.

L. BODIN, B. GOLDEN (1981). Classification in vehicle routing and scheduling. *Networks 11*, 97-108.

G.G. BROWN, G.W. GRAVES (1981). Real-time dispatch of petroleum tank trucks. *Management Sci. 27*, 19-32.

N. CHRISTOFIDES, J.E. BEASLEY (1984). The period routing problem. *Networks 14*, 237-256.

N. CHRISTOFIDES, S. EILON (1969). An algorithm for the vehicle dispatching problem. *Oper. Res. Quart. 20*, 309-318.

N. CHRISTOFIDES, A. MINGOZZI, P. TOTH (1981a). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Math. Programming 20*, 255-282.

N. CHRISTOFIDES, A. MINGOZZI, P. TOTH (1981b). State space relaxation procedures for the computation of bounds to routing problems. *Networks 11*, 145-164.

N. CHRISTOFIDES, A. MINGOZZI, P. TOTH (1981c). *Exact Algorithms for the Travelling Salesman Problem with Time Constraints, Based on State-Space Relaxation*, Unpublished manuscript.

G. CLARKE, J.W. WRIGHT (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res. 12*, 568-581.

R.W. CONWAY, W.L. MAXWELL, L.W. MILLER (1967). *Theory of Scheduling*, Addison-Wesley, Reading, MA.

A. CROES (1958). A method for solving traveling salesman problems. *Oper. Res. 5*, 791-812.

G.B. DANTZIG, J.H. RAMSER (1959). The truck dispatching problem. *Management Sci. 6*, 80-91.

C.J. DATE (1975). *An Introduction to Database Systems*, Addison-Wesley, Reading, MA.

R. DAVIS (1980). Meta-rules: reasoning about control. *Artificial Intelligence 15*, 179-222.

G. DESANCTIS (1984). Computer graphics as decision aids: directions for research. *Decision Sci. 15*, 463-487.

M. DESROCHERS (1986). *An Algorithm for the Shortest Path Problem with Resource Constraints*, Publication 421A, Centre de recherche sur les transports, Université de Montréal.

M. DESROCHERS, J.K. LENSTRA, M.W.P. SAVELSBERGH, F. SOUMIS (1988). Vehicle routing with time windows: optimization and approximation. B.L. GOLDEN, A.A. ASSAD (eds.). *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam, to appear.

M. DESROCHERS, F. SOUMIS (1985a). *A Generalized Permanent Labelling Algorithm for the Shortest Path Problem with Time Windows*, Publication 394A, Centre de recherche sur les transports, Université de Montréal.

M. DESROCHERS, F. SOUMIS (1985b). *A Reoptimization Algorithm for the Shortest Path Problem with Time Windows*, Publication 397A, Centre de recherche sur les transports, Université de Montréal.

J. DESROSIERS, Y. DUMAS, F. SOUMIS (1986a). *The Multiple Vehicle Many to Many Routing and Scheduling Problem with Time Windows*, Cahiers du GERAD G-84-13, Ecole des Hautes Etudes Commerciales de Montréal.

J. DESROSIERS, Y. DUMAS, F. SOUMIS (1986b). A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *Amer. J. Math. Management Sci. 6*, 301-326.

J. DESROSIERS, Y. DUMAS, F. SOUMIS (1987). *Vehicle Routing Problem with Pick-up, Delivery and Time Windows*, Cahiers du GERAD, Ecole des Hautes Etudes Commerciales de Montréal.

J. DESROSIERS, P. PELLETIER, F. SOUMIS (1984). Plus court chemin avec contraintes d'horaires. *RAIRO Rech. Opér. 17(4)*, 357-377.

J. DESROSIERS, M. SAUVÉ, F. SOUMIS (1985). *Lagrangian Relaxation Methods for Solving the Minimum Fleet Size Multiple Traveling Salesman Problem with Time Windows*, Publication 396, Centre de recherche sur les transports, Université de Montréal.

J. DESROSIERS, F. SOUMIS, M. DESROCHERS (1984). Routing with time windows by column generation. *Networks 14*, 545-565.

J. DESROSIERS, F. SOUMIS, M. DESROCHERS, M. SAUVÉ (1985). Routing and scheduling with time windows solved by network relaxation and branch-and-bound on time variables. J.-M. Rousseau (ed.). *Computer Scheduling of Public Transport 2*, North-Holland, Amsterdam, 451-469.

D.R. DOLK (1986). Data as models: an approach to implementing model management. *Decision Support Systems 2*, 73-80.

D.R. DOLK, B.R. KONSYNSKI (1984). Knowledge representation for model management systems. *IEEE Trans. Software Engrg. SE-10*, 619-628.

Y. DUMAS (1985). *Confection d'itinéraires de véhicules en vue du transport de plusieurs origines à plusieurs destinations*, Publication 434, Centre de recherche sur les transports, Université de Montréal.

Y. DUMAS, J. DESROSIERS (1986). *A Shortest Path Problem for Vehicle Routing with Pick-up, Delivery and Time Windows*, Cahiers du GERAD G-86-09, Ecole des

Hautes Etudes Commerciales de Montréal.

M.L. FISHER (1986). Interactive optimization. *Ann. Oper. Res. 4*, 541-556.

M.L. FISHER, R. JAIKUMAR (1978). *A Decomposition Algorithm for Large-Scale Vehicle Routing*, Working Paper 78-11-05, Department of Decision Sciences, University of Pennsylvania.

M.L. FISHER, R. JAIKUMAR (1981). A generalized assignment heuristic for vehicle routing. *Networks, 11*, 109-124.

M.L. FISHER, R. JAIKUMAR, L. VAN WASSENHOVE (1986). A multiplier adjustment method for the generalized assignment problem. *Management Sci. 32*, 1095-1103.

R. FOURER, D.M. GAY, B.W. KERNIGHAN (1987). *AMPL: A Mathematical Programming Language*, AT&T Bell Laboratories, Murray Hill, N.J.

M.R. GAREY, D.S. JOHNSON (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Fransisco.

T.J. GASKELL (1967). Bases for vehicle fleet scheduling. *Oper. Res. Quart. 18*, 281-295.

A.M. GEOFFRION (1986). *An Introduction to Structured Modeling*, Working paper No. 338, Western Management Institute, University of California, Los Angeles.

A.M. GEOFFRION (1987). *The theory of structured modeling*. Working paper No. 346, Western Management Science Institute, University of California, Los Angeles.

F. GLOVER, D. KLINGMAN (1987). *Creating Exploitable Structure in Linear and Integer Programs*, Working Paper CBDA 128, Center for Business Decision Analysis, University of Texas.

M. GOETSCHALCKX, C. JACOBS (undated). *The Vehicle Routing Problem with Backhauls*, Manuscript, Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta.

B.L. GOLDEN, A.A. ASSAD (1986). Vehicle routing with time-window constraints: algorithmic solutions. *Amer. J. Math. Management Sci. 6*, 251-428. (special issue).

I.P. GOLDSTEIN, B. ROBERTS (1979). Using frames in scheduling. P.H. WINSTON, R.H. BROWN (eds.). *Artificial Intelligence: An MIT perspective*, MIT Press, Cambridge, MA.

R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math. 5*, 287-326.

GUAN MEIGU [KWAN MEI-KO] (1962). Graphic programming using odd or even points. *Chinese Math. 1*, 273-277.

M. GUIGNARD (1984). *Lagrangean Decomposition: an Improvement over Lagrangean and Surrogate Duals*, Report 62, Department of Statistics, University of Pennsylvania, Philadelphia.

G.Y. HANDLER, P.B. MIRCHANDANI (1979). *Location on Networks: Theory and Algorithms*, MIT Press, Cambridge, MA.

F.R.A. HOPGOOD, D.A. DUCE, J.R. GALLOP, D.C. SUTCLIFFE (1983). *Introduction to the Graphical Kernel System (GKS)*, Academic Press, London.

W. JAQUES (1859). *The Icosian Game*, published and sold wholesale by John Jaques and Son, London.

J.-J. JAW, A.R. ODONI, H.N. PSARAFTIS, N.H.M. WILSON (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Res. Part B 20B*, 243-257.

C.V. JONES (1987). User interfaces. Unpublished manuscript; E.G. COFFMAN, JR., J.K. LENSTRA, A.H.G. RINNOOY KAN (eds.). *Handbook in Operations Research and Management Science, Volume 3: Computation*, North-Holland, Amsterdam, to appear.

C.V. JONES (1988). The 3-dimensional Gantt chart. *Oper. Res.*, to appear.

K.O. JÖRNSTEN, M. NASBERG, P.A. SMEDS (1985). *Variable Splitting - a New Lagrangean Relaxation Approach to Some Mathematical Programming Models*, Report LITH-MAT-R-85-04, Department of Mathematics, Linköping Institute of Technology.

P.G.W. KEEN (1986). Decision support systems: the next decade. E.R. MCLEAN, H.G. SOL (eds.). *Decision Support Systems: a Decade in Perspective*, North-Holland, Amsterdam, 221-237.

A.W.J. KOLEN, A.H.G. RINNOOY KAN, H.W.J.M. TRIENEKENS (1987). Vehicle routing with time windows. *Oper. Res. 35*, 266-237.

B.J. LAGEWEG, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1981). *Computer-Aided Complexity Classification of Deterministic Scheduling Problems*, Report BW 138, Mathematisch Centrum, Amsterdam.

B.J. LAGEWEG, J.K. LENSTRA, E.L. LAWLER, A.H.G. RINNOOY KAN (1982). Computer-aided complexity classification of combinatorial problems. *Comm. ACM 25*, 817-822.

G. LAPORTE, Y. NOBERT (1983). Generalized travelling salesman problem through n sets of nodes: an integer programming approach. *INFOR 21*, 61-75.

S. LIN (1965). Computer solutions to the traveling salesman problem. *Bell System Tech. J. 44*, 2245-2269.

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

S. MARTELLO, P. TOTH (1981). An algorithm for the generalized assignment problem. J.P. BRANS (ed.). *Operations Research 81*, North-Holland, Amsterdam. 589-603.

K. MENGER (1930). Das Botenproblem. K. MENGER (ed.) (1932). *Ergebnisse eines Mathematischen Kolloquiums 2*, Teubner, Leipzig, 9. Kolloquium (5.II.1930), 12.

C. MILLER, A. TUCKER, R. ZEMLIN (1960). Integer programming formulation of travelling salesman problems. *J. Assoc. Comput. Mach. 7*, 326-329.

M. MINSKY (1975). A framework for representing knowledge. P.H. WINSTON (ed.). *The Psychology of Computer Vision*, McGraw-Hill, New York.

J. MYLOPOULOS (1980). An overview of knowledge representation. M.L. BRODIE, S.N. ZILES (eds.). *Proc. ACM Workshop on Data Abstraction, Databases, Conceptual Modeling*, 5-12.

J. MYLOPOULOS, H.J. LEVESQUE (1984). An overview of knowledge representation. M.L. BRODIE, J. MYLOPOULOS, J.W. SCHMIDT (eds.). *On Conceptual*

Modeling, 3-17.

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.

C.S. ORLOFF (1976). Route constrained fleet scheduling. *Transportation Sci. 10*, 149-168.

H. PSARAFTIS (1983a). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Sci. 17*, 351-360.

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

H.N. PSARAFTIS (1983). Analysis of an $O(N^2)$ heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. *Transportation Res. Part B 17B*, 133-145.

D. RONEN (1987). Perspectives on practical aspects of truck routing and scheduling. *European J. Oper. Res.*, to appear.

G.T. ROSS, R.M. SOLAND (1975) A branch and bound algorithm for the generalized assignment problem, *Math. Programming 8*, 91-103.

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

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

M.W.P. SAVELSBERGH (1987). *Local Search for Constrained Routing Problems*, Report OS-R8711, Centre for Mathematics and Computer Science, Amsterdam.

T.R. SEXTON, L.D. BODIN (1985a). Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. *Transportation Sci. 19*, 378-410.

T.R. SEXTON, L.D. BODIN (1985b). Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. *Transportation Sci. 19*, 411-435.

B. SHNEIDERMAN (1987). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA.

M.M. SOLOMON (1983). *Vehicle Routing and Scheduling with Time Window Constraints: Models and Algorithms*, Report 83-02-01, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia.

M.M. SOLOMON (1986). On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Networks 16*, 161-174.

M.M. SOLOMON, E.K. BAKER, J.R. SCHAFFER (1988). Vehicle routing and scheduling problems with time window constraints: implementations of solution improvement procedures. B.L. GOLDEN, A.A. ASSAD (eds.). *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam, to appear.

B. SÖRENSEN (1986). *Interactive Distribution Planning*, Ph.D. thesis, Technical University of Denmark, Lyngby.

F. SOUMIS, J. DESROSIERS, M. DESROCHERS (1985). Optimal urban bus routing with scheduling flexibilities. *Lecture Notes in Control and Information Sciences 59*, Springer, Berlin, 155-165.

R.F. SPROULL, W.R. SUTHERLAND, M.K. ULLNER (1985). *Device-Independent Graphics*, McGraw-Hill, New York.

H. STERN, M. DROR (1979). Routing electric meter readers. *Comput. & Oper. Res. 6*, 209-223.

A.J. SWERSEY, W. BALLARD (1984). Scheduling school buses. *Management Sci. 30*, 844-853.

E.R. TUFTE (1983). *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT.

T.J. VAN ROY, L.A. WOLSEY (1987). Solving mixed integer programming problems using automatic reformulation. *Oper. Res. 35*, 45-57.

SAMENVATTING

In het standaard voertuigrouteringsprobleem moet een aantal voertuigen vanuit een centraal depot een aantal klanten bezoeken. Elk voertuig heeft een beperkte capaciteit en elke klant heeft een bepaalde vraag. Het probleem is routes voor de voertuigen te vinden zodat alle klanten worden bezocht, de totale vraag van de klanten in een route de capaciteit van het betreffende voertuig niet overschrijdt, en de totale lengte van de routes minimaal is.

Dit probleem en talloze variaties treden op in een grote verscheidenheid van praktische situaties. Overheid en industrie besteden een aanzienlijk deel van hun budget aan distributie- en transportactiviteiten. Een effectief routeringssysteem kan enorme besparingen in tijd, geld en energie opleveren.

In dit proefschrift worden methoden en technieken uit de mathematische besliskunde en informatica behandeld die aangewend kunnen worden om op verscheidene niveaus ondersteuning te verlenen aan diegenen die dagelijks met de problematiek van fysieke distributie te maken hebben.

Het eerste deel beschrijft hoe realistische, veelvuldig in de praktijk optredende, randvoorwaarden verwerkt kunnen worden in de in de literatuur bestudeerde modellen en algoritmen. Randvoorwaarden die aan de orde komen zijn: tijdvensters bij klanten, de combinatie van zowel bestellen als afhalen bij klanten, en volgorderestricties tussen klanten onderling. Er wordt zowel aandacht besteed aan optimaliseringsalgoritmen als aan benaderingsalgoritmen, waarbij de nadruk gelegd wordt op de laatste. Bij de benaderingsalgoritmen wordt gekeken naar constructiemethoden, die een toegelaten oplossing creëren op basis van nog ongestructureerde gegevens, naar methoden voor onvolledige optimalisering, die heuristische regels toepassen om een volledige aftelling van de oplossingsruimte te voorkomen, en naar methoden voor iteratieve verbetering, die een gegeven oplossing proberen te verbeteren door locale veranderingen aan te brengen. Het onderzoek richt zich hierbij vooral op de vraag hoe op een efficiënte manier, dat wil zeggen met een minimale extra inspanning, rekening gehouden kan worden met randvoorwaarden.

Het tweede deel beschrijft interactieve planningssystemen. Dergelijke systemen ondersteunen planningsactiviteiten door de integratie van menselijk inzicht en mechanische algoritmen. Zij zijn een gevolg van het inzicht dat in vele planningssituaties een planner niet geheel vervangen kan worden door kwantitatieve technieken, maar dat deze wel een belangrijke bijdrage kunnen leveren. In interactieve planningssystemen worden de taken dan ook verdeeld tussen mens en machine in overeenstemming met beider kwaliteiten. Een machine is onverslaanbaar bij het oplossen van volledig gespecificeerde gedetailleerde problemen. De mens is superieur in het herkennen van globale patronen en het waarnemen van ad hoc randvoorwaarden. Dit heeft echter tot gevolg dat er veel aandacht besteed moet worden aan het ontwerpen van de gebruikersinterface, d.w.z. dat deel van het systeem dat de communicatie tussen mens en machine verzorgt.

Een belangrijke plaats is in dit deel ingeruimd voor CAR, een interactief systeem voor 'computer aided routing'. CAR kan worden gebruikt als hulpmiddel voor de operationele distributieplanning; met CAR produceert men eenvoudig economische ritten en tijdschema's voor voertuigen.

De grote verscheidenheid van distributieproblemen en het grote aantal algoritmen maken het bijna onmogelijk voor een onervaren, en zelfs voor een ervaren, logistiek manager om een algoritme te kiezen die geschikt is voor zijn specifieke probleemsituatie. Het laatste deel geeft een blauwdruk van een systeem dat het modelleren van probleemsituaties en het selecteren en construeren van algoritmen voor de oplossing van het resulterende model ondersteunt. Het onderzoek heeft zich daarbij tot nu toe gericht op de representatie en manipulatie van informatie en kennis. Als eerste wordt een classificatieschema behandeld waarmee zowel praktijk problemen als problemen uit de literatuur op een eenvoudige wijze kunnen worden beschreven. Daarna wordt geschetst hoe modellen, algoritmen en kennis kunnen worden gerepresenteerd en gemanipuleerd om selectie en constructie van algoritmen mogelijk te maken.

STELLINGEN

behorende bij het proefschrift van

MATHIEU WILLEM PAUL SAVELSBERGH

COMPUTER AIDED ROUTING

(1) Indien leerboeken in de mathematische besliskunde dat deel van de complexiteitstheorie behandelen dat zich bezighoudt met NP-volledigheid en polynomiale reducties, verdient het de voorkeur 'bounded tiling' als het universele probleem te gebruiken in plaats van 'satisfiability'.

(2) Zij gegeven een graaf $G=(V,E)$ met voor iedere kant $e \in E$ een gewicht c_e, een wortel $w \in V$ en voor iedere punt $v \in V \setminus \{w\}$ een tijdvenster $[r_v, d_v]$. Beschouw een opspannende boom T van G. Definieer $t_w = 0$ en, voor elke $v \in V \setminus \{w\}$, $t_v = \max\{r_v, t_u + c_{\{u,v\}}\}$, waarbij u de voorganger van v is op het pad in T van w naar v. T heet *tijdtoegelaten* als $t_v \leqslant d_v$ voor elke $v \in V \setminus \{w\}$. Het *gewicht* van T is gedefinieerd als de som van de gewichten van zijn kanten. Het probleem van het vinden van een tijdtoegelaten opspannende boom van minimaal gewicht is oplosbaar in polynomiale tijd indien $d_v = \infty$ voor alle $v \in V \setminus \{w\}$.

(3) Het probleem zoals gedefinieerd in Stelling 2 is oplosbaar in polynomiale tijd indien $c_e = 1$ voor alle $e \in E$.

(4) Het probleem zoals gedefinieerd in Stelling 2 is NP-lastig indien $r_v = 0$ voor alle $v \in V \setminus \{w\}$ en $c_e \in \{1,2\}$ voor alle $e \in E$.

(5) Zij gegeven een volledige graaf $G=(V,E)$ en een constante C. Beschouw het probleem van het vinden van een opspannende boom van minimaal gewicht met de eigenschap dat ieder punt een graad heeft die niet groter is dan C. Een benaderingsalgoritme voor dit probleem is de volgende extensie van Prim's algoritme voor het vinden van een minimale opspannende boom: beginnend met een boom die slechts bestaat uit één punt, wordt telkens de kortste uitgaande kant toegevoegd die geen circuit en geen graadoverschrijding veroorzaakt.

Beschouw nu de partitie van de puntenverzameling die verkregen wordt door beginnend met een minimale opspannende boom telkens de langste kant te verwijderen die twee punten met graadoverschrijding verbindt, en door vervolgens voor de overblijvende punten met graadoverschrijding telkens de langste kant te verwijderen totdat er geen sprake meer is van een graadoverschrijding.

De bovengenoemde extensie van Prim's algoritme leidt tot dezelfde opspannende boom wanneer begonnen wordt met punten uit dezelfde deelverzameling van de partitie.

(6) Zowel voor het handelsreizigersprobleem als voor het handelsreizigersprobleem met tijdvensters geldt dat verificatie van 2-optimaliteit van een oplossing $O(n^2)$ tijd kost op één processor, waarbij n het aantal steden is. Het gebruik van n processoren reduceert deze verificatietijd tot $O(n)$. Echter, wanneer gebruik gemaakt wordt van n^2 processoren kan 2-optimaliteit van een oplossing van het handelsreizigersprobleem in polylogaritmische tijd geverifieerd worden, terwijl voor het handelsreizigersprobleem met tijdvensters nog steeds lineaire tijd nodig is wanneer gebruik gemaakt wordt van de technieken zoals beschreven in deel I van dit proefschrift.

(7) Combinatorische optimalisering houdt zich voor een belangrijk deel bezig met het ontwerpen van methoden die sneller tot resultaten leiden dan volledige aftelling. De jarenlange ervaring in het ontwikkelen van geavanceerde zoekstrategieën kan een wezenlijke bijdrage leveren aan het onderzoek op het gebied van de kunstmatige intelligentie.

(8) Verdieping, de in Van Dale gegeven betekenis van etage, kan beter worden vervangen door verhoging.

(9) Het groeiend aantal voordeurdelers heeft niet geleid tot een verhoogde afzet van gedeelde voordeuren.

(10) Huis-aan-huis verspreid reclamedrukwerk kan beter rechtstreeks bij scholen en verenigingen worden afgeleverd.

(11) Het plaatsen van rivaliserende voetbalsupporters in één vak draagt op snelle wijze bij tot eliminatie van voetbalvandalisme.

(12) De geneugten van landelijk wonen zijn in sterke mate afhankelijk van de kwaliteit van het hang- en sluitwerk.