Routing and Scheduling

14

Routing and scheduling are part of the operative decisions required for running *daily operations*. The prerequisites are as follows. Typically, we are given known capacities that cannot be extended at short notice so that resource scarceness cannot always be prevented. Furthermore, detailed information about the demand to be fulfilled is available.

The central challenge of operative decision making is now to match demand with the available capacities on a daily basis. Here, two basic decision tasks have to be solved (separately or in parallel): (1) demand has to be assigned to resources and (2) schedules have to be set up. A *schedule* describes the sequence in which the assigned tasks are executed and at which starting points individual operations are initiated. The major difficulty is to ensure that the available capacities are not exceeded and that resources are deployed at the highest efficiency.

In Sect. 14.1 we introduce a typical case for operative decision making. Section 14.2 introduces *mathematical graphs* as a tool for representing decision scenarios in a network structure. Additionally, first insights into the algorithmic processing of graph-data as the basic ingredient for decision making in network structures are provided. The consideration of complex restrictions during the deployment of a resource is discussed in Sect. 14.3 by means of the so-called *traveling salesman problem* (TSP) in which the sequencing of operations to build a schedule for a resource is in the focus of decision making. The integrated consideration of *assignment and scheduling/sequencing* decision problems under limited resource availability is addressed in Sect. 14.4 in the context of the *capacitated vehicle routing problem* (CVRP). In Sect. 14.5 a short introduction into the scheduling of the production machines is given. Finally, Sect. 14.6 summarizes the key aspects of this chapter.

Find additional case-studies, Excel spreadsheet templates, and video streams in the *E-Supplement* to this book on www.global-supply-chain-management.de!

D. Ivanov et al., *Global Supply Chain and Operations Management*, Springer Texts in Business and Economics, https://doi.org/10.1007/978-3-319-94313-8_14

The Learning Objectives for This Chapter

- Learn about typical operative process planning tasks from transportation as well as production.
- Understand the representation of networks in mathematical graphs.
- Define decision models in mathematical graphs.
- Select adequate algorithms for solving optimization models defined in graphs.
- Understand basic ingredients of heuristic algorithms for solving complex routing and scheduling models with several constraints.
- Understand basic objectives and trade-offs in routing and scheduling.
- Compute optimal and sub-optimal (heuristic) production schedules.

14.1 Introductory Case Study RED SEA BUS TRAVEL

The leading actor in this chapter is Christina. She has just completed her study program in Global Supply Chain and Operations Management. Equipped with a recently acquired degree, she wants to gain some job experience abroad. So she applies for several jobs in the well-known Hurghada Red Sea area in Egypt. After several interviews, she finally gets a job at RED SEA BUS TRAVEL (RSBT). This company offers transport services in the Hurghada region. International travel agencies book RSBT service operations in order to ensure that their tourists are moved to their preferred holiday region.

On her first working day, Christina is assigned to the planning and dispatching office of RSBT in downtown Hurghada. After several complicated years of business due to a lack of tourists, RSBT has detected an increasing number of bookings from travel agencies, but also from individual customers. These bookings can be classified into the following four categories:

– Limousine Services (LS)
– Sightseeing Tours (SST)
– Airport Arrival Transfer (AAT)
– Airport Departure Transfer (ADT)

RSBT receives all bookings at short notice, e.g. on the day before the transport service is requested. Service (transport) processes have to be set up in order to cover customer demand. RSBT operates a fleet of (mini)buses of different sizes and has qualified drivers. Assignment of bookings to buses has to be decided, and for each bus the daily schedule has to be determined.

Christina's task is now to analyze the four products, LS, SST, AAT, and ADT, and to make suggestions about how the day-to-day deployment of the available buses (which are the available resources) must be carried out in order to use these resources in the most efficient manner while fulfilling all booking requests. Having accompanied several transport services during her first week at RSBT, Christina has identified the core planning challenges in the four business sectors:

LS: What is the *shortest (quickest) path* through the local street network to travel from the limousine service headquarters to the airport? How can this path be identified?
SST: In which *sequence* should all the tourist attractions be visited so that visitors have enough time to enjoy their leisure time in the hotel?
AAT: What is the minimal travel *distance* to bring all inbound passengers associated with an inbound flight to their hotels? What is the minimal number of required buses?
ADT: Which bus should *pick up* which guests from which hotel in order to take them to the airport on time if customers do not accept a pickup more than 5 h earlier than their scheduled flight departure?

Christina is now looking for suitable planning models for the four types of transportation service situations. Her intention is to make the determination of the processes traceable, but, at the same time, she wants to ensure that efficiency of the deployment resources is as high as possible.

14.2 Shortest Paths in a Network

This section addresses the optimization of travel paths in a network structure. In particular, we are looking for the shortest (or quickest) connections between a given start location and a given destination location. These two locations are both part of a network, but there is no direct connection available between them. Instead, it is necessary to determine a sequence of concatenated direct connections between intermediately passed connections/points that connects the start and terminus locations.

14.2.1 Outline of the Shortest Path Problem (SPP) in a Network

Christina first turns towards the LS business. In the first step, she asks the booking department for data about the usual and most frequently booked places for limousine transport. Her survey results in the observation that six locations contribute to the daily LS operations (see Table 14.1).

The fare for an LS booking is calculated based on the travel distance between the booked pickup location and the booked delivery location. From discussions with the RSBT customer care manager, Christina has learned that there are several complaints from customers who think that they have to pay too much, since drivers do not follow the shortest travel route.

Christina wants to find out if these complaints are justified and she decides to talk to some of the drivers. After her interviews, she was convinced that the complaints were not justified. All of the complaints were a result of a misunderstanding: the infrastructure in the area is quite different from the typical European street network. Since all hotels have been built around the populated area downtown, there are only

Table 14.1 Locations for LS

Notation	Locations
A	Reefside—The Fisherman's Village
B	The Palace Hotel
C	Downtown 5star
D	Mermaid's Inn
E	Red Sea Diver's Base
F	Hurghada International Airport

streets connecting the biggest hotels. This means going from a pickup point to a delivery point often requires a visit to several hotels. The street network is significantly less dense than in a typical European city and "felt detours" are needed when travelling in the area.

In order to provide more transparency for customers and instruct the limousine drivers, Christina decides to create a list for each pair of pickup and delivery point combinations. She wants to add the shortest route to each pair (to inform the drivers) and the corresponding travel distance (to inform the customers).

In the first attempt to compile the desired list, Christina collects data from the street map. She marks the six locations A to F (Fig. 14.1). Then, she looks for streets connecting these locations. Whenever she finds out that there is a street connecting a hotel with the airport or the airport with a hotel or a hotel with another hotel, she connects the origin location with the terminal location by an arc, that is, an arc represents an existing street connecting these two points where the arc direction indicates the direction of travel (Fig. 14.1). Finally, she appends the length (given in kilometers) of such a street (section) between two points. From the compiled simplified map, she learns, for example, that there is a street that goes from A (Reefside—The Fisherman's Village) to D (Mermaid's Inn), but this street cannot be used to go from D to A. The length of the street section from A to D is 3 km. But what is the distance of a trip from E to F? Christina is looking for a comfortable method to determine the least travel distance (i.e., to solve the shortest path problem [SPP]) between a pair of pickup and delivery locations not directly connected by an arc.

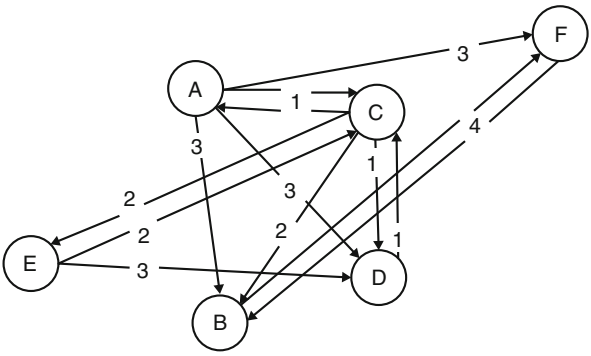

Fig. 14.1 Street network connecting the Airport (F) with the most important hotels (A–E) in the Hurghada area

14.2.2 Mathematical Graphs

A mathematical *graph* is a tool to describe network structures. A *network* consists of several points and connections between these points. In a mathematical graph these points are represented by *nodes* and all nodes are collected in a *node set* which is often labelled *N*. If there is a connection between two points, then there is also a connection (i.e., an *arc*) between the two corresponding nodes, say *i* and *j*, taken from the node set N (i, j $\in$ N). If it is only necessary to know that these two nodes are connected, then this connection is expressed by putting the two connected nodes into a set {i; j}. This set is called an *edge*. In most situations it is necessary to save information about the type of connection between the nodes i and j. For example, if i as well as j are cities served by an airline, it is necessary to know whether an offered flight goes from i to j and/or from j to i. It is therefore necessary to distinguish a connection from i to j and a connection from j to i. In such a situation the connection is established by the ordered pair (i; j) which indicates that there is a connection originating from i and going to j. If there is also a connection from j to i then a second ordered pair (j; i) is used to represent this additional connection. An ordered pair (i; j) connecting two nodes is called an *arc*. All required arcs and/or edges are saved in another set, called the arc (or edge) set typically labelled E. Information associated with a node or an arc (edge) can be stored by mappings f and g, so that f(i), g(i; j), or g({i; j}) respectively are kept as stored information. The quadruple (N; E; f; g) is called a *weighted mathematical graph.*

14.2.3 The SPP as Graph-Based Optimization Model

Christina wants to represent the LS problem setting in a mathematical graph. She starts by setting up a graph-based optimization model for the network structure shown in Fig. 14.1. Therefore, it is necessary to represent the given information as a mathematical graph G = (Ω; Θ; d) with evaluated arcs. The node set Ω consists of the six nodes Ω = {A; B; C; D; E; F}. The arc set Θ comprises exactly those arcs connecting the nodes as shown in Fig. 14.1, so that Θ:={(A; B), (A; C), (A; D), (A; F), (B; F), (C; A), (C; B), (C; D), (C; E), (D; C), (E; C), (E; D), (F; B)}. We incorporate the real-valued mapping d to assign the travel distance d(i; j) to each arc (i; j) $\in$ Θ.

In order to prepare the analysis of the derived graph, we introduce several definitions. These definitions help us to discuss the specific properties of a graph-based decision model.

Let (i; j) as well as (k; l) be two arcs in a given graph G. We call arc (k; l) a "successor of arc (i; j)" if and only if the two nodes j and k coincide, i.e. if and only if j = k. In such a situation, we also say that arc (k; l) follows arc (i; j) if and only if (k; l) is a successor of (i; j) in the given graph G. In our example arc (F; B) is a successor of arc (A; F), which means that arc (F; B) follows arc (A; F).

We are now prepared to introduce the term "*path*" into a network G. Let s and t be nodes from the node set Ω of G. In our example, s might be node A and t could be

node E. Let $(i_0; i_1), (i_1; i_2), \ldots, (i_{k-1}; i_k)$ be a finite sequence of following arcs with the properties $i_0 = s$ and $i_k = t$. We call the arc sequence $(i_0; i_1), (i_1; i_2), \ldots, (i_{k-1}; i_k)$ a path in G from s to t. For short, we call this arc sequence an s-t-path in G.

$$L((i_0; i_1), (i_1; i_2), \ldots, (i_{k-1}; i_k)) := d(s, i_1) + d(i_1; i_2) + \ldots + d(i_{k-1}; t) \quad (14.1)$$

The length of a given s-t-path $(i_0; i_1), (i_1; i_2), \ldots, (i_{k-1}; i_k)$ in G is defined by Eq. (14.1). In a case where all nodes contained in path $(i_0; i_1), (i_1; i_2), \ldots, (i_{k-1}; i_k)$ are pairwise different, we call the s-t-path a "simple path".

The property "pairwise different" refers to a situation in which each two nodes of a given subset of the node set Ω are not the same. A simple path in the graph shown in Fig. 14.1 is, for example, the path (A; D), (D; C), (C; E). In contrast, the path (A; D), (D; A), (A; F) is not a simple path since node A is visited more than once.

Using the aforementioned definitions, we are now prepared to describe Christina's task to determine the shortest path between a pair of nodes in the given network structure formally. Let s be the pickup point of an LS booking and let t be the planned drop-off point of this LS booking. Christina is looking then for a simple s-t-path $(s; i_1), (i_1; i_2), \ldots, (i_{k-1}; t)$ in G with minimal length $L((s; i_1), (i_1; i_2), \ldots, (i_{k-1}; t))$.

14.2.4 Dijkstra's Algorithm for the Identification of a Shortest S-T-Path

There is a very efficient and quick algorithm available to identify the shortest s-t-path in a given graph G. The only requirement for the applicability of this algorithm is that all arcs (i; j) of G have a non-negative length d(i; j). Applied to determine a shortest s-t-path in G, this algorithm does a little more than needed: it calculates the shortest paths from *s* to every other node in G. The basic idea of this algorithm was a contribution of by E.W. Dijkstra (1959). For this reason, the algorithm is often called the "Dijkstra-Algorithm".

Dijkstra's algorithm works as follows. Let $|\Omega|$ be the number of nodes in the given graph G, in which we are going to identify the shortest s-t-path.

Algorithm Start In the beginning, each node is assigned a so-called temporary label value l(v). This value equals the shortest path length known so far (not the path itself!) from start node *s* to node *v*, i.e. it carries the length of the shortest s-v-path found so far. Since, in the beginning, no s-v-path is known, we set l(v) equal to infinity ($l(v) := \infty$), for all $v \in \Omega$, $v \neq s$, but we set $l(s) = 0$.

Algorithm Iteration Dijkstra's algorithm iterates the following steps $|\Omega|$ times.

1. Each iteration starts with the selection of a node k with a temporary label value l(k). If there is such a node we select node k with a minimal temporary label value among all temporarily labelled nodes. If there is no temporarily labelled node left, then the algorithm terminates.

2. The label value l(k) of the selected node k is declared to be "permanent" and will not be changed any more in later iterations.
3. For all remaining nodes $w \in \Omega \backslash \{k\}$ with a label value l(w) which is not yet permanently labelled, we check the following: if the arc (k; w) is contained in G and if $l(k) + d(k;w) < l(w)$, then we update l(w) by l(w):=l(k) + d(k; w). We have found a shorter s-w-path in G. If there is no such arc (k; w) or if $l(k) + d(k; w) \geq l(w)$, then we do not change l(w) and check the next node $w \neq k$.

The following definitions help to ease the execution of the Dijkstra algorithm for even larger networks. Let v be a node from the node set Ω of graph G. In step (ii) of the iteration of Dijkstra's algorithm, we need to analyze all those nodes in Ω that are directly connected with v by an arc that originates from v. The termination node of such an arc is called a "successor of v". Let SUCC(v) be the set that contains all temporarily labelled successors of v, i.e. SUCC(v):={$k \in \Omega$l(v; k) $\in \Theta$ and k is temporarily labelled}.

Let (i; j) be an arc taken from the arc set Θ of graph G. We define i as a "predecessor" of j in G. For short, we write pred(j): = {i} if (i; j) $\in \Theta$. In the Dijkstra algorithm, we use the predecessor convention to recursively store the shortest s-t-path, e.g. the sequence of visited nodes in the shortest s-t-path. Starting from terminal node t, we store pred(t) and for the node pred(t) we store pred(pred(t)) and so on (right-to-left path determination).

Let Ω^T be the subset of the node set Ω that contains all nodes that are currently labelled temporarily and let Ω^P be the subset of Ω formed by the nodes that are already labelled permanently.

Dijkstra algorithm	
(a)	Set Ω^T:=V and Ω^P:= ∅
(b)	Set l(v):= ∞ and pred(v):= ∅ for all $v \in \Omega \backslash \{s\}$, l(s):=0 and pred(s):= ∅, l(s) = 0
(c)	Proceed with steps (d)–(g) if $\Omega^T \neq \emptyset$, otherwise go to (h)
(d)	Select a node $k \in \Omega^T$ so that l(k) is minimal over Ω^T.
(e)	Declare the label l(k) as permanent and update $\Omega^T := \Omega^T \backslash \{k\}$ and $\Omega^P := \Omega^P \cup \{k\}$
(f)	For all $w \in$ SUCC(k) that fulfil $l(k) + d(k; w) < l(w)$ update l(w):=l(k) + d(k; w) and pred (w):={k}
(g)	Goto (c)
(h)	Stop the algorithm

Such an algorithm description is quite close to a computer programming code. It is written in pseudocode. Pseudocode descriptions are often used in order to make the "real" implementation (coding) with a programing language like BASIC, Java, or C++ easier.

The Dijkstra algorithm starts with the initialization of the two node sets Ω^T, as well as Ω^P (the symbol ∅ represents an empty set) in the steps (a) and (b). The steps (c)–(g) are repeated as long as there is still a node with a temporary label available. In each repetition, exactly one node is moved from the set of temporarily labelled nodes

into the set of permanent nodes (d), (e) and the temporary labels are updated if a shorter s-w-path to a node w is found (f).

Christina has read about the Dijkstra algorithm and the shortest path problem. She applies it now to the problem situation shown in Fig. 14.1. She starts with the identification of the shortest path between point E and point F.

In order to ease the repetitive applications of steps in the Dijkstra algorithm it seems to be appropriate and helpful to represent the working data and intermediate results in a table as shown in Table 14.2. The recent label values are contained in the columns headed by the nodes. The recent pred(X)-value of node X is given in brackets to the right of the recent label value. Label values declared as permanent are marked by *. Each row represents one iteration of the Dijkstra algorithm.

Iteration 0 (Start): $\Omega^P = \varnothing$ and $\Omega^T = \{A; B; C; D; E; F\}$. The label value l(E) of the starting node E is set to 0. All remaining nodes are temporarily labelled by ∞.

Iteration 1: $\Omega^P = \varnothing$ and $\Omega^T = \{A; B; C; D; E; F\}$. Node E is labelled with the minimal temporary label value 0. For this reason, E is moved from Ω^T to Ω^P ("permanently labelled"). SUCC(E) = {C; D}. Since l(E) + d(E; C) = 0 + 2 < ∞, we update l(C):=2 and set pred(C):={E}. Since l(E) + d(E;D) = 0 + 3 < ∞, we update l(D):= 3 and set pred(D):={E}.

Iteration 2: $\Omega^P = \{E\}$ and $\Omega^T = \{A; B; C; D; F\}$. Node C is labelled with the smallest temporary label value. For this reason, C is moved from Ω^T to Ω^P ("permanently labelled"). SUCC(C) = {A; B; D}. Since l(C) + d(C; A) = 2 + 1 < ∞, we update l(A):=3 and update pred(A):={C}. Since l(C) + d(C; B) = 2 + 2 < ∞, we update l(B):=4 and update pred(B):={C}. Since l(C) + d(C; D) = 2 + 1 $\geq$ 3, we do not update l(D) und pred(D).

Iteration 3: $\Omega^P = \{C; E\}$ und $\Omega^T = \{A; B; D; F\}$. Both nodes A and D are labelled with the same minimal temporary label value 3. Node A is selected since it comes first in a lexicographic order. For this reason, we move A from Ω^T to Ω^P and update SUCC(A):={B; F}. Since l(A) + d(A; B) = 3 + 3 $\geq$ 4, we preserve l (B) and pred(B). Since l(A) + d(A; F) = 3 + 3 < ∞, we update l(F):=6 as well as pred(F):={A}.

Table 14.2 Structured table-based presentation of the iterations of the Dijkstra algorithm

Iteration	Node recently selected and labelled as permanent	Recent label values					
		A	B	C	D	E	F
0 (start)	–	∞	∞	∞	∞	0(−)	∞
1	E	∞	∞	2(E)	3(E)	0*(−)	∞
2	C	3(C)	4(C)	2*(E)	3(E)	0*(−)	∞
3	A	3*(C)	4(C)	2*(E)	3(E)	0*(−)	6(A)
4	D	3*(C)	4(C)	2*(E)	3*(E)	0*(−)	6(A)
5	B	3*(C)	4*(C)	2*(E)	3*(E)	0*(−)	6(A)
6	F	3*(C)	4*(C)	2*(E)	3*(E)	0*(−)	6*(A)

Iteration 4: $\Omega^P = \{A; C; E\}$ and $\Omega^T = \{B; D; F\}$. Node D has the minimal temporary label value 3. Therefore, we move D from Ω^T to Ω^P and update SUCC(D):= ∅. No temporary label value must be checked to be updated.

Iteration 5: $\Omega^P = \{A; C; D; E\}$ and $\Omega^T = \{B; F\}$. Node B is labelled with the smallest temporary label value, which is 4. Thus, we move B from Ω^T to Ω^P and we set SUCC(B) = {F}. Since l(B) + d(B; F) = 4 + 4 ≥ 4, we preserve l(F) and we do not update prec{B}.

Iteration 6: $\Omega^P = \{A; B; C; D; E\}$ and $\Omega^T = \{F\}$. Node F is labelled with the minimal temporary label value 6. We therefore move F from Ω^T to Ω^P and set SUCC(B):= ∅. No further checks are necessary.

Iteration 7: $\Omega^P = \{A; B; C; D; E; F\}$ and $\Omega^T = ∅$. There is no further node available that has been labelled temporarily. The Dijkstra algorithm terminates here.

The permanent label value l(F) of the designated goal node gives the length of the shortest E-F-path in graph G. The path is determined by a recursive backtrack-analysis of the final pred-values. We have found that pred(F):=A, pred(A):=C, pred (C):=E. The shortest E-F-path in G is ((E; C), (C; A), (A; F)).

There is a chance of the existence of more than one E-F-path with the same minimal distance. The Dijkstra algorithm can find only one shortest E-F-path. Christina can use the Dijkstra algorithm to determine the shortest distances between each pair of nodes in the Hurghada network.

14.3 Round Trip Planning/Travelling Salesman Problem

Now that Christina has successfully solved the planning challenge for the LS operations, she turns towards the next open challenge. In the SST department of RSBT, it is necessary to determine round trips of buses that contain exactly one visit for each location contained in a given list of sightseeing objects.

Christina examines the SST situation and she finds out that there are some similarities with the shortest path identification problem which she solved as explained in Sect. 14.2. First, the SST challenge also exists in a network structure. Second, for a given list of points of interests, a connecting path in the given network structure is required. Third, also in the SST situation, a path with minimal length is searched. However, Christina discovers an important difference between the LS and SST situations. In the LS challenge, only the starting as well as the terminating node of the requested s-t-path is given. There are no requests to visit other nodes. In contrast, the necessity to visit all locations contained in the aforementioned list of points of interests is postulated. For this reason, Christina concludes that the Dijkstra algorithm is not a sufficient tool to solve the SST challenge, since this algorithm does not provide the opportunity to manage the constraint that each node contained in the network is visited exactly once.

In this section we accompany Christina while she learns how optimization problems on network structures can be solved if restrictions on the solution must be considered. In Sect. 14.3.1, the underlying basic decision problem of the SST

situation is introduced and explained in detail. In Sect. 14.3.2, a mathematical optimization model for a restricted network optimization problem is introduced. Techniques to determine feasible solutions for these models are presented in Sect. 14.3.3.

14.3.1 Travelling Salesman Problem

Christina wants to know if her SST challenge has already been solved by anybody else. She remembers a lecture she visited during her studies. It was called "Operations Research Methods for Management Decision Problems". She sends an email to one of her friends from the course and asks him to send her a copy of the presentations delivered in this lecture. Several days later, she receives these printouts and starts scanning the documents.

After a few minutes, she discovers the picture of a network (Fig. 14.2) that attracts her attention. The slide that contains this graphic is headed by the words "Travelling Salesman Problem".

Christina starts reading the remarks given by the instructor. She reads that the *Traveling Salesman Problem (TSP)* represents the basic decision task of many vehicle deployment problems. It is quite similar to the shortest path problem since in the TSP it is also necessary to fulfil a transportation task in a network structure with minimal resource utilization.

However, the TSP is distinct from the shortest path problem since all nodes in the network have to be visited, but no node is allowed to be visited more than once (which means that each available node must be visited *exactly once*). There is also a distinct node, called the starting node or the depot, from which the travelling salesman starts his journey and to which he returns after all other nodes have been visited exactly once. The salesman wants to keep his total sum of travel distances between the visited nodes as small as possible.

Christina summarizes these findings and transfers them to the SST challenges. She is convinced that the TSP represents exactly the SST problem: there is a set of nodes given and these nodes (in the SST case: the given points of interests) must all be visited by a vehicle, but the length of the travel path must be minimal.

Christina continues to read the instructor's notes. Without going into detail, she understands that the TSP can be modelled as a *linear program* (cf. Chap. 12), but that it is impossible to apply the standard approach for linear programs, which is the Simplex algorithm, to solve this model. For this reason, her instructor proposes applying other decision supporting methods called heuristics. Now, Christina's attention is awakened and she decides to study the information given in the course material about the TSP, its modeling, and the heuristic model solving techniques in detail.

First, Christina wants to understand the case that motivates Fig. 14.2. The outline of the case study is as follow. A sales representative lives in Hamburg (city numbered 1 according to Table 14.3). Each week, she has to visit the remaining 13 cities in Northern Germany, which are given in Table 14.3 (cities 2–14). She wants to visit all these cities in one trip (if necessary, she will have an overnight stay

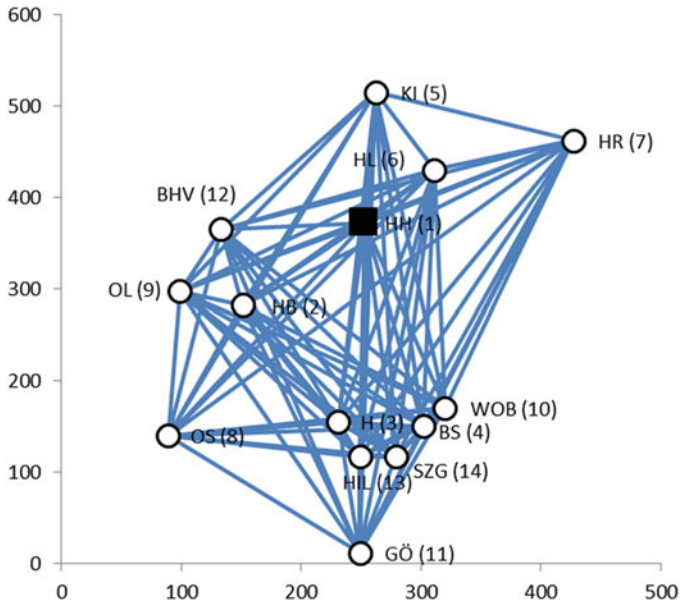

Fig. 14.2 Problem setting of the travelling salesman problem (TSP)

in one of the cities) in order to visit all the retail outlet stores of her company. According to her contract with this company, she has to pay for the fuel costs on her own. In order to keep the fuel expenses as low as possible, the sales representative searches for a least-distance round trip that starts in Hamburg, terminates in Hamburg, and visits all remaining 13 cities exactly once.

Christina continues to learn more about the case. The sales representative (who is in the role of the travelling saleswoman) can travel directly from each given city and each given other city as indicated by the graph in Fig. 14.2. Such a graph in which each node has a connection to each other node is called a complete graph. At first look, such a network structure seems to make the determination of a round trip easier than the determination of a round trip in a general graph like the one given in Fig. 14.1: Detours on the way from one node to another node are not necessary.

► **Practical Insights** A high number of nodes and arcs in the network let the determination of the shortest path become a management problem of high *complexity*. The following calculation emphasizes this statement. How many round trips are available that visit each node exactly once? For the first customer node, the sales representative can select from 13 so far unvisited nodes. For the second customer node to be visited, she can still select from 12 remaining customer locations and so on. In the given example, this leads to $13 \cdot 12 \cdot \ldots \cdot 2 \cdot 1 = 13! = 6{,}227{,}020{,}800$ possible trips that visits each node exactly once. Such a path that contains a node exactly once and that is closed (start and end node coincide) is called a Hamiltonian path. Now assume that the identification and the calculation of the length of each individual Hamiltonian

Table 14.3 Coordinates of major cities in Northern Germany

Name of location (abbreviation)	i	Coordinates	
		x_i	y_i
Hamburg (HH)	1	253	372
Bremen (HB)	2	151	282
Hannover (H)	3	231	155
Braunschweig (BS)	4	302	150
Kiel (KI)	5	262	515
Lübeck (HL)	6	311	425
Rostock (HR)	7	427	462
Osnabrück (OS)	8	89	140
Oldenburg (OL)	9	99	297
Wolfsburg (WOB)	10	320	170
Göttingen (GÖ)	11	249	12
Bremhaven (BHV)	12	133	365
Hildesheim (HIL)	13	249	117
Salzgitter (SZG)	14	279	117

path lasts 1 s. The compilation of a list of all evaluated Hamiltonian paths would last around 1200 days. For this reason, this so-called complete enumeration is not an appropriate approach with which to identify a shortest Hamiltonian path due to practicality.

Christina is happy. She knows now that the SST challenges match the TSP challenges. Therefore, she is sure that she can solve her SST task after she has learned how the TSP is solved. She continues to study the lectures notes about the TSP and finds out that, again, a model-based approach is proposed to identify a solution for the given decision task. First, a model for the TSP is needed and, second, an algorithm for identifying a solution of the model is required.

14.3.2 A Mixed-Integer Linear Program for TSP-Modelling

All $N = 14$ locations to be visited are numbered by 1, 2, …, N in order to ease further description of the problem (cf. Table 14.3). These locations form the node set. An arc (i; j) connects two nodes and indicates that it is possible to travel from i to j. The node set in the TSP is equal to $\{1, \ldots, N\} \times \{1, \ldots, N\} \backslash \{(1; 1), \ldots, (N; N)\}$. Therefore, the graph in the TSP is complete.

The basic idea for modeling the TSP is that each arc (i; j) contained in the aforementioned node set is either contained in the Hamiltonian path (the round trip through all N nodes) or not. In the first case mentioned, node j is visited immediately after node I, but in the latter case node j is not visited immediately after i has been left. The TSP decision problem can be reduced to the question of which arcs form the requested Hamiltonian path and which arcs are ignored. In order to represent these binary decisions, the family of binary decision variables x_{ij} ($i \in \{1, \ldots, N\}$, $i \in \{1,$

. . ., N}) is introduced. Each decision variable x_{ij} is either 0 or 1, and x_{ij} represents the decision whether arc (i; j) is contained in the Hamiltonian path or not. We declare x_{ij} to be 1 if and only if arc (i; j) is included in the Hamiltonian path. It equals 0 if this is not true.

Let d(i; j) be the distance between node i and node j if the arc (i; j) is used.

$$Z(x_{11}, x_{12}, \ldots, x_{NN}) = \sum_{i=1}^{N} \sum_{j=1}^{N} d(i;j) \cdot x_{ij} \rightarrow min \tag{14.2}$$

Equation (14.2) determines the sum of travel distances of all arcs that are selected to be included in the Hamiltonian path. This value $Z(x_{11}, \ldots, x_{NN})$ must be minimized.

$$\sum_{j=1}^{N} x_{ij} = 1 \quad \forall i \in \{1, \ldots, N\} \tag{14.3}$$

$$\sum_{j=1}^{N} x_{ji} = 1 \quad \forall i \in \{1, \ldots, N\} \tag{14.4}$$

In order to ensure that the selected arcs form a Hamiltonian path through the node set {1, . . ., N} it is necessary to restrict the selection of arcs. First, each node must be left once (Eq. 14.3) and the salesman has to go to each node exactly once (Eq. 14.4).

It is not sufficient (but it is necessary) that the two restrictions (14.3) as well as (14.4) are fulfilled in order to ensure that a Hamiltonian path through $N = \{1, \ldots, N\}$ is determined. Figure 14.3 shows a selection of arcs that fulfills Eqs. (14.3) and (14.4), but a Hamiltonian Path is not achieved. Instead, two so-called short-cycles are generated. One of these two short cycles does not contain the start node 1. Therefore, it is necessary to add other constraints in order to ensure that a Hamiltonian path is generated.

$$u_i - u_j + N x_{ij} \leq N - 1 \quad \forall i, j \in \{2, \ldots, N\} \tag{14.5}$$

The article of Desrochers and Laporte (1991) proposed the constraint family (Eq. 14.5) as short-cycle elimination constraints. They demonstrate that by means of these constraints and with the incorporation of the real-valued u_i-decision variables, short cycles that do not contain the start node 1 can be prevented.

$$x_{ij} \in \{0; 1\} \ \forall i, j \in \{1, \ldots, N\} \text{ and } u_i \text{ real-valued for all } i \in \{1, \ldots, N\} \tag{14.6}$$

Jointly, Eqs. (14.5) and (14.6) ensure that only round trips that contain the designated starting point 1 are allowed.

The TSP is now represented by the linear program (Eqs. 14.2–14.6). Since some of the decision variables included in this model are limited to integer (binary) values while other decision variables are of real value, this model falls into the class of a

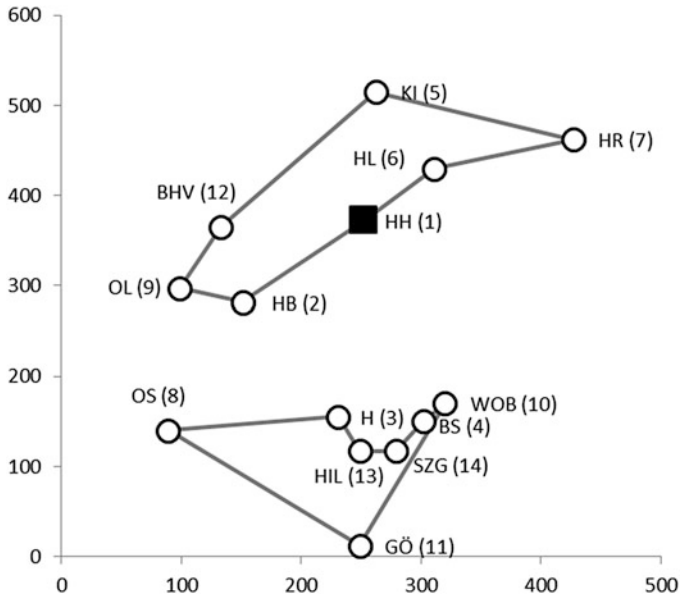

Fig. 14.3 Infeasible TSP solution with too short cycles

mixed-integer linear program (MILP). A linear program containing only integer-valued decision values falls into the class of integer linear programs (ILP) according to Grünert and Irnich (2005).

Having studied the derivation of the mathematical model and learned that the model consists of a linear objective function as well as a collection of linear constraints, Christina re-implements the reported problem instance in an Excel spreadsheet model and tries to solve this model with the Excel Solver add-in. Quite rapidly, the solver returns a feasible solution. Christina studies the returned decision variable values. She is worried by the proposed values: she finds the proposal to set an x-decision variable to the value 0.25. She asks herself how she can interpret this value. After a while, she finds out the reason for this unexpected proposal: she forgot to declare the x-decision variables to be binary. She corrects this shortcoming and restarts the solver. Unexpectedly, the solver is unable to return a feasible solution. The calculation lasts more than 20 min so she decides to interrupt it, being disappointed that there is no straightforward way to solve the TSP.

From her previous experience, Christina has learned that the Simplex algorithm fails to return a feasible solution since this algorithm is unable to ensure that integer values are returned. An explicit limitation of the decision variable domains to binary or integer values significantly prolongs the model solving times. Christina supposes that in the latter situation another algorithm is used. Her interest is aroused. She wants to know if there are other algorithmic approaches to solve mathematical optimization models. She returns to the textbook and continues to read the instructor's ideas about solving a TSP-like problem.

14.3.3 Heuristic Search for High Quality Round Trips

An algorithmic approach for solving a decision model by exploiting a rational strategy of planning without mathematical proof that the proposed (return) feasible model solution cannot be dominated by another feasible solution is called a *heuristic* or a non-optimizing algorithm. In contrast, an algorithm for which the returned solution proposal is proven to be minimal (maximal) is called an *exact* or optimization algorithm or approach. The Simplex algorithm is an example of an exact algorithm. Heuristics are incorporated into model solvers in case no exact algorithm is available or if the processing times provided by an exact algorithm are expected to be prohibitively long. Since the *processing time* for solving relatively small instances of a TSP is in general quite long it seems reasonable to think about a heuristic approach for the TSP.

This paragraph introduces heuristics for the management of TSP-like planning tasks. Two types of algorithms are distinguished. First, we discuss the construction of a first feasible solution for the TSP, i.e. we explain the construction of a first Hamiltonian path in the given network. Second, we discuss approaches for the improvement of the Hamiltonian path, i.e. we want to reduce the travel length associated with the Hamiltonian path. Therefore, we need to analyze the existing Hamiltonian path in order to find out which modifications are promising for a reduction in the length of the Hamiltonian path. The most promising modifications are tentatively implemented and, if they lead to a reduction in travel distance, finally implemented.

Algorithms for the Construction of an Initial Feasible Model Solution
Joereßen and Sebastian (1998) describe an intuitive approach for the construction of a Hamiltonian path. The central idea is to consecutively insert the nodes after the last visited node until all nodes are contained in the path. A Hamiltonian path is created. This approach is also known as a "*nearest neighborhood heuristic*".

It starts with an empty path. In the first step, the starting node (in our example node 1) is inserted. Next, the node with the shortest distance from the starting node is inserted immediately after the starting node. This path extension is iterated until no further node remains. This path construction procedure follows the rational strategy of keeping the additional travel distance associated with serving the next customer after the previously inserted customer as low as possible. Therefore, the applied rational strategy can be characterized as "*greedy*". Consequently, this construction heuristic is called the "greedy construction procedure". A major shortcoming of the greedy construction procedure is its myopic insertion strategy. After all nodes have been inserted into the node sequence, it is necessary to return to the start node. The travel distance for the return to the depot is not considered in the calculation of the additional travel distance to serve the next customer. Consequently, the greedy strategy cannot be proven to return the shortest possible Hamiltonian path. However, it returns a first Hamiltonian path.

Task 14.1 TSP Solution with Greedy Heuristic
Consider a transportation company that has to deliver products from Hamburg to 13 cities in Germany according to the TSP. The data refer to Table 14.3 and the distances are reflected in Fig. 14.4.

Of course, the lengths d(i, j) of the arcs are very important for the decision about the inclusion of the next node. Typically, these values are composed in the distance matrix as shown for the example from Table 14.3 in Fig. 14.4. The entry in the i-th row in column j corresponds to the length of the arc (i; j). Headers are not considered in this numbering.

Greedy construction procedure	
(a)	Initialize: L = 0, start = 1, k = 1;
(b)	Repeat steps (c)–(i) until all columns are blocked
(c)	DEST:=min{d(START, q)| column q is not blocked}
(d)	L:=L + d(START, DEST);
(e)	Block row of START as well as column associated with START;
(f)	Stop[k]:=START;
(g)	START:=DEST;
(h)	k = k + 1;
(i)	Goto (b);
(j)	Stop[k]: = 1;
(k)	L:=L + d(START, 1);
(l)	End;

This pseudocode represents the greedy construction procedure that consists of an initialization phase (a), iteration loops (b)–(i), and a termination phase (j)–(l). During the initialization phase, the travel distance is set to 0 and the first node in the path is set to the starting node start: =1 (home of the salesman). The iteration counter k is initialized.

Exactly one node is appended to the already existing partial path in each iteration. Each iteration starts with checking whether there are still unblocked columns. Such an unblocked column represents a node that has not yet been appended. Among all unblocked columns (available nodes), a node DEST with the least distance d (START, DEST) from the last appended node START is chosen (c). Afterwards, the so far travelled path length L is increased by L(START; DEST) (d). Next, the row as well as the column associated with DEST is blocked (e). The recently selected node DEST is appended to the already existing partial path (f) and DEST will be set to the last inserted node (g). The iteration counter is increased by 1 (h) and the procedure jumps back to the beginning of the loop (i). If no further unblocked column is available, the path is closed, i.e. the arc from the last appended node to the starting node is selected (j) and the travel distance is updated (k). Further details can be found in the examples in the E-Supplement.

In the example, we have framed the cells representing the arcs (1; 6), (2; 3), (3; 13), (4; 10), (5; 7), (6; 5), (7; 12), (8; 1), (9; 2), (10; 11), (11; 8), (12; 9), (13; 14) as

i \ j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1		136	218	227	143	79	196	284	171	213	360	120	255	256
2	136		150	201	258	215	330	155	54	203	287	85	192	209
3	218	150		71	361	282	364	143	194	90	144	232	42	61
4	227	201	71		367	275	336	213	251	27	148	273	62	40
5	143	258	361	367		102	173	413	272	350	503	198	398	398
6	79	215	282	275	102		122	361	248	255	418	188	314	310
7	196	330	364	336	173	122		467	367	311	484	310	388	375
8	284	155	143	213	413	361	467		157	233	205	229	162	191
9	171	54	194	251	272	248	367	157		255	322	76	234	255
10	213	203	90	27	350	255	311	233	255		173	270	89	67
11	360	287	144	148	503	418	484	205	322	173		372	105	109
12	120	85	232	273	198	188	310	229	76	270	372		274	288
13	255	192	42	62	398	314	388	162	234	89	105	274		30
14	256	209	61	40	398	310	375	191	255	67	109	288	30	

Fig. 14.4 Distance matrix for the TSP instance defined by the nodes in Table 14.3 (values for d(i; j) in kilometers)

well as (14; 4). These arcs lead to the Hamiltonian path 1, 6, 5, 7, 12, 9, 2, 3, 13, 14, 4, 10, 11, 8, 1. The length is 1746 km. This path can be found in Fig. 14.5.

Figure 14.5 visualizes the proposal generated by the greedy construction procedure. The generated Hamiltonian path is not optimal, i.e. it is possible to reduce the travel length. Within the route segment $1 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 12$, as well as in the route segment $9 \rightarrow 2 \rightarrow 3 \rightarrow 13 \rightarrow 14 \rightarrow 4 \rightarrow 10 \rightarrow 11 \rightarrow 8 \rightarrow 1$, a modification of the visiting order leads to a shorter Hamiltonian path. This is not a surprising observation since the primary goal of the greedy construction procedure is the generation of a feasible (but not necessarily a least-distance) Hamiltonian path. The idea of inserting one customer after another, even with least marginal costs, does not result in a least-distance Hamiltonian path.

The phenomenon observed here refers to a general shortcoming of the heuristic search for solutions if several constraints must be considered while trying to optimize the objective function. It either tries to fulfil the optimization property or it tries to achieve feasibility of the generated solution proposals. At first glance, it seems that the greedy construction procedure fails for the TSP, but it is worth getting a first feasible Hamiltonian path, since it is possible to improve the objective function using so-called improvement heuristics that modify the initial feasible solution. These improvement heuristics focus on the minimization of the objective function value, but retain the feasibility of the modified solution.

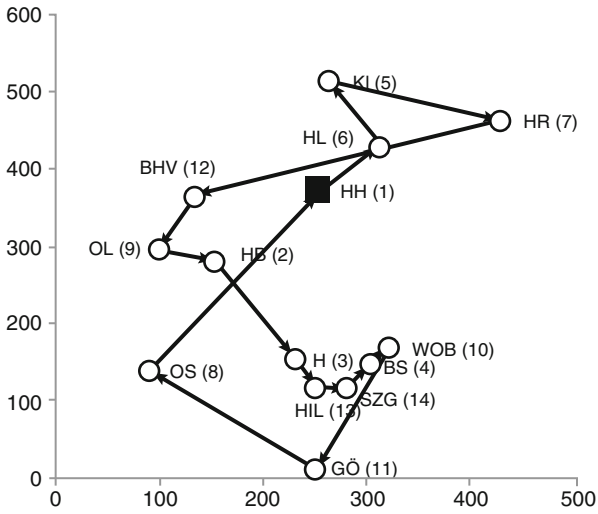

Fig. 14.5 Visualization of the greedy construction procedure proposal

Improvement Heuristics for Travel Distance Reduction

We design and apply improvement heuristics in order to update an existing, feasible decision problem solution. During the update, it is necessary to preserve feasibility, but the objective function value should be brought closer to the (unknown) optimization goal. For a given Hamiltonian path in the TSP, an improvement heuristic tries to find another Hamiltonian path with reduced travel distance by modifying the visiting sequence as proposed by the construction heuristic. Detours should be eliminated. The 2-opt-improvement heuristic presented here is based on ideas discussed by Croes (1958). In its most simple realization, the 2-opt heuristic tries to reduce the length of a given Hamiltonian path by swapping two adjacent nodes in the path. Therefore, the swap is applied tentatively. Then, it is checked to see whether length reduction is achieved. In this case, the tentative swap is confirmed. Otherwise, the swap is cancelled (Fig. 14.6).

We now apply the 2-opt-improvement logic to reduce the travel distance by swapping adjacent nodes in the available Hamiltonian path proposed by the greedy construction heuristic. We start with the partial path 1; 6; 5; 7 and try to swap nodes 6; 5, i.e. we tentatively replace 1; 6; 5; 7 with 1; 5; 6; 7 (cf. Fig. 14.6).

Iteration 1: The sub-path 1; 6; 5; 7 has a length of 354 km. Swapping 6 and 5 will lead to the sub-path 1; 5; 6; 7 with a length of 367 km. This swap is therefore not beneficial and it is not confirmed.

Iteration 2: The sub-path 6; 5; 7; 12 has a length of 605 km. Swapping 5 and 7 will lead to the sub-path 6; 7; 5; 12 of length 493 km. Since the executed swap leads to a length reduction it is confirmed and the sub-path 6; 5; 7; 12 is replaced by the sub-path 6; 7; 5; 12.

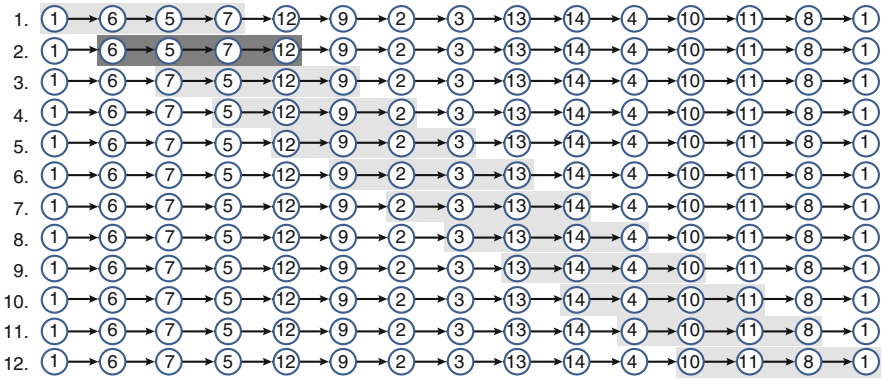

Fig. 14.6 Processing of 2-opt-improvement-algorithm

Iteration 3: The sub-path 7; 5; 12; 9 has a length of 447 km. Swapping nodes 12 and 5 leads to the sub-path 7; 12; 5; 9 of length 780 km. This swap is therefore not beneficial and it is not confirmed.

Iteration 4: The sub-path 5; 12; 9; 2 has a length of 328 km. Swapping nodes 12 and 9 leads to sub-path 5; 9; 12; 2 with a length of 433. This swap is not confirmed.

Iteration 5: The sub-path 12; 9; 2; 3 has a length of 280 km. Swapping nodes 2 and 9 leads to sub-path 12; 2; 9; 3 with a length of 333 km. This swap is not confirmed.

Iteration 6: The sub-path 9; 2; 3; 13 has a length of 264 km. Swapping nodes 2 and 3 leads to sub-path 9; 3; 2; 13 with a length of 536 km. This swap is not confirmed.

Iteration 7: The sub-path 2; 3; 13; 14 has a length of 222 km. Swapping nodes 3 and 13 leads to sub-path 2; 13; 3; 14 with a length of 295 km. This swap is not confirmed.

Iteration 8: The sub-path 3; 13; 14; 4 has a length of 112 km. Swapping nodes 13 and 14 leads to sub-path 3; 14; 13; 4 with a length of 153 km. This swap is not confirmed.

Iteration 9: The sub-path 13; 14; 4; 10 has a length of 97 km. Swapping nodes 14 and 4 leads to sub-path 13; 4; 14; 10 with a length of 169 km. This swap is not confirmed.

Iteration 10: The sub-path 14; 4; 10; 11 has a length of 240 km. Swapping nodes 4 and 10 leads to sub-path 14; 10; 4; 11 with a length of 242 km. This swap is not confirmed.

Iteration 11: The sub-path 4; 10; 11; 8 has a length of 405 km. Swapping nodes 10 and 11 leads to sub-path 4; 11; 10; 8 with a length of 554 km. This swap is not confirmed.

Iteration 12: The sub-path 10; 11; 8; 1 has a length of 662 km. Swapping nodes 11 and 8 leads to sub-path 10; 8; 11; 1 with a length of 798 km. This swap is not confirmed (Fig. 14.7).

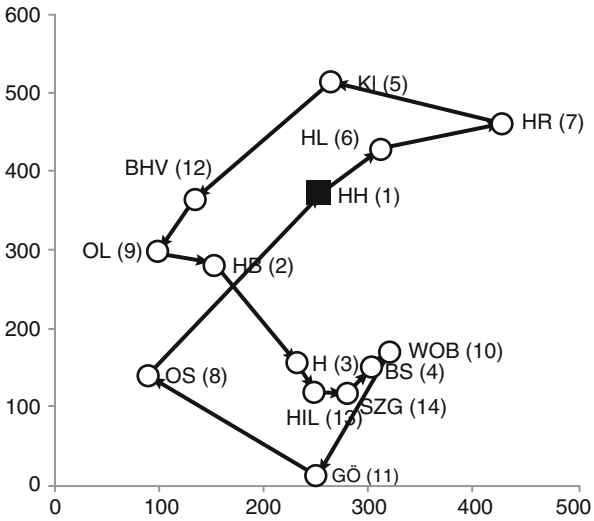

Fig. 14.7 Hamiltonian path after first application of the 2-opt-improvement heuristic

The existing Hamiltonian path is now completely processed. A length reduction of 112 km has been achieved. The improved Hamiltonian path is shown in Fig. 14.7. Despite significant improvement, we see immediately that further improvements are needed. However, the required improvement cannot be achieved by interchanging adjacent nodes, so a 2-opt improvement procedure call is useless. In the situation reported here, it is beneficial to invert the sub-path 3; 13; 14; 4; 10; 11; 8, i.e. to replace this sub-path by 8; 11; 10; 4; 14; 13; 3 (see Fig. 14.8). However, the identification of a sub-path whose inversion is beneficial is a challenging task in itself.

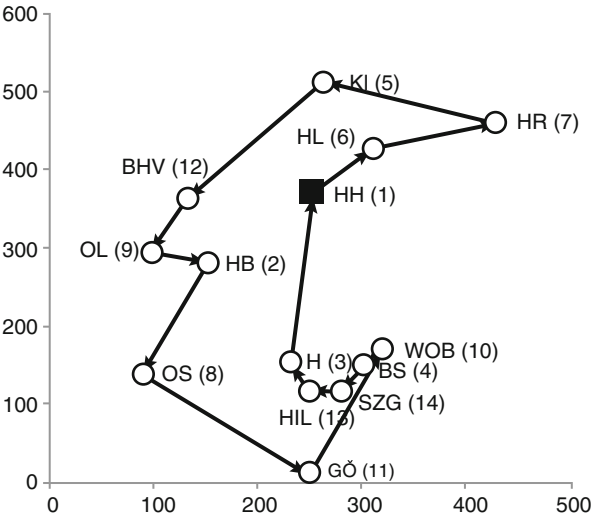

Fig. 14.8 Solution of inversion of the sub-path from 3 to 8

In general, a reapplication of the 2-opt procedure to the already improved solution can be useful for identifying further length-saving swapping. In our example, swapping of nodes 6 and 7 is now possible (but was not possible in the first application of the 2-opt procedure). Sometimes nodes are moved from a position at the beginning of a Hamiltonian path to a later position or vice versa.

14.4 Vehicle Routing

Having solved the first two operative vehicle deployment tasks (LS and SST) successfully, Christina turns to the next large business area, which is the local transfer of inbound tourists. RSBT provides this service to international travel agencies. These agencies sell all-inclusive travel packages to customers. Such a package includes

- a return flight ticket from the selected origin airport to Hurghada international airport (HRG)
- accommodation in the selected hotel in the Hurghada region including breakfast or half board or full board or all-inclusive food & beverage
- transfer on day of arrival with a pickup at the airport and a drop-off at the selected hotel
- transfer on the day of departure from the hotel to the airport.

Christina finds out that RSBT generates a significant amount of income from the transfer services, but the profits are quite small. She discusses this observation with the responsible division manager. The manager mentions that the transfer of incoming passengers (AAT) is easier to plan than the transfer of outbound passengers (ADT) since in the ADT setting up pickup times at the hotels has to be considered. In the AAT setup, such time windows do not play any role.

14.4.1 Case Study ORION: Vehicle Routing at UPS

ORION (**O**n-**R**oad **I**ntegrated **O**ptimization and **N**avigation) is a routing optimization system that suggests an optimized route to get from point A to point Z and all potential stops in between while also considering specified delivery windows. It aims to determine routes by a heuristic that uses predominantly right-hand-turns in order to increase efficiency by maximizing both time savings and safety (therefore saving on repairs, etc.). ORION uses a database of 250 million addresses, customized map data, and takes into account different variables such as distances, delivery times, different types of customers, and package types. ORION constantly re-evaluates alternative routing options, up to the last moment before a driver leaves the depot and thereby provides optimal routing instructions to UPS drivers—in the second phase planned for the ORION project, UPS plans to enable live recalculations even during delivery.

What Has Been the Business Value of ORION?

Through the deployment of ORION, UPS had optimized 10,000 routes by the end of 2013 and aims to optimize all 55,000 routes being served in the US by 2017. The optimization of routes has had effects on three dimensions: efficiency, sustainability, and service level. The results stated refer to the pilot year 2013.

By optimizing routes, UPS managed to reduce the total distance travelled by trucks by 20 million miles and to deliver 350,000 additional packages. Estimations indicate that a reduction of 1 mile per day per driver for 1 year results in savings amounting to $50 M.

These efficiency gains in turn positively affect the use of resources, as a reduction in miles travelled causes a diminution of CO_2 emissions (14,000 metric tons) and leads to gasoline savings (5.7 million liters of fuel). At the same time, UPS also increased its service called "My Choice" that allows consumers to actively choose delivery preferences, reroute shipments, and adjust delivery locations and dates as needed.

What Are the Disadvantages of ORION?

The costs connected with the development of soft- and hardware as well as the setup of vehicle routing solutions such as ORION can be extremely high. UPS dedicates 700 employees to the operation and maintenance of ORION, even though there are no official comments regarding the costs directly attributable to ORION.

UPS' overall annual budget of $1.0B on technology suggests that the investment in innovation and improvement of technology is material. Also, the lead time needed for the setup can often be considerable. In UPS' case, the ORION algorithm was initially developed in a laboratory and tested at various UPS sites from 2003 to 2009. In order to gather the huge amount of data necessary to develop the ORION system, UPS installed GPS tracking equipment and vehicle sensors on UPS delivery trucks in 2008. The company prototyped ORION at eight sites between 2010 and 2011, before deploying the system to six beta sites in 2012.

Furthermore, the *complexity* of vehicle routing problems in the real world is often very high, requiring special experts for the implementation of VRP solutions. The complexity can easily be understood when thinking of the number of possibilities for determining only one route: an average UPS driver has about 125 customers to reach a day. The number of options for scheduling a route for 125 different locations is 125! which is a number with 210 digits, which by far exceeds the number of nanoseconds the earth has existed. Another hint for the complexity of the problem is the fact that after 6 years of research, UPS was able to determine an algorithm that takes 90 pages to describe and 1000 pages of code to implement.

Finally, at this stage, ORION is not able to consider *dynamic data* such as weather, traffic jams, or accidents which might slow down a driver's route, and is therefore only providing an optimal solution at the moment a vehicle leaves the depot and not later when changes might occur.

14.4.2 Decision Situation Outline

Christina and the manager decide together that because of time, they have to separate the AAT from the ADT operations planning. Furthermore, they decide to analyze the AAT setup first. Since Christina is not familiar with the AAT business she asks for an example in order to get an insight into the preparation and execution of some AAT services. The manager selects a certain day from the last month. On that day, there was one evening flight that carried overall 29 passengers to the HRG airport. Different travel agencies have booked the corresponding AAT service at RSBT.

Overall, 14 hotels have to be served. The number of passengers q_i to be brought to a hotel H_i varies between 1 and 4 as shown in Table 14.4. The hotels are distributed around the airport (Fig. 14.9).

Next, Christina wants to know how the passengers are transported to the hotels. She learns that four minibuses are available to fulfil the AAT services. Each minibus offers ten passenger seats. All minibuses start their operations from a large parking lot at the airport. From there, a minibus moves to the exit of HRG's arrival hall where the passengers to be served enter the minibus. Next, a minibus goes to the requested hotel(s) where passengers are dropped off. Within one service trip, a minibus regularly visits one or more hotels before it returns empty to the parking lot at HRG.

The AAT manager tells Christina that he has the feeling that the performance of the AAT operations is quite low. In particular, he thinks that the distances travelled (and therefore trip durations) are quite high. As a consequence, the number of deployed vehicles seems to be quite high. Overall, AAT operations seem to produce unnecessary travel costs (fuel costs resulting from detours) as well as personal costs (for the drivers). In order to demonstrate this, the AAT manager produces Table 14.5 showing

Table 14.4 Hotels to be served in the AAT example

Hotel (code)	Hotel number i	Coordinates		Number of passengers
		x	y	q_i
Marina[a] (H1)	1	103	30	1
Blue Lagoon (H2)	2	143	75	2
Three Palms (H3)	3	123	113	2
Golf Resort (H4)	4	126	88	4
Delphin (H5)	5	117	81	2
Pyramid (H6)	6	38	102	2
Inn at the Beach (H7)	7	81	53	2
Sharky Corner (H8)	8	120	41	1
Diver's Paradise (H9)	9	97	101	1
Red Sea Exclusive (H10)	10	60.75	60.75	3
Water Palace (H11)	11	8.75	78.75	2
Grand Hotel Landside (H12)	12	103	58.5	4
Desert Castle (H13)	13	101.25	87.75	2
Wonder World (H14)	14	87.5	22	1

[a]It is a three star rated hotel

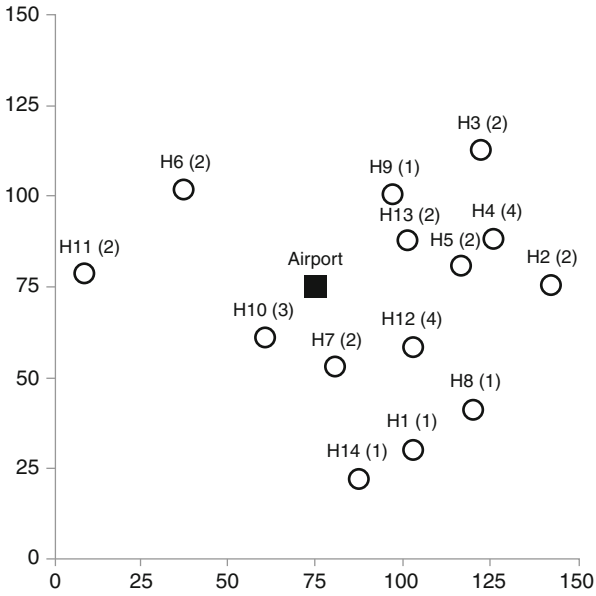

Fig. 14.9 Map of hotels in the example

the trips executed by the four vehicles. Four vehicles are deployed and the total distance travelled is 232.42 km + 285.77 km + 204.08 km + 150.81 km = 873.07 km.

14.4.3 Current Approach for the Route Compilation

Having seen only the vehicle trips already executed, Christina feels uncomfortable in giving a statement about the suitability of the proposed trips. She looks for hints that support the evaluation of the proposed service processes (the vehicle trips). In a first attempt to estimate the quality of the proposed trips she visualizes the proposed solution (the trips) in the map provided in Fig. 14.9. Therefore, she connects the hotels by lines in a sequence proposed by the visiting sequences shown in Table 14.5. As the result, she gets the route map shown in Fig. 14.10.

Vehicle 1 follows a continuous line, but vehicle 2 travels along the dotted line. In addition, vehicle 3 operates along the dashed line, while the fourth vehicle follows the grey line. Christina thinks about generating trips in the AAT business and she tries to uncover similarities and differences compared to the two previous investigations. Here is what she finds out:

1. Since several vehicles are available it is necessary to distribute all nodes representing locations to be visited among the available vehicles. No node is allowed to be left uncovered.
2. In order to save travelled distances, each node must be served by one visit, i.e. by exactly one vehicle.

Table 14.5 Vehicle trips in the AAT example

t	X	y	Occupied seats at departure	Route of
Airport	300	300	9	Vehicle 1
H1	412	121	8	Trip length: 232.42
H2	570	301	6	
H3	490	450	4	
H4	505	353	0	
Airport	300	300		
Airport	300	300	8	Vehicle 2
H5	467	323	6	Trip length: 285.77
H6	150	407	4	
H7	322	212	2	
H8	481	165	1	
H9	388	402	0	
Airport	300	300		
Airport	300	300	9	Vehicle 3
H10	243	243	6	Trip length: 204.08
H11	35	315	4	
H12	412	234	0	
Airport	300	300		
Airport	300	300	3	Vehicle 4
H13	405	351	1	Trip length: 150.81
H14	350	88	0	
Airport	300	300		

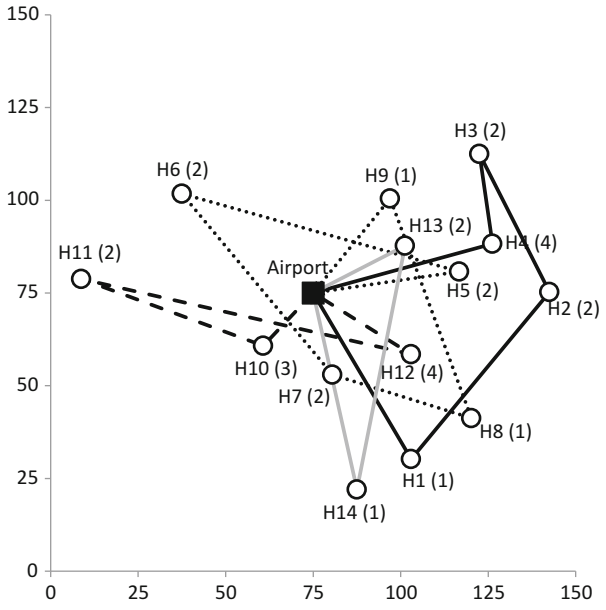

Fig. 14.10 Vehicle trips in the AAT-example

3. Since the vehicle capacity is limited, there is a constraint that must be considered: the maximal payload capacity is not allowed to be exceeded.
4. For each vehicle, a TSP has to be solved (a Hamiltonian path through the common starting point and the locations assigned to this vehicle).

According to the first two properties of AAT's current situation, it is necessary to make a decision about the assignment of customer locations to the vehicles, i.e. the set of customer nodes is clustered into pairwise disjoint sets which are called tours. Each tour is extended by the central position (the depot) from which all vehicles are located before the trip execution starts and to which all vehicles return after they have served all assigned customers (all deliveries have been executed).

The third property implies that the clustering is a decision that is constrained, e.g. some clustering decisions are not allowed (they are infeasible). The fourth observation demonstrates that there is also a sequencing problem (a TSP) which must be solved.

Christina leans back and summarizes her findings. The current AAT situation is a constrained combined clustering and sequencing problem. In order to find out if such a decision situation has been discussed before, she starts an Internet search and learns that the current AAT decision was a problem that had been quite well and intensively studied: it is called the *capacitated vehicle routing problem* (CVRP).

14.4.4 Capacitated Vehicle Routing Problem

Solving a CVRP is the process of finding optimal transportation routes for a given fleet of vehicles under capacity constraints. There are different variations of the CVRP in regard to the specific constraints that an organization faces. In Fig. 14.11, the most important CVRP determinants are summarized.

Consider the most important CVRP determinants. First, *objectives* may relate to time, distance, costs, or sustainability. Second, the consideration of *customers* may be subject to different numbers of customers, demand patterns, and time windows. Third, the *drivers* can be classified regarding different working periods and regulations for working conditions and time.

Fourth, the *fleet* can be considered in different ways depending on the size of vehicles (homogenous or heterogeneous), capacity utilization rules (e.g., LTL: less than truck load or FTL: full truck load), and depot of references (e.g., the requirement on returning to the start depot). Fifth, representation of the *road network* may be different for different CVRP cases. Finally, the *depots* can be considered as single or multiple. The solution methods used for the CVRP consider *heuristic* and *exact* approaches.

► **Practical Insights** CVRP solutions in practice are typically based on heuristic methods since the complexity of CVRP is even higher than in TSP. In addition, up-to-date information technology allows the use of real-time data about traffic jams, weather conditions, etc. This makes it possible to

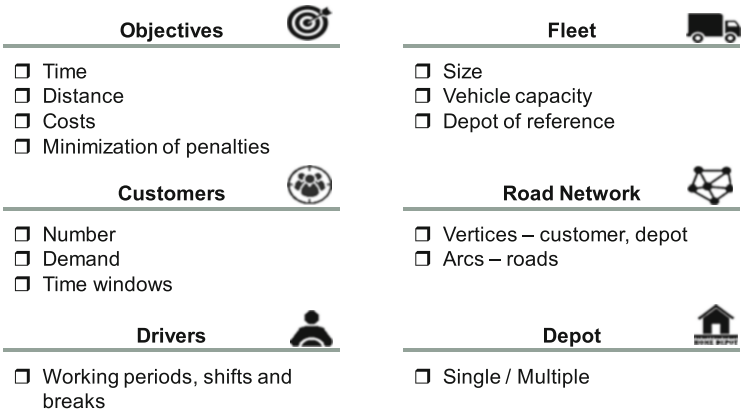

Fig. 14.11 CVRP characteristics

implement *dynamic* or *adaptive* vehicle routing models in combination with sophisticated software to increase flexibility and customer service level.

Having learned that the AAT challenge can be represented as CVRP, Christina wants to know how the vehicle trips (routes) are derived from the requests submitted by the travel agencies. She discusses this issue with the AAT manager. First, she learns that all requests are submitted from the travel agencies to AAT 3 days prior to the due day, which is the execution day. Second, she is told that the consecutively arriving requests are processed in the order of their arrival ("first come/first serve"). Hotel H1 belongs to the first request received, H2 to the second, and so on. The request received first is assigned to the first vehicle, i.e. vehicle 1 visits H1. The second request received is appended to this route after the first request, i.e. H2 is visited by vehicle 1 immediately after H1 has been visited. The next requests are processed similarly as long as the residual capacity (number of free seats) of the vehicle currently under consideration is large enough to serve the next request, i.e. as long as adding the next request would not result in exceeding the capacity of the vehicle. The route of this vehicle is then closed, i.e. the return of the vehicle to the central depot (parking lot) is added and the route of this vehicle is not changed any further. A new vehicle route is opened and the drop-off location for the request that has caused excess capacity is inserted as the first customer location in the route of the second vehicle. Altogether, a greedy strategy similar to the strategy introduced for the TSP is applied to the CVRP. Two differences must be mentioned.

1. The next request is not determined by the smallest distance increase, but by the arrival time of the request.
2. A capacity limit assigned to each vehicle is observed. When the addition of the next request would lead to a capacity excess, a new vehicle route is started.

The pseudocode description of the applied construction procedure for the CVRP is as follows.

Construction Procedure for the CVRP	
(a)	Initialize: START = 0; DEST = 1; k = 1; v = 1; cap_used(v) = 0; L(v) = 0; PATH[v,k] = START;
(b)	Repeat steps (c)-(p) until DEST > CUSTOMERS
(c)	If CAP_used(v) + q(DEST) ≤ CAP(v) then goto (j)
(d)	PATH[v;k + 1]: = 0;
(e)	L:=L + d(START,0);
(f)	v:=v + 1;
(g)	L(v):=0;
(h)	cap_used(v):=0;
(i)	k = 0; START:=0;
(j)	L(v):=L(v) + d(START,DEST);
(k)	cap_used(v): = cap_used(v) + q(DEST);
(l)	k:=k + 1;
(m)	PATH[v;k]:=DEST;
(n)	START:=DEST;
(o)	DEST:=DEST + 1
(p)	Goto (b);
(q)	Return PATH;
(r)	End;

All vehicles start at the depot node 0 and have the remaining payload capacity CAP[v]. The vehicle currently under consideration is represented by v. The assigned payload of vehicle v is labelled by cap_used(v), and L(v) carries the length of the route of vehicle v. PATH[v;k] contains the node that is the k-th customer location in the route of vehicle v. START gives the last visited node and DEST contains the node associated with the request at position DEST in the sequence of requests.

The CVRP construction procedure starts with the initialization of the used variables (a). In each iteration, the steps (b)–(p) are processed. Iteration starts by checking whether there are still unassigned customer locations (b). After that, the assignment of the next request to vehicle v is tested to see if it will lead to an excess in capacity in this vehicle (c). If this is not the case, then the trip length is updated (j), the used capacity is increased (k), the vehicle customer location is appended to the existing partial route (m) and the next customer to be considered is selected (n)–(o). If the capacity test fails, i.e. if the next customer cannot be assigned to the current vehicle, then the depot is appended as the last visited location to the route (d) and the return distance is added (e). Afterwards, the route of the next vehicle is initialized (f)–(i).

Christina asks for the ratio behind the planning logic represented by the CVRP construction procedure. She learns that this planning procedure is very simple and can be applied very quickly. In studying this information, Christina is sure that she can improve the route plan, because the proposed CVRP construction procedure does not

consider any information about the geographic location of the hotels to be visited. This results in unnecessary detours. For example, she refers to the dashed and to the dotted vehicle routes in Fig. 14.10. Here, the two vehicles are "zig-zagging" through the area; travelling back and forth between different regions causes detours.

14.4.5 The Sweep Algorithm

Task 14.2 Solving the CVRP by the Sweep Algorithm
Consider the locations with transformed coordinates (Table 14.6).

In order to overcome the shortcomings associated with the CVRP construction procedure, Christina thinks about opportunities to identify and exploit similar locations of different hotels. One characteristic of a customer location (hotel) is its direction in relation to the central reference point which is the airport parking lot at point (75; 75). Another characteristic of a hotel location is the distance between the airport parking lot and the hotel. If and only if both values, direction as well as distance from the airport of different hotels, are similar then both hotels are situated close together. However, if the direction components are similar then it is perhaps beneficial to assign both customers to the same vehicle since detours are unlikely and the vehicle is already heading in the same direction for both customers (hotels) (Fig. 14.12).

The analysis of the x-y-coordinates that describe the hotel locations does not help Christina to find similarities between the hotel sites. For this reason, she thinks about transforming the x-y-coordinates into polar-coordinates. Therefore, she defines the grey vertical reference line as shown in Fig. 14.12 and determines the direction of a

Table 14.6 Locations with transformed coordinates

	No.	(x; y)-coordinates			Polar coordinates		Angle
Location	i	x	y	q_i	r_i	deg_i	α_i
Airport	0	75	75	0	0	0	0
H6	6	37.50	101.75	2	46.06	0.95	54.51
H11	11	8.75	78.75	2	66.36	1.51	86.78
H10	10	60.75	60.75	3	20.15	2.36	135.01
H14	14	87.50	22.00	1	54.45	3.37	193.27
H7	7	80.50	53.00	2	22.68	3.39	194.04
H1	1	103.00	30.25	1	52.79	3.70	212.04
H8	8	120.25	41.25	1	56.45	4.07	233.29
H12	12	103.00	58.50	4	32.50	4.18	239.50
H2	2	142.50	75.25	2	67.50	4.72	270.21
H5	5	116.25	80.75	2	42.14	4.85	277.84
H4	4	126.25	88.25	4	52.94	4.96	284.50
H13	13	101.25	87.75	2	29.18	5.16	295.91
H3	3	122.50	112.50	2	60.52	5.38	308.30
H9	9	97.00	100.50	1	33.68	5.57	319.22

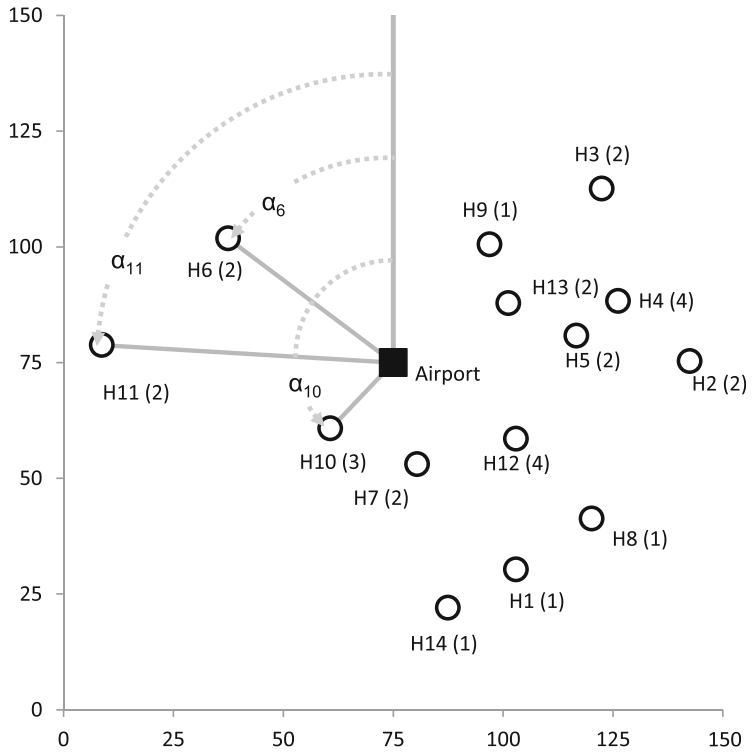

Fig. 14.12 Determination of polar-coordinates of locations in relation to the *vertical grey line*

hotel, e.g. H6, by drawing a line from the airport's coordinates to H6. Each hotel, i.e. H6, is now defined by the distance r_i between the airport and the hotel ("radius") plus the direction as defined by the angle α_i between the vertical reference line and the direction line to the hotel side. In column 7 of Table 14.6, this angle is given by deg_i of the grey arc (the length of the circumference of a complete circle = 6.282). If deg_i is multiplied by 360°/6.282 then we get the angle α_i.

Christina now sorts the requests according to increasing α_i-values. She gets the request sequence shown in columns 1 and 2 in Table 14.6. She saves this sequence in the list SEQ, so that SEQ[1] = H6, SEQ[2] = H11, SEQ[3] = H10, SEQ[4] = H14, SEQ[5] = H7, SEQ[6] = H1, SEQ[7] = H8, SEQ[8] = H12, SEQ[9] = H2, SEQ [10] = H5, SEQ[11] = H4, SEQ[12] = H13, SEQ[13] = H3, SEQ[14] = H9.

Sweep-procedure for the CVRP	
(a)	Initialize: START = 0; REQ = 1; DEST = SEQ[REQ]; k = 1; v = 1; cap_used(v) = 0; L (v) = 0; PATH[v,k] = START;
(b)	Repeat steps (c)–(q) until DEST > CUSTOMERS
(c)	If CAP_used(v) + q(DEST) ≤ CAP(v) then goto (j)
(d)	PATH[v;k + 1]: = 0;
(e)	L:=L + d(START,0);
(f)	v:=v + 1;

(continued)

(g)	L(v):=0;
(h)	cap_used(v):=0;
(i)	k = 0; START:=0;
(j)	L(v):=L(v) + d(START,DEST);
(k)	cap_used(v): = cap_used(v) + q(DEST);
(l)	k:=k + 1;
(m)	PATH[v;k]:=SEQ[REQ];
(n)	START:= SEQ[REQ];
(o)	REQ:=REQ + 1;
(p)	DEST:=SEQ[REQ];
(q)	Goto (b);
(r)	Return PATH;
(s)	End;

The aforementioned pseudocode depicts the modified construction procedure that now incorporates the determined sequence of requests ordered by increasing angle α_i. This procedure appends requests in the sequence SEQ to a vehicle as long as this vehicle has enough capacity. When vehicle capacity is not sufficient to serve the next request according to the SEQ-order, it continues with the next vehicle. If we fix the lower corner of the vertical grey line at the depot at (75; 75) and if we move the other corner counterclockwise around the depot then the grey line "sweeps" consecutively over all locations in the sequence SEQ. For this reason, the procedure represented in the pseudocode is called the "sweep algorithm for the capacitated vehicle routing problem" (Gillett and Miller 1974).

Christina applies the sweep algorithm to the AAT example and gets the route collection printed in Table 14.7. She adds up the travel distances of the three generated routes and finds out that they add up thus: 239.45 km + 209.02 km + 172.42 km = 620.90. Compared to the travel distance sum achieved from the application of the construction procedure shown in Fig. 14.12, which was 873.07 km, Christina calculates a travel distance saving of ≈29% as a consequence of the variation of the sequence in which the request locations are appended to the existing routes. She learns that a careful analysis of the locations to be inserted next at the end of a tentative route significantly contributes to saving travel distances. The resulting vehicle routes are plotted in Fig. 14.13.

The number of vehicles required is also reduced if the sweep procedure is applied. However, Christina concludes that this is an achievement that has not been addressed by the sweep procedure. Her analysis of the sweep procedure reveals that the reduction is not addressed here. However, Christina realizes that a reduction in the number of vehicles used has the potential to contribute to the reduction of the total sum of travelled distances. Her statement is mainly inspired by the following observation. If there are two vehicles following two routes, then each vehicle must travel to the first customer in its route and each vehicle has to travel back from the last visited customer location to the depot. We have two outbound route segments from the depot with length d1 (to node p) and d2 (to node q) and the inbound travel distances from the nodes s and t to the depot of length d3 and d4. If both routes are integrated, then we can save one outbound segment (say of the second vehicle) and one inbound segment which summarize to d2 + d4. If this sum is larger than the

Table 14.7 Vehicle trips in the AAT example proposed by the sweep-algorithm

Location	x	Y	Occupied seats at departure	Route of
Airport	75.00	75.00	10	Vehicle 1
H6	37.50	101.75	8	Trip length: 239.45
H11	8.75	78.75	6	
H10	60.75	60.75	3	
H14	87.50	22.00	2	
H7	80.50	53.00	0	
Airport	75.00	75.00		
Airport	75.00	75.00	10	Vehicle 2
H1	103.00	30.25	9	Trip length: 209.02
H8	120.25	41.25	8	
H12	103.00	58.50	4	
H2	142.50	75.25	2	
H5	116.75	80.75	0	
Airport	75.00	75.00		
Airport	75.00	75.00	9	Vehicle 3
H4	126.25	88.25	5	Trip length: 172.42
H13	101.25	87.75	3	
H3	122.50	112.50	1	
H9	97.00	100.50	0	
Airport	75.00	75.00		

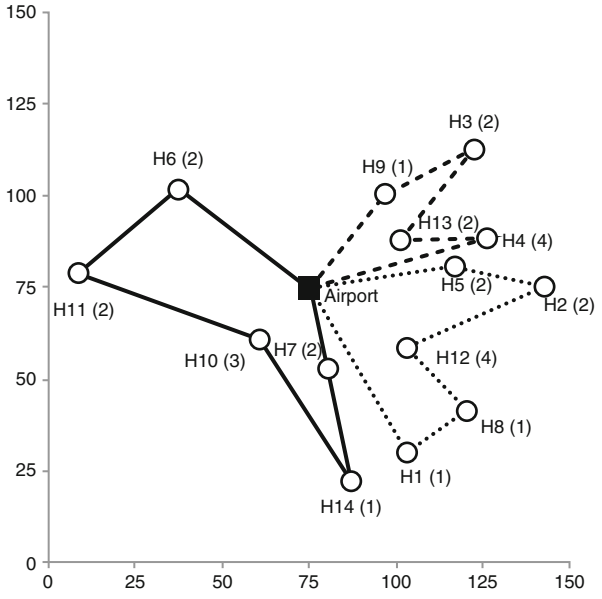

Fig. 14.13 Routes proposed by the sweep algorithm

distance of the additional route segment from s to p of length d(s, p), i.e. if $d2 + d4 - d(s, p) > 0$, then the combination of the two trips realizes travel distance reductions. Clarke and Wright (1964) have proposed a so-called saving algorithm that starts with pendulum routes between the depot and a single customer location and continues to combine two routes to save travel distances. Further details on ADT scenario can be found in the example in the E-Supplement.

14.5 Machine Scheduling

Christina wants to thank her study colleague (his name is Mark) for providing all the information about the TSP that enabled her to understand the algorithmic ideas and improve the operations at RSBT. She calls him one evening and they talk for a long time about their current professional involvement. Mark mentions that he is working on scheduling problems, but in the manufacturing business where he is responsible for setting up machine schedules. Christina is interested and invites him to talk about his experiences.

14.5.1 The Problem of Scheduling a Machine

Mark explains the general setup of a simple one-machine scheduling challenge in a manufacturing environment (Fig. 14.14).

There is a machine that is able to process one manufacturing task at a time. A manufacturing task is part of a customer *order*. All received orders are collected in the order or job backlog. At a specific time, all jobs are prepared for fulfilment. Fulfilling a job means using the machine to execute the manufacturing tasks requested by the customer that is associated with a job.

All jobs from the job backlog are ordered and jobs from the backlog are consecutively processed by the machine in the order determined by the job sequence $o_1, o_2, \ldots, o_N$. The executed tasks create the assets $A_1, A_2, \ldots, A_N$ requested by the customers. The basic decision problem associated with the outlined machine scheduling scenario is to determine the job *processing sequence* that determines the *schedule of the machine*.

Christina wants to know how a manufacturing request can be evaluated in order to find an appropriate job processing sequence. Mark explains that each individual job is associated with different attributes. According to these attributes, jobs can be compared and prioritized. Thonemann (2010) proposed the following three attributes to compare jobs in a job backlog. First, the *due date* d_i of job i is used. The due date determines the latest permitted completion time associated with a job. Second, each job has a specific *processing or flow time* p_i, which is the time needed to process job i on the machine. Third, the *release time* t_i of job i is used to characterize a job. The release time is the time at which the job is completely specified so that its processing can start.

Christina interrupts Mark and tells him that she would have been faced with a similar situation if she had handled the TSP. She had experienced that finding good

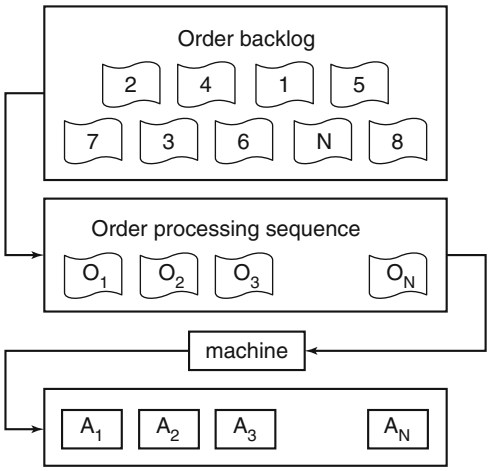

Fig. 14.14 Machine scheduling problem

sequences was a very challenging task due to the strong increase in possible permutations of operations. Furthermore, she wants to know about the planning objectives in machine scheduling. Mark continues and states that there are several commonly used *performance indicators*:

The *completion time* of job *i* determines the time when the job is completely fulfilled, i.e. it is the time when the job "leaves" the machine. The completion time is typically denoted by c_i.

The *flow (lead-time) time* f_i of job *i* is the difference between the completion time c_i and the release time r_i of a job *i*. It can be interpreted as the time span necessary to respond to a customer job release.

Lateness or *tardiness* TD_i of job *i* measures the amount of time in excess of the due date associated with a job *i.*

Work-in-process inventory and *capacity utilization.*

Lead time is composed of processing time, set-up time, waiting time, lying time, and transportation time (Fig. 14.15).

Set-up time is needed to prepare the machine for processing the next job. *Lying time* is needed because of technological restrictions (e.g., drying). *Transportation* time reflects the physical movement of jobs through the machines. *Waiting time* exists because a job cannot be processed at the next machine since this machine is occupied with another job. Even the waiting time and sometimes the set-up time can be affected by sequencing.

Completion times, flow times, and tardiness are performance indicators for quantifying scheduling decisions from the perspective of an individual request. In order to evaluate the quality of a complete job processing sequence, additional performance indicators are used.

The *makespan* MS(S) associated with a job processing sequence S is defined as the length of the period that starts with the release of the first job and ends with the completion of the last job in the sequence S.

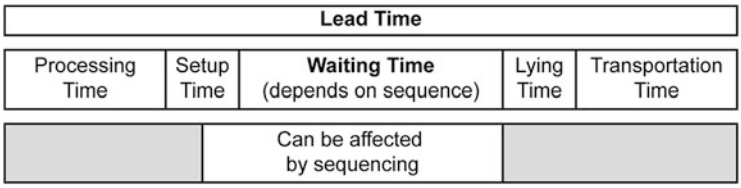

Fig. 14.15 Lead time

The *average completion time* ACT(S) associated with sequence S is the average value of the completion times of the processed jobs if the jobs are processed in the sequence S by the machine.

The *maximal completion time* MCT(S) is the maximal value observed among all completion times after the application of the job processing sequence S.

Mark concludes his short introduction to machine scheduling with the remark that all the performance indicators can be used as primary planning goals. This creates *trade-offs* or goal conflicts since some of the goals contradict each other.

A typical trade-off in production scheduling is lead time and capacity utilization. Just imagine that you are in a supermarket on a Tuesday morning. Probably, you are the only person there at this time. So you will not have to wait anywhere and will exit the supermarket in a short time. The time of your shopping will be short. But what about the supermarket? If you are the only customer, their capacity utilization is very low. The same effect can be encountered in manufacturing and service systems in regard to customer order lead time and resource capacity utilization.

This typical *trade-off in scheduling* is called *the scheduling dilemma*. According to the scheduling dilemma, we cannot achieve a reduction in lead time and an increase in capacity utilization at the same time. Mark gives an example. Students at the Berlin School of Economics and Law arrive at the subway station Berliner Strasse, which is about a 10 min walk from the university. They can also use bus 104 to get to the school from the subway station. Assume that the buses run every 10 min and every minute one student comes to the bus stop. In this case, the waiting times and the lead times will be quite long, but the bus will take ten students and have good capacity utilization. Should the buses run every minute, the lead times would be very short (no waiting time), but the capacity utilization of each bus would be low.

14.5.2 Priority Rule-Based Scheduling

After she has learned that the schedule generation is also based on the sequencing of operations, as is the visit sequencing in the TSP, Christina wants to know how appropriate job processing sequences are determined. She asks Mark whether he also uses "greedy" strategies to add operation by operation to a partial sequence. Mark confirms her speculation, but tells Christina that there are some sort of *priority* or *dispatching rules* whose applications make the sequencing straightforward.

Mark explains that a priority rule is a "rule of thumb" to sort a set of jobs according to increasing or decreasing values of the release times r_i, the flow times f_i, or the due dates d_i. The application of such a "simple" rule seems to be promising since the evaluation value of an individual job is not affected by the earlier or the subsequently processed job. This is a major difference from sequencing in the TSP when the travel distances between consecutively visited nodes impacts the overall objective which is the total sum of travelled distances. Priority rules can be applied both for *job shop* and *flow shop* (cf. Chap. 9).

Task 14.3 Scheduling by Priority Rules

In order to demonstrate the effects of determining a job processing sequence using a priority rule, Mark introduces an example with eight jobs. The attributes of the jobs from the job backlog are summarized in Table 14.8.

The most intuitive approach for the determination of the job processing sequence is to process the jobs in the sequence of their arrival or release time. Those jobs arriving first will be processed first. This sequencing rule is called *First Come/First Serve (FCFS).* It does not apply to any "intelligent" sorting, but it is a fair approach.

Table 14.9 shows the machine schedule resulting from the FCFS-processing job. The makespan is 44–1 = 43 time units. The average completion time equals 27 time units, but the maximal completion time is 44 time units. Five jobs are completed with delays. The average tardiness is 6.75 time units and the maximal tardiness is 23 time units. On average, the flow time is 22.5 time units.

The application of FCFS does not consider processing times or due dates while determining the sequence of processing. In order to enable a quick delivery of jobs, it is useful to try to minimize the completion times of jobs. Here, it is beneficial to process those jobs first that can be processed rapidly in order to make the machine available for the next job as quickly as possible. According to the *Shortest-Processing-Time rule (SPT-rule)* the jobs are sorted by increasing processing time. The job with the shortest processing time in the job backlog is processed first.

The SPT-generated job processing sequence is listed in column 1 of Table 14.10. Again, the makespan is 43 time units, but the average completion time is reduced from 27 time units to 23.25 time units. The maximal completion time remains 44 time units. Four jobs are still delayed with a slightly reduced tardiness of 6 time units. However, the maximal tardiness is prolonged to 26 time units, but the average flow time is reduced to 18.75 time units.

Neither the application of the SPT rule nor the application of FCFS is able to control the maximal tardiness since neither of these two rules considers the due date information associated with the jobs. In order to minimize the maximal tardiness, the *Earliest Due Date rule (EDD-rule*) is proposed. According to the EDD-rule, the

Table 14.8 Job backlog

Job i	1	2	3	4	5	6	7	8
Release time r_i	1	2	3	5	6	7	8	1
Duration p_i	4	2	3	7	2	8	3	4
Due date d_i	14	29	16	23	24	18	31	14

Table 14.9 Job processing sequence and machine schedule generated by FCFS

Job i	Release time r_i	Duration p_i	Due date d_i	Start time	Finish time	Tardiness	Flow time	Delayed?
1	1	4	14	8	12	0	11	0
2	2	2	29	12	14	0	12	0
3	3	3	16	14	17	1	14	1
4	4	7	28	17	24	0	20	0
5	5	7	23	24	31	8	26	1
6	6	2	24	31	33	9	27	1
7	7	8	18	33	41	23	34	1
8	8	3	31	41	44	13	36	1

Table 14.10 Job processing sequence and machine schedule generated by SPT

Job i	Release time r_i	Duration p_i	Due date d_i	Start time	Finish time	Tardiness	Flow time	Delayed?
2	2	2	29	8	10	0	8	0
6	6	2	24	10	12	0	6	0
3	3	3	16	12	15	0	12	0
8	8	3	31	15	18	0	10	0
1	1	4	14	18	22	8	21	1
4	4	7	28	22	29	1	25	1
5	5	7	23	29	36	13	31	1
7	7	8	18	36	44	26	37	1

most urgent job is processed first, followed by the job with the second highest urgency.

The makespan remains 43 time units (since there is no slack time in the schedules) as shown in Table 14.11. The average completion time is 29.5 time units and the maximal completion time equals 44 time units. An average flow time of 25 time units is observed. The application of the EDD-rule leads to a significant reduction in maximal tardiness from 23 (FCFS) and 26 (EDD), respectively, down to 13 time units. Unfortunately, the number of delayed requests is increased (six jobs). The same development is observed for the average tardiness, which is increased by to 7 time units compared to 6 time units (SPT) and 6.75 time units (FCFS).

► **Practical Insights** The application of a priority rule enables the minimization of a single performance indicator value. There is no priority rule known that minimizes all three performance indicators (1) number of delayed requests (2) average completion time and (3) maximal delay. However, priority rules are easy to apply. This is why these simple heuristics are widely used in practice. In sophisticated scheduling software, so-called *meta-heuristics* such as *genetic algorithms* and *ant colony optimization* (ACO) are used. They provide solutions of very high quality.

Table 14.11 Job processing sequence and machine schedule generated by EDD

Job i	Release time r_i	Duration p_i	Due date d_i	Start time	Finish time	Tardiness	Flow time	Delayed?
1	1	4	14	8	12	0	11	0
3	3	3	16	12	15	0	12	0
7	7	8	18	15	23	5	16	1
5	5	7	23	23	30	7	25	1
6	6	2	24	30	32	8	26	1
4	4	7	28	32	39	11	35	1
2	2	2	29	39	41	12	39	1
8	8	3	31	41	44	13	36	1

14.5.3 Scheduling Algorithm of Moore

Moore (1968) published a paper about a one-machine scheduling algorithm that minimizes the number of delayed jobs. The basic idea is to extend the EDD rule by identifying those jobs that block the machine. Those jobs are then moved to the end of the sequence with the hope that now one or even more jobs can be started earlier so that the number of delayed jobs can be reduced. Mark is going to explain to Christina how the Moore algorithm works by means of the example previously introduced.

First, the job backlog is scheduled according to the EDD rule. If there are no delayed jobs, then the schedule construction is finished. Otherwise, the first delayed job according to the EDD-induced schedule is marked; here job 7 is the first delayed job (Table 14.12).

The basic idea of the Moore algorithm is to find out the cause of this delay. Moore supposed that the marked or a previously processed job is responsible for the delay. To this end, Moore suggested moving the labelled job 7 or one of the previously scheduled jobs (job 1 or job 3) to the end of the sequence so that all subsequent jobs can be started earlier. In order to identify the most appropriate job to be moved to the end, Moore proposed comparing the processing times p_i of the marked job and of the earlier scheduled jobs and selecting the job among the aforementioned jobs with the longest processing time in order to achieve a maximal left shift of the job starting times. In the example (Table 14.12), job 7 exhibits the longest processing time $p_7 = 8$ so that it is moved to the end of the sequence. This job is marked by * in Table 14.13.

In the next iteration, the first delayed job (job 4) is marked. This job and job 5 have the same maximal processing time of 7 time units among all requests started after the marked job. Job 5 is selected (since it appears first in the intermediate sequence) and job 5 is moved to the end of the sequence (Table 14.14).

Table 14.14 contains the final scheduling sequence (1; 3; 6; 4; 2; 8; 7; 5). None of the so far unmoved jobs is delayed so that the algorithm terminates. The minimal number of delayed jobs is two.

Table 14.12 Moore-algorithm (start): job processing sequence and machine schedule generated by EDD and marked first delayed job 7

Job i	Release time r_i	Duration p_i	Due date d_i	Start time	Finish time	Tardiness	Flow time	Delayed?
1	1	4	14	8	12	0	11	0
3	3	3	16	12	15	0	12	0
7	**7**	**8**	**18**	**15**	**23**	**5**	**16**	**1**
5	5	7	23	23	30	7	25	1
6	6	2	24	30	32	8	26	1
4	4	7	28	32	39	11	35	1
2	2	2	29	39	41	12	39	1
8	8	3	31	41	44	13	36	1

Table 14.13 Moore-Algorithm (2nd iteration): Job processing sequence and machine schedule generated by EDD and marked first delayed job 4

Job i	Release time r_i	Duration p_i	Due date d_i	Start time	Finish time	Tardiness	Flow time	Delayed?
1	1	4	14	8	12	0	11	0
3	3	3	16	12	15	0	12	0
5	5	7	23	15	22	0	17	0
6	6	2	24	22	24	0	18	0
4	**4**	**7**	**28**	**24**	**31**	**3**	**27**	**1**
2	2	2	29	31	33	4	31	1
8	8	3	31	33	36	5	28	1
7*	7	8	18	36	44	26	37	1

Table 14.14 Moore-Algorithm (3rd iteration): Final job processing sequence and machine schedule with minimal number of delayed requests

Job i	Release time r_i	Duration p_i	Due date d_i	Start time	Finish time	Tardiness	Flow time	Delayed?
1	1	4	14	8	12	0	11	0
3	3	3	16	12	15	0	12	0
6	6	2	24	15	17	0	11	0
4	4	7	28	17	24	0	20	0
2	2	2	29	24	26	0	24	0
8	8	3	31	26	29	0	21	0
7*	7	8	18	29	37	19	30	1
5*	5	7	23	37	44	21	39	1

14.5.4 Scheduling Two Machines in a Flow Shop

Christina detects similarities between the machine scheduling problem outlined by Mark and the TSP. There are the jobs which correspond to the requests in the TSP and there are resources: in the TSP there is a vehicle, but in Mark's setup there is a

manufacturing machine. Christina wonders if there is a machine scheduling setup with two or even more machines that corresponds to the CVRP (Fig. 14.16).

She asks Mark about this question. Mark affirms, but tells Christina that there are several multi-machine setups. Christina wants to know what the schedule generation is like in such a multi-machine scenario. Mark starts with a description of the system setup (Fig. 14.16). Each job requires two production steps. In the first step, the work piece is painted and in the second step the painted piece is protected by a coating. Of course, it is necessary that painting precedes coating for all jobs. Each job *i* is characterized by the two processing times p_i^{paint} for the painting procedure as well as p_i^{coat} for the coating procedure (Table 14.15).

Mark explains that there is a famous approach to this setting for minimizing the completion time of the complete job backlog. This algorithm was originally proposed by Johnson (1954) and is hence called *Johnson's algorithm*. This algorithm follows the principle of keeping the *idle times* of the second machine as short as possible. This is achieved by starting those jobs first on the first machine with the shortest p_i^{paint}-values. These jobs are quickly processed by the first machine and then forwarded to the second machine in order to avoid idle times on the last mentioned machine. The starting sequences of the jobs on the two machines should be kept unchanged. Therefore, it is sufficient to define the job processing sequence only for the first machine. The determined sequence then also applies for the second machine.

The Johnson's algorithm starts by splitting up the job backlog into two subsets J_1 and J_2. All jobs i whose processing times p_i^{paint} are shorter than their processing times p_i^{coat} are put into J_1. The remaining jobs are collected in the set J_2. In the example setting, the first set is $J_1 := \{1; 2; 5; 7\}$ and the second set is $J_2 := \{3; 4; 6; 8\}$.

In the second step, both sets are ordered. The set J_1 is sorted by increasing processing times p_i^{paint} and the set J_2 is sorted by decreasing processing times p_i^{coat}. In the example, we have the sequence $S_1 := (2; 5; 7; 1)$ as well as the sequence $S_2 := (4; 6; 8; 3)$.

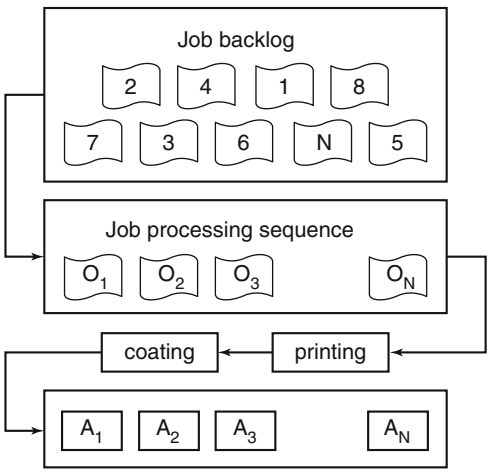

Fig. 14.16 Two machine scheduling situation

Table 14.15 Job backlog for the two-machine set-up

Job i	1	2	3	4	5	6	7	8
Release time	3	4	5	6	7	8	9	10
Paint duration p_i^{paint}	4	2	3	7	2	8	3	4
Coating duration p_i^{coat}	7	6	1	5	4	3	4	2

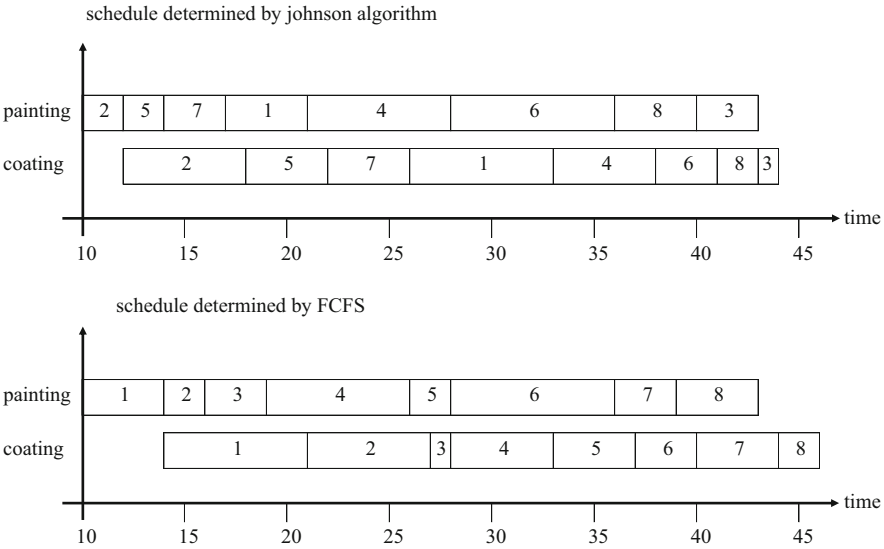

Fig. 14.17 Two machine schedules generated by FCFS and Johnson's algorithm

In the third step, sequence S_1 is combined with sequence S_2 by appending S_2 to S_1. In the example, we get the complete job backlog sequence $S:=(S_1, S_2) = (2; 5; 7; 1; 4; 6; 8; 3)$.

Mark outlines the generated schedule in a *Gantt-chart* (i.e., a special form of schedule representation) and compares it with the schedule that results from starting the jobs in the order in which they have been released (Fig. 14.17). It can be observed that makespan has been reduced with the help of Johnson's algorithm.

14.5.5 Further Challenges in Machine Scheduling

In reality, there is a variety of other more complicated machine scheduling situations (Kovalyov et al. 2007; Dolgui et al. 2010; Berrichi and Yalaoui 2013). Li and Chen (2010) investigated a one-machine set-up in which the exact processing times of the jobs in the backlog are unknown, but estimated by a distribution function. Albers (1997) analyzed a setup in which the job backlog is not available. Instead, no arriving job is scheduled before it arrives or without considering a job arriving subsequently.

If two or more machines must be scheduled in order to fulfil a set of jobs, two general setups are distinguished. The case outlined by Mark falls into the category of *flow production* (Dolgui and Proth 2010; Pinedo 2010): all jobs are processed in the same sequence on all available machines. In the *job shop* setup (Mattfeld 1996; Werner 2013), each job maintains an individual machine processing sequence. In many industries, flexible flow and job shops with alternative parallel machines are used (Kyparisis and Koulamas 2006; Ivanov and Sokolov 2012). In the process industry, continuous flows represent an additional challenge (Shah 2004; Bożek and Wysocki 2015; Ivanov et al. 2016a).

Next, SC coordination belongs to future trends in scheduling research and practice. In this area, *integration of scheduling and routing* problems is studied (Agnetis et al. 2006; Chen 2010). In addition, integration of *inventory-routing* decisions as well as supplier selection and scheduling is a promising research area (Sawik 2013).

Finally, *uncertainty* challenges scheduling and routing decisions. In this setting, the issues of schedule *stability and robustness* along with the development of rescheduling policies has become more and more important (Kolisch and Hess 2000; Vieira et al. 2003; Artigues et al. 2005; Aytug et al. 2005; van de Vonder et al. 2007; Hazir et al. 2010; Gomes et al. 2013; Sotskov et al. 2013; Harjunkoski et al. 2014; Liu and Ro 2014).

14.6 Key Points

In this chapter, you have been introduced to typical operative process planning tasks from transportation as well as production. In particular, you are now familiar with the shortest path problem, the travelling salesman problem, the CVRP, with the single machine scheduling problem, and a simple two-machine flow-shop problem.

You have learned the basics about the formulation of decision models on mathematical graphs and you are now familiar with the concept of modeling operative decision tasks as graph-based optimization models. Decision variables are used to represent atomic decision tasks and constraints must be considered in order to compile a feasible solution for a complex decision problem from the atomic decisions.

You are aware now that a careful selection of adequate algorithms for solving optimization models defined on graphs is important. For some decision problems, such models as the one presented for the shortest path problem, there are exact algorithms. As an example, you have been introduced to the Dijkstra algorithm.

Christina leans back and recapitulates on what she has learned. First, a network-based optimization model for TSP was introduced. This model falls into the category of a mixed-integer linear problem since some decision variables must be integers or even binary. Within such a model, it is possible to add constraints to control the decision variable value determination.

Compared to the shortest path problem in a network, which does not have any constraints about nodes to be visited, the complexity of the TSP is significantly

increased. Therefore, it is reasonable to refrain from using exact algorithms for the identification of the best possible Hamiltonian path (with the least travel distance). Instead heuristics are proposed to approximate an optimal Hamiltonian path. There are different types of heuristics. Construction heuristics are used to set up an initial feasible solution for a constraint optimization model. Improvement heuristics (e.g. a 2-opt improvement procedure) are used to improve the objective function value of the best found feasible solution, i.e. to replace an existing feasible solution with another feasible solution having an improved objective function value). Christina is happy since she can use the proposed techniques to model and solve RSBT's SST challenge.

Section 14.4 introduced the decision problem class of combined assignment and sequencing problems from operation fleet management. The basic decision task is described in the CVRP. We started the discussion of the CVRP with an outline of a typical planning situation. By means of this example, we discussed the sophisticated decision challenges of a simultaneously conducted partition of the set of requested locations, and the sequencing of the locations in order to determine vehicle routes. We have seen that failures in the assignment of requests to vehicles typically result in detours and an increase in the number of required vehicles. Both issues contribute to additional costs for fulfilling customer requests.

In order to contribute to keeping the fulfilment costs as low as possible, we proposed analysis of customer locations, i.e. the locations that require a visit. For this reason, we proposed sorting all locations by means of their angle relative to a reference line. We then proposed the sweep algorithm, which exploits information about the vicinity of different locations. The sweep algorithm tries to compile closely situated locations into one route in order to keep the sum of travelled distances low. We are now prepared to manage all decision situations that are in the form of the AAT challenge.

Finally, we have learned how to design simple heuristics for complex routing and scheduling models. But you are also aware that the computation of (sub)-optimal model solutions is often very complicated. Nevertheless, you understand the importance of formulating an appropriate decision model as the interface between applications and computers. The formulation is the most important ingredient for the setup of computerized decision support systems.

Toth and Vigo (2002) discussed model formulations for the CVRP. These models are used to apply special solver tools like CPLEX or LINGO in order to derive proven optimal solutions to the CVRP. However, depending on the actual data of a CVRP scenario and depending on the number of available vehicles and requests to be served, the processing times are often prohibitive so that for practical real-world problem settings heuristics are applied preferentially (Gendreau et al. 2002).

For other decision models, it is necessary to incorporate heuristics like the greedy heuristic or the sweep heuristic. There are also priority rules that can be applied to sequence tasks in machine scheduling. You can distinguish between construction and improvement procedures as basic ingredients of heuristic algorithms for solving complex models with several constraints. Fleet routing and machine scheduling belong to the most exciting tasks in SCOM!

Bibliography

Agnetis A, Hall NG, Pacciarelli D (2006) Supply chain scheduling: sequence coordination. Discret Appl Math 154(15):2044–2063

Albers S (1997) Better bounds for online scheduling. SIAM J Comput 29(2):459–473

Andersson A, Hoff A, Christiansen M, Hasle G, Løkketangen A (2010) Industrial aspects and literature survey: combined inventory management and routing. Comput Oper Res 37:1515–1536

Artigues C, Billaut J-C, Esswein C (2005) Maximization of solution flexibility for robust shop scheduling. Eur J Oper Res 165(2):314–328

Aytug H, Lawley MA, McKay K, Mohan S, Uzsoy R (2005) Executing production schedules in the face of uncertainties: a review and some future directions. Eur J Oper Res 161(1):86–100

Berrichi A, Yalaoui F (2013) Efficient bi-objective ant colony approach to minimize total tardiness and system unavailability for a parallel machine scheduling problem. Int J Adv Manuf Tech 68 (9–12):2295–2310

Blazewicz J, Ecker K, Pesch E, Schmidt G, Weglarz J (2001) Scheduling computer and manufacturing processes, 2nd edn. Springer, Berlin

Bożek A, Wysocki M (2015) Flexible job shop with continuous material flow. Int J Prod Res 53 (4):1273–1290

Chen Z-L (2010) Integrated production and outbound distribution scheduling: review and extensions. Oper Res 58(1):130–148

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. Oper Res 12(4):568–581

Croes G (1958) A method for solving traveling-salesman problems. Oper Res 6(6):791–812

Desrochers M, Laporte G (1991) Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. Oper Res Lett 10:27–36

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1:269–271

Doerner KF, Gronalt M, Hartl RF, Kiechle G, Reimann M (2008) Exact and heuristic algorithms for the vehicle routing problem with multiple, interdependent time windows. Comput Oper Res 35:3034–3048

Dolgui A, Proth J-M (2010) Supply chain engineering: useful methods and techniques. Springer, Berlin

Dolgui A, Eremeev AV, Kovalyov MY, Kuznetsov PM (2010) Multi-product lot-sizing and scheduling on unrelated parallel machines. IIE Trans 42(7):514–524

Dolgui A, Ivanov D, Sethi SP, Sokolov B (2018) Scheduling in production, supply chain and industry 4.0 systems by optimal control. Int J Prod Res:1–22. https://doi.org/10.1080/00207543.2018.1442948

Gendreau M, Laporte G, Potvin J-Y (2002) Metaheuristics for the capacitated VRP. In: Toth P, Vigo D (eds) The vehicle routing problem. Society for Industrial and Applied Mathematics, Philadelphia, pp 129–154

Gillett BE, Miller LR (1974) A heuristic algorithm for the vehicle-dispatch problem. Oper Res 22 (2):340–349

Gomes MC, Barbosa-Póvoa AP, Novais AQ (2013) Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly a mathematical programming approach. Int J Prod Res 51(17):5120–5141

Grünert T, Irnich S (2005) Optimierung im transport - band I: grundlagen. Shaker, Aachen

Harjunkoski I, Maravelias CT, Bongers P, Castro PM, Engell S, Grossmann IE, Hooker J, Méndez C, Sand G, Wassick J (2014) Scope for industrial applications of production scheduling models and solution methods. Comput Chem Eng 62:161–193

Hazir O, Haouari M, Erel E (2010) Robust scheduling and robustness measures for the discrete time/cost trade-off problem. Eur J Oper Res 207(2):633–643

Ivanov D, Sokolov B (2012) Dynamic supply chain scheduling. J Sched 15(2):201–216

Ivanov D, Sokolov B, Dolgui A, Werner F, Ivanova M (2016a) A dynamic model and an algorithm for short-term supply chain scheduling in the smart factory industry 4.0. Int J Prod Res 54 (2):386–402

Ivanov D, Dolgui A, Sokolov B (2016b) Robust dynamic schedule coordination control in the supply chain. Comput Ind Eng 94(1):18–31

Ivanov D, Dolgui A, Sokolov B (2018) Scheduling of recovery actions in the supply chain with resilience analysis considerations. Int J Prod Res:1–18. https://doi.org/10.1080/00207543.2017.1401747

Joereßen A, Sebastian H-J (1998) Problemlösung mit Modellen und Algorithmen. Teubner, Stuttgart

Johnson SM (1954) Optimal two- and three-stage production schedules with setup times included. Nav Res Logist Q 1(1):61–68

Kolisch R, Hess K (2000) Efficient methods for scheduling make-to-order assemblies under resource, assembly area and part availability constraints. Int J Prod Res 38(1):207–228

Kovalyov MY, Ng CT, Cheng TCE (2007) Fixed interval scheduling: models, applications, computational complexity and algorithms. Eur J Oper Res 178:331–342

Kyparisis GJ, Koulamas CP (2006) Flexible flow shop scheduling with uniform parallel machines. Eur J Oper Res 168:985–997

Li Y, Chen R (2010) Stochastic single machine scheduling to minimize the weighted number of tardy jobs. In: Cao B-Y, Wang E, Guo S-Z, Chen S-L (eds) Fuzzy information and engineering 2010. Springer, Berlin, pp 363–368

Liu Z, Ro YK (2014) Rescheduling for machine disruption to minimize makespan and maximum lateness. J Sched 17(4):339–352

Maccarthy BL, Liu J (1993) Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. Int J Prod Res 31(1):59–79

Mattfeld DC (1996) Evolutionary search and the job shop. Physica-Verlag, Heidelberg

Moore JM (1968) An n job, one machine sequencing algorithm for minimizing the number of late jobs. Manag Sci 15(1):102–109

Pinedo ML (2010) Theory, algorithms, and systems, 4th edn. Springer, New York

Ritzinger U, Puchinger J, Hartl RF (2016) A survey on dynamic and stochastic vehicle routing problems. Int J Prod Res 54(1):215–231

Sawik T (2013) Integrated selection of suppliers and scheduling of customer orders in the presence of supply chain disruption risks. Int J Prod Res 51(23–24):7006–7022

Shah N (2004) Process industry supply chains: advances and challenges. Comput Aided Chem Eng 18:123–138

Solomon MM (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 5(2):254–265

Sotskov YN, Lai T-C, Werner F (2013) Measures of problem uncertainty for scheduling with interval processing times. OR Spectr 35(3):659–689

Thonemann U (2010) Operations management: konzepte, methoden und anwendungen, 2nd edn. Pearson, München

Toth P, Vigo D (2002) Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discret Appl Math 123(1–3):487–512

Van de Vonder S, Demeulemeester E, Herroelen W (2007) A classification of predictive-reactive project scheduling procedures. J Sched 10(3):195–207

Vieira GE, Herrmann JW, Lin E (2003) Rescheduling manufacturing systems: a framework of strategies, policies and methods. J Sched 6(1):35–58

Werner F (2013) A survey of genetic algorithms for shop scheduling problems. In: Siarry P (ed) Heuristics: theory and applications. Nova Science, New York, pp 161–222

References for Sect. 14.4.1

Konrad A (2013) Meet ORION, software that will save ups millions by improving drivers' routes. http://www.forbes.com/sites/alexkonrad/2013/11/01/meet-orion-software-that-will-save-ups-millions-by-improving-drivers-routes/. Accessed 19 Jan 2015

Noyes K (2014) The shortest distance between two points? At UPS, it's complicated. Fortune Online, July 25. http://fortune.com/2014/07/25/the-shortest-distance-between-two-points-at-ups-its-complicated/. Accessed 2 Dec 2014

Shontell A (2011) Why UPS is so efficient: "Our trucks never turn left". Business Insider, March 24. http://www.businessinsider.com/ups-efficiency-secret-our-trucks-never-turn-left-2011-3. Accessed 28 Nov 2014

Wohlsen M (2013) The astronomical math behind UPS' new tool to deliver packages faster. Wired Magazine Online, December 13. http://www.wired.com/2013/06/ups-astronomical-math/. Accessed 9 Dec 2014