Chapter 8

Combinatorial Optimization: Route Planning

There are a great many of combinatorial optimization problems that genetic algorithms have been applied on so far. In the following we will concentrate on two selected route planning problems with a lot of attributes which are representative for many combinatorial optimization problems, namely the traveling salesman problem (TSP) and the vehicle routing problem (VRP).

The traveling salesman problem is certainly one of the classical as well as most frequently analyzed representatives of combinatorial optimization problems with a lot of solution methodologies and solution manipulation operators. Comparing the TSP to other combinatorial optimization problems, the main difference is that very powerful problem-specific methods as for example the Lin-Kernighan algorithm [LK73] and effective branch and bound methods are available that are able to achieve a global optimal solution in very high problem dimensions. These high-dimensional problem instances with a known global optimal solution are very well suited as benchmark problems for metaheuristics as for example GAs.

The VRP as well as its derivatives, the capacitated VRP (CVRP) and the capacitated VRP with time windows (CVRPTW) which will be introduced in this chapter, are much closer to practical problem situations in transport logistics, and solving them requires the handling of implicit and explicit constraints. There are also no comparable powerful problem-specific methods available, and metaheuristics like tabu search and genetic algorithms are considered the most powerful problem solving methods for VRP which is a different but not less interesting situation than handling the TSP problem.

8.1 The Traveling Salesman Problem

The TSP is quite easy to state: Given a finite number of cities along with the cost of travel between each pair of them, the goal is to find the cheapest way of visiting all the cities exactly once and returning to your starting point. Usually the travel costs are symmetric. A tour can simply be described by the order in which the cities are visited; the data consist of integer weights

assigned to the edges of a finite complete graph and the objective is to find a Hamiltonian cycle, i.e., a cycle passing through all the vertices, of minimum total weight. In this context, Hamiltonian cycles are commonly called tours.

Already in the early 19th century the TSP appeared in literature [Voi31]. In the 1920s, the mathematician and economist Karl Menger [Men27] published it in Vienna; it reappeared in the 1930s in the mathematical circles of Princeton. In the 1940s, it was studied by statisticians (Mahalanobis, see [Mah40], e.g., and Jessen, see for instance [Jes42]) in connection with an agricultural application. The TSP is commonly considered the prototype of a hard problem in combinatorial optimization.

8.1.1 Problem Statement and Solution Methodology

8.1.1.1 Definition of the TSP

In a formal description the TSP is defined as the search for the shortest Hamiltonian cycle of a graph whose nodes represent cities. The objective function f represents the length of a route and therefore maps the set $\mathcal{S}$ of admissible routes into the real numbers $\mathbb{R}$ [PS82]:

$$f : \mathcal{S} \rightarrow \mathbb{R}$$

The aim is to find the optimal tour $s^* \in \mathcal{S}$ such that $f(s^*) \leq f(s_k), \forall s_k \in \mathcal{S}$. In order to state the objective function f we have to introduce a distance matrix $[d_{ij}], d_{ij} \in \mathbb{R}^+$ whose entries represent the distance from a city i to a city j. In that kind of representation the cities are considered as the nodes of the underlying graph. If there is no edge between two nodes, the distance is set to infinity.

Using the notation given in [PS82], that $\pi_k(i)$ represents the city visited next after city i in a certain tour s_k, the objective function is defined as

$$f(s_k) = \sum_{i=1}^{n} d_{i\pi_k(i)} \tag{8.1}$$

By this definition the general asymmetric TSP is specified. By means of certain constraints on the distance matrix it is possible to define several variants of the TSP. A detailed overview about the variants of the TSP is given in [LLRKS85]. The most important specializations consider symmetry, the triangle-inequality, and Euclidean distances:

Symmetry

A TSP is defined to be symmetric if and only if its distance matrix is symmetric, i.e., if

$$d_{ij} = d_{ji}, \forall i, j \in 1, \ldots, n \tag{8.2}$$

If this set of equalities is not satisfied for at least one pair (i, j), for example if "one-way streets" occur, we denote the problem as an asymmetric TSP.

The Triangle Inequality

Symmetric as well as asymmetric TSPs can, but don't necessarily have to satisfy the triangle inequality:

$$d_{ij} \leq (d_{ik} + d_{kj}), \forall (i, j, k \in 1, \ldots, n) \tag{8.3}$$

i.e., that the direct route between two cities must be shorter than or as long as any route including another node in between.
A reasonable violation of the triangle inequality is possible especially when the entries in the distance matrix are interpreted as costs rather than as distances.

Euclidean Distances

As a rather important subset of symmetric TSP satisfying the triangle inequality we consider the so-called Euclidean TSP. For the Euclidean TSP it is mandatory to specify the coordinates of each node in the n-dimensional Euclidean space. For the 2-dimensional case the entries d_{ij} of the distance matrix are consequently given by the Euclidean distance

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{8.4}$$

whereby x_i and y_i denote the coordinates of a certain city i.

In contrast to most problems that occur in practice, many TSP benchmark tests use Euclidean TSP instances. Anyway, GA-based metaheuristics do not take advantage of the Euclidean structure and can therefore also be used for Non-Euclidean TSPs.

8.1.1.2 Versions of the TSP

Motivated by certain situations appearing in operational practice, some more variants of the TSP have emerged. Appreciable standardizations that will not be taken into further account within the scope of this book are the following ones:

Traveling Salesman Subtour Problems (TSSP)

In contrast to the TSP not all cities have to be visited in the context of the TSSP; only those cities are visited that are worth being visited which implies the necessity of some kind of profit function in order to decide if the profit is higher than the travel expenses. Vice versa, depending on the actual implementation, this can also be realized by the introduction of penalties for not visiting a certain node (city).

Postman Problems

For postman problems (e.g., [Dom90]) not certain sets of nodes (cities) have to be visited but rather given sets of edges (which can be interpreted as streets

of houses) have to be passed at least once with the goal to minimize the total route length. Therefore, the aim is a suitable selection of the edges to be passed in a certain order for obtaining minimal cost.

Time Dependent TSP

In time dependent TSPs the cost of visiting a city j starting from a city i does not only depend on d_{ij} but also on the position in the total-route or, even more general, on the point of time a certain city is visited [BMR93].

Traveling Salesman Problem with Time Windows (TSPTW)

Like the TSP, the TSPTW is stated as finding an optimal tour for a set of cities where each city has to be visited exactly once. Additionally to the TSP, the tour must start and end at a unique depot within a certain time window and each city must be visited within its own time window. The cost is usually defined by the total travel distance and/or by the total schedule time (which is defined as the sum of travel time, waiting time, and service time) [Sav85].

8.1.1.3 Review of Optimal Algorithms

Total Enumeration

In principle, total enumeration is applicable to all integer optimization problems with a finite solution space: All points of the solution space $\mathcal{S}$ are evaluated by means of an objective-function storing the best solution so far. As the TSP has a worst case complexity of $\mathcal{O}(n!)$, total enumeration is only applicable to very small problem instances. For example, even for a rather small and simple 30-city symmetric TSP one would have to consider $\frac{(n-1)!}{2} = \frac{(29)!}{2}$ possible solutions which would require a computational time of about $1.4 * 10^{12}$ years assuming the use of a very powerful computer which can evaluate 100,000 million routes per second.

Integer Programming

In order to apply integer programming to the TSP it is mandatory to introduce a further $n \times n$ matrix $X = [x_{ij}]$ with $x_{ij} \in \{0, 1\}$ where x_{ij} indicates whether or not there is a connection from city i to city j. Thus, the optimization problem can be stated in the following way:
Find

$$\min(\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}) \tag{8.5}$$

such that

$$\sum_{j=1}^{n} x_{ij} = 1; \forall i \in \{1, \ldots, n\} \tag{8.6}$$

$$\sum_{i=1}^{n} x_{ij} = 1; \forall j \in \{1, \ldots, n\} \tag{8.7}$$

$$x_{ij} \geq 0; \forall i, j \in \{1, \ldots, n\} \tag{8.8}$$

These constraints ensure that each city has exactly one successor and is the predecessor of exactly one other city. The representation given above is also called an assignment problem and has firstly been applied to the TSP by Dantzig [DR59]. However, the assignment problem alone does not assure a unique Hamiltonian cycle, i.e., it is also possible that two or more subcycles exist which does not specify a valid TSP.

Hence, for the TSP it is necessary to state further conditions in order to define an assignment problem without subcycles, and therefore the integer property of the assignment problem does not hold any more. Similar to linear programming, in integer programming the admissible solution space can be restricted by the given constraints – but the corners of the emerging polyhedron won't represent valid solutions in general. In fact, only a rather small number of points inside the polyhedron will represent valid solutions of the integer program.

As described in [Gom63] Gomory tried to overcome this drawback by introducing the cutting-plane method that introduces virtual constraints, the so-called cutting-planes, in order to ensure that all corners of the convex polyhedron are integer solutions. The crux in the construction of suitable cutting planes is that it requires a lot of very problem-specific knowledge.

Grötschel's dissertation [Grö77] was one of the first contributions that considered a special TSP instance in detail and a lot of articles about the solution of specific TSP-benchmark problems have since then been published (as for example in [CP80]) with problem-specific cutting-planes.

Unfortunately, the methods for constructing suitable cutting-planes are far away from working in an automated way and require a well versed user. Therefore, the main area of application of integer programming for the TSP is the exact solution of some large benchmark problems in order to get reference problems for testing certain heuristics.

8.1.2 Review of Approximation Algorithms and Heuristics

During the last four decades a variety of heuristics for the TSP has been published; Lawler et al. have given a comparison of the most established ones in [LLRKS85]. Operations research basically distinguishes between methods that are able to construct new solutions routes, called route building heuristics

or construction heuristics, and methods that assume a certain (valid) route in order to improve it, which are understood as route improving heuristics.

Nearest Neighbor Heuristics

The nearest neighbor algorithm [LLRKS85] is a typical representative of a route building heuristics. It simply considers a city as its starting point and takes the nearest city in order to build up the Hamiltonian cycle. At the beginning this strategy works out quite well whereas adverse stretches have to be inserted when only a few cities are left.

Figure 8.1 shows a typical result of nearest neighbor heuristics applied to a TSP instance that demonstrates its drawbacks.

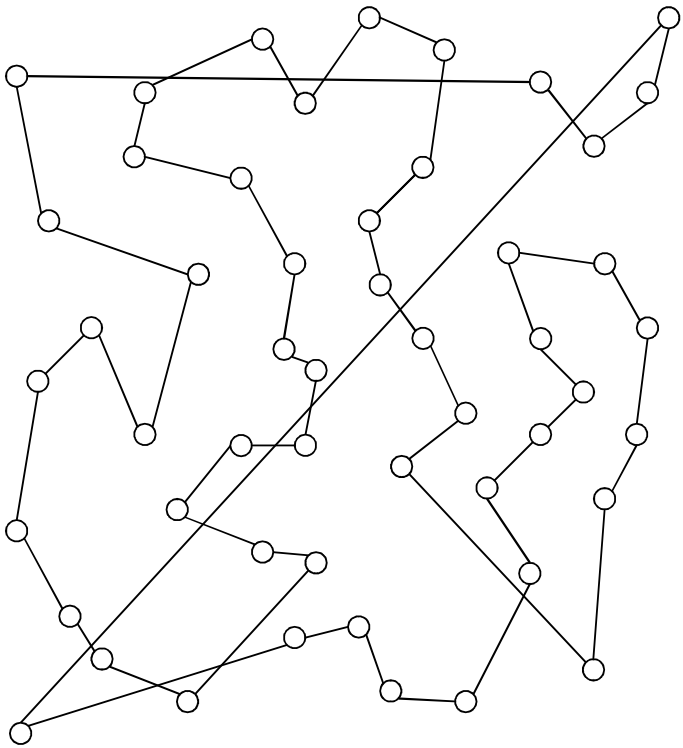

FIGURE 8.1: Exemplary nearest neighbor solution for a 51-city TSP instance ([CE69]).

Partitioning Heuristics

Applying partitioning heuristics to the TSP means splitting the total number of cities into smaller sets according to their geographical position. The emerging subsets are treated and solved as independent TSPs and the solution of the original TSP is given by a combination of the partial solutions.

The success rate of partitioning heuristics for TSPs very much depends on the size and the topological structure of the TSP. Partitioning heuristics do not perform well in the general case. Particularly suitable for partitioning heuristics are only higher dimensional Euclidean TSPs with rather uniformly distributed cities. Algorithms based on partitioning have been presented in [Kar77], [Kar79], and [Sig86].

Local Search

Typical representatives of route improving heuristics are the so-called k-change methods that examine a k-tuple of edges of a given tour and test whether or not a replacement of the tour segments effects an improvement of the actual solution quality.

A lot of established improvement methods are based upon local search strategies also causing the nomenclature "neighborhood search." The basic idea is to search through the surroundings of a certain solution s_i in order to replace s_i by an eventually detected "better" neighbor s_j.

The formal description of the neighborhood structure is given by $N \subseteq \mathcal{S} \times \mathcal{S}$ with $\mathcal{S}$ denoting the solution space. The choice of N is up to the user with the only restriction that the corresponding graph has to be connected and undirected, i.e., the neighborhood structure should be designed in a way that any point in the solution space is reachable and that s_i being a direct neighbor of s_j implies s_j being a direct neighbor of s_i.

Mainly for reasons of implementation the following formal definition has been established:

$$N \subseteq \mathcal{S} \times \mathcal{S}$$

with

$$N(s_i) := \{s_j \in \mathcal{S} \mid (s_i, s_j) \in \mathrm{N}\}.$$

Choosing a neighborhood of larger size can cause problems concerning computational time whereas a rather small neighborhood increases the probability of getting stuck in a local optimum [PS82]. The search process for a better solution in the neighborhood is performed successively until no better solution can be detected. Such a point is commonly referred to as a local minimum with respect to a certain neighborhood structure and the neighborhood structure is termed definite if and only if any local optimum coincides with the global optimum (optima) s^* due to the neighborhood. Unfortunately, the verification of a definite neighborhood itself mostly is a NP-complete problem [PS82].

The 2-Opt Method

The most popular local edge-recombination heuristic is the 2-change replacement of two edges. In this context the neighborhood is defined in the following way:

> A tour s_i is adjacent (neighboring) to a tour s_j *if and only if* s_j can be derived from s_i by replacing two of s_i's edges.

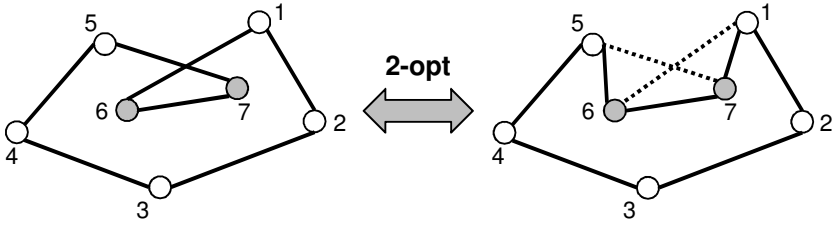

FIGURE 8.2: Example of a 2-change for a TSP instance with 7 cities.

Numbering the cities in the order they are visited with $c_1 \dots c_n$ yields the following representation of two adjacent routes:

$$(c_1 \dots c_i c_{i+1} \dots c_j c_{j+1} \dots c_n) \longleftrightarrow (c_1 \dots c_i c_j \dots c_{i+1} c_{j+1} \dots c_n)$$

Figure 8.2 illustrates one possible 2-change operation for a small TSP instance. In this example (assuming that the left tour is transformed to the right tour) the two edges $5-7$ and $6-1$ are removed and the edges $5-6$ and $7-1$ are inserted in order to reestablish a valid tour.

Any route s_j can be derived from any other route s_i by at most $(n-2)$ 2-change operations [AK89] and any solution s_i has exactly $\frac{n(n-1)}{2}$ neighboring (adjacent) solutions. For a symmetrical TSP (as indicated in the example of Figure 8.2) the number of neighboring solutions reduces to $\frac{n(n-2)}{2}$ [GS90].

Already half a century ago Croes [Cro58] published a solution technique for the TSP which is based upon the 2-change method: The algorithm has to check if an existing route s_i can be upgraded by the 2-change operator and perform it where applicable. This process is repeated until $f(s_j) \geq f(s_i)$ for all s_j that can be generated by using 2-change and the resulting route is called 2-optimal. Unfortunately, it is very unlikely that a 2-optimal tour is globally optimal.

The 3-Opt Method

The 3-opt method is very similar to the 2-opt method with the exception that not two but three edges are replaced. Considering a route with n nodes being involved $\frac{n(n-1)(n-2)}{2}$ different 3-change operations are possible [GS90].

$$(c_1 \dots c_i c_{i+1} \dots c_j c_{j+1} \dots c_k c_{k+1} \dots c_n) \longleftrightarrow (c_1 \dots c_i c_{j+1} \dots c_k c_{i+1} \dots c_j c_{k+1} \dots c_n)$$

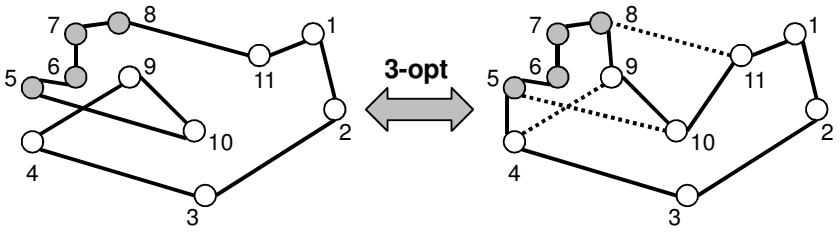

FIGURE 8.3: Example of a 3-change for a TSP instance with 11 cities.

Figure 8.3 illustrates one possible 3-change operation for a small TSP instance. In this example (assuming that the left tour is transformed to the right tour) the three edges $4-9$, $5-10$, and $8-11$ are removed and the edges $4-5$, $8-9$, and $10-11$ are inserted in order to reestablish a valid tour.

Also already half a century ago, Bock [Boc58] was the first one who applied the 3-opt method to the TSP. Similar as for the 2-opt method the final route was derived by successively applying 3-change operations terminates to a so-called 3-optimal solution. The probability to obtain a global optimal solution using the 3-opt method was empirically detected to be about $2^{-\frac{n}{10}}$ [Lin65].

The k-opt Method

In principle, the k-opt method is the consequential generalization of the methods described previously: k edges are replaced in a k-change neighborhood structure and a route is called k-optimal if it cannot be improved by any k-change. If $k = n$ then it is proven that the k-optimal solution is the global solution [PS82]. But as the complexity of locating a k-optimal solution is given by $\mathcal{O}(n^k)$ [GBD80], the computational effort is still enormous even for rather small values of k. A very efficient implementation for Euclidean traveling salesman problems is the Lin-Kernighan algorithm [LK73]. An efficient implementation is given in [Hel00].

8.1.3 Multiple Traveling Salesman Problems

The multiple traveling salesman problem (MTSP) describes a generalization of the TSP in the sense that there is not just one traveling salesman performing the whole tour but rather a set of salesmen, each serving a subset of the cities involved. Therefore, one of the cities has to be selected as the location for the depot representing the starting as well as the end point of all routes. So the MTSP is a combination of the assignment problem and the TSP. Usually a tour denotes the set of cities served by one traveling salesman and the number of tours is specified by m.

In literature there are mainly two definitions of the MTSP:

- In Bellmore's definition (given in [BH74]) the task is to find exactly m tours in such a way that each city in a tour and the depot are visited exactly once with the objective to minimize the total way.
- The second definition of the MTSP (as given in [Ber98], e.g.) does not postulate exactly but at most m routes and the goal is to minimize the total distance if each tour includes the depot and each city is visited exactly once in some tour.

At first sight the second definition seems more reasonable because there is no comprehensible reason why one should consider m tours if there is a solution involving only $(m-1)$ tours, for example. Still, one has to be aware of the fact that in the second definition with no additional constraints the solution will always be a single tour including all cities for any distance matrix fulfilling the triangle inequality.

8.1.4 Genetic Algorithm Approaches

Sequencing problems as for example the TSP are among the first applications of genetic algorithms, even if the classical binary representation as suggested in [Gol89] is not particularly suitable for the TSP because crossover hardly ever produces valid descendants.

In the following we will discuss some GA coding standards for the TSP as proposed in the relevant GA and TSP literature:

8.1.4.1 Problem Representations and Operators

Adjacency Representation

In the adjacency representation [GGRG85] a tour is represented as a list of n cities where city j is listed in position i if and only if the tour leads from city i to city j. Thus, the list

$$(7 \quad 6 \quad 8 \quad 5 \quad 3 \quad 4 \quad 2 \quad 1)$$

represents the tour

$$3-8-1-7-2-6-4-5.$$

In the adjacency representation any tour has its unique adjacency list representation. An adjacency list may represent an illegal tour. For example,

$$(3 \quad 5 \quad 7 \quad 6 \quad 2 \quad 4 \quad 1 \quad 8)$$

represents the following collection of cycles:

$$1-3-7, \quad 2-5, \quad 4-6, \quad 8$$

Obviously, the classical crossover operator(s) (single or n-point crossover) are very likely to return illegal tours for the adjacency representation. Therefore, the use of a repair operator becomes necessary.

Other operators for crossover have been defined and investigated for this kind of representation:

- Alternating Edge Crossover :
  The alternating edge crossover [GGRG85] chooses an edge from the first parent at random. Then, the partial tour created in this way is extended with the appropriate edge of the second parent. This partial tour is extended by the adequate edge of the first parent, etc. By doing so, the partial tour is extended by choosing edges from alternating parents. If an edge is chosen which would produce a cycle into the partial tour, then the edge is not added; instead, the operator randomly selects an edge from the edges which do not produce a cycle.
  For example, the result of an alternating edge crossover of the parents

  $$(2 \quad 3 \quad 8 \quad 7 \quad 9 \quad 1 \quad 4 \quad 5 \quad 6) \qquad (7 \quad 5 \quad 1 \quad 6 \quad 9 \quad 2 \quad 8 \quad 4 \quad 3)$$

  could for example be

  $$(2 \quad 5 \quad 8 \quad 7 \quad 9 \quad 1 \quad 6 \quad 4 \quad 3)$$

  The first edge chosen is $(1-2)$ included in the first parent's genetic material; the second edge chosen, edge $(2-5)$, is selected from the second parent, etc. The only randomly introduced edge is $7-6$ instead of $7-8$. Nevertheless, experimental results using this operator have been discouraging. The obvious explanation seems to be that good subtours are often disrupted by the crossover operator. Ideally, an operator ought to promote longer and longer high performance subtours; this has motivated the development of the following operator.

- Subtour Chunks Crossover:
  Using the subtour chunks crossover [GGRG85] an offspring is constructed from two parent tours in the following way: A random subtour of the first parent is chosen, and this partial tour is extended by choosing a subtour of random length from the second parent. Then, the partial tour is extended by taking subtours from alternating parents. If the use

of a subtour, which is selected from one of the parents, would lead to an illegal tour, then it is not added; instead, an edge is added which is randomly chosen from the edges that do not produce a cycle.

- Heuristic Crossover:
  The heuristic crossover [GGRG85] starts with randomly selecting a city for being the starting point of the offspring's tour. Then, the edges starting from this city are compared and the shorter of these two edges is chosen. Next, the city on the other side of the chosen edge is selected as a reference city. The edges which start from this reference city are compared and the shortest one is added to the partial tour, etc. If at some stage a new edge introduces a cycle into the partial tour, then the tour is extended with an edge chosen at random from the remaining edges which do not introduce cycles.

The main advantage of the adjacency representation is that it allows schemata analysis as described in [OSH87], [GGRG85], [Mic92]. Unfortunately, the use of all operators described above lead to poor results; in particular, the experimental results with the alternating edge operator have been very poor. This is because this operator often destroys good subtours of the parent tours. The subtour chunk operator which chooses subtours instead of edges from the parent tours performs better than the alternating edge operator. However, it still has quite a low performance because it does not take into account any information available about the edges. The heuristic crossover operator performs far better than the other two operators; still, the performance of the heuristic operator is not remarkable either [GGRG85].

Ordinal Representation

When using the ordinal presentation as described in [GGRG85] a tour is also represented as a list of n cities; the i-th element of the list is a number in the range from 1 to $n - i + 1$, and there an ordered list of cities serving as a reference point is also used.

The easiest way to explain the ordinal representation is probably by giving an example. Assume, for instance, that the ordered list L is given as

$$L = (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7).$$

Now the tour

$$1 - 2 - 7 - 5 - 6 - 3 - 4$$

in ordinal representation is given as

$$T = (1 \quad 1 \quad 5 \quad 3 \quad 3 \quad 1 \quad 1).$$

This can be interpreted in the following way: The first member of T is 1, which means that in order to get the first city of the tour we take the first element of

the list L and remove it from the list. So the partial tour is 1 at the beginning. The second element of T is also 1 so the second city of the route is 2 which is situated at the first position of the reduced list. After removing city 2 from the list, the next city to add is in position 5 according to T, which is city 7 in the again reduced list L, etc. If we proceed in this way until all elements of L are removed, we will finally find the tour $1-2-7-5-6-3-4$ with the corresponding ordinal representation $T = (1 \quad 1 \quad 5 \quad 3 \quad 3 \quad 1 \quad 1)$. The main advantage of this rather complicated ordinal representation lies in the fact that the classical crossover can be used. This follows from the fact that the i-th element of the tour representation is always a number in the range from 1 to $n-i+1$. It is self-evident that partial tours to the left of the crossover point do not change whereas partial tours to the right of the crossover point are split in a quite random way and, therefore, the results obtained using ordinal representation have been generally poor (approximately in the dimension of the results with adjacency representation) [GGRG85], [LKM+99].

Path Representation

The path representation is probably the most natural representation of a tour. Again, a tour is represented as a list of n cities. If city i is the j-th element of the list, city i is the j-th city to be visited. Hence the tour

$$1-2-7-5-6-3-4$$

is simply represented by

$$(1 \quad 2 \quad 7 \quad 5 \quad 6 \quad 3 \quad 4).$$

Since the classical operators are not suitable for the TSP in combination with the path representation, other crossover and mutation operators have been defined and discussed. As this kind of representation will be used for our experiments in Chapter 10, we shall now discuss the corresponding operators in a more detailed manner:

- Partially Matched Crossover (PMX):
  The partially matched crossover operator has been proposed by Goldberg and Lingle in [GL85]. It passes on ordering and value information from the parent tours to the offspring tours: A part of one parent's string is mapped onto a part of the other parent's string and the remaining information is exchanged.

  Let us for example consider the following two parent tours

  $$(a \quad b \quad c \quad d \quad e \quad f \quad g \quad h \quad i \quad j) \quad \text{and} \quad (c \quad f \quad g \quad a \quad j \quad b \quad d \quad i \quad e \quad h).$$

  The PMX operator creates an offspring in the following way: First, it randomly selects two cutting points along the strings. As indicated in Figure 8.4, suppose that the first cut point is selected between the fifth

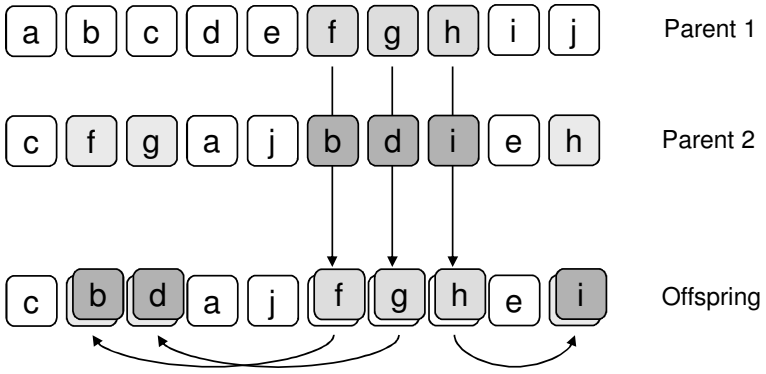

FIGURE 8.4: Example for a partially matched crossover (adapted from [Wen95]).

and the sixth element and the second one between the eighth and ninth string element. The substrings between the cutting points are called the mapping sections. In our example they define the mappings $f \leftrightarrow b$, $g \leftrightarrow d$, and $h \leftrightarrow i$. Now the mapping section of the first parent is copied into the offspring resulting

$$(\square \quad \square \quad \square \quad \square \quad \square \quad f \quad g \quad h \quad \square \quad \square)$$

Then the offspring is filled up by copying the elements of the second parent; if a city is already present in the offspring then it is replaced according to the mappings. Hence, as illustrated in Figure 8.4, the resulting offspring is given by

$$(c \quad b \quad d \quad a \quad j \quad f \quad g \quad h \quad e \quad i)$$

The PMX operator therefore tries to keep the positions of the cities in the path representation; these are rather irrelevant in the context of the TSP problem where the most important goal is to keep the sequences. Thus, the performance of this operator for the TSP is rather poor, but we can easily imagine that this operator could perform well for other combinatorial optimization problems like the machine scheduling problem even if it has not been developed for such problem instances.

- Order Crossover (OX):
  The order crossover operator has been introduced by Davis in [Dav85]. For the first time it employs the essential property of the path representation, that the order of cities is important and not their position.

  It constructs an offspring by choosing a subtour of one parent preserving the relative order of the other parent. For example let us consider the

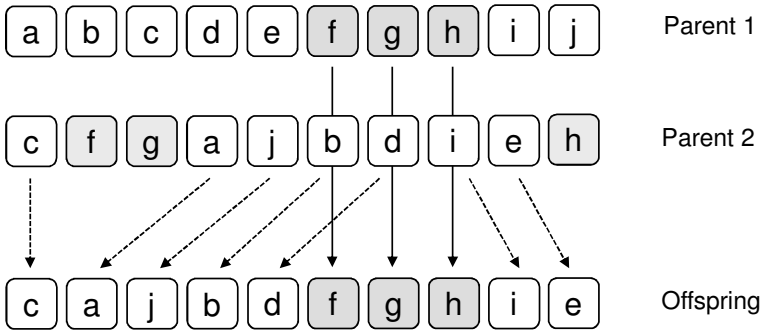

FIGURE 8.5: Example for an order crossover (adapted from [Wen95]).

following two parent tours

$$(a \quad b \quad c \quad d \quad e \quad f \quad g \quad h \quad i \quad j) \text{ and } (c \quad f \quad g \quad a \quad h \quad b \quad d \quad i \quad e \quad j)$$

and suppose that we select the first cut point between the fifth and sixth position and the second cut point between the eighth and ninth position. For creating the offspring the tour segment between the cut points of the first parent is copied into it, which gives

$$(\square \quad \square \quad \square \quad \square \quad \square \quad f \quad g \quad h \quad \square \quad \square)$$

Then the selected cities of the first parent's tour segment are canceled from the list of the second parent and the blank positions of the child are filled with the elements of the shortened list in the given order (as illustrated in Figure 8.5), which gives

$$(c \quad a \quad b \quad d \quad i \quad f \quad g \quad h \quad e \quad j)$$

Since a much higher number of edges is maintained, the results are unequivocally much better compared to the results achieved using the PMX operator.

- Cyclic Crossover (CX):
  The cyclic crossover operator, proposed by Oliver et al. in [OSH87], attempts to create an offspring from the parents where every position is occupied by a corresponding element from one of the parents.

  For example, again consider the parents

  $$(a \quad b \quad c \quad d \quad e \quad f \quad g \quad h \quad i \quad j) \text{ and } (c \quad f \quad g \quad a \quad h \quad b \quad d \quad i \quad e \quad j)$$

  and choose the first element of the first parent tour as the first element of the offspring. As node c can no longer be transferred to the child

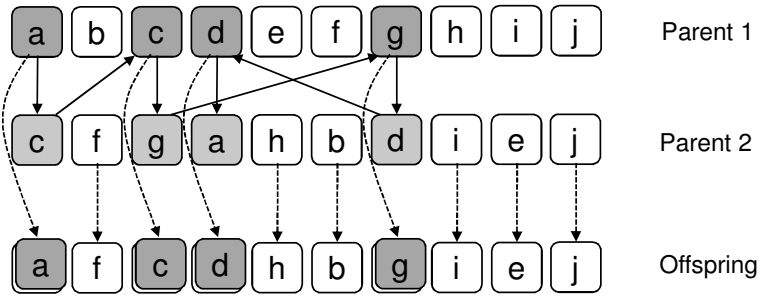

FIGURE 8.6: Example for a cyclic crossover (adapted from [Wen95]).

from the second parent, we visit node c in the first parent and transfer it to the offspring which makes it impossible for the first parent's node g to occupy the same position in the child. Therefore, g is taken from parent 2 and so on. This process is continued as long as possible, i.e., as long as the selected node is not yet a member of the offspring. In our example this is the case after four successful copies resulting in the following partial tour:

$$(a \quad \square \quad c \quad d \quad \square \quad \square \quad g \quad \square \quad \square \quad \square).$$

The remaining positions can then simply be taken from one of the two parents; in this example, which is graphically illustrated in Figure 8.6, these are taken from from parent 2).
Oliver et al. [OSH87] concluded from theoretical and empirical results that the CX operator gave slightly better results than the PMX operator. Anyway, the results of both position preserving operators CX and PMX are definitely worse than those obtained with OX which fortifies our basic assumption that in the context of the TSP it is much more important to keep sequences rather than positions.

- Edge Recombination Crossover (ERX):
  Even if the main aim of the OX operator is to keep the sequence of at least one parent there are still quite a lot of new edges in the offspring.[1] Whitley et al. [WSF89] tried to overcome this drawback and came up with the edge recombination crossover which has been designed with the objective of keeping as many edges defined by the parents as possible. Indeed it can be shown that about $95\% - 99\%$ of each child's edges occur in at least one of the two respective parents [WSF89]. Therefore, the ERX operator for the first time represented an almost mutation-free

[1] In the present contents "new" means that those edges do not occur in any of the two parents.

crossover operator, but unfortunately this can only be achieved by a quite complicated and time consuming procedure:
The ERX operator is an operator which is suitable for the symmetrical TSP as it assumes that only the values of the edges are important and not their direction. Pursuant to this assumption, the edges of a tour can be seen as the carriers of heritable information. Thus, the ERX operator attempts to preserve the edges of the parents in order to pass on a maximum amount of information to the offspring whereby the breaking of edges is considered as an unwanted mutation. The problem that usually occurs with operators that follow an edge recombination strategy is that they often leave cities without a continuing edge [GGRG85] whereby these cities become isolated and new edges have to be introduced.
The ERX operator tries to avoid this problem by first choosing cities that have few unused edges; still, there has to be a connection with a city before it can be selected. The only edge that the ERX operator may fail to enforce is the edge from the final city to the initial city which inhibits the ERX operator of working totally mutation free. When constructing an offspring (descendant), we first have to construct a so-called "edge map" which gives the edges for each of the parents that start or finish in it. Then, the ERX works according to the following algorithm [WSF89]

1. Choose the initial city from one of the two parent tours. It might be chosen randomly or according to criteria outlined in step 4. This is the "current city."
2. Remove all occurrences of the "current city" from the left hand side of the edge map.
3. If the current city has entities in its edge list go to step 4; otherwise, go to step 5.
4. Determine which of the cities in the edge-list of the current city has the fewest entities in its own edge list. The city with the fewest entities becomes the "current city"; ties are broken at random. Proceed with step 2.
5. If there are no remaining unvisited cities, then terminate; otherwise randomly choose an unvisited city and continue with step 2.

We will explain the functioning of ERX on the basis of a small example which has also been used in [Wen95]. Consider for instance the tours

$$(1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9) \quad \text{and} \quad (4\ \ 1\ \ 2\ \ 8\ \ 7\ \ 6\ \ 9\ \ 3\ \ 5)$$

The edge map for our example parent tours is given in Table 8.1.

Table 8.1: Exemplary edge map of the parent tours for an ERX operator.

city	connected cities
1	9, 2, 4
2	1, 3, 8
3	2, 4, 9, 5
4	3, 5, 1
5	4, 6, 3
6	5, 9, 7
7	6, 8
8	7, 9, 2
9	8, 1, 6, 3

According to the procedure given before, we select city 1 as the initial city. The edge map of city one shows that cities 9, 2, and 4 are the candidates for becoming the next current city. As city 9 actually has 4 (8,1,6,3) further links, we have to decide between the cities 2 and 4 which both have 3 further links. Choosing city 4 as the next current city we obtain 3 and 5 as the next candidates, etc. Proceeding in that way, we might finally end up with the offspring tour

$$(1 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 2 \quad 3 \quad 9)$$

which for this special case of our example is totally mutation free, i.e., all edges of the offspring occur in at least one of the two parents.

As common sequences of the parent tours are not taken into account by the ERX operator an enhancement, commonly denoted as "enhanced edge recombination crossover (EERX)", has been developed [SMM+91]. The EERX additionally gives priority to those edges starting from the current city which are present in both parents.

For mutation in the context of applying genetic algorithms to the TSP, the $2-change$ and $3-change$ techniques have turned out to be very successful [WSF89]. A comprehensive review of mutation operators for the TSP is given in [LKM+99]. In the following some of the most important mutation operators are described which are also applied in the experimental part of this book:

- Exchange Mutation:
  The exchange mutation operator selects two cities of the tour randomly and simply exchanges them. In various publications the exchange mutation operator is also referred to as swap mutation, point mutation, reciprocal exchange, or order-based mutation [LKM+99].

- Insertion Mutation:
  The insertion mutation operator [Mic92] randomly chooses a city, removes it from the tour and inserts it at a randomly selected place. An

alternative naming for insertion mutation is position-based mutation [LKM+99].

- Simple Inversion Mutation:
  The simple inversion mutation operator [Hol75], which is used in the TSP experiments of the book, randomly selects two cut points and simply reverses the string between them.

- Inversion Mutation:
  The inversion mutation operator [Fog93] randomly selects a subtour, removes it, and inserts it in reverse order at a randomly chosen position. An alternative naming for inversion mutation is cut-inversion mutation [LKM+99].

8.2 The Capacitated Vehicle Routing Problem

In principle, the vehicle routing problem (VRP) is a m-TSP where a demand is associated with each city, and the salesmen are interpreted as vehicles each having the same capacity. A survey of the VRP is for example given in [Gol84]. During the later years a number of authors have "renamed" this problem the capacitated vehicle routing problem (CVRP). The sum of demands on a route cannot exceed the capacity of the vehicle assigned to this route; as in the m-TSP we want to minimize the sum of distances of the routes. Note that the CVRP is not purely geographic since the demand may be constraining.

The CVRP is the basic model for a number of vehicle routing problems:

If a time slot, in which customers have to be visited, is added to each customer, then we get the "vehicle routing problem with time windows" (VRPTW or CVRPTW). In addition to the capacity constraint, a vehicle now has to visit a customer within a certain time frame given by a ready time and due date. It is generally allowed that a vehicle may arrive before the ready time (in this case it simply waits at the customer's place), but it is forbidden to arrive after the due date. However, some models allow early or late servicing but with some form of additional cost or penalty. These models are denoted "soft" time window models (as for example in [Bal93]).

If customers are served from several depots, then the CVRP becomes the "multiple depots vehicle routing problem" (MDVRP); in this variant each vehicle starts and returns to the same depot. The problem can be solved by splitting it into several single depot VRP problems if such a split can be done effectively. Another variant of the CVRP is the "vehicle routing problem with length constraints" (VRPLC or CVRPLC). Here each route is not allowed to exceed a given distance; this variant is also known as the "distance constrained

vehicle routing problem" (DVRP) in case there are no capacity restrictions and the length or cost is the only limiting constraint.

In the "split delivery" model the demand of a customer is not necessarily covered by just one vehicle but may be split between two or more. The solutions obtained in a split delivery model will always be at least as good as for the "normal" CVRP and we often might be able to utilize the vehicles better and thereby save vehicles.

Finally we shall also mention the "pickup and delivery" variant where the vehicles not only deliver items but also pick up items during the routes. This problem can be varied even more according to whether the deliveries must be completed before starting to pick up items or the two phases can be interleaved.

All of these problems have in common that they are "hard" to solve. For the VRPTW exact solutions can be found within reasonable time for some instances including up to about 100 customers. A review of exact methods for the VRPTW is given in Subsection 8.2.1.2.

Often the number of customers combined with the complexity of real-life data does not permit solving the problems exactly. In these situations it is commendable to apply approximation algorithms or heuristics. Both can produce feasible, but not necessarily optimal solutions; whereas a worst-case deviation is known for approximation algorithms, nothing is known a priori for heuristics. Some of these inexact methods will be reviewed in Subsection 8.2.1.3.

If the term "vehicle" is interpreted more loosely, numerous scheduling problems can also be modeled as CVRPs or VRPTWs. An example is the following one: For a single machine we want to schedule a number of jobs for which we know the flow time and the time to go from one running job to the next one. This scheduling problem can be regarded as a VRPTW with a single depot, a single vehicle, and the customers representing the jobs. The cost of changing from one job to another is equal to the distance between the two customers, and the time it takes to perform an action is the service time of the job. For a general description of the connection between routing and scheduling see for instance [vB95] or [CL98].

8.2.1 Problem Statement and Solution Methodology

8.2.1.1 Definition of the CVRP

In this section we present a mathematical formulation of the general vehicle routing problem with time windows (VRPTW or CVRPTW) as the (capacitated) vehicle routing problem ((C)VRP) is fully included within this definition under special parameter settings. The formulation is based upon the model defined by Solomon [SD88].

In this description the VRPTW is given by a fleet of homogeneous vehicles

$\mathcal{V}$, a set of customers $\mathcal{C}$, and a directed graph $\mathcal{G}$. The graph consists of $|\mathcal{C}| + 2$ vertices, whereby the customers are denoted as $1, 2, \ldots, n$ and the depot is represented by the vertices 0 (the "driving-out depot") and $n + 1$ (the "returning depot"). The set of vertices $0, 1, \ldots, n + 1$ is denoted as $\mathcal{N}$; the set of arcs $\mathcal{A}$ represents connections between customers and between the depot and customers, where no arc terminates in vertex 0 and no arc originates from vertex $n + 1$. With each arc (i, j), where $i \neq j$, we associate a cost c_{ij} and a time t_{ij}, which may include service time at customer i.

Each vehicle j has a capacity q_j and each customer i a demand d_i. Furthermore, each customer i has a time window $[a_i, b_i]$; a vehicle can arrive before a_i, but service does not start before a_i); however, the vehicle must arrive at the customer before b_i. In the general description, the depot also has a time window $[a_0, b_0] = [a_{n+1}, b_{n+1}]$, the scheduling horizon. Vehicles may not leave the depot before a_0 and must be back before or at time b_{n+1}.

It is postulated that q, a_i, b_i, d_i, and c_{ij} are nonnegative integers, while the t_{ij} values are assumed to be positive integers. Furthermore, it is assumed that the triangular inequality is satisfied for both c_{ij} values as well as the t_{ij} values.

This model contains two sets of decision variables, namely x and s. For each arc (i, j), where $i \neq j, i \neq n + 1, j \neq 0$, and each vehicle k we define x_{ijk} in the following way:

$$x_{ijk} = \begin{cases} 0 & , \text{ if vehicle } k \text{ does not drive from vertex } i \text{ to vertex } j \\ 1 & , \text{ if vehicle } k \text{ drives from vertex } i \text{ to vertex } j \end{cases}$$

The decision variable s_{ik} is defined for each vertex i and each vehicle k denoting the time vehicle k starts to service customer i. If the given vehicle k doesn't service customer i, then s_{ik} does not mean anything. We assume $a_0 = 0$ and therefore $s_{0k} = 0$ for all k.

The goal is to design a set of routes with minimal cost, one for each vehicle, such that

- each customer is serviced exactly once,
- every route originates at vertex 0 and ends at vertex $n + 1$, and
- the time windows and capacity constraints are complied with.

The mathematical formulation for the VRPTW is stated as follows [Tha95]:

$$\min \sum_{k\in\mathcal{V}} \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}} c_{ij} x_{ijk} \quad s.t. \tag{8.9}$$

$$\sum_{k\in\mathcal{V}} \sum_{j\in\mathcal{N}} x_{ijk} = 1 \quad \forall i \in \mathcal{C} \tag{8.10}$$

$$\sum_{i\in\mathcal{C}} d_i \sum_{j\in\mathcal{N}} x_{ijk} \leq q \quad \forall k \in \mathcal{V} \tag{8.11}$$

$$\sum_{j\in\mathcal{N}} x_{0jk} = 1 \quad \forall k \in \mathcal{V} \tag{8.12}$$

$$\sum_{i\in\mathcal{N}} x_{ihk} - \sum_{j\in\mathcal{N}} x_{hjk} = 0 \quad \forall h \in \mathcal{C}, \forall k \in \mathcal{V} \tag{8.13}$$

$$\sum_{i\in\mathcal{N}} x_{i,n+1,k} = 1 \quad \forall k \in \mathcal{V} \tag{8.14}$$

$$s_{ik} + t_{ij} - K(1 - x_{ijk}) \leq s_{jk} \quad \forall i, j \in \mathcal{N}, \forall k \in \mathcal{V} \tag{8.15}$$

$$a_i \leq s_{ik} \leq b_i \quad \forall i \in \mathcal{N}, \forall k \in \mathcal{V} \tag{8.16}$$

$$x_{ijk} \in \{0, 1\} \quad \forall i, j \in \mathcal{N}, \forall k \in \mathcal{V} \tag{8.17}$$

The constraint (8.10) states that each customer is visited exactly once, and (8.11) implies that no vehicle is loaded with more than its capacity allows. Equations (8.12), (8.13), and (8.14) ensure that each vehicle leaves depot 0, leaves again after arriving at a customer, and finally arrives at the depot $n+1$. Inequality (8.15) states that a vehicle k cannot arrive at j before $s_{ik} + t_{ij}$ if it is traveling from i to j, whereby K is a large scalar. Finally, constraints (8.16) ensure that the time windows are adhered to and (8.17) are the integrality constraints. In this definition an unused vehicle is modeled by driving the empty route $(0, n + 1)$.

As already mentioned earlier, the VRPTW is a generalization of TSP and CVRP; in case the time constraints (8.15) and (8.16) are not binding, the problem becomes a CVRP. This can be achieved by setting $a_i = 0$ and $b_i = M$ (where M is a large scalar) for all customers i. In this context it should be noted that the time variables enable us to formulate the CVRP without subtour elimination constraints. If only one vehicle is available, then the problem is in fact a TSP.

8.2.1.2 Exact Algorithms

Almost all papers proposing an exact algorithm for solving the CVRP or the VRPTW use one or a combination of the following three principles:

- Dynamic programming
- Lagrange relaxation-based methods
- Column generation

The dynamic programming approach for the VRPTW was presented in [KRKT87]. This paper is inspired by an earlier publication [CMT81] where Christofides et al. used the dynamic programming paradigm to solve the CVRP.

Lagrange relaxation-based methods have been published in a number of papers using slightly different approaches. There are approaches applying variable splitting followed by Lagrange relaxation as well as variants applying the k-tree approach followed by Lagrange relaxation. In [FJM97] Fisher et al. presented a shortest path approach with side constraints followed by Lagrangean relaxation. The main problem, which consists of finding the optimal Lagrange multipliers that yield the best lower bounds, is solved by a method using both subgradient optimization and a bundle method. Kohl et al. [KM97] managed to solve problems of 100 customers from the Solomon test cases; among them some previously unsolved problems.

If a linear program contains too many variables to be solved explicitly, it is possible to initialize the linear program with a smaller subset of variables and compute a solution of this reduced linear program. Afterwards one has to check whether or not the addition of one or more variables, currently not in the linear program, might improve the solution; this check is commonly done by the computation of the reduced costs of the variables. An introduction to this method (commonly called "column generation method") can for example be found in [BJN$^+$98].

Again, similar as for the TSP it takes well versed users in order to benefit from the mentioned exact algorithms - especially if they are applied to large problems.

Therefore, the main area of application of exact methods in the context of CVRP is to locate the exact solution of some large benchmark problems in order to get some reference-problems for testing certain heuristics which can easily be applied to practical problems of higher dimension.

8.2.1.3 Approximation Algorithms and Heuristics

The field of inexact algorithms for the CVRP has been very active - far more active than that of exact algorithms, and a long series of papers has been published over the recent years. Heuristic algorithms that build a set of routes from scratch are typically called route-building heuristics, while an algorithm that tries to produce an improved solution on the basis of an already available solution is denoted as route-improving.

The Savings Heuristic

The savings heuristic has been introduced in [CW64]. At the beginning of the algorithm, each of the n customers (cities) is considered to be delivered with an own vehicle. For every pair of two cities a so-called savings value is calculated; this value specifies the reduction of costs which is achieved when

the two routes are combined. Then the routes are merged in descending order of their saving values if all constraints are satisfied. According to [Lap92] the time complexity of the savings heuristic is given as $\mathcal{O}(n^2 \log n)$.

A lot of papers based on savings heuristics have been published. Especially Gaskell's approach [Gas67] is appreciable in this context as it introduces a different weighting of the savings with respect to the length of the newly inserted route-part as well as the so-called parallel savings algorithm that not only examines a pair but rather a n-tuple of routes.

The Sweep Heuristic

The sweep heuristic has been introduced by Gillett and Miller [GM74]. It belongs to the so-called "successive methods" in the sense that the ultimate goal of this approach is not necessarily the location of the final solution but rather the generation of reasonable valid tours which can be optimized by some kind of route improving heuristic.

The fundamental idea of the sweep heuristic can be described as follows: Imagining a watch hand that is mounted at the depot, the sweep heuristic builds up the first tour starting from an arbitrary angle and takes the cities in the order the watch hand sweeps over them as long as all constraints are fulfilled. Then the next cluster is established in the same way.

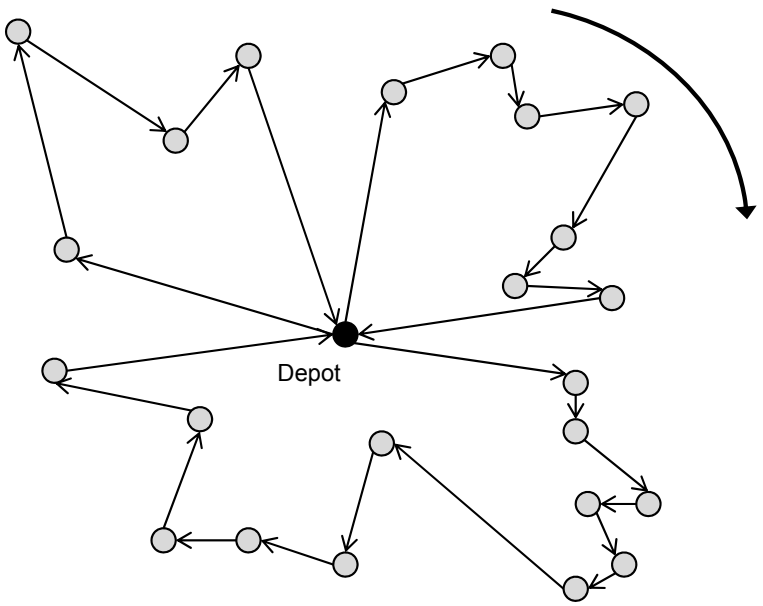

FIGURE 8.7: Exemplary result of the sweep heuristic for a small CVRP.

Figure 8.7 shows a possible solution achieved by a simple sweep heuristic. The more customers can be assigned to a route, the better the sweep heuristic typically performs. The time complexity of sweep heuristics is $\mathcal{O}(n \log n)$, which is equal to the complexity of a sorting algorithm.

The Push Forward Insertion Heuristic

In [Sol87] Solomon describes and evaluates three insertion heuristics for the VRPTW. Here a new route is started by a customer which minimizes a certain cost function cf_1. All unrouted customers are then evaluated for insertion between any two customers i and j of the partial route according to another cost function cf_2. If no feasible insertion is possible for all customers, cf_1 is evaluated again for all unrouted customers in order to determine the starting customer of a new route. Three different possible criteria for selecting the next customer are the following ones:

- Farthest customer from the depot first
- Customer with the earliest due date first
- Customer with the minimum equally weighted direct route-time and distance first

The third function basically describes the closest customer that will be directly reached in time. During the evaluation of their performance Solomon states that generally neither is better than the other. The farthest customer first criterion is suited for problems with shorter scheduling horizons, while selecting the customers regarding the earliest due date gives better results in situations where the scheduling horizons are longer, i.e., where vehicles have to visit more customers. The third alternative for cf_1 was not examined closer as it was not used in conjunction with the best performing alternative for cost function cf_2.

As Solomon notes, the three insertion heuristics that are described as alternatives for cf_2 are guided by both geographical and temporal criteria. The insertion heuristic, which is described first and termed *I1*, performed best in a number of test cases. Basically it extends the savings heuristic insofar as it takes into account the prolongation of the arrival time at the next customer. This function evaluates the difference between scheduling a customer directly and servicing it in an existing route between two customers. Mathematically it can be described as

$$I1(i,u,j) = \lambda t_{0u} - (\alpha_1(t_{iu} + t_{uj} - \mu t_{ij}) + \alpha_2(b_{j_u} - b_j)) \tag{8.18}$$

with the following restrictions: $\lambda, \mu, \alpha_1, \alpha_2 \geq 0$ and $\alpha_1 + \alpha_2 = 1$. In this case the VRP cost function c_{ij} equals to 1 for each pair of different customers i and j.

Tests with a choice of some configurations for λ, μ, α_1, and α_2 showed that good results were achieved with $\lambda = 2$, $\mu = 1$, $\alpha_1 = 0$, and $\alpha_2 = 1$, thus using only time-based savings instead of distance-based savings.

The name "push forward insertion heuristic" stems from a more efficient computation of the feasibility of an insertion. At each point, where a customer could be inserted, the time at which the vehicle would arrive later at the preceding customer is propagated sequentially through the route. As soon as this time becomes 0 the insertion is feasible as the remaining customers would not be serviced later than they already are. If the old partial route is feasible, then the new one thus will also be feasible. If the push forward value surpasses the due date at a customer, then an infeasible insertion is encountered and the rest of the route does not have to be checked. In the worst case this method still needs to perform the calculation for every customer in the tour. Feasibility regarding the capacity constraints, at least for the VRP variants without pickup & delivery, is easier to compute.

Solomon concludes that a hybridization of *I1* with a sweep heuristic could achieve excellent initial solutions with a reasonable amount of computation. Such an approach can be found in [TPS96] where cf_1 is a function taking into account three different properties: distance, due date, and the polar angle. The mathematical description reads

$$cf_1(u) = -\alpha t_{0u} + \beta b_u + \gamma \varphi_u \tag{8.19}$$

with empirically derived weights $\alpha = 0.7$, $\beta = 0.2$, and $\gamma = 0.1$.

Other Methods

The problem of building one route at a time (which is done when using the heuristics described above) is usually that the routes generated in the latter part of the process are of worse quality because the last unrouted customers tend to be scattered over the geographic area. Potvin and Rousseou [PR93] tried to overcome this problem of the insertion heuristic by building several routes simultaneously where the initialization of the routes is done by using Solomon's insertion heuristic:

On each route the customer farthest away from the depot is selected as a "seed customer." Then, the best feasible insertion place for each unserviced customer is computed and the customer with the largest difference between the best and the second best insertion place is inserted. Even if this method works out better than the Solomon heuristic it is still quite far away from optimum. Russell elaborates further on the insertion approach in [Rus95].

Another approach built up upon the classical insertion heuristic is presented in [AD95]. Defined in a very similar way to the Solomon heuristics, every unrouted customer requests an offer and receives a price for insertion from every route in the schedule. Then unrouted customers send a proposal to the route with the best offer, and each route accepts the best proposal among the

customers with the fewest number of alternatives. Therefore, more customers can be inserted in each iteration. If a certain threshold of routes is violated, then a certain number of customers is removed and the process is started again. The results of Antes and Derigs are comparable to those presented in [PR93]. As a matter of principle it has turned out that building several routes in parallel results in better solutions than building the routes one by one.

Similar to the route first schedule second principle mentioned previously, Solomon also suggests doing it the other way round in the "giant tour heuristic" [Sol86]. First all customers are scheduled in a giant route and then this route is divided into a number of routes. In the paper no computational results are given for the heuristic. Implementations of route-building heuristics on parallel hardware are reported for example in [FP93] and [Lar99].

8.2.2 Genetic Algorithm Approaches

Applying genetic algorithms to vehicle routing problems with or without time constraints is a rather young field of research and therefore, even if a lot of research work is done, no widely accepted standard representations or operators have yet been established. In the following we will in short discuss some of the more popular or promising proposals:

- A genetic algorithm for the VRPTW has been presented in [TOS94]. This algorithm uses the already mentioned cluster first route second method whereby clustering is done by a genetic algorithm while routing is done by an insertion heuristic. The GA works by dividing each chromosome into K divisions of N bits. The algorithm is based on dividing the plane by using the depot as the center and assuming the polar angle to each customer. Each of the divisions of a chromosome then represents the offset of the seed of a sector; the seeds here are polar angles that bound the sector and thereby determine the members of the sector.

- The genetic algorithm of Potvin and Bengio [PB96] operates on chromosomes of feasible solutions. The selection of parent solutions is stochastic and biased towards the best solutions. Two types of crossover, called RBX and SBX, are used. They rarely produce valid solutions and the results therefore have to undergo a repair phase as the algorithm only works with feasible solutions. The reduction of routes is often obtained by two mutation operators and the routes are optimized by local search every k iterations.
  The approach described in [PTMC02] is similar, but does not use trip delimiters in the actual representation. Instead, the routes that a solution is composed of are saved separately as ordered sets. The number of routes is not predefined and practically only limited by the number of customers. The crossover described in [PTMC02] is biased insofar as it uses distance information to decide where to insert a segment, though

an unbiased generic variant of this crossover has been defined later in [TMPC03]. A repair method is applied (after applying the crossover) which constructs a feasible solution by removing customers that are visited twice. Additionally, any route that has become too long (so that the demand or time window constraints would not be satisfied) is split before the violating customer. This algorithm has been applied on several benchmark instances of the CVRP and CVRPTW problem variants.

- An encoding similar to the TSP is used in [Zhu00] and [Pri04]. As described in these publications, a GA optimizes solutions for the problem using a path representation that does not include the depot. The representation is thus similar to the TSP and the subtours are identified deterministically whenever needed. [Pri04] describes a genetic algorithm that is applied to the DVRP using a splitting procedure that will find the optimal placement of the depots; this guarantees the feasibility of solutions in any case. Additionally, classic operators like the order crossover (as used for tackling the TSP) are applied without modifications. This approach does not rely solely on unbiased operators and is hybridized with a local search procedure, while [Zhu00] makes use of biased crossover operators in which distance information is used for determining the point where a crossing might occur. In this approach an initial population, consisting of individuals created by suited construction heuristics as well as randomly generated individuals, is also used. Additionally, the mutation probability is adjusted depending on the diversity in the population leaving a minimum probability of 6%.

- A cellular genetic algorithm has been proposed in [AD04]. It uses an encoding with unique trip delimiters such that the customers are assigned numbers from 0 to $(|\mathcal{C}| - 1)$ while the trip delimiters are numbers from $|\mathcal{C}|$ to $(|\mathcal{C}| + |\mathcal{V}| - 1)$. The representation of a solution thus consists of a string of consecutive numbers and is syntactically equal to a TSP path encoding. This allows the use of crossover operators known from the TSP such as the ERX. For mutating solutions the authors use insertion, swap, and inversion operators which are similar to relocate, exchange, and 2-opt as described below. The difference is that swap and inversion are used in an inter- as well as an intraroute way. There is also a local search phase which is conducted after every generation; in this phase all individuals are optimized by 2-opt and λ-interchanges. The best results have been achieved using both methods and setting $\lambda = 2$.

8.2.2.1 Crossover Operators

Sequence-Based Crossover (SBX)

The sequence-based crossover (SBX) operator has been proposed for the CVRP in [PB96], but it is applicable also to other VRP variants as well.

Generally, it assumes a path representation with trip delimiters, but can be applied on a representation without trip delimiters in which case the subtours have to be calculated first.

It works by breaking a subtour in each parent and linking the first part of one parent to the second part of another parent. The newly created subtour is then copied to a new offspring individual and completed with the genetic information of one of the parents. This behavior is exemplarily illustrated in Figure 8.8.

This operator is very likely to create ill-formed children with duplicate or unrouted customers; therefore the authors also propose a repair method which creates syntactically valid genetic representations. However, feasibility cannot be guaranteed in every case as it is not always possible to find a feasible insertion space for all unrouted customers. In such a case the offspring is discarded and the crossover is executed anew with a new pair of parents. It is stated in [PB96] that when applied on the Solomon benchmark set [Sol87] 50% of the offspring are infeasible.

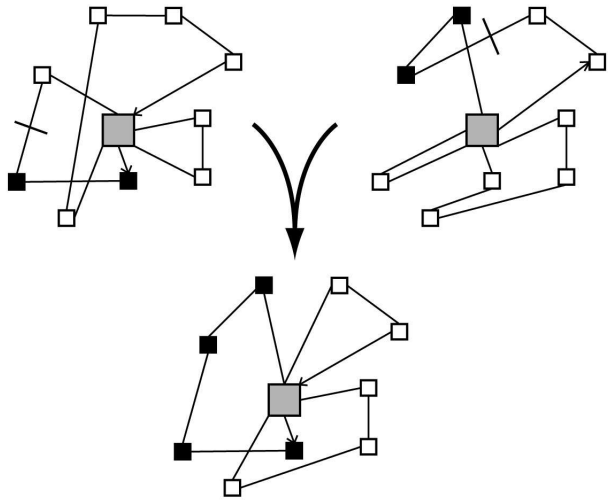

FIGURE 8.8: Exemplary sequence-based crossover.

Let us for example consider the following tours in path representation with trip delimiters where the depot is denoted as 0 and all other values represent customers.

$$(0 \quad 1 \quad 2 \quad 3 \quad 0 \quad 4 \quad 5 \quad 6 \quad 0) \quad \text{and} \quad (0 \quad 2 \quad 5 \quad 3 \quad 0 \quad 1 \quad 4 \quad 6 \quad 0)$$

In this case the SBX would randomly select two cut points in both solutions, for example at customer 2 in the first parent and customer 4 in the second

one. Then the first half of the route in the first solution is concatenated with the second half of the route in the second solution yielding

$$(0 \quad 1 \quad 2 \quad 6 \quad 0)$$

This route then replaces the route with the selected customer in the first solution; thus the solution becomes

$$(0 \quad 1 \quad 2 \quad 6 \quad 0 \quad 4 \quad 5 \quad 6 \quad 0)$$

Obviously, now customer 6 is served twice, while customer 3 is not served at all and the repair procedure will have to correct this situation: First it will remove all duplicate customers in all routes except the new one which was just formed by the concatenation. This results in

$$(0 \quad 1 \quad 2 \quad 6 \quad 0 \quad 4 \quad 5 \quad 0)$$

Then it will try to route all unserviced customers in that location in which the detour is minimal. For this example let us assume that this is after customer 5 and the final offspring solution thus is

$$(0 \quad 1 \quad 2 \quad 6 \quad 0 \quad 4 \quad 5 \quad 3 \quad 0)$$

Route-Based Crossover (RBX)

The route-based crossover (RBX) operator has also been proposed in [PB96]. It differs from the SBX insofar as subtours are not merged, but rather a complete subtour of one parent is copied to the offspring individual filling up the rest of the chromosome with subtours from the other parent. This procedure is exemplarily illustrated in Figure 8.9. Again, the operator does not produce feasible solutions in all cases and a repair procedure is needed.

Let us again consider the following tours in path representation with trip delimiters where 0 denotes the depot and all other values represent customers.

$$(0 \quad 1 \quad 2 \quad 3 \quad 0 \quad 4 \quad 5 \quad 6 \quad 0) \quad \text{and} \quad (0 \quad 2 \quad 5 \quad 3 \quad 0 \quad 1 \quad 4 \quad 6 \quad 0)$$

The RBX randomly selects a complete route in solution 1, in this case for example the first route:

$$(0 \quad 1 \quad 2 \quad 3 \quad 0)$$

This route then replaces a route in the second solution and thus we get

$$(0 \quad 1 \quad 2 \quad 3 \quad 0 \quad 1 \quad 4 \quad 6 \quad 0)$$

Obviously, now customer 1 is served twice, while customer 5 is not served at all. The same repair procedure as the one described for the SBX operator is applied here as well, resulting in the following possible final solution:

$$(0 \quad 1 \quad 2 \quad 3 \quad 0 \quad 4 \quad 6 \quad 5 \quad 0)$$

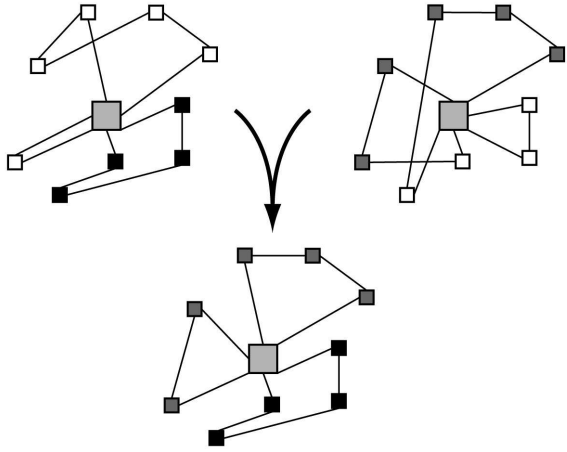

FIGURE 8.9: Exemplary route-based crossover.

Other Crossover Operators

The crossover that is introduced in [PTMC02] does not necessarily concatenate the opposite ends of two partial routes such as the SBX, but inserts a partial route of one solution into a good location of another solution. The fitness of such a location is determined by the distance between the customer which would precede the new partial route and the first customer of that partial route. Such an approach works well when solving the CVRP, but as is noted in [TMPC03] it does not help in the CVRPTW; this is in fact not surprising as distance alone might not be sufficient enough to determine the fitness of an insert when there are additional constraints which to a certain degree determine the shape of a route. Thus, a generic variant of the crossover is proposed which works similar to the RBX, except that it does not replace, but rather appends the new route to a given solution. Removing customers which are served twice then is the only necessary repair method. Such a removal also has the benefit that any solution remains feasible if it has been feasible before.

Other genetic algorithm approaches such as the one described in [Pri04] build on a path representation without trip delimiters as in the TSP. This allows the application of those crossover operators that have been mentioned in Section 8.1.4.1 without modifications. In [Zhu00] the PMX is compared to two new crossovers called "heuristic crossover" and "merge crossover" that take into account spatial and temporal features of the customers. However, these new operators achieve slightly better results only for those problem instances in which the customers are clustered, whereas for the other cases of randomly placed customers and a mix of random and clustered customers PMX showed better results.

8.2.2.2 Mutation Operators

Relocate

The relocate operator moves one of the customers within the solution string from one location to another randomly chosen one. An example of this behavior is illustrated in Figure 8.10.

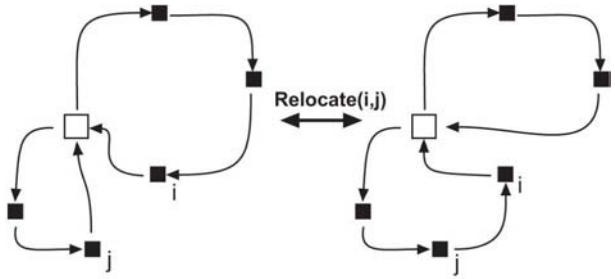

FIGURE 8.10: Exemplary relocate mutation.

Exchange

The exchange operator selects two customers within different tours and switches them so that they are served by the other vehicle, respectively. Both relocate and exchange operators are similar to a $(1, 0)$ and $(1, 1)$ λ-exchange defined by Osman [Osm93]. An example of the exchange behavior is shown in Figure 8.11.

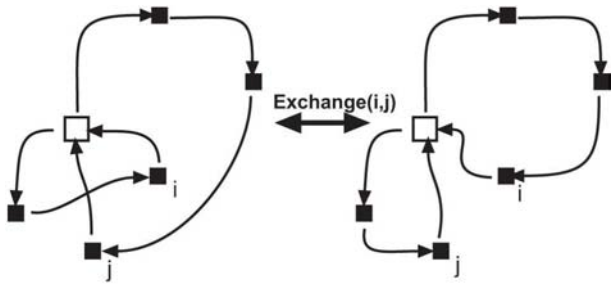

FIGURE 8.11: Exemplary exchange mutation.

2-Opt

The 2-opt operator selects two sites within the same route and inverts the route between them, so that the vehicle travels in the opposite direction. An example of this behavior is given in Figure 8.12.

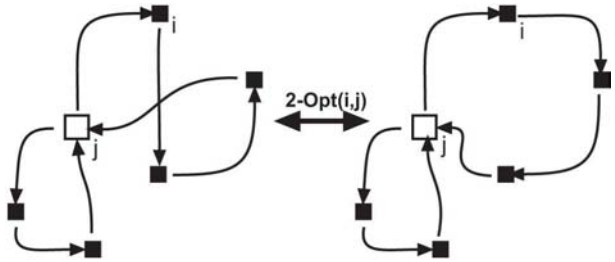

FIGURE 8.12: Example for a 2-opt mutation for the VRP.

2-Opt*

The 2-opt* operator behaves like a one point crossover operator in a tour: It first selects two customers in two different tours and creates two new tours; the first tour here consists of the first half of the first tour unified with the second half of the second tour, and the second tour consists of the first half of the second tour unified with the second half of the first tour. An example for the behavior of this operator is illustrated in Figure 8.13.

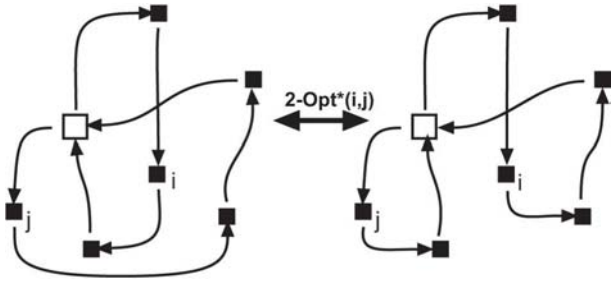

FIGURE 8.13: Example for a 2-opt* mutation for the VRP.

Or-Opt

The or-opt operator takes a number of consecutive customers, deletes them and inserts them at some point of the same tour. An example of this behavior is given in Figure 8.14.

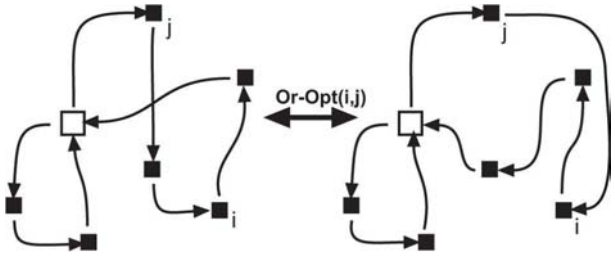

FIGURE 8.14: Example for an or-opt mutation for the VRP.

One Level Exchange (M1)

The one level exchange (M1) operator is mentioned in the context of the GA proposed in [PB96]. It tries to eliminate routes by inserting the customers into other routes while maintaining a feasible solution. This operator favors small routes with higher probability, because these are in general easier to remove. The probability is chosen such that a trip of size N is half as likely to be chosen as one of size $\frac{N}{2}$.

Two Level Exchange (M2)

The two level exchange (M2) operator looks one level deeper than the M1 operator as it removes routes from the whole route, but tries harder for each of the customers. After selecting a trip using the same bias towards smaller routes as the M1, each customer is tried to be inserted instead of another customer in a different route which in turn is tried to be inserted in any other place (except the originally selected trip). If such a feasible insertion is found, then the second selected customer is inserted and the first customer is inserted at the second customer's original place. This operator is more likely to find feasible insertion places, but requires quite a lot of computational effort; in the worst case the runtime complexity is $O(N^2)$.

Local Search (LSM)

Another operator which is also proposed in [PB96] applies several or-opt exchanges until a local optimum is reached. It first selects all possible combinations of three consecutive customers and tries to insert them in any other

place while still maintaining the feasibility of the solution. If no solution can be found the process is repeated with two consecutive customers and finally with every single customer. Once a better solution has been found, the operator starts again with three consecutive customers and continues until no further improvement is possible. Local search methods can improve solutions on the one hand, but on the other hand they also might reduce the diversity in a population when applied to several individuals which could end in the same local minimum.