Universiteit Twente
de ondernemende universiteit

ORTEC

The development of a generic scheduling approach for the PLANWISE optimizer

Graduation assignment

E. M. Alvarez

Supervisors: Dr. Ir. J.M.J. Schutten
Dr. J. L. Hurink
Dr. Ir. A. J. R. M. Gademann

Enschede, 16th December 2008

Preface

In this report, I present the results I obtained while executing my Master's assignment at ORTEC. With the completion of this assignment, I complete my Master program in Industrial Engineering and Management at the University of Twente.

The execution of my Master's assignment has taken me one year. During this year, I have learned many things in the area of scheduling. In addition, ORTEC gave me the opportunity to perform a number of projects in different areas. Overall, it was an enjoyable year in which I was able to do many things.

I now take the opportunity to thank a number of people. First of all, I thank my supervisors, Marco Schutten and Johann Hurink of the University of Twente and Noud Gademann of ORTEC, for all their support during the execution of my assignment. I could always turn to them for useful insights and feedback on my work.

I also thank my colleagues at ORTEC for making my time at ORTEC so enjoyable. I appreciate the support they have given me during the execution of my assignment and the enjoyable work environment they provided. I especially thank Daniël Dam for all the daily support during the first few months of my assignment and all the guys at Software Development for the help they gave me with programming the scheduling approaches (and solving all those impossible bugs I managed to create).

Last, but certainly not least, I thank all my family and friends for their love and support. During my assignment, they have always shown interest in my work and they were always willing to listen when I needed them.

Summary

In this assignment, we focus on creating scheduling approaches for the optimizer in PLANWISE, a scheduling system created by ORTEC that supports the user in scheduling operations on different resources, such as machines and operators. The scheduling approaches must be able to solve a scheduling problem specified by the optimizer. In other words: The approaches must be able to create a schedule in which a set of operations is scheduled on a set of resources.

The scheduling approaches have to meet two requirements:

1. The approaches should be generic and be able to handle different scheduling situations that occur in practice.
2. The approaches need to be able to handle manual adjustments made by users. More specifically: The approaches must be able to solve scheduling problems that contain fixed operations. These operations have already been scheduled on a set of resources and they may not be rescheduled: i.e. they may not be reassigned to different resources and they may not obtain different starting and completion times.

To create scheduling approaches that meet these requirements, we first determine which scheduling problem the approaches should be able to solve. After considering the scheduling situations at different PLANWISE customers and potential customers, we create a general scheduling problem that contains the constraints and objectives mentioned most often by the PLANWISE customers. In the resulting scheduling problem, we have a set of jobs that each consist of one or more operations. These operations each need to be scheduled on a set of resources for their processing. The objective is now to schedule the operations on the available resources such that the total tardiness of the jobs in the schedule is minimized. The main constraints that we consider in our scheduling problem are general finish-start precedence relations, unique resources, preemptive and non-preemptive downtimes, setup times, routing constraints, resource-dependent processing times, and fixed operations.

Subsequently, we determine which scheduling approaches we use to solve our scheduling problem. After performing a literature study on different scheduling approaches, we create two approaches to solve the general scheduling problem: One approach based on an integrated architecture and the regret-based random sampling algorithm and one approach based on the hierarchical architecture and local search algorithms. These approaches are able to handle the following constraints of the general scheduling problem: Finish-start precedence relations, unique resources, preemptive and non-preemptive downtimes, and setup times. The integrated approach is also able to handle routing constraints. In addition, we expect that the approaches are also able to handle fixed operations.

We test both approaches on a set of scheduling instances and we subsequently compare the performance of the approaches to each other. In this comparison, we use the integrated approach as the benchmark approach, because ORTEC currently uses this approach in certain PLANWISE optimizers. Based on the test results, we can make a number of conclusions: First, on the tested instances the hierarchical approach generally performs better than the integrated approach with respect to tardiness minimization. Among the different versions of the hierarchical approach, we obtain the best results when we use simulated annealing to solve the assignment problem. In addition, a good initial solution and frequent iteration between the assignment and sequencing steps are also beneficial for the performance of the approach.

However, improvements can still be made to both approaches with respect to both tardiness minimization and calculation time. Based on test results, we are able to conclude that the characteristics of a scheduling instance have a lot of influence on the performance of the integrated approach. Specifically, the characteristics of the jobs in the scheduling instance have a lot of influence on the decision sets created to choose a next operation to schedule and the operation chosen from this decision set. Therefore, further research is required to fine tune the integrated approach to the scheduling instance. Also, further tests need to be done to determine the influence of the initial solution and communication scheme on the performance of the hierarchical approach. Finally, both approaches need to be adjusted so they are able to handle all constraints of the general scheduling problem: First, both approaches need to be adjusted so they can handle resource-dependent processing times. In addition, at present the approaches can only solve scheduling problems where all resources have the same intervals of preemptive downtime. In practice, however, resources often have different intervals of preemptive downtime. Finally, the hierarchical approach must be adjusted so it is able to handle routing constraints.

Table of contents

1. Introduction

This assignment takes place at ORTEC, a company specialized in advanced planning and optimization software and consulting services. ORTEC is active in several different areas, such as workforce scheduling, vehicle routing and dispatch, and production and project planning. In the area of production and project planning, ORTEC currently has a scheduling system called PLANWISE. PLANWISE is a real-time system that supports the user (a human scheduler) in the scheduling of activities on different resources, such as machines, employees, and tools. To support PLANWISE users even further, ORTEC would now like to implement a generic optimizer in PLANWISE that can create a schedule automatically. This optimizer requires a scheduling approach that is able to schedule a set of activities on a set of resources. In doing so, the approach should ensure that a set of restrictions is met when creating the schedule. In addition, the approach should optimize the schedule on a set of objectives.

In this assignment, we focus on creating a generic scheduling approach for the PLANWISE optimizer. We can divide the assignment in three main sections: First, we need to determine which scheduling problem we wish to solve with the scheduling approach: We need to determine which set of restrictions the approach should be able to handle and the objectives that the approach should optimize when creating a schedule. Second, we need to create different scheduling approaches to solve the scheduling problem. Finally, we need to compare the different approaches to each other to determine which approach is most suitable.

The remainder of the report is as follows: In Chapter 2, we describe in more detail the problem that we consider in this assignment. Here, we also mention the research questions that we wish to answer and we describe our research approach. In Chapter 3, we subsequently give an introduction into scheduling and we describe how scheduling takes place in PLANWISE. Next, we mention different scheduling situations that we encounter in practice in Chapter 4. Based on these scheduling situations, we determine the general scheduling problem that we consider in the remainder of the assignment. We present this scheduling problem in Chapter 5. In Chapter 6, we present the results of a literature study on scheduling approaches for the general scheduling problem. Based on this study, we create two main scheduling approaches, which we describe in Chapter 7. We subsequently test both approaches on a number of scheduling instances and we provide the results of these tests in Chapter 8. Finally, in Chapter 9 we present some conclusions and we give some recommendations for further research.

2. Problem description

2.1 Introduction

In Chapter 1, we mentioned that ORTEC has a scheduling system called PLANWISE that supports the user in scheduling activities on different resources. At the moment, scheduling occurs mainly manually in this system: The user himself needs to choose which activity he will subsequently schedule and on which resource and at what time he will schedule the activity. The function of PLANWISE is to show the user which possibilities there are for scheduling a certain activity (e.g. which resources are available for scheduling and at what times they are available).The so-called engine in the system calculates and propagates all consequences and generates warnings in case of violations.

ORTEC now wishes to implement an optimizer function in PLANWISE that can create a schedule automatically. The user should be able to specify the activities that need to be scheduled and the resources on which the activities need to be scheduled. Subsequently, the optimizer needs to perform the following steps:

1. It needs to translate the request made by the user into a scheduling problem that needs to be solved. The scheduling problem specifies which restrictions need to be met when scheduling the activities on the resources. In addition, the problem specifies which objectives need to be optimized when creating the schedule.
2. It needs to solve the scheduling problem. To solve this scheduling problem, the optimizer requires a *scheduling approach.* This scheduling approach actually creates the schedule: It assigns the activities to different resources and it determines the starting and completion times of the activities.
3. It needs to present the solution given by the scheduling approach to the user. In PLANWISE this is done by presenting the resulting schedule in a graphical planning board.

The scheduling approach used in the optimizer needs to meet a number of requirements:

1. It should be able to handle certain scheduling situations that occur often in practice. ORTEC wishes to use the scheduling approach to solve scheduling problems that occur at different kinds of customers. Therefore, the scheduling approach should be able to handle several restrictions that occur often in practice. The approach should also be able to optimize schedules based on objectives that occur in practice. In addition, the scheduling approach should be easily adjustable to different scheduling situations. In other words: It should be possible to easily adjust the scheduling approach to a different set of constraints or objectives.
2. A PLANWISE user should be able to use the scheduling approach to develop schedules interactively. Besides generating complete schedules, the approach should thus be able to cope with manual adjustments made by users. The set of activities S that is scheduled manually by the user may not be rescheduled by the scheduling approach: Each activity in S has been assigned to a set of resources. This set may not be adjusted by the scheduling approach. In addition, each activity in S has a fixed starting and completion time; the scheduling approach may not assign different times to the operations in S.

In addition to these requirements, ORTEC considers it an added benefit if a PLANWISE user can also use the optimizer for capacity planning. Sometimes, ORTEC customers wish to roughly determine whether they have sufficient capacity to schedule the set of orders that they have at present. In addition, these customers wish to determine how much capacity they have to accept new orders. To answer these questions, a scheduling approach should be implemented in the optimizer that is able to determine whether a feasible schedule can be found where the set of current orders is scheduled on the available set of resources. In addition, if the approach cannot find a feasible schedule, it should be able to indicate which

resources are bottleneck resources and solve a new scheduling problem that contains a larger number of bottleneck resources. For instance, the scheduling approach may not be able to find a feasible plan, because there is only one machine of type A. The approach should then specify that machine A is the bottleneck resource and it should suggest a schedule where activities are scheduled on more machines of type A.

At present, certain versions of PLANWISE already contain an optimizer. The scheduling approaches used in these optimizers have been developed specifically for a certain customer. Therefore, these approaches are not sufficiently generic to handle different scheduling situations. However, each scheduling approach is based on a random sampling algorithm[1]. ORTEC therefore wishes to know whether an approach based on this algorithm is a suitable approach or whether a different approach should be used. In addition, ORTEC would like to obtain an optimizer based on a generic scheduling approach.

In this assignment, we focus on creating different generic scheduling approaches and comparing these approaches to each other to determine which approach is most suitable to use in the PLANWISE optimizers. During the assignment, we focus on the two requirements that the scheduling approach needs to meet. To limit the scope of the assignment, we do not consider possibilities for using the scheduling approach for capacity planning.

2.2 Research questions

In the previous section, we mentioned that we focus on creating a number of generic scheduling approaches and comparing these scheduling approaches to each other. In addition, we mentioned that we use a scheduling approach to solve a specific scheduling problem. Therefore, we need to perform three main steps in this assignment: First, we need to determine which scheduling problem we wish to solve. Subsequently, we need to develop different scheduling approaches to solve the scheduling problem. Finally, we need to compare the different approaches to each other. In this graduation assignment, we thus need to answer the following research questions:

1. *Which general scheduling problem do we want to solve?*
    a. *Which kinds of scheduling situations do we want to handle with the optimizer?*
    b. *Which objectives and constraints should the approach be able to handle to incorporate the chosen scheduling situations?*

2. *Which scheduling approaches do we use to solve the main scheduling problem?*

3. *How well do the approaches perform?*

2.3 Research approach

In this section, we describe the research approach that we use to answer our research questions. For each research question we mention the activities performed to answer that question.

Determining the general scheduling problem
As we previously mentioned, we first need to determine which scheduling problem we want to solve. To determine this scheduling problem, we determine the scheduling situations at current and potential PLANWISE customers. We do so by interviewing ORTEC employees. During these interviews, we focus on the characteristics of the activities that need to be scheduled and on the kinds of resources on which the activities need to be scheduled.

[1] We describe this random sampling algorithm in Section 6.1

In addition, we focus on relevant scheduling constraints and objectives. Once we have determined the different scheduling situations, we are able to determine which situations occur most frequently and which constraints and objectives are most relevant in these scheduling situations. Based on this information, we define our general scheduling problem.

Developing scheduling approaches for the general scheduling problem
As we mentioned, ORTEC has versions of PLANWISE that contain an optimizer. The scheduling approaches used in these optimizers are all based on a random sampling algorithm. In this assignment, we therefore develop a new scheduling approach based on this algorithm that is able to solve the general scheduling problem. We also develop an alternative scheduling approach to solve the general scheduling problem. To do so, we perform a literature study to determine which general approaches are suitable to solve the general scheduling problem. We then develop the alternative approach based on the results of the literature study.

Evaluating the different approaches
Once we have developed an approach based on random sampling and an approach based on alternative algorithms, we compare both approaches to each other by testing them on a set of scheduling instances. Here, we use the approach based on random sampling as the benchmark approach and we compare the results of the alternative approach to those of the random sampling approach. During this comparison, we predominantly focus on the quality of the resulting schedules.

3. An introduction into scheduling

In this assignment, we focus on creating a scheduling approach to solve scheduling problems. Therefore, in this chapter we define what we mean by the concept 'scheduling' and we introduce the basic scheduling concepts that PLANWISE uses.

In Section 3.1, we first give a theoretical explanation of scheduling: We present our definition of scheduling and mention where scheduling is located in the planning framework. In Section 3.2, we mention how scheduling occurs in PLANWISE and we state some important scheduling concepts that are used in PLANWISE.

3.1 Scheduling in theory

In this research, we use the definition of scheduling as given by Baker (1974). We thus define scheduling as the allocation of resources over time to perform a collection of activities. Baker mentions that the solving of a scheduling problem amounts to the answering of two kinds of questions:

- Which resources will be allocated to perform each activity?
- When will each activity be performed?

In the remainder of this document, we use the term 'assignment' to specify the allocation of resources to activities.

Within the planning framework of Hans et al. (2007), scheduling can be found at the operational level. At this level, several aspects have already been fixed by decisions taken at higher planning levels. Examples of such higher-level decisions are determining demand forecasts and rough predictions for future production mix and volume, and allocating sufficient resources to deal with the incoming demand. At the scheduling level, thus, the nature of the activities to be scheduled has already been described and the configuration of the resources available has already been determined (Baker, 1974). In other words: We consider the activities to be scheduled and the resources on which they can be scheduled as inputs in a scheduling problem.

3.2 Scheduling with PLANWISE

As previously mentioned, we focus on developing a scheduling approach for the optimizer for PLANWISE. In this section, we therefore describe PLANWISE in a bit more detail, focusing in particular on the way that PLANWISE defines activities and resources.

With respect to the activities, we can define a number of different concepts. They are:

- **Orders:** In PLANWISE, an order is defined as a customer order that consists of a set of order lines. Each order consists of a time frame in which the order is to be carried out.
- **Order lines:** Customer orders can be split into different order lines. Each order line specifies the demand for a product that needs to be delivered to the customer and the quantity to be delivered of that product. An order line, for example, may specify that 20 units of type B books must be delivered.
- **Operations:** Each product is made by performing one or more operations. In our example, one book of type B is made by first printing the pages of the book and then binding the pages together. The operations are those that are actually scheduled on the resources; each operation requires a certain amount of time and capacity from the resources.

§ **Resource demands:** Each operation has one or more resource demands. The resource demands specify which subset of the resources is suitable for processing the operation. For each type of resource required, there is a resource demand that specifies which skills (called 'properties' in PLANWISE) the resource must have and at what skill levels. For example, to print pages we require two types of resources: A resource that can print ink on paper and a resource that can service the printing resource (i.e. an operator). We then need to define two resource demands, one for each type of resource required, that specify the skills that the resource must have. For example, the printing resource should be able to print black ink on paper. The resource demands also specify the quantities required of each resource type.

Figure 3.1 shows the relationships between the different concepts just described:

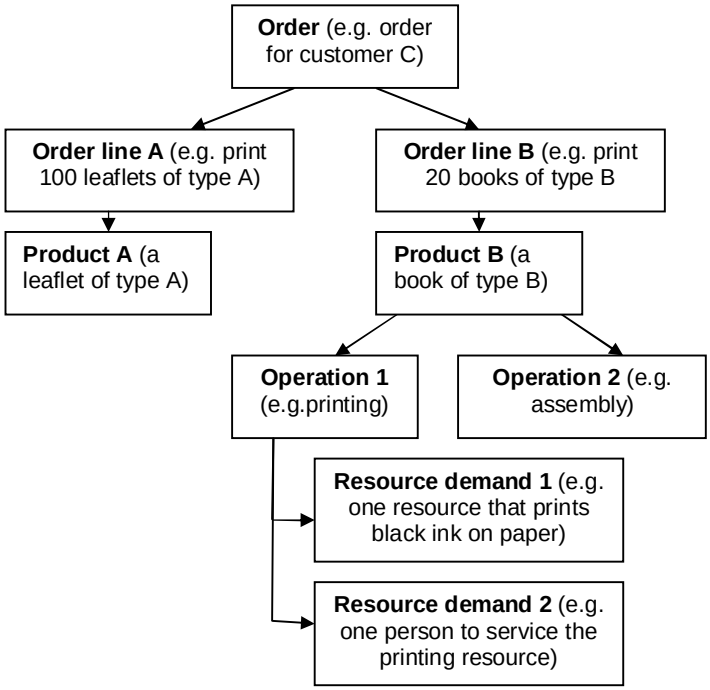

Figure 3.1: Different PLANWISE concepts with respect to activities

As mentioned, the operations are those that need to be scheduled. Each operation needs to be scheduled on a set of resources. Resources are those instruments that can be used to perform different operations. Some examples of resources are machines, employees, tools, and spaces.

Each resource has a certain amount of capacity available during each time period. This available capacity is determined based on the resource's availability (at each point in time a resource can either be available or unavailable) and its capacity (the resource's capacity in the event that the resource is available). We assume that all resources are unique: Each resource thus has a capacity of one in the periods in which it is available. PLANWISE specifies the availability of a resource by using an availability profile. This profile specifies the time intervals in which a resource is available. In addition to available capacity, each resource possesses a number of skills (i.e. properties) at a certain skill level for each property. The skill level specifies whether, and how well, the resource possesses that property. For instance: The skill level may specify whether instructor A is suited to teach a

specific course. In addition, the skill level may also specify that instructors A and B are both suited to teach the course, but that instructor B is better suited to do so than instructor A.

With the available information on the different activities and resources, PLANWISE is able to aid the user in generating a schedule. The user first needs to assign each operation to a set of resources. PLANWISE aids the user by specifying the set of resources on which the operation can be scheduled to satisfy a specific resource demand. PLANWISE does this by comparing the properties and skill levels demanded by that resource demand to those provided by the resource. Once the user has assigned an operation to a set of resources, he must determine a starting and completion time for that operation. Now, PLANWISE aids the user by calculating the earliest possible time that the operation can start. In calculating this time, PLANWISE considers, amongst others, the predecessors of an operation (i.e. the set of operations that must be completed before the operation can start) and the availability profiles of the resources to which the operation is assigned.

In the remainder of this report, we refer to order lines as jobs and to activities as operations. We do so, because scheduling literature often describes scheduling problems in terms of jobs and operations.

4. Scheduling situations

In this chapter, we describe the different scheduling situations encountered at current and possible future PLANWISE clients. Each PLANWISE client has its own particular constraints and characteristics that need to be considered when developing schedules. In addition, each client operates in a different environment with different kinds of operations and resources. In this chapter, we describe these different aspects in more detail. In addition, we describe the scheduling situations for a representative set of 15 PLANWISE customers based on these aspects.

Pinedo and Chao (1999) mention that scheduling problems are often characterized by three aspects: Machine configuration, the processing constraints, and the objectives used for a particular schedule. This scheme is based on the classification scheme introduced by Graham et al. (1979) and revised by Lawler et al. (1982). In addition to these aspects, ORTEC also classifies customers based on the customer's scheduling environment. We now use these four aspects to characterize the scheduling situation at each customer.

In Section 4.1, we first discuss the processing constraints. In Section 4.2, we subsequently discuss the scheduling objectives. We then discuss the other two types of classifications in Section 4.3. Finally, we give conclusions on the scheduling situations in Section 4.4. We then use these conclusions to create the scheduling problem of the next chapter (Chapter 5).

4.1 Processing constraints

When scheduling operations, a scheduling approach needs to consider many different processing constraints. Pinedo and Chao (1999) and Schutten (1996) mention a number of constraints that can be found in scheduling problems. However, since the number of constraints experienced by PLANWISE customers is a more extensive list, we only mention the constraints experienced by these customers.

The following are some of the constraints that are experienced by the PLANWISE clients:

- **Resource capacity:** Each resource has a finite amount of capacity. The capacity of a resource specifies the maximum workload that can be scheduled on that resource at any moment in time. As a result, only a limited number of operations can be scheduled on a resource at any moment. As we mentioned in Section 3.2, we assume that each resource is unique. As a result, at most one operation can be scheduled on a resource at any moment in time.
- **Resource eligibility:** Usually, a resource demand cannot be satisfied by every resource; each resource demand can only be satisfied by a subset of the available resources.
- **Precedence relations:** Precedence relations are those constraints that specify time lags between the start or completion of operations belonging to the same job (De Reyck and Herroelen, 1999). There are four main types of precedence constraints: The Finish-Start (F-S) relations are the most common precedence relations. They specify minimal and maximal time lags between the completion time of an operation and the starting time of a successor. With Start-Start (S-S) relations, the minimal and maximal time lags apply between the starting times of the operations. A similar reasoning applies to Finish-Finish (F-F) relations and Start-Finish (S-F) relations.
- **Multiple resources:** Each operation may require multiple resources to be processed. For instance, we may need a machine and an operator to process an operation. The starting time of the operation now depends on the completion times of preceding operations on all resources to which the operation has been assigned.
- **Transportation times:** Transportation time may apply between two operations belonging to the same job. If this transportation time is fixed between two operations,

it becomes a special case of a F-S precedence relation. Transportation time may also depend on the resources on which the operations are scheduled. In this case, we thus will not know how much transportation time must be scheduled until we have assigned both operations to a set of resources. In some cases, we might require a set R^t of resources to transport a product. For instance, we might need to transport a table top and four table legs to the assembly department, where they will be assembled into a table. To transport the table top and legs we might need a container. To perform the transportation in this case, we need to schedule the transportation on each of the resources in R^t.

- **Personnel constraints:** ORTEC's customers at times have personnel constraints. These constraints are based on laws and collective labour agreements and they specify, amongst others, the maximum number of hours an employee is allowed to work both on a daily and a weekly basis.
- **Resource availability:** Resources are often not continuously available for processing. Schutten (1996) defines the periods in which resources are unavailable as downtime. He distinguishes between two types of downtime: Preemptive and non-preemptive downtime. In the case of preemptive downtime, an operation may start before the downtime and finish after it. In contrast, in the case of non-preemptive downtime each operation needs to be completely processed either before or after the downtime.
- **Changeover times:** Schutten (1996) mentions that a resource may have to be set up before it can process a next operation. As a result, an operation cannot start before the previous operation on the resource has been completed and the setup (or changeover) has been performed. Setups can either be sequence-dependent or sequence-independent. When a setup is sequence-dependent, the length of the setup depends both the operation just completed and the operation to be processed (Pinedo and Chao, 1999). With a sequence-independent setup, in contrast, the length of the setup only depends on the operation being processed next. In some cases, a set of resources R^c is also required to perform the setup. As a result, the setup cannot be performed unless each resource in R^c is available to perform the setup.
- **Use of the same resources:** In addition to eligibility constraints, there may be an additional constraint that all operations belonging to a job must be processed on the same set of resources. A job, for instance, may be to give a group of people a course that consists of several modules (i.e. operations). In this case, we may require that the same instructor (i.e. resource) gives all modules of the course to that group of people.
- **Materials:** Certain operations require input of materials. In addition, certain operations produce material output. The material stock will thus increase and decrease over time. For this reason, checks need to be present to ensure that sufficient materials are present for production and that the material stocks do not exceed a certain maximum.
- **Eligibility between resources:** Eligibility constraints may also apply between different resources. For example, only a subset of employees may be allowed to service a machine. Also, a tool can only be used on a subset of machines.
- **Transfer batches:** Certain customers of ORTEC also work with production and transfer batches. Schutten (1996) defines a production batch as a number of identical products that is processed contiguously on a resource. The transfer batch is defined as the number of items that is transported simultaneously to another resource. In practice, the batch sizes of both kinds of batches do not need to be equal: Often, the transfer batch is smaller than the production batch. In this case it is not necessary to wait until the entire production batch has been completed on the first resource to start with processing on the subsequent resource; it is sufficient to wait for the completion of a transfer batch.

- **Shared resources:** ORTEC's customers sometimes have resources that are shared between different locations. We distinguish two situations: Resources with a fixed location and resources without a fixed location. If a resource has a fixed location (e.g. the main office), a specific amount of transportation time from this location needs to be included in the schedule before the production of an operation can take place. If, the resource does not have a fixed location, however, the transportation time is dependent on the present location of the resource and the location where the operation needs to be performed.
- **Combining and splitting different customer orders:** In some production environments, it might be efficient to combine operations from different jobs in a single production operation O. For example, it may be beneficial to print labels for different customers on the same sheet of paper instead of using a different sheet for each customer. If the operations of different jobs are combined, they must also be split in a later production stage. When creating and splitting a combined operation, the precedence relations must be considered between the combined operation and the other operations that belong to the jobs that have been combined: O may not start until all predecessors of the operations in O have been completed. Similarly, the successors of the operations in O may not start until O has been completed.
- **Waiting time between operations:** Some waiting time may need to be scheduled between the completion of an operation and the start of the operation's successor. Waiting time between operations is a special case of the F-S precedence relation with a minimal time lag: The minimal time lag is equal to the amount of waiting time.
- **Use of different resources for two operations:** This constraint specifies that a set O^d of operations belonging to the same job may not be performed on the same set of resources: Each operation in O^d must be performed on a different set of resources. This constraint may apply, for instance, when a product needs to be developed and tested: Development and testing must then be done by two different people.
- **Routing constraints:** In some environments, the physical routings between the different resources are fixed. The different routings are also known as the organization's factory layout. Routing constraints specify the set of resources that may succeed each other: If, for instance, there is a routing from resource A to resource B, then an operation may be scheduled on resource A and its successor on resource B. Otherwise, this is not possible. These constraints are special cases of the resource eligibility constraints, since they only specify which resources are suitable for an operation. We mention them separately, however, because in this case the eligibility of a resource for an operation depends on the set of resources to which the operation's predecessor is assigned.
- **Resource dependent processing times:** Certain operations can be performed on more than one resource. However, the processing time of the operation may differ depending on the resource to which the operation is assigned. This is modelled in PLANWISE by the use of modes: Each operation can be performed in one or more modes, where the mode determines the operation's processing time and its resource requirements.
- **Product carriers:** When producing certain products, product carriers are sometimes required for the different operations needed to produce the product. For example, when we need to perform maintenance on a ship, we require a maintenance dock (the product carrier) where the ship is kept for the duration of the maintenance activities. ORTEC assumes that product carriers are occupied for the entire duration of production, even when no processing takes place. The maintenance dock in our example is thus unavailable from the moment the ship enters the dock until the moment that all maintenance activities are completed and the ship leaves the dock.

Table 4.1 specifies how often each type of constraint has been mentioned:

Constraint	Number of times mentioned
Resource capacity	15
Resource eligibility	12
Precedence relations	10
Multiple resources	9
Transportation times	6
Use of the same resources	6
Personnel constraints	5
Resource availability	5
Changeover times	4
Materials	3
Eligibility between resources	3
Transfer batches	3
Shared resources	2
Combining and splitting different customer orders	2
Waiting time between operations	2
Routing constraints	2
Use of different resources for two operations	1
Resource dependent processing times	1
Product carriers	1

Table 4.1: Constraints mentioned by (potential) PLANWISE customers

Before we continue with the section on objectives, we want to make one final note with respect to preemption: Pinedo and Chao (1999) mention that in some environments the processing of an operation may be interrupted in favour of a different operation being processed, for example a rush order. ORTEC, however, assumes that an operation is never interrupted once its processing has started. We thus assume that operations are never preempted.

4.2 Objectives

When developing schedules, we can focus on several kinds of objectives. Pinedo and Chao mention (1999) the following objectives as being important basic objectives:

1. *Throughput and makespan objectives:* The throughput of a resource is equivalent to the output rate of that resource. An objective might thus be to maximize this throughput. The makespan, on the other hand, is defined as the time that the last job leaves the system. In other words: The makespan of a schedule is equal to the maximum completion time of the operations in the schedule. Now, the objective is to minimize this makespan. Heuristics that tend to minimize the makespan when there are a finite number of jobs also tend to maximize the throughput rate when the flow of jobs is constant over time (i.e. when the release and due dates of the jobs are spread evenly over the planning horizon. This in contrast to a situation where the jobs are clustered together).
2. *Due date related objectives:* Several objectives are related to due dates. Some examples of due date related objectives are minimizing the maximum lateness, minimizing the number of tardy jobs, and minimizing the total weighted tardiness
3. *Objectives related to production costs:*
    a. Setup costs: The related objective here is to minimize the setup costs. Note that setup costs are not necessarily proportional to setup times. So minimizing setup costs may lead to different schedules being chosen than when setup time is minimized.

b. Work-in-Process inventory costs: The objective here is to minimize WIP inventory. Some reasons to minimize this inventory are that the inventory ties up capital and increases handling costs. Large amounts of WIP can also clog operations. Certain performance measures, such as the average throughput time (the average time it takes for a job to traverse the system), can be used as a surrogate for WIP.
c. Finished Goods inventory costs: The objective here is to minimize the inventory costs of the finished goods.
d. Personnel costs: Costs are associated with the assignment of a particular shift to a particular time slot. The objective is to minimize the total costs associated with certain shift patterns. An important factor to consider here is the costs required for overtime: These costs are often significantly higher than the costs of regular time.

Pinedo and Chao also mention other objectives, such as the minimization of the sum of the earliness (which could be an objective in a Just-in-Time environment, where a job should not be completed until just before its due date) or generating schedules that are as robust as possible (in a robust schedule, necessary changes that need to be made as a result of a disruption are minimal).

When developing schedules, PLANWISE customers focus on a number of different objectives. Some of these objectives are common and can be found in the list of basic objectives. In addition, customers often also have their own specific objectives. We now mention the most common types of objectives used by PLANWISE customers:

1. *Due date related objectives:* The most frequently used objectives are due date related objectives. All customers have as most important goal to develop schedules where jobs are scheduled on or before their due dates. However, this goal translates into different objective functions for different customers. In equipment scheduling environments (equipment scheduling is mentioned in Section 4.3), for instance, jobs need to take place a specific moments. When developing schedules, a job is thus either on time or it cannot be scheduled at the required time. In these environments, the focus therefore is on maximizing the number of jobs scheduled (and thus on time). In other environments, however, the focus was on minimizing the total (weighted) tardiness of the jobs.
2. *Objectives related to the use of resources:* Several PLANWISE customers also focus on how resources are used, but they consider these goals to be of less importance than the due date related goals. In this category, customers again have mentioned several objective functions. The most frequently mentioned objectives are:
    a. Minimizing the number of used resources: Customers want to minimize the number of resources used to perform the operations.
    b. Maximizing the number of preferred resources used: The preferred resource of resource A is that resource that, if available, must be used in combination with resource A. Often, the preferred resource is an operator. This operator is then the preferred person to operate machine A. Preferred resources can also apply for a certain job. If, for instance, customer B is used to having engineer A, then engineer A should be scheduled for jobs from customer B as much as possible.
    c. Balancing the workload as much as possible over the different resources: Customers sometimes wish to balance the total workload as much as possible over different resources. This objective often applies to human resources: A company, for instance, may want to spread the workload over different operators. In addition, certain companies wish to schedule the operations in one department in such a way that the next department has a more balanced workload. For instance, one company wishes to schedule operations in the

production department in such a way that the workload is evenly balanced over time in the assembly department.

4.3 Additional classifications of scheduling situations

In this section, we describe the possible machine configurations and scheduling environments that (potential) PLANWISE customers could have. In scheduling literature, a scheduling problem is partially characterized by the machine configuration used. ORTEC also classifies the scheduling situations at its customers by describing the scheduling environment at the customer.

Machine configurations

The machine configuration specifies the machine characteristics of a problem. Based on the classification scheme by Graham et al. (1979) and Lawler et al. (1982), we can specify five basic machine configurations:

1. *Single machine models:* In these models there is a single resource on which a set of operations needs to be scheduled. These models can, naturally, be used in systems with only one resource. However, they can also be used, for instance, to model a single bottleneck resource in a multi-resource environment. Since the operation sequence at the bottleneck typically determines the performance of the entire system, it might be helpful to first schedule operations on the bottleneck resource and then schedule operations on the other resources.
2. *Parallel machine models:* In these models, we have a workstation that consists of a number of resources in parallel. We may, for example, have a printing area consisting of a number of printers. When operations arrive at the workstation, they need to be processed on one of the resources of that workstation. The resources at a workstation may be identical, thus enabling an operation to be processed on any one of the resources. It may also occur that the resources are not identical. In this case, certain operations may only be processed on a subset of the resources. In addition, the processing time of the operation may vary depending on the chosen resource. The single machine model previously described is a special case of the parallel machine model: The workstation then consists of only one resource.
3. *Flow shop models:* In flow shop models, we have jobs consisting of one or more operations. Each operation needs to be processed on a different resource. However, the route for each job is identical. The operations thus need to be scheduled in a fixed sequence. The flow shop is a special case of the flexible flow shop, which consists of a number of stages in series with a number of resources in parallel at each stage. At each stage, an operation needs to be processed on one of the parallel resources.
4. *Job shop models:* The flow shop model is also a special case of the job shop model. As in a flow shop, the operations of a job need to be performed in a certain sequence. However, in a job shop, the sequence in which the operations of a job need to be processed depends on the job (this in contrast to the flow shop, where the sequence is fixed for all jobs). Job shops, in turn, are special cases of the flexible job shop, where work centres have multiple resources in parallel. Each operation now needs to be scheduled on one of the parallel resources.
5. *Open shop models:* In open shops, the operations of a job can be processed in any order. As a result, no precedence relations exist between the different operations of a job. The operations of a job, however, cannot be processed simultaneously.

The machine configurations just described are the most basic configurations. In practice, however, we do not encounter any PLANWISE customers that have these basic configurations, because these customers often have additional practical constraints that characterize their scheduling situations. In the basic configurations, for instance, each operation only requires one resource to be processed. In practice, we often require multiple

resources to perform an operation. In addition, two operations belonging to the same job may be performed simultaneously in practice if the one operation is not a (direct or indirect) predecessor or successor of the other operation. In the basic configurations, however, at most one operation of a job may be performed at any moment in time.

The scheduling situations at the PLANWISE customers thus differ from the basic configurations described. Nevertheless, we are still able to specify which basic configuration most resembles the scheduling situation at that customer. Table 4.2 specifies the machine configurations for the set of PLANWISE customers:

Machine configuration	Number of customers
Single machine	0
Parallel machine	4
Flow shop	0
Flexible flow shop	4
Job shop	1
Flexible job shop	6
Open shop	0

Table 4.2: The machine configurations of PLANWISE customers

As we can see, the machine configurations most frequently encountered are those where operations can be performed on different resources. For many PLANWISE customers, an important part of scheduling is thus the assignment of operations to resources: Users need to choose a resource for an operation before they can give the operation a start and completion time.

Brucker and Thiele (1996) mention that all basic machine configurations are special cases of the general-shop problem. In a general shop problem, the operations of a job do not need to be performed in a sequence: There can be arbitrary precedence relations between any pair of operations. As a result, it may occur that several operations need to be completed before a certain operation can start. Similarly, an operation can also have multiple successors.

Scheduling environments

ORTEC distinguishes three kinds of scheduling environments based on the different kinds of customers that it has. We distinguish the different environments based on the main resource types scheduled in the environments. In addition, each environment has its own specific constraints:

- *Task scheduling:* With task scheduling, the general idea is to assign operations to persons with the right qualifications and competences (i.e. the right properties). The use of machines and equipment plays a limited role in this kind of scheduling. Eligibility constraints, precedence relations, and resource-dependent processing times (i.e. a more experienced person can perform an operation faster than a less experienced person) are some of the main constraints in task scheduling problems.
- *Equipment scheduling:* The idea of equipment scheduling is to schedule operations on equipment (e.g. a crane) and the resources related to this equipment, such as tools and equipment operators. The main constraints that play a role in task scheduling also apply in equipment scheduling problems. In addition to these restrictions, an important aspect of equipment scheduling is the fixed time intervals in which a job needs to take place: When a customer calls to use a piece of equipment, he specifies a time interval in which he requires the equipment. For instance, the customer wishes to have a crane at his disposal from 2 pm to 6 pm on Thursday. Since each job must be done in a specific time interval, the release date of a job is equal to the start of the interval and the due date of a job is equal to the end of the interval. As a result, jobs in an equipment environment do not have any slack: Each job must start on its release date and it must end on its due date. The starting and

completion times for the operations of a job are thus fixed as well. Since we do not need to determine starting and completion times for the operations of a job, we only need to assign the operations to a set of resources to solve the equipment scheduling problem.

- *Production scheduling:* This is the most detailed scheduling environment. Production scheduling entails the scheduling of operations on several kinds of resources, such as machines, tools, and people. In this kind of scheduling, several additional factors need to be considered that do not play a role in the other kinds of scheduling environments. Examples of such factors are the use and storage of materials, changeovers, and layout constraints. Task scheduling is a special case of production scheduling: The restrictions that apply in task scheduling also apply in production scheduling.

Table 4.3 specifies the scheduling environments in which PLANWISE customers operate:

Scheduling environment	Number of customers
Task scheduling	6
Equipment scheduling	4
Production scheduling	5

Table 4.3: Scheduling environments of PLANWISE customers

4.4 Conclusions

In the previous sections, we have described the scheduling situations at different PLANWISE customers by looking at the constraints that customers need to deal with, the objectives that they wish to optimize, and their machine configurations and scheduling environments. In this section, we use the information from the previous sections to determine which constraints and objectives we need to include in our scheduling problem.

With respect to the different scheduling environments, we notice that PLANWISE customers are divided equally over the different scheduling environments. When creating the general scheduling problem, we choose to focus on the production scheduling environment. As we mentioned before, task scheduling is a special case of production scheduling. Therefore, a scheduling approach that solves a production scheduling problem will also be suitable to solve a task scheduling problem. We do not focus on creating an equipment scheduling problem, because this environment differs in a number of ways from the production scheduling environment: As we mentioned before, in the equipment scheduling environment each operation has a specific starting and completion time. As a result, an equipment scheduling problem only consists of assigning a set of operations to a set of resources. Therefore, we might require a different approach to solve an equipment scheduling problem compared to a production scheduling problem. An equipment scheduling problem also requires a different objective function from a production scheduling environment. Since the production and task scheduling environments occur more frequently among the PLANWISE customers, we choose to create a production scheduling problem.

We also consider the machine configurations of the customers in determining which constraints need to be included in the general scheduling problem. As we can conclude from Table 4.2, the basic configurations that occur most frequently are the parallel machine configuration, the flexible flow shop and the flexible job shop. We choose to focus on the flexible job shop, because the other two types of configurations are special cases of the flexible job shop.

We thus wish to create a production scheduling problem that is able to handle flexible shops. As we mentioned before, a flexible shop contains jobs, where each job consists of one or more operations that need to be performed in a certain sequence. Each operation requires one resource to be performed and this resource must be chosen from a set of suitable resources. Therefore, the general scheduling problem that we consider in the remainder of this assignment must contain at least two basic constraints: Precedence relations and eligibility constraints. The precedence relations specify the predecessors and successors of each operation and the eligibility constraints specify which subset of resources is suitable to schedule each operation on. Since we want to have a practical scheduling problem, we choose to specify general Finish-Start (FS) precedence relations in the general scheduling problem instead of the precedence relations used in the flexible job shop. As a result, each operation has a set of predecessors and a set of successors. In addition, we allow two operations of a job to be performed simultaneously if the operations are not related in any way (i.e. the one operation is not a (direct or indirect) predecessor or successor of the other operation).

The general scheduling problem that we consider should thus contain general precedence relations and eligibility constraints. In addition to these restrictions, we also include an additional set of constraints in the scheduling problem. As we can see from Table 4.1, certain practical restrictions are experienced by several PLANWISE customers. Therefore, we choose to include them in the general scheduling problem. In addition, certain restrictions are also interesting to include, because ORTEC thinks that these restrictions could be important restrictions for PLANWISE customers in the future.

In total, we include the following restrictions in the general scheduling problem:

1. Precedence relations
2. Resource eligibility constraints
3. Resource capacity restrictions
4. Multiple resources
5. Resource availability constraints
6. Changeover times
7. Resource dependent processing times
8. Routing constraints

With respect to objectives, we noticed that customers have mentioned several different types of objectives. Of these different objectives, customers have mentioned the due date related objectives most frequently. In addition, they also consider these objectives to be the most important objectives to focus on. Therefore, we focus on these types of objectives when solving the general scheduling problem.

When we compare the different due date related objectives, we see that customers predominantly focus on minimizing the (weighted) tardiness of the jobs or minimizing the number of tardy jobs. Tardiness minimization has been mentioned more frequently by customers in production and task scheduling, whereas the minimization of tardy jobs has been mentioned most frequently by customers in equipment scheduling. Since we focus on a production scheduling environment, we choose to minimize the tardiness of the jobs in the general scheduling problem.

5. The general scheduling problem

In Chapters 3 and 4 we introduced scheduling problems in general and we described different scheduling situations that we may encounter in practice. Based on these chapters, we now define the general scheduling problem that we consider in the remainder of the assignment.

In this chapter, we also discuss two other topics: The description of a solution for the scheduling problem and the critical set of an operation. We discuss these topics here, because we frequently use them in subsequent chapters.

We discuss the main scheduling problem in Section 5.1. Subsequently, we discuss the description of a solution in Section 5.2. Finally, we discuss the critical set of an operation in Section 5.3.

5.1 The general scheduling problem

For our general scheduling problem we consider a modified general shop problem. The problem can be formulated as follows:

We have n jobs $J_1, J_2, ..., J_n$, each with a release date r_j and due date d_j. Each job J_j consists of a set of operations $O_{1j}, O_{2j}, ..., O_{n_j,j}$. Here, n_j denotes the number of operations of job J_j. All operations are non-preemptive: Once an operation starts, it may not be interrupted in favour of another operation.

Finish-Start precedence relations may apply between any two operations of a job. These precedence relations specify the sequence in which the operations of a job need to be performed: If a precedence relation applies from operation O_{ij} to operation O_{hj} $\left(j = 1,2,...,n; i, h = 1,2,...,n_j\right)$, then O_{ij} must be completed before O_{hj} can start. Since O_{ij} must be completed before O_{hj} can start, we define O_{ij} to be a job predecessor of O_{hj}. Similarly, we define O_{hj} to be a job successor of O_{ij}. Nonnegative minimal time lags apply between the completion of an operation and the start of a job successor of that operation (or, similarly, between a job predecessor of an operation and that operation).

We also have a set R of unique resources on which the operations need to be scheduled. Each resource works according to a calendar that specifies the periods in which the resource has preemptive downtime. In addition, each resource may also have periods of non-preemptive downtime.

Each operation O_{ij} can be performed in n_{ij} modes. For each mode V_{ijm} $(m = 1,2,...,n_{ij})$, the operation has a processing time p_{ijm} and a set of resource demands $U_{ijm1}, U_{ijm2}, ..., U_{ijmn_{ijm}}$ that need to be satisfied (n_{ijm} denotes the number of resource demands belonging to operation O_{ij} when executed in mode V_{ijm}). Each resource demand U_{ijmk} $\left(j = 1,2,...,n; i = 1,2,...,n_j; m = 1,2,...,n_{ij}; k = 1,2,...,n_{ijm}\right)$ requires a_{ijmk} suitable resources.

For each resource demand U_{ijmk}, there is a subset R_{ijmk} $(R_{ijmk} \subset R)$ that specifies the resources that are suitable for satisfying that resource demand. Since all resources are unique, the resource demand U_{ijmk} must be satisfied by a_{ijmk} different resources from R_{ijmk}.

For the given scheduling problem, three different types of decisions have to be made:

- For each operation O_{ij}, we need to choose one mode V_{ijm^*} in which the operation will be performed.
- For each resource demand U_{ijm^*k} corresponding to the chosen mode V_{ijm^*}, we must assign a_{ijm^*k} suitable resources to operation O_{ij}.
- For each operation O_{ij} we need to determine a starting time S_{ij}. The completion time C_{ij} of the operation follows from the starting time, the processing time p_{ijm^*}, and the preemptive downtimes on the resources assigned to this operation. In Section 5.2, we provide more detailed information on the way that starting and completion times are calculated.

Constraints

In the scheduling problem we consider the following constraints:

1. *Precedence relations:* As we mentioned before, Finish-Start (F-S) precedence relations may apply between any two operations of a job. Between operation O_{ij} and its job successor O_{hj} the following constraint thus applies: $C_{ij} + lag_{ihj}^{fs\min} \leq S_{hj}$. Here, $lag_{ihj}^{fs\min}$ denotes the minimal time lag. Note that we only model minimum time lags: We do not consider maximum time lags in this problem. If operations O_{ij} and O_{lj} are not related to each other (i.e. the one operation is not a (direct or indirect) job predecessor or successor of the other operation), the operations may be performed simultaneously.
2. *Resource capacity constraints:* As previously mentioned, each resource is unique. Therefore, at most one operation can be scheduled on a resource at any moment in time.
3. *Resource availability constraints:* As we previously mentioned, there can be periods of both preemptive and non-preemptive downtime on each resource. The preemptive downtimes on a resource are specified by the calendar of the resource and they often occur on a regular basis. For instance, the resource's calendar may specify that the resource is always unavailable during the weekend. Operations may be scheduled over preemptive downtimes: An operation may thus start before a preemptive downtime and finish after the downtime. In addition to preemptive downtimes, a resource can also have incidental periods of non-preemptive downtime. An example of incidental downtime is when a resource is unavailable because preventive maintenance is being performed on that resource. Operations may not be scheduled over non-preemptive downtimes: Each operation must be scheduled either entirely before the downtime or entirely after the downtime.
4. *Changeovers:* Z_{ijhlr} units of changeover time need to be scheduled between the completion of operation O_{ij} and the start of operation O_{hl} if O_{hl} is scheduled directly after O_{ij} on resource R_r, i.e. in this case the starting time of O_{hl} must be at least the completion time of O_{ij} plus Z_{ijhlr}.
5. *Routing constraints:* We have a set R^F of forbidden resource pairs. If the pair $\{R_r, R_v\}$ $(R_r, R_v \in R)$ is included in the set R^F, it is forbidden to schedule an operation O_{ij} on resource R_r and a job successor O_{hj} of O_{ij} on R_v.

6. *Fixed operations:* We have a set O^{fixed} of fixed operations. Each operation O_{ij} in O^{fixed} has already been scheduled: The operation has been assigned to a set of resources and it has been given a starting time S_{ij} and completion time C_{ij}. These operations may not be rescheduled: They may not be reassigned to different resources and their starting and completion times may not be changed. Each operation in O^{fixed} needs to be treated as a non-preemptive downtime on the set of resources to which the operation is assigned. In other words: Any other operations scheduled on the same resources must be scheduled either entirely before, or entirely after, the fixed operation. Note that precedence relations may apply between a fixed operation and the other operations belonging to the same job.

Objectives

Each job has a certain tardiness value T_j that specifies the amount of time by which the completion time of the job exceeds its due date. Let $C_j = \max_i(C_{ij})$ be the completion time of job J_j. The tardiness of job J_j is then determined as $T_j = \max(C_j - d_j, 0)$.

In this scheduling problem, the objective is to minimize the total tardiness of the jobs, i.e. $\min \sum_j T_j$.

Before we proceed with the next section, we give a number of additional comments with respect to the scheduling problem:

- In the scheduling problem, we have changeover time Z_{ijhlr} between two adjacent operations O_{ij} and O_{hl} on resource R_r. In Section 4.1, we mentioned that in practice we sometimes require a set of resources R^C to perform the changeover. For instance, we may require an operator to perform the changeover. In this scheduling problem, however, we do not consider changeovers with resource requirements. In other words: The set R^C is always empty.
- In addition to the constraints previously specified, we also implicitly consider the following set of constraints in the general scheduling problem:
    - *Eligibility constraints:* We incorporate eligibility restrictions in the problem when we specify the set of resources included in the subset R_{ijmk}. The resources included in the subset are suitable resources for resource demand U_{ijmk}, whereas the remaining resources are not suitable.
    - *Multi-resource operations:* Since we are able to specify operations with more than one resource demand, we are able to incorporate multi-resource operations in the scheduling problem.
    - *Resource-dependent processing times:* In the scheduling problem we are able to specify multiple modes in which an operation can be performed. For each mode, we can specify a different processing time. As a result, we can create resource-dependent processing times by specifying a different mode for each production speed in which a resource can operate. Subsequently, by including only those resources that operate on the right speed in the subset R_{ijmk}, we ensure that an operation is only assigned to the resources that actually operate at that specific speed.

5.2 Characterization of a solution for the scheduling problem

In the previous section, we described the general scheduling problem. There, we mentioned that we need to make three different types of decisions:

1. Determining the mode in which each operation is performed.
2. Determining the resources on which each operation is performed.
3. Determining a starting and completion time for each operation.

We thus obtain a solution for our scheduling problem by making three choices for each operation:

1. The mode in which the operation is performed.
2. The set of resources to which the operation is assigned. The chosen set of resources depends on the mode in which the operation is performed, because the mode determines the set of resource demands for the operation.
3. The position that the operation has on each resource to which it is assigned. The positions of the operations on a resource specify the sequence in which those operations are performed on that resource. If an operation has position 5, we thus know that the operation is the fifth operation in the sequence on that resource. Based on the position of the operation, we can determine which operations need to be completed before the operation can be scheduled: For instance, if operation O_{ij} has position 5 on a resource, we know that we need to schedule the operations on positions 1 to 4 on the resource before operation O_{ij}. Since the operation only obtains a position on the resources to which it is assigned, we need to know to which set of resources an operation has been assigned before we can assign positions to the operation on these resources.

Once we have specified these issues for each operation, we are able to create a schedule: Since we have assigned each operation to a set of resources, we are able to determine which operations are scheduled on a specific resource. In addition, we can determine the sequence in which the operations are performed on each resource by looking at the positions that operations have on that resource: We first schedule the operation with position 1 on the resource, subsequently we schedule the operation with position 2, and so forth.

Once we know the assignment of operations to resources and the sequence in which operations are performed on each resource, we can assign starting and completion times to each operation O_{ij} by finding the first time interval in which O_{ij} can be scheduled. Naturally, this time interval must be after the completion time of the job predecessors and resource predecessors of O_{ij}. In addition, we must ensure that the set of restrictions in Section 5.1 are satisfied: For instance, there must be sufficient time between the start of O_{ij} and the completion of a job predecessor to incorporate the minimal time lag between the two operations. Similarly, there must be sufficient time to perform a changeover between O_{ij} and each of its resource predecessors. In addition, the preemptive and non-preemptive downtimes on resources must be considered when determining both the start and the duration of the interval.

We characterize a solution q to the general scheduling problem as follows: $q = (M_q, A_q, W_q)$. We now describe each of the elements of which q is comprised:

1. We have a function $M_q(O_{ij})$ that assigns a mode to operation O_{ij} in solution q. This means that M_q specifies the mode assignment of solution q.

2. We have a function $A_q(O_{ij}, M_q)$ that specifies a set of resources for each resource demand U_{ijmk} that O_{ij} has when it is performed in the mode obtained through $M_q(O_{ij})$. This means that A_q specifies for each operation in q the set of resources assigned to that operation to satisfy the resource demands that the operation has corresponding to the mode assigned to the operation in M_q.
3. We have a function $W_q(O_{ij}, M_q, A_q)$ that specifies for each resource R_r assigned to O_{ij} by $A_q(O_{ij,} M_q)$ the position on which O_{ij} is processed on R_r. This means that W_q specifies for each resource $R_r \in R$ the sequence in which the operations assigned to R_r in q are scheduled.

Once the mode assignment, the resource assignment, and the sequence are fixed for each operation, we can develop a schedule. Let $P(O_{ij})$ specify the direct predecessors of operation O_{ij}. In other words: The set $P(O_{ij})$ contains the operations that must be scheduled before O_{ij} can be scheduled. $P(O_{ij})$ consists of two kinds of operations:

a) The job predecessors of O_{ij}.
b) The resource predecessors of O_{ij} on each resource R_r in $A_q(O_{ij}, M_q)$. The resource predecessor of O_{ij} on R_r is that operation that has a position directly in front of operation O_{ij} on R_r.

Once $P(O_{ij})$ is known for each operation, we can actually assign starting and completion times to the operations. We do so by repeating the following steps until all operations have been scheduled:

1. Choose an unscheduled operation O_{ij} for which the set $P(O_{ij})$ is empty or for which all direct predecessors have already been scheduled.
2. Calculate the earliest possible starting time ES_{ij} for O_{ij} by only considering the predecessors of O_{ij}. This time must be at least r_j, the release date of the job to which O_{ij} belongs. For $O_{hl} \in P(O_{ij})$ let $ES(O_{hl})_{ij}$ be the earliest possible starting time of O_{ij} with respect to only O_{hl}. If O_{hl} is a job predecessor (i.e. $l = j$), then $ES(O_{hl})_{ij}$ is equal to C_{hl} plus the minimal time lag between O_{hl} and O_{ij}. If O_{hl} is a resource predecessor, then $ES(O_{hl})_{ij}$ is defined equal to C_{hl}. ES_{ij} then can be calculated as $\max\{r_j, \max_{O_{hl} \in P(O_{ij})} ES(O_{hl})_{ij}\}$. Note that we do not include any changeover time in $ES(O_{hl})_{ij}$. We consider the changeover times in step 3.
3. Determine the actual starting and completion time of O_{ij}:
    a. Create an *aggregate availability profile* that specifies the intervals of downtime, both preemptive and non-preemptive, present on each of the resources in $A_q(O_{ij}, M_q)$. Based on this profile, we can determine the intervals of time in which all resources are available. We create the aggregate availability profile by considering the individual availability profiles of each resource in $A_q(O_{ij}, M_q)$. The availability profile for a resource specifies the downtimes, both preemptive and non-preemptive, present on the resource.

The non-preemptive downtimes can either be incidental downtimes or fixed operations.

b. Determine the earliest time interval $[s_{ij}', c_{ij}']$ after ES_{ij} that does not include a non-preemptive downtime on any of the resources in $A_q(O_{ij}, M_q)$ in which all resources $R_r \in A_q(O_{ij}, M_q)$ have p_{ijm} units of *simultaneous* uptime. The starting time S_{ij} of O_{ij} is then equal to the start s_{ij}' of the interval and the completion time C_{ij} is equal to the end c_{ij}' of the interval. By considering the aggregate availability profile, we can determine when the resources in $A_q(O_{ij}, M_q)$ have simultaneous uptime. When determining the earliest interval, we must consider two factors:
   i. There must be sufficient uptime before the time interval to schedule a possible setup between O_{ij} and the previous operation on that resource (either a fixed operation or the resource predecessor of O_{ij} on the resource).
   ii. There must be sufficient uptime after the time interval to schedule a possible setup between O_{ij} and the next fixed operation on the resource (since we only schedule an operation once all its predecessors have been scheduled, it is not possible to have a setup between O_{ij} and a resource successor of O_{ij}).

For clarity, we now present an example to demonstrate how to calculate starting and completion times of operations:

We have two operations, O_{11} and O_{22}, that both need to be scheduled on an operator and a machine. Operation O_{11} has no predecessors and can start at time 0. Operation O_{22} has two predecessors: Resource predecessor O_{11} and a job predecessor. Operation O_{22} can start at time 6 if we only consider its job predecessor. The processing time of both operations is 1 day.

Both the operator and the machine have periods of preemptive and non-preemptive downtime. In Figure 5.1, we present the availability profiles of both resources. The timeline is given in days in the figure:

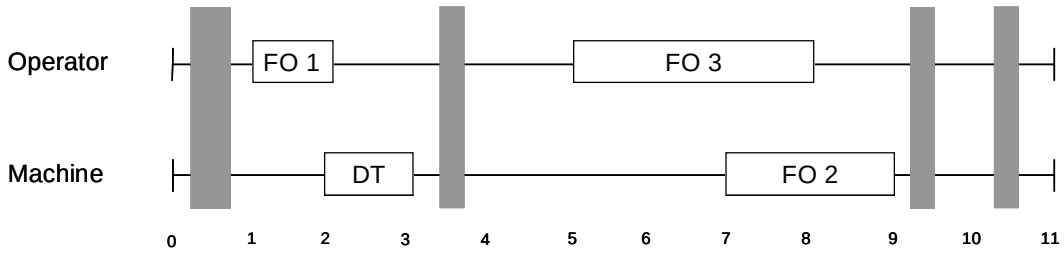

Figure 5.1: An example to demonstrate the calculation of starting and completion times for operations.

In Figure 5.1, the periods of preemptive downtime are marked in dark grey. The first preemptive downtime has a duration of 0.5 and is over the interval [0.25, 0.75], whereas the other three downtimes have a duration of 0.25 each and are over the intervals [3.25, 3.5], [9.25, 9.5], and [10.25, 10.5]. The four blocks on the different resources indicate the non-preemptive downtimes: On the operator, we have two fixed operations and on the machine we have one period of actual downtime (indicated with DT) and one fixed operation (FO 2).

We schedule operations O_{11} and O_{22} as follows:

- *We first schedule operation O_{11}:*
  2. *Since operation O_{11} does not have any predecessors, ES_{11} is equal to 0.*
  3. *In Figure 5.1, we can see that there are three time intervals where neither resource contains a non-preemptive downtime. These intervals are [0,1], [3,5], and [9,11]. We cannot schedule O_{11} in [0,1], because the uptime in this interval is 0.5, which is smaller than the processing time of the operation. In interval [3,5], we have a total uptime of 1.75, which is greater than the processing time of O_{11}. However, we must also consider if we have sufficient time to perform setups between FO 1 and O_{11}, between O_{11} and FO 3, and between O_{11} and FO 2. Assuming that no setups are required, then the earliest interval in which we can schedule O_{11} is [3,4.25]. This interval consists of the processing time of O_{11} and the non-preemptive downtime of 0.25. O_{11} obtains a starting time S_{11} of 3 and a completion time C_{11} of 4.25.*
- *Since we have scheduled O_{11} and the job predecessor of O_{22}, we are now able to schedule O_{22}:*
  2. *ES_{22} is max(4.25, 6), so O_{22} cannot start before time 6.*
  3. *The only possible interval in which O_{22} can be scheduled, is the interval [9,11]. If we assume that a setup is required between FO 2 and O_{22} of 0.125, then the earliest interval in which we can schedule O_{22} is [9.125, 10.625]. O_{22} obtains a starting time S_{22} of 9.125 and a completion time C_{22} of 10.625.*

In Figure 5.2, we can see the resulting schedule. The scheduled operations are indicated by the light grey boxes. The striped box between FO 2 and O_{22} is the setup between the two operations:

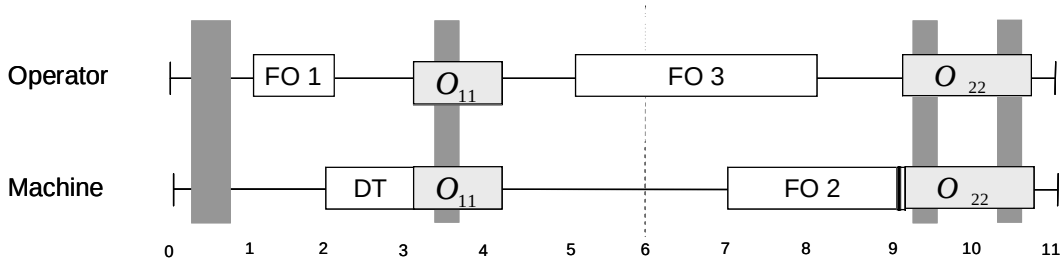

Figure 5.2: The resulting schedule

Note that we only construct schedules that are semi-active: Given a mode assignment, a resource assignment, and a sequence, each operation starts as early as possible.

5.3 The critical set of an operation

In the remainder of this document, we frequently work with the critical set of an operation in a schedule. In this section, we therefore describe what we mean by a critical set and we give some definitions that we use often. We first give an intuitive definition of the critical set. Subsequently, we give a more formal definition.

Intuitively, we define the critical set of an operation O_{ij} as the set of relevant operations that prohibit O_{ij} from being scheduled at an earlier time. There are two kinds of operations in the critical set of O_{ij}:

1. Critical predecessors of O_{ij}: A critical predecessor is an operation that directly determines the earliest starting time of operation O_{ij}. Each critical predecessor is either a job predecessor or a resource predecessor of O_{ij}.
2. Operations that directly or indirectly determine the earliest starting time of a critical predecessor of O_{ij}.

We determine the critical predecessors of O_{ij} as follows: For each (job or resource) predecessor O_{hl} of O_{ij}, we determine the earliest possible time $ET(O_{hl})_{ij}$ that O_{ij} can start if we only consider O_{hl}. For job predecessors we use a different way to calculate $ET(O_{hl})_{ij}$ than for resource predecessors:

- For each job predecessor $O_{hl}(l = j)$, $ET(O_{hl})_{ij}$ is equal to the completion time of O_{hl} plus the minimal time lag between O_{hl} and O_{ij}.
- For each resource predecessor O_{hl}, we determine whether there are any non-preemptive downtimes between the completion time C_{hl} of O_{hl} in the schedule q and the starting time S_{ij} of O_{ij} in q. If so, $ET(O_{hl})_{ij}$ is equal to the end time of last non-preemptive downtime in the interval $[C_{hl}, S_{ij}]$ plus the setup time, if any, between the downtime and O_{ij} (the setup time may apply if the downtime is a fixed operation). If there are no non-preemptive downtimes in the interval $[C_{hl}, S_{ij}]$, then $ET(O_{hl})_{ij}$ is equal to the completion time of O_{hl} plus possible setup time between O_{hl} and O_{ij}.

Once we have determined $ET(O_{hl})_{ij}$ for each predecessor, we determine $\max\{r_j, \max_{O_{hl} \in P(O_{ij})} ET(O_{hl})_{ij}\}$, which specifies the earliest possible starting time for O_{ij} if we only consider the predecessors of the operation and the release date of the job to which O_{ij} belongs. We define $O_{hl} \in P(O_{ij})$ to be a critical predecessor of O_{ij} if and only if $ET(O_{hl})_{ij}$ is equal to this value. We now determine the critical set $COS(O_{ij})$ of operation O_{ij} recursively as the set of all its critical predecessors plus the critical set of these critical predecessors.

We now clarify the steps to determine $COS(O_{ij})$ by giving some examples.

For the first example, given in Figure 5.3, we use the following simple schedule consisting of three jobs. A precedence relation applies between operation O_{11} and its job successor O_{21}. Similarly, there is a precedence relation from O_{12} to O_{22} of job J_2. Operation O_{13} does not have any predecessors and it has the release date of job J_3 as starting time (indicated by the striped line in the figure).

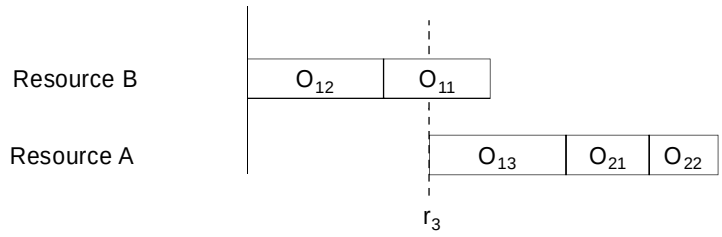

Figure 5.3: An example schedule

In Figure 5.3, $COS(O_{22})$ is the set $\{O_{13}, O_{21}\}$. Operation O_{22} has two predecessors: Job predecessor O_{12} and resource predecessor O_{21}. When comparing the two predecessors, we see that O_{21} has the latest completion time. Therefore, O_{21} is the only critical predecessor of O_{22}. Similarly, operation O_{21} has one critical predecessor: Resource predecessor O_{13}.Since O_{13} does not have any predecessors, we are done: $COS(O_{22})$ consists of operations O_{21} and O_{13}.

As we can see in the example, operations O_{12} and O_{11} are not a part of $COS(O_{22})$, because neither operation influences the earliest starting time of O_{22} or any operation on the critical set of O_{22}.

In a second example, given in Figure 5.4, we have a schedule that is similar to the first schedule. As in the example in Figure 5.3, we have precedence relations between the operations of jobs J_1 and J_2. However, in this example we also have a minimal time lag between operations O_{11} and O_{21}. The duration of the minimal time lag is given by the length of the arrow in Figure 5.4.

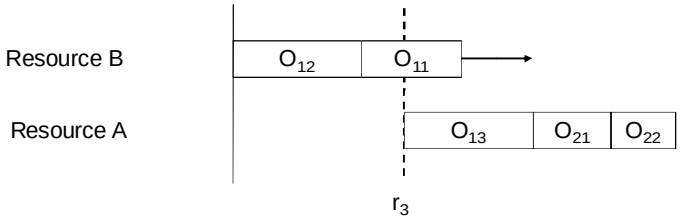

Figure 5.4: A second example schedule

In the example of Figure 5.4, $COS(O_{22})$ is the set $\{O_{12}, O_{11}, O_{13}, O_{21}\}$: As in Figure 5.3, O_{21} is the only critical (resource) predecessor of O_{22}. In this case, however, O_{22} has both O_{11} and O_{13} as critical predecessors, because $ET(O_{11})_{21}$ is equal to $ET(O_{13})_{21}$. Operation O_{11} only has one predecessor, operation O_{12}, which is also the only critical predecessor of O_{11}. Since both operations O_{12} and O_{13} do not have any predecessors, we have found all operations belonging to $COS(O_{22})$.

Finally, we include an example to show the influence of non-preemptive downtimes on the critical set of an operation. Again, we have a schedule that is similar to the schedules in the previous two examples: We have precedence relations between the operations of jobs J_1 and J_2. However, now we also have a non-preemptive downtime on resource A (indicated by the grey block). The schedule is given in Figure 5.5:

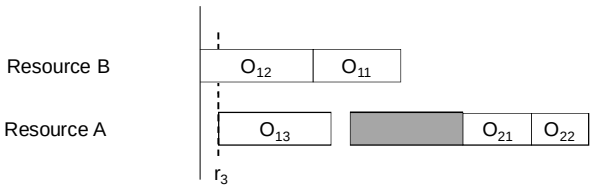

Figure 5.5: A third example schedule

In Figure 5.5, $COS(O_{22})$ is again the set $\{O_{13}, O_{21}\}$: As in the previous examples, O_{21} is the only critical (resource) predecessor of O_{22}. When determining $COS(O_{21})$, we again have operations O_{11} and O_{13} as predecessors. In this case, however, $ET(O_{11})_{21}$ is equal to C_{11}, whereas $ET(O_{13})_{21}$ is equal to the end time of the non-preemptive downtime. Since $ET(O_{13})_{21}$ is greater than $ET(O_{11})_{21}$, O_{13} is the only critical predecessor of operation O_{22}. As in previous examples, O_{13} does not have any predecessors.

Note, that we do not consider any kind of non-preemptive downtime as a possible operation for $COS(O_{22})$: If the grey block had been a fixed operation, we still would not have considered it as a possible operation to include in $COS(O_{22})$. We choose not to consider fixed operations for the critical set, because these operations cannot be rescheduled in any way: The operations cannot be scheduled earlier in time and they cannot be scheduled on a different set of resources. Therefore, we cannot adjust these operations so O_{22} can be scheduled earlier in time.

The critical set of an operation is relevant when we determine the set of operations in a schedule that determine the schedule's tardiness. We define this set of operations as the set of *bottleneck operations*. Each bottleneck operation in the schedule directly or indirectly determines the tardiness of the schedule. Therefore, to reduce the tardiness of a schedule, we must ensure that at least one bottleneck operation in the schedule obtains an earlier starting and completion time.

We distinguish two kinds of bottleneck operations:

1. Operations that directly determine the tardiness of a schedule: As we mentioned in Section 5.1, we determine a schedule's tardiness by determining the tardiness of each job in the schedule. The tardiness of a job is determined by the completion time of the latest operation that belongs to the job. The latest operation of each job thus directly determines the tardiness of the schedule. We define the set T_q as the set that contains the latest operation of each tardy job in schedule q.
2. Operations that indirectly determine the tardiness of a schedule: In addition to the operations in T_q, the critical set of each operation in T_q also influences the schedule's tardiness. As we mentioned before, the critical set of an operation prohibits that operation from being scheduled earlier in time. In other words: The critical set of an operation determines the starting time (and hence the completion time) of an operation. Therefore, the critical sets of the operations in T_q indirectly determine the schedule's tardiness.

As mentioned, we need to reassign or resequence at least one bottleneck operation in a schedule if we want to decrease the tardiness of that schedule. In subsequent chapters we demonstrate how the set of bottleneck operations is used in different scheduling approaches for solving scheduling problems.

6. Approaches for solving the general scheduling problem

In the previous chapter, we defined the scheduling problem for which we develop a scheduling approach. In this chapter, we focus on the different scheduling approaches to solve this scheduling problem.

In Section 5.2, we mentioned that we need to make three different types of decisions to solve the general scheduling problem:

1. We need to determine the mode in which each operation is performed.
2. We need to determine the set of resources on which each operation is performed.
3. We need to determine a starting and completion time for each operation.

The scheduling problem therefore contains three different subproblems:

1. *The mode assignment problem:* This is the problem of assigning each operation in the scheduling problem to a specific mode. By solving this problem, we obtain M_q.
2. *The resource assignment problem:* This is the problem of assigning each operation in the scheduling problem to a set of resources to satisfy the operation's resource demands. By solving this problem, we obtain A_q.
3. *The sequencing problem:* This is the problem of determining for each resource in the scheduling problem the sequence in which the operations assigned to that resource are processed. By solving this problem, we obtain W_q.

As we previously mentioned, we are able to create a schedule once we have determined M_q, A_q, and W_q. The procedure to determine starting and completion times for the operations in the schedule is given in Section 5.2.

When studying scheduling approaches found in literature, we thus need to focus on those approaches that can be used to solve a scheduling problem that contains a mode assignment problem, a resource assignment problem, and a sequencing problem. We did not find a scheduling problem in literature in which all these subproblems need to be solved. The scheduling problems found in literature that contain most similarities with the scheduling problem of the previous chapter are:

1. *Flexible shop scheduling problems:* The general scheduling problem closely resembles a general shop problem. Therefore, we consider these kinds of problems in scheduling literature as well. We also consider the most extensive special case of the general shop, namely the flexible job shop, because the flexible job shop also has important similarities to the general scheduling problem of the previous chapter: In the flexible job shop, (basic) precedence relations apply between operations belonging to the same job. Therefore, a scheduling approach for this kind of problem must consider the job predecessors of an operation when scheduling that operation. Another important similarity to our scheduling problem is resource flexibility: In the flexible job shop problem, each operation needs to be scheduled on one of the different suitable resources. Therefore, scheduling approaches for this scheduling problem must include a procedure to assign operations to resources.
2. *Multi-mode resource-constrained project scheduling problems (MRCPSP):* We also consider MRCPSP, because this type of problem shows even stronger similarities to our scheduling problem than the flexible job shop problem. In the MRCPSP, we again have a set of jobs that consist of a number of operations (called activities in project scheduling) between which general precedence relations may apply. In these problems, each operation needs to be performed in one out of several possible processing modes (ways), where a processing mode uniquely determines the operation's processing time and resource requirements. A set of resources, each with

a limited amount of capacity in each time period, is available to satisfy the resource requirements of the operations (Kolish and Drexl, 1997). Two kinds of decisions need to be made to solve a MRCPSP: First, each operation must be assigned to a processing mode. Second, each operation must obtain a starting time. The most important similarities between the MRCPSP and our scheduling problem are:

a. General precedence relations between operations: Operations in the MRCPSP may have multiple job predecessors and job successors.
b. Multi-resource operations: The processing mode of an operation may specify that an operation needs to be scheduled on multiple types of resources.
c. Resource flexibility: As in the general scheduling problem, we need to assign each operation to a set of resources before we can determine starting times for the operations. In the MRCPSP, the assignment of operations to resources is done by choosing a processing mode in which the operation will be performed. Note that the assignment of a mode to an operation in the MRCPSP is not completely similar to the mode assignment in the scheduling problem of the previous chapter: In that scheduling problem, we still need to assign operations to a set of resources after having assigned the operation to a mode. In the MRCPSP, in contrast, the set of resources on which an operation needs to be scheduled, is fixed once we have assigned a processing mode to the operation.

Both the flexible shop problems and the MRCPSP contain an assignment problem and a sequencing problem: In the flexible shop problems the assignment problem consists of assigning each operation in the flexible shop to a resource to process the operation, whereas in the MRCPSP the assignment problem consists of assigning each operation in the scheduling problem to a processing mode. By considering the scheduling methods used to solve the assignment problem in these types of problems, we may be able to determine a suitable method to solve the mode assignment and resource assignment problems in the general scheduling problem of the previous chapter. We may also be able to determine a suitable method to solve the sequencing problem by considering the methods used to solve the sequencing problem in the flexible shop problems and MRCPSP: In both the flexible shop and the MRCPSP, we need to determine starting and completion times for each operation. Therefore, we might be able to use the methods for the sequencing problem to determine positions for the operations in the general scheduling problem.

In addition to the flexible shop and MRCPSP, we also consider some special cases of these problem types when looking for scheduling approaches in literature. The special cases that we consider are the job shop scheduling problem (a special case of the flexible job shop problem) and the singe resource-constrained project scheduling problem (RCPSP, which is a special case of the MRCPSP). In both types of special cases, each operation has already been assigned to a specific mode or a specific resource. As a result, these problem types do not contain an assignment problem. However, we still consider these special cases because scheduling approaches for these cases need to determine starting and completion times for each operation. Therefore, we might be able to use these approaches to solve the sequencing problem in the general scheduling problem.

When we consider scheduling approaches found in literature to solve the problem types previously mentioned, we notice that the different scheduling approaches are based on one of the following two main solution architectures:

1. *The hierarchical architecture:* In the hierarchical architecture, the assignment of operations to resources and the sequencing of operations on the resources are treated separately. First, the assignment problem is solved, in which operations are assigned to a set of resources. In this problem, the sequencing of the operations is of little or no importance. Once the assignment problem has been solved, the assignment of operations to resources is considered fixed. The sequencing problem

is then solved for the given assignment (i.e. positions are determined for the operations on the resources to which the operations have been assigned). This structure is based on the idea of decomposing the original problem in order to reduce its complexity.

2. *The integrated architecture:* Brandimarte (1993) refers to this structure as the concurrent architecture. In the integrated architecture, the assignment and sequencing problems are considered simultaneously: After assigning an operation to a set of resources, the operation immediately obtains a position on each resource in this resource set. This in contrast to the hierarchical architecture, where each operation first obtains a resource assignment before the operations obtain positions on their resource sets.

Irrespective of the structure used, each scheduling approach must assign each operation to a set of suitable resources and determine a position for the operation on each resource to which the operation is assigned. To do so, all scheduling approaches found in literature use a set of basic algorithms. Therefore, the remainder of this chapter is as follows: In Section 6.1, we first discuss the basic algorithms that have been used in the different scheduling approaches. Subsequently, we discuss the scheduling approaches themselves. In Section 6.2, we first discuss the scheduling approaches based on the hierarchical architecture. In this section, we also mention the algorithms that different authors have used in the different approaches. Then, in Section 6.3, we discuss scheduling approaches based on the integrated architecture. Finally, in Section 6.4, we make conclusions on the suitability of the different approaches and algorithms for the general scheduling problem introduced in the previous chapter.

6.1 Algorithms mentioned in literature

We previously mentioned that all scheduling approaches consist of a set of basic algorithms. In this section, we discuss the different basic algorithms that authors have used to create the different scheduling approaches.

As we mentioned, we focus on those scheduling approaches that have been used to solve (flexible) job shop problems, general shop problems, and (M)RCPSP's. These different problem types are all NP-hard problems if we wish to minimize the tardiness of the schedule (Brucker, 1998). For these types of problems, no scheduling approaches have yet been created to solve the problem to optimality in polynomial time. Since we focus on solving real-life scheduling problems, we must limit the calculation time of the scheduling approach so it can be used in practical settings. Therefore, we do not consider any exact scheduling approaches in this assignment; we only focus on scheduling approaches based on heuristic algorithms.

We consider two kinds of algorithms: Constructive algorithms and local search algorithms. A constructive algorithm builds a schedule from scratch. A local search algorithm, in contrast, takes an existing schedule and tries to improve upon this schedule. We first discuss the constructive algorithms. Afterwards, we discuss local search algorithms.

Constructive algorithms

We have encountered a number of constructive algorithms in scheduling literature. The algorithms that we have encountered most frequently are priority-rule-based algorithms, the shifting bottleneck heuristic, truncated exponential algorithms, and multi-pass algorithms.

Priority-rule-based algorithms

Kolish and Padman (2001) mention priority-rule-based algorithms as algorithms that can be used in scheduling approaches for the RCPSP and its multi-mode variant. Scheduling approaches based on these algorithms work as follows: A partial schedule is extended in a stage-wise fashion to generate a feasible schedule. In each stage, a set of operations is chosen and these operations are scheduled on a set of resources.

Priority-rule-based algorithms are made up of two components: A schedule generation scheme and a set of priority rules. The schedule generation scheme determines which operations may be considered when determining the set of operations to be scheduled in a specific stage. In other words: the generation scheme determines which operations belong to the decision set in each stage. Subsequently, the priority rules are used to choose the set of operations that will be scheduled in the stage. In the flexible job shop problem (FJS) and the MRCPSP, priority rules are also used to choose the resource or mode to which the operation is assigned.

Kolish and Padman mention two types of schedule generation schemes: The serial method, (Kelley, 1963) and the parallel method (Bedworth and Bailey, 1982). In the serial scheme, we choose one operation in each stage and we schedule this operation on a set of resources. To choose which operation we will schedule in the stage, we create a decision set consisting of all unscheduled operations that do not have any job predecessors or whose job predecessors have all been scheduled. An important aspect of the serial scheme is that the scheme does not focus on the time that an operation can actually be scheduled. In other words: If we have two operations whose job predecessors have been scheduled, we include both operations in the decision set even if one of the operations can be scheduled much earlier than the other operation. As a result, we do not necessarily build up a schedule from left to right: It may occur that we first schedule an operation O_{ij} on a resource and that we subsequently schedule an operation O_{hl} before O_{ij} on this resource.

When we create schedules with the parallel scheme, we focus both on the job predecessors of an operation and the actual time that an operation can start when we create a decision set. With this scheme, we construct a schedule from left to right: We first schedule the operations that can start the earliest and we subsequently continue to schedule operations that can start later in time. We reach a new stage in the parallel scheme as soon as we reach a time t^* where we could possibly schedule a new operation. There are different moments when we can reach a new stage. Some possible moments are when a scheduled operation is completed (in this case t^* is equal to the completion time of the operation), when we reach the end time of a non-preemptive downtime (t^* is equal to this end time), or when we reach the release date of a job (t^* is equal to this release date). In each stage, we create a decision set that consists of operations that meet two criteria:

1. The operation is unscheduled and it either has no job predecessors or it has job predecessors that have been scheduled and completed by time t^*.
2. We can actually schedule the operation at time t^* on the set of resources to which the operation is assigned. Here, we assume that it is known in advance to which resources an operation is assigned.

Once we have created the decision set, we schedule a subset of the operations in the decision set. We do so by repeating the following steps until the decision set is empty:

1. Choose one operation from the decision set and schedule the operation at time t^*.
2. Remove all operations from the decision set that can no longer be scheduled at time t^*.

Once we have scheduled a subset of operations, we find a new value for t^* and we proceed to the next stage. We proceed in this way until we have scheduled all operations.

The procedure just mentioned for determining whether an operation belongs to the decision set assumes that each operation has already been assigned to a set of resources. In the scheduling problem of the previous chapter, however, the resource assignment for an operation is not known in advance. As a result, we do not know to which resources an operation has been assigned when we need to create a decision set. This problem also applies when we need to solve a FJS problem or an MRCPSP. In scheduling literature, this problem is solved by including an operation in the decision set if there is at least one resource (in the FJS) or one mode (in the MRCPSP) on which the operation can be scheduled at time t^*. For the scheduling problem of the previous chapter we can use a similar approach: We include an operation in the decision set if the operation can be scheduled at time t^* in at least one of its modes. This means that, for at least one mode, there must be a set of resources to which the operation can be assigned that satisfies the resource demands of the mode. In addition, it must be possible to schedule the operation at time t^* on this set of resources.

Once the decision set has been created in a stage, priority rules are used to actually choose an operation from the decision set to schedule as the next operation. When the FJS problem or the MRCPSP is solved, these rules are also used to choose a resource or mode to assign the operation to. To solve the scheduling problem of the previous chapter, we need several priority rules: One to choose an operation from the decision set, one to choose a mode in which the operation is performed, and one to choose a set of resources to satisfy each resource demand of the mode. Pinedo and Chao (1999) discuss different priority rules, but they use the term 'dispatching rules'. They make a distinction between basic and composite dispatching rules, where a basic rule is defined as a function of attributes of operations, resources, or both. Here, an attribute is any property associated with either an operation (e.g. its due date) or a resource (e.g. the speed of the resource). A composite rule, on the other hand, is a ranking expression that combines a number of basic dispatching rules. In a composite rule, the overall priority of an operation is determined by the basic rules used and their accompanying scaling parameters. The scaling parameter properly scales the contribution of each basic rule in the total ranking.

In addition to the distinction between basic and composite rules, Pinedo and Chao also classify priority rules in static and dynamic rules and local and global rules. Static rules are not time-dependent. As a result, the priorities of an operation remain the same in every stage. Dynamic rules, in contrast, are time-dependent. The priorities of an operation thus change in every stage. An example of a dynamic rule is the minimum slack first rule: This rule chooses the operation with the minimum amount of slack among the operations in the decision set. The slack of an operation depends on the moment in time when the slack is measured. We determine slack as follows: Slack = MIN(internal due date[2] – processing time – t ,0). Here, t specifies the moment in time when the slack is measured. For instance, we wish to determine which operation from the decision set will be scheduled next on resource R_r. Assuming that R_r becomes available at time t, the minimum slack rule determines the slack of each operation in the decision set at time t. Subsequently, the rule chooses the operation with the least amount of slack.

[2] The internal due date of an operation specifies the latest moment in time that the operation can finish such that the remaining operations of the job can be scheduled before the due date of the job. In other words: If an operation O_{ij} is scheduled after its internal due date, then there will be at least one operation O_{gj} whose completion time will be later than the due date of job J_j.

Local and global priority rules differ in the kind of information each type of rule uses: A local rule only uses information on the operations in the decision set or the resource to which the operation is assigned. A global rule, in contrast, may use information on other operations or other resources. In a job shop, for instance, we may consider an operation's job successor when we determine which operation will be scheduled as next operation: If the job successor needs to be processed on a resource that is available immediately, we may choose to give the operation a higher priority. On the other hand, if the successor needs to be processed on a resource that is not available for processing soon (for instance, because a lot of operations are already scheduled on the resource), we may choose to give the operation a lower priority, since the successor cannot start any time soon.

Truncated exponential algorithms

Brandimarte (1993) mentions some general heuristic algorithms for solving the flexible job shop problem. The truncated exponential algorithms are among the algorithm types he mentions. These algorithms are derived from exact algorithms (e.g. branch and bound), when the condition assuring optimality of the solutions is relaxed. One example of such an algorithm is the beam search algorithm. This algorithm is derived from branch and bound. In branch and bound, we (implicitly or explicitly) consider all solutions in the solution space and we choose the solution with the best objective value. In beam search, in contrast, we only consider the subset of the solutions in the solution space that seem most promising. From this subset, we subsequently choose the best solution. To increase the probability of finding the most promising solutions, the beam search algorithm uses a two-stage approach: First, a set of criteria are used to roughly determine which solutions in the solution space have most potential. The solutions that meet the criteria are retained, whereas the remaining solutions are discarded permanently. We subsequently use a set of criteria to evaluate the remaining solutions more thoroughly. The solutions that seem most promising after this evaluation are kept. From this set of solutions we subsequently choose the solution with the best objective value.

Shifting bottleneck heuristic

The shifting bottleneck heuristic of Adams et al. (1988) can be used for minimizing a schedule's makespan in the job shop environment. In the classical job shop, each operation has been assigned to a resource. The scheduling problem is thus to determine a sequence for the operations assigned to each resource. The shifting bottleneck heuristic solves this scheduling problem by sequencing the operations on one resource at a time. In each step, the heuristic identifies the bottleneck resource among the resources not yet sequenced and it sequences the operations assigned to this resource. Subsequently, the algorithm locally re-optimizes all previously sequenced resources. Both the identification of the bottleneck resource and the local re-optimization of the sequenced resources are done by solving single-machine scheduling problems.

To solve the classical job shop, the heuristic repeats the following steps until all resources have been scheduled:

1. Determining the bottleneck resource among the resources not yet sequenced: To determine the bottleneck resource, the heuristic takes the following steps:
    a. For each not sequenced resource, the heuristic solves a single machine problem of minimizing the makespan. In doing so, the heuristic considers the job predecessor of an operation when determining the earliest starting time of that operation. Note that the job predecessor O_{gj} of an operation O_{ij} may be processed on a resource that has already been scheduled. In this case, the operation sequence on that resource determines the earliest starting time of O_{gj} and, hence, the earliest starting time of O_{ij} on its resource.
    b. The resource with the largest makespan is considered the bottleneck resource.

2. After having sequenced the operations on the bottleneck resource, the operation sequences on each of the previously scheduled operations is re-determined, keeping in mind the sequence on the new resource.

Schutten (1998) mentions that the shifting bottleneck heuristic can be extended to solve more general scheduling problems than the classical job shop. For instance, the heuristic can be extended to incorporate downtimes, changeover times, and multi-resource operations. In addition, the heuristic can be used for other objectives than makespan minimization.

Multi-pass algorithms

In the previously mentioned algorithms, only one schedule is constructed. Some of these algorithms require very little computation time. Baker (1974) therefore suggests that the procedure by which a single schedule is obtained, is repeated a number of times with some simple variations. The best schedule then can be selected from the generated schedules.

Generally, two different approaches of multi-pass procedures have been proposed:

- One schedule generation scheme (i.e. a serial or parallel scheme) and different priority rules are used. This approach is called the multi-priority rule approach. Baker (1974) mentions an approach closely related to this approach, where a family of appropriate rules are identified that are distinct only in terms of a single parameter. The algorithm is now repeated for several parameter values and the best schedule is then chosen.
- One scheduling scheme and one priority rule are used, but the choice made by the priority rule is biased through a random device. This approach is called multi-pass sampling. There are three basic variations of multi-pass sampling (Kolish and Drexl, 1996):
    - *Random sampling:* This variation assigns the same probability to each item in the decision set.
    - *Biased random sampling:* This form of sampling biases the probabilities for each item in the decision set by relating this probability to the priority values of the items. A priority rule is used to create an ordered list of the items in the decision set: The smaller the position of the item in the list, the higher the priority that the item has. Subsequently, each item in the list is given a probability that depends on its position in that list (Baker, 1974).
    - *Parameterized regret-based random sampling:* In parameterized regret-based random sampling, the selection probability assigned to an item depends on the regret factor that the item has. The regret factor on an item specifies the amount of 'regret' if the item is not chosen. Therefore the higher the regret factor on an item, the greater the probability assigned to the item. Assuming that an item with a higher priority should get a higher probability, we determine the regret factor r_j of item j as follows: $r_j = p_j - \min_j(p_j)$. Here, p_j is the priority of item j. Subsequently, we calculate the probability of an item as follows: $prob_j = \frac{(r_j+1)^{\alpha}}{\sum_j (r_j+1)^{\alpha}}$. Here, α is a parameter that specifies the level of bias: An α of zero results in random sampling, whereas a large value of α results in a more deterministic choice. As we can see from the previous equations, the algorithm considers the absolute difference in priority values instead of the priority values themselves: The regret factor is equal to the difference in priority values and this factor is the one used to determine an item's probability.

Local search algorithms

Hurink (-) mentions that local search algorithms are based on the idea of iteratively moving through the set of feasible schedules. We move through the set of schedules by taking a given schedule (which is called the current solution in scheduling literature) and choosing a new schedule based on this given schedule (and possibly on previous schedules that have been visited). The choice for a new schedule is restricted to schedules that are in some way close to the given schedule. These schedules belong to the neighbourhood of the given schedule and they are called the neighbours of the given schedule.

Once we have moved to the new schedule, this schedule becomes the new given schedule. We then continue to move to other schedules in the solution space in a similar way until a set of stopping criteria are met.

In scheduling literature, we have found a number of local search algorithms that are used in scheduling approaches for the (flexible) job shop and (M)RCPSP. The algorithms that we have encountered most frequently are the iterative improvement algorithm, simulated annealing, threshold accepting, tabu search, and genetic algorithms. Note, that in the remainder of this chapter, and in subsequent chapters, we refer to the given schedule as the current schedule.

Iterative improvement

Papadimitriou and Steiglitz (1982) give an introduction in iterative improvement. However, they refer to the algorithm as 'local search'. The idea behind iterative improvement is that only true improvements are accepted. In other words: We move from the current schedule to a neighbouring schedule only if the objective function value of the neighbour is better than the value of the current schedule. The algorithm is repeated with a new solution until no improving candidate is found.

The iterative improvement algorithm just described has one important disadvantage: Since a neighbour is only accepted if it has a better objective function value than the current schedule, the algorithm stops as soon as it reaches a schedule that does not have any neighbours with a better objective value. This schedule is called a local optimum. Often, however, this local optimum is not the best schedule in the solution space (i.e. the global optimum).

Simulated annealing

Van Laarhoven and Aarts (1987) created the simulated annealing (SA) algorithm as an algorithm that is able to escape local optima. As we mentioned before, the iterative improvement algorithm only accepts a neighbour if the neighbour has a better objective value than the current schedule. The SA algorithm, in contrast, can also accept a neighbour if the neighbour has a worse objective value than the current schedule. As a result, SA can escape a local optimum and reach other areas of the solution space.

The basic SA algorithm is as follows: We start with an initial schedule. In every step, we randomly choose a neighbour schedule q'. If the objective value of q' is better than the objective value of the current solution q, we accept q' and we move to q'. Otherwise, we accept q' with a probability that depends on the deterioration Δ of the objective function value. The probability of acceptance is computed as $e^{-\Delta/T}$. Here, Δ is equal to $ov(q) - ov(q')$, with $ov(q)$ being the objective value of the current solution q, and T being a control parameter (the "temperature"). During the course of the algorithm, T is gradually reduced according to some cooling scheme. As a result, the probability of accepting a deteriorating move decreases during the annealing process. The algorithm ends once T reaches a value below a certain minimum temperature (i.e. the end temperature).

The Simulated Annealing algorithm has the following parameters:

1. *An initial temperature*: This is the temperature at the start of the algorithm.
2. *The end temperature*: The end temperature is the stop criterion in the algorithm. The algorithm ends once the temperature becomes lower than this minimum temperature.
3. *The Markov chain length*: In the Simulated Annealing algorithm, we perform a number of steps with the same temperature. The Markov chain length (MCL) specifies this number of steps. Once this number of steps has been performed, we reduce the temperature and subsequently we perform MCL steps with the new temperature.
4. *Decrease factor*: The decrease factor determines by how much the temperature is reduced after MCL steps have been performed. To obtain a new temperature, the current temperature is multiplied by the decrease factor. Therefore, the decrease factor has a value between zero and one.

Threshold accepting

Ducek and Sheuer (1990) designed threshold accepting as a partially deterministic version of the SA algorithm. As with SA, the algorithm randomly chooses a neighbour in every step. The mechanism for accepting the neighbour solution, however, differs from SA: The threshold accepting algorithm only accepts an inferior neighbour if the difference between its objective function value and the value of the current solution is smaller than a threshold t. The threshold is a positive control parameter that decreases as the number of iterations increases and converges to zero. As with SA, the probability of accepting a deteriorating move thus decreases in the course of this algorithm.

Tabu search

The tabu search (TS) algorithm is a local search technique that has been created by Glover (1986). As with the SA and threshold accepting, this technique also tries to avoid getting stuck in a local optimum. In TA, we try to avoid local optima by forbidding (or penalizing) certain moves. The algorithm examines all solutions in the neighbourhood (or in a subset of the neighbourhood when the neighbourhood is too big to be explored efficiently) and moves to the best possible neighbour, even if this neighbour is not a better solution than the current solution. To avoid cycling, a list, called the tabu list, is installed that contains a certain number of the last solutions encountered. If a solution is in the list, the move to that solution is forbidden. Alternatively, a characteristic or an attribute of the moves can be recorded. These characteristics are used to develop aspiration criteria that determine which tabu restrictions can be overridden, thus removing a tabu classification otherwise applied to a move.

Genetic algorithms

Pinedo and Chao (1999) mention genetic algorithms as a general-purpose scheduling algorithm. The algorithm has been developed by Holland (1975). Genetic algorithms view solutions as individuals or members of a population. Each individual is characterized by its fitness, which is measured by its objective function value. The genetic algorithm generally starts with an initial population, which is comprised of a subset of feasible solutions. During each iteration, the algorithm replaces the current population by a next population. The current population can be changed in two ways: By mutation and by recombination or crossover. A mutation is applied with a given probability to each solution in the current population and it changes the solution slightly. A recombination, in contrast, produces a new subset of solutions. This is done by choosing pairs of solutions from the current population (i.e. parents) and for each pair two new solutions (i.e. children) are obtained by combining the parents. Which pairs of solutions are chosen, depends on the fitness of the solutions: In general, the solutions are assigned a probability based on their fitness (i.e. if they are ‘fitter’, they have a higher probability of being chosen). Subsequently, the pairs of solutions are randomly chosen.

From solutions obtained by mutation and recombination, a subset is subsequently chosen as the new population. Often, the best k solutions are chosen to form the new population. The population size often remains constant over time.

Pinedo and Chao mention that the use of genetic algorithms has both advantages and disadvantages. One advantage of the algorithm is that it can be applied to a problem without having to know much about the structure of that problem. Also, the algorithm can be coded very easily and it often gives good solutions. However, compared with more rigorous problem-specific approaches, the algorithm might need a relatively large amount of computation time to obtain comparable solutions.

6.2 Scheduling approaches based on the hierarchical architecture

As mentioned at the beginning of this chapter, the assignment of operations to resources and the sequencing of operations on the resources are treated separately in the hierarchical architecture. As a result, scheduling approaches based on this structure consist of two different steps:

1. A step in which the assignment problem is solved: In this step, the approach determines an assignment A_q for the schedule. Here, the approach does not focus, or barely focuses, on determining a sequence W_q for the operations assigned to the different resources.
2. A step in which the sequencing problem is solved: In this step, the approach assumes that the assignment A_q of operations to resources is known. Based on this assignment, the approach now determines the positions that each operation will obtain on the resources to which that operation has been assigned.

As we mentioned in Section 5.2, we must have an assignment A_q before we are able to specify W_q. As a result, scheduling literature considers the assignment problem as the higher level problem that needs to be solved. The sequencing problem is the lower level problem that can only be solved once an assignment has been specified.

Schedule approaches based on the 'pure' hierarchical architecture solve the assignment problem by only specifying A_q. In other words: The approach does not specify a sequence W_q for the schedule. This form of the hierarchical architecture has the important disadvantage that we cannot determine how an assignment affects the objective value of a schedule: In the scheduling problems that we consider in scheduling literature and the scheduling problem of the previous chapter, the objective is either to minimize the makespan of the schedule or to minimize the tardiness of the jobs in the schedule. To determine the quality of an assignment on either type of objective value, each operation in the schedule must be given a completion time. To determine the completion times for the operations, however, the sequence W_q for the schedule must have been determined first.

To determine the quality of an assignment A_q, we thus might need to solve the sequencing problem for that assignment. As we previously mentioned, the sequencing problems that we consider are NP-complete problems: Both the job shop problem and the RCPSP are NP-complete if we wish to minimize tardiness. Since we focus on solving practical problem instances (i.e. instances that tend to be large), we expect it to be very time-consuming to solve the sequencing problem for each possible assignment. In scheduling literature, authors therefore choose to evaluate each assignment A_q by using a simple method $f_{simple}(A_q)$ to

solve the sequencing problem for the assignment. The authors subsequently choose a set of assignments that seem most promising and they solve the sequencing problem again for these assignments. Now, however, the authors use a more extensive sequencing method $f_{extensive}(A_q)$.

In the remainder of this section we describe the scheduling approaches based on the hierarchical architecture that we have found in scheduling literature. We first describe the general structures of which the scheduling approaches are comprised. Subsequently, we give examples of hierarchical scheduling approaches that certain authors have used to solve a certain scheduling problem. Here, we specify the general structure of the scheduling approach and the algorithms that have been used to solve the assignment and sequencing problems.

The general structure of hierarchical scheduling approaches

Brandimarte (1993) mentions that scheduling approaches based on the hierarchical architecture can be classified according to the information flow between the method used to solve the assignment problem and the method used to solve the sequencing problem. We refer to these flows as communication schemes. Brandimarte mentions two types of schemes:

- *The one-way communication scheme:* In this scheme, the assignment problem is solved first and subsequently the sequencing problem is solved once. When we solve the assignment problem, we thus obtain a set of assignments. For each assignment A_q in the set we subsequently solve the sequencing problem.
- *The two-way communication scheme:* In this scheme, iteration takes place between the method that solves the assignment problem and the method that solves the sequencing problem. The schedules that we obtain when we solve the sequencing problem for a given assignment A_q can thus be used as input to create a new assignment $\overline{A_q}$. For this new assignment we can again solve the sequencing problem and so forth. We continue to iterate in this way until a certain stop criterion is met.

When using a hierarchical architecture, we must thus determine whether we exit a scheduling approach once we have solved the sequencing problem for a given assignment or whether we determine new assignments based on the schedules created when solving the sequencing problem.

In addition to determining a communication scheme, we must also specify how we determine a set of assignments and for which assignments we subsequently solve the sequencing problem. De Reyck and Herroelen (1999) have experimented with different possibilities. They mention the following:

1. *Truncated enumeration:* Approaches based on truncated enumeration systematically enumerate a number of assignments and subsequently solve the sequencing problem for each assignment.
2. *Random enumeration:* Approaches based on random enumeration randomly generate a number of assignments. As with truncated enumeration, the sequencing problem is solved for each of the generated assignments.
3. *Local search methods:* These types of approaches start with an initial solution q. Subsequently, they use local search to improve the schedule's assignment: The approaches create a neighbourhood of q, where each neighbour q' has a different assignment $A_{q'}$ from A_q. A simple procedure $f_{simple}(A_{q'})$ is used to determine a

sequence $W_{q'}$ for the given assignment. From the neighbourhood, the schedule with the best objective value is chosen and the process is repeated until a stop criterion is met. The assignment A_{q*} of the best schedule $q*$ found is subsequently fixed, and a more detailed procedure $f_{\det ailed}(A_{q*})$ is then used to solve the sequencing problem for the given assignment. Some algorithms used for the local search are fastest descent (the first schedule in the neighbourhood with a better objective value is chosen), steepest descent (the best schedule in the neighbourhood is chosen), and tabu search.

Examples of hierarchical approaches

In the previous section, we described some factors to consider when creating a hierarchical scheduling approach. In this section, we give some examples of the hierarchical scheduling approaches that different authors have used.

Brandimarte (1993) considers two hierarchical scheduling approaches to solve the flexible job shop problem: One approach based on the one-way communication scheme and one approach based on the two-way scheme. In the one-way approach, priority rules are used to create an initial schedule q. Subsequently, the assignment A_q of operations to resources is considered fixed. The approach then uses a more detailed procedure $f_{\det ailed}(A_q)$ based on the tabu search algorithm to solve the sequencing problem for the given assignment. The procedure uses a neighbourhood $N_{seq}(q)$ that contains schedules where a subset of operations is chosen and each operation in this subset is swapped with an adjacent operation. The remaining operations maintain the positions they had on their resources in q. The procedure performs a fixed number of iterations, where each iteration consists of creating a neighbourhood $N_{seq}(q)$ of the current schedule and choosing the non-tabu neighbour with the lowest makespan. As we can see, the approach only creates one assignment and it subsequently considers this assignment to be fixed. Brandimarte mentions that the approach can be performed multiple times, each time with a different set of priority rules to create an assignment. Then, the approach can be used to create schedules with different assignment and then choose the schedule with the best objective value.

The approach based on the two-way scheme differs in two important ways from the one-way approach. First, in the two-way scheme we are able to determine a new assignment $A_{q'}$ after solving the sequencing problem. Second, the approach now uses tabu search instead of priority rules to solve the assignment problem. The two-way approach uses the following procedure to solve the assignment problem: The procedure performs a predetermined number of iterations. In each iteration, the procedure creates a neighbourhood $N_{assign}(q)$ by reassigning one operation on a critical path[3] of the current schedule q to a different resource and giving this operation the position on the resource that reduces the makespan of the schedule. All other operations remain assigned to the same set of resources as in q. Note that each schedule q' in $N_{assign}(q)$ obtains both a new resource assignment $A_{q'}$ and a new sequence $W_{q'}$. From $N_{assign}(q)$ the procedure chooses the non-tabu neighbour with the lowest makespan. When the assignment problem is solved for the first time, priority rules are

[3] Each schedule has one or more critical paths. Each critical path specifies a chain of operations that determine the makespan of the schedule: If the completion time of any of the operations on the critical path is increased, the makespan of the schedule will increase as well.

used to obtain an initial schedule. Later, the best schedule found when solving the sequencing problem is used as the initial schedule. Like the one-way approach, the two-way approach also uses $f_{\det ailed}(A_q)$ to solve the sequencing problem for a given assignment.

Brandimarte mainly focuses on minimizing the schedule's makespan. However, he mentions that the approaches can easily be adjusted to minimize the total (weighted) tardiness of a schedule. To use the approaches for tardiness minimization, however, we must not focus on the operations on a critical path of the schedule. Now, we need to focus on the operations that determine the tardiness of the schedule. For the scheduling problem of the previous chapter we thus need to focus on the bottleneck operations as described in Section 5.3.

Fattahi et al. (2007) developed different hierarchical approaches for the flexible job shop problem. All approaches work according to the same basic structure: First, a set of neighbours is created of the current schedule, where each neighbour q' has an assignment $A_{q'}$ that differs from the current schedule. For each neighbour, the procedure $f(A_{q'})$ is subsequently used to solve the sequencing problem, leading to a sequence $W_{q'}$ for the neighbour. One neighbour is subsequently chosen and this neighbour becomes the new current schedule if a set of criteria are met. The process is then repeated until a specific stop criterion is met and we exit the approach. All different approaches use a two-way communication scheme: By solving the sequencing problem for a neighbour, we obtain a complete schedule for which we can determine the makespan. We subsequently use the makespan of a neighbour to determine which neighbour becomes the new current schedule.

Two different algorithms are used to determine the set of neighbours with a different assignment: SA and TS. These two algorithms are also used to solve the sequencing problem for each neighbour. By combining the different algorithms, we obtain the following four approaches: An approach where SA is used to create the neighbours with different assignments and solve the sequencing problem, an approach where TS is used to create assignment neighbours and solve the sequencing problem, an approach where SA is used to solve create assignment neighbours and TS is used to solve the sequencing problem, and an approach where TS is used to create assignment neighbours and SA is used to solve the sequencing problem.

Both SA and TS use the same assignment and sequencing neighbourhoods: To solve the assignment problem, an assignment neighbourhood is created with schedules where one operation is reassigned to a new resource and all other operations remain assigned to the same resources as in the current schedule. The sequencing problem is solved by creating a sequencing neighbourhood with schedules where one resource is chosen and the positions of two adjacent operations on this resource are swapped.

An initial schedule q is obtained as follows: First, the resource assignment A_q is determined by assigning each operation to the resource on which the operation has the smallest processing time. Subsequently, W_q is determined by randomly assigning a position to each operation on its resource.

Zribi et al. (2007) also propose a hierarchical architecture for solving the flexible job shop problem. In this approach, both the assignment and sequencing sub-problems are solved once, with the assignment problem being solved first. When solving the assignment problem, the authors focus on minimizing the weighted sum of the workload on the critical resource (i.e. the resource with the greatest workload) and the total workload on all resources. Here, the workload of a resource is equal to the sum of the processing times of the operations scheduled on that resource. To solve the assignment problem, the authors first create an

initial assignment by using the localization approach. The idea behind this approach is to assign each operation to the least loaded resource, i.e. the resource with the smallest workload after including the processing time of the new operation. The assignment is subsequently improved by using tabu search. A neighbourhood is created by reassigning one operation to the resource that leads to the best schedule with respect to workload minimization.

By solving the assignment algorithm, the authors obtain an assignment A_q. This assignment is subsequently used as input when solving the sequencing problem. The authors now focus on makespan minimization and they use a hybrid genetic algorithm to solve the sequencing problem. An initial set of schedules (i.e. an initial population) is created that consists of two types of schedules: Schedules created by using different priority rules and schedules created by randomly assigning positions to operations. Each schedule q is stored as a list $d = \{O_{ij},...,O_{hl}\}$ that specifies the sequence in which the operations are performed: The schedule is obtained by scheduling the operations in the order in which they are found in the list.

In each iteration, the authors obtain a new population as follows: First, they obtain a new set of schedules by mutation and recombination of the current set of schedules. A mutated version of schedule q is created by taking an operation O_{ij} from list d and giving the operation a new position in the list. This position must be between the positions that the job predecessor and job successor of O_{ij} have in the list. In recombination, two schedules are combined to form two new schedules: In the list d_a' that represents the new schedule q_a', there is one job whose operations have the same positions in d_a' as they had in d_a. The other operations are assigned to the remaining positions in the list according to the order that these operations have in d_b: If an operation O_{ij} has an earlier position in d_b than an operation O_{hl}, then O_{ij} will also have an earlier position than O_{hl} in d_a'. Consider the following example: We have two schedules, represented by the lists $d_a = \{O_{11}, O_{12}, O_{22}, O_{21}, O_{32}, O_{31}, O_{13}, O_{23}\}$ and $d_b = \{O_{12}, O_{11}, O_{22}, O_{21}, O_{31}, O_{13}, O_{32}, O_{23}\}$. A possible schedule that we can create through recombination is the schedule, represented by $d_a' = \{O_{11}, O_{12}, O_{22}, O_{21}, O_{13}, O_{31}, O_{32}, O_{23}\}$. In d_a', the operations of job J_1 have the same positions as in d_a. The remaining operations need to be performed in the following order: $O_{12} \rightarrow O_{22} \rightarrow O_{13} \rightarrow O_{32} \rightarrow O_{23}$, which is the order that the operations have in d_b. Operation O_{12} thus obtains the first free position (position 2), operation O_{22} obtains the next free positions (position 3), and so forth.

Once the authors have obtained a new set of schedules, they apply iterative improvement to the schedules obtained through recombination to obtain the new population. They create a neighbourhood by taking an operation on a critical path of the schedule and swapping this operation (if possible) with its critical resource predecessor.

De Reyck and Herroelen (1999) use a hierarchical architecture for solving the multi-mode resource-constrained project scheduling problem with generalized precedence constraints. They develop several scheduling approaches, amongst others approaches based on truncated and random enumeration, steepest descent, and tabu search. A more detailed explanation of each approach is given in the previous section. In all cases, an initial assignment is obtained by assigning to each operation the mode with the smallest associated duration. For solving the resource-constrained problem (i.e. the sequencing problem), a truncated version of branch and bound is used.

The different procedures are compared with each other for several problem cases and the authors are able to conclude that tabu search consistently performs better than other procedures.

6.3 A scheduling approach based on the integrated architecture

In scheduling literature, we have also found approaches based on the integrated architecture. Here, we discuss the approach used by Dauzère-Pérès et al. (1998) to solve a practical flexible job shop. In this job shop, each operation may need several resources to be performed and each resource may be selected from a given set of candidate resources. Also, each operation has a set of job predecessors and a set of job successors. The authors propose a neighbourhood where one operation on a critical path is moved from its current position between operations O_{ij} and O_{hl} to a position between two different operations on a (possibly different) resource. The operation is reinserted in such a way that a feasible schedule is obtained. This neighbourhood structure is used to perform tabu search.

The authors generate an initial solution by using a two-step procedure: First, they assign the operations to the resources in order to balance the workload on each resource. They then develop a feasible schedule by using a FIFO dispatching rule. With tabu search, neighbours are subsequently developed and the best neighbour is chosen. The tabu list is then updated with a record that consists of three elements: The moved operation, its predecessor, and the resource on which the operation is scheduled.

6.4 Evaluation of found literature

As mentioned at the beginning of this chapter, the scheduling approach that we use needs to solve the scheduling problem mentioned in Chapter 5. In this section, we therefore evaluate the different approaches and algorithms found in literature and determine which seem to be most promising for our scheduling problem.

When comparing the hierarchical architecture to the integrated architecture, we can conclude that the hierarchical architecture seems the most promising for our scheduling problem. This is because the hierarchical architecture can be adjusted more easily to different scheduling situations: In Section 4.3, for instance, we concluded that many (potential) PLANWISE customers operate in an equipment environment. In such an environment, operations have fixed time windows. As a result, it is only relevant to solve the assignment problem in such a case. In the integrated architecture, in contrast, both sub-problems are combined into one problem. Therefore, it is more difficult to easily incorporate different scheduling situations.

As previously mentioned, hierarchical architectures either use a one-way or a two-way communication scheme. We think that for our scheduling problem it is more suitable to use a two-way scheme instead of a one-way scheme. We expect to find better solutions with this scheme compared to the one-way scheme: As we mentioned before, we solve the sequencing problem for a given assignment A_q. As a result, we reduce the solution space when we solve the sequencing problem to those schedules that have assignment A_q. Since we are not allowed to iterate between sequencing and assignment in the one-way scheme, we remain stuck in a specific part of the solution space once we have chosen an assignment. In the two-way scheme, however, we avoid this problem: Now, we are able to iterate between assignment and sequencing. As a result, we are able to create a new assignment $\overline{A_q}$ based on the results of the sequencing step. We can then use this new assignment to reach a different part of the solution space. In other words: When we use the two-way

scheme, we are still able to reach different kinds of assignments (and thus different areas of the solution space).

In addition to choosing a communication scheme, we must determine for which assignments we choose to solve the sequencing problem. For instance, we could choose to solve the sequencing problem for all assignments, or we could choose a set of assignments in a systematic way and subsequently solve the sequencing problem for this set of assignments. De Reyck and Herroelen (1999) mentioned three possibilities for making this choice, which have been described in Section 6.2. Of the different possibilities we prefer the local search method: With this method, we can use a simple procedure $f_{simple}(A_q)$ to determine a sequence W_q, and hence a schedule, for each a assignment A_q. We can subsequently compare different assignments to each other by looking at the objective values for the resulting schedules. We can thus choose the most promising assignments based on the objective values that we can reach with the assignment. Subsequently, we can use a detailed procedure to solve the sequencing problem for the promising assignments.

As we mentioned in Section 6.1, we have found a number of algorithms that are used in methods to solve the assignment problem or the sequencing problem. In Section 5.1 we mentioned that we require a scheduling approach to solve a modified general shop scheduling problem. Therefore, we require a generic algorithm (i.e. the algorithm should not only be suitable for a job shop problem). In addition, the algorithm should be flexible, so it can be easily adjusted to the situation at a specific customer. Based on the literature study, we conclude that the most suitable algorithms to meet these demands are local search algorithms and multi-sampling algorithms. These types of algorithms are not problem specific: In scheduling literature, they have been used to solve (flexible) job shops, practical general shops, and (M)-RCPSP. In addition, these types of algorithms have also been used to optimize a schedule on different types of objective functions. For instance, these types of algorithms have been used for makespan minimization and tardiness minimization. We thus know that these types of algorithms can be used to solve scheduling problems with different types of constraints and objective functions. The different algorithm types can also be adjusted easily to different scheduling problems, because the algorithm types are all very generic. If, for instance, we use a method based on local search and we wish to adjust this method to a different scheduling problem, we must only adjust the procedure used to create a neighbourhood. However, the structure of the method remains intact: In each iteration we still choose one neighbour from the neighbourhood of the current schedule and we move to this neighbour if certain criteria are met. Similarly, in methods based on multi-sampling algorithms, we only need to adjust the procedure to create a single schedule. The scheduling approach, however, still consists of creating a number of schedules and subsequently choosing the best schedule created.

Of the different local search algorithms, we prefer the Simulated Annealing (SA) and Tabu Search (TS) algorithms for our scheduling problem. The greatest advantage of these algorithms is that they provide possibilities for escaping local optima. Algorithms like iterative improvement and threshold acceptance, in contrast, have less (or no) possibilities to examine the complete solution space. In addition, SA and TS can be implemented relatively easily and they can be used for solving both the assignment and the sequencing subproblem. Finally, we prefer these algorithms over the genetic algorithm, because they are easier to explain and understand.

In addition to local search algorithms, we think that multi-sampling algorithms might also be suitable for our scheduling problem. Like local search algorithms, multi-sampling algorithms are very flexible: They can easily be adjusted to incorporate different scheduling situations. We can use such algorithms, for instance, to optimize schedules on different criteria. In addition, we can incorporate practical constraints relatively easily in these algorithms.

Truncated exponential algorithms, in contrast, are highly problem dependent. For these kinds of algorithms we thus require more effort to adjust the algorithm to a different scheduling situation. We also prefer multi-sampling algorithms to the shifting bottleneck heuristic: Both kinds of algorithms are able to incorporate many different practical constraints. We think, however, that we can incorporate more kinds of practical constraints in multi-sampling algorithms than in the shifting bottleneck heuristic. For instance, in the general scheduling problem we have multi-resource operations. Since resources may have different periods of preemptive downtime, we must consider all resources in a set S simultaneously in order to schedule an operation on the resources in S[4]. In the shifting bottleneck heuristic, however, we solve a scheduling problem by solving a number of single-machine problems. The heuristic is thus not able to consider multiple resources simultaneously.

Of the different multi-sampling variants, we prefer the regret-based random sampling (RBRS) variant, because it is the most flexible variant: By adjusting the bias parameter α we can create both situations in which a item is chosen randomly from the decision set and situations in which the item with the highest regret value has by far the greatest probability of being chosen. We also prefer RBRS, because the variant looks at the difference between priority values instead of the priority values themselves. The scheduling approach should be able to solve different scheduling situations. Since we expect that operations will have different priority values in different scheduling situations, we do not want the height of priority values alone to influence the outcome of the algorithm.

Like priority-rule based algorithms, the RBRS algorithm requires a schedule generation scheme. Both the serial and the parallel generation scheme are suitable for our scheduling problem. The serial scheme, however, has the advantage of being the simplest scheme to implement: When determining a decision set for the next operation to schedule, we only need to determine whether the operation's job predecessors have been completed. For the parallel scheme, in contrast, we also need to determine whether the operation can be scheduled at a time t. For the general scheduling problem we thus need to determine if there is at least one mode in which the operation can start at time t. To determine whether an operation O_{ij} can start at time t in mode V_{ijm}, we need to determine whether there is a set of resources S on which we can schedule O_{ij} at time t, such that all resource demands of V_{ijm} are satisfied. In the general scheduling problem, however, each resource has a calendar that specifies the periods of preemptive downtime on that resource. If all resources work according to the same calendar, we are sure that we can schedule operation O_{ij} on the set S at time t if we can schedule O_{ij} on each resource in S at time t. If the calendars of the resources differ, however, a situation may occur where O_{ij} can be scheduled at time t on each resource in S, but that O_{ij} still cannot be scheduled on the set S at time t. Consider the following example:

We need to determine whether operation O_{ij} can be scheduled at time 1. Operation O_{ij} has a processing time of 2. One possible set S on which we can schedule O_{ij} consists of resource A and resource B. Figure 6.1 gives the availabilities of the two resources. Resource A has the following intervals of preemptive downtime (indicated by the grey areas): [2,3] and [4,5]. Resource B has preemptive downtime in the interval [2.5, 3.5] and a fixed operation that is scheduled at time 5.

[4] In Figure 6.1, we give an example to demonstrate that we must consider all resources in set S simultaneously to determine whether an operation can be scheduled on that set.

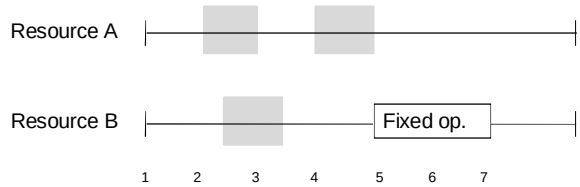

Figure 6.1: The availabilities of resource A and resource B.

As we can see in Figure 6.1, operation O_{ij} can start on resource A at time 1. O_{ij} then finishes at time 4. Similarly, operation O_{ij} can start on resource B at time 1 and then finish at time 4. However, it is not possible to schedule operation O_{ij} on resource A and resource B at time 1: O_{ij} can only be processed if both resources are available. Therefore, to schedule O_{ij} on time 1 on both resources, the resources must have at least 2 units of uptime that take place simultaneously in the interval [1,5] (O_{ij} must be completed before time 5, because it may not be scheduled over the fixed operation on resource B). When we consider the periods of simultaneous uptime, we notice that the resources only have 1.5 units of uptime in interval [1,5] (the resources are available in intervals [1,2] and [3.5, 4]).Thus, O_{ij} cannot be scheduled on set S at time 1.

In practical scheduling situations, we expect different resources to have different calendars. We thus need to consider the resources in S simultaneously to determine whether O_{ij} can be scheduled on set S at time t. Since we need to consider all resources in the set simultaneously, it may be a time-consuming process to determine whether O_{ij} can be scheduled in mode V_{ijm} at time t: There may be many possible sets of resources that can satisfy the resource demands of V_{ijm}. Each set must be considered separately to determine whether O_{ij} can be scheduled on that set at time t. In addition, the calculation time also increases as the number of resources in a set S increases.

For the general scheduling problem it can thus be time-consuming to determine whether an operation can be scheduled at time t and, hence, to create a single schedule. Since we develop many different schedules in the RBRS algorithm, it must not take a lot of time to develop a single schedule. A parallel scheme, however, might still be interesting in the future: In the parallel scheme, there is a greater focus on using the available free capacity of resources. As a result, schedules obtained with a parallel scheme will probably contain less idle time on resources compared to schedules obtained with a serial scheme.

7. Created scheduling approaches

In this chapter, we discuss the scheduling approaches that we use to solve the general scheduling problem mentioned in Chapter 5. We use two different approaches to solve the scheduling problem: An integrated approach based on regret-based random sampling (RBRS) and a hierarchical architecture based on local search algorithms.

We use an integrated approach based on RBRS, because certain PLANWISE versions already contain an optimizer based on this scheduling approach. This approach has thus already been used to solve different practical scheduling problems. Based on our findings in the literature, we also develop and implement an alternative approach for scheduling based on the hierarchical solution approach. In this approach, we solve both the assignment problem and the sequencing problem by using local search algorithms.

As mentioned in Section 2.3, the integrated approach serves as our benchmark approach: We compare the schedules obtained by this approach to the schedules obtained by using the hierarchical architecture.

The remainder of this chapter is as follows: We first discuss the integrated approach in Section 7.1 and 7.2 and subsequently we discuss the hierarchical approach in Sections 7.3 and 7.4. In Sections 7.1 and 7.3, we present the general structure of the different approaches: Here, we explain the idea behind the approaches and we specify in general terms how the approaches work. Then, in Sections 7.2 and 7.4, we discuss the actual implementation of the two approaches: Here, we specify, amongst others, the neighbourhood structures that we use and the algorithms on which the scheduling approaches are based.

When creating the different approaches, we assume that each operation is performed in a default mode. This mode does not change during the execution of the approach. Therefore, when describing the approaches, we do not focus on assigning or reassigning an operation to a different mode.

7.1 A general description of the integrated approach with RBRS

In the integrated approach based on RBRS, a number of different schedules are generated. The idea behind the approach is that it takes relatively little computation time to generate one schedule. As a result, we can generate multiple schedules and subsequently choose the best one.

We generate a single schedule by using a constructive procedure. In this procedure, we use the RBRS algorithm at decision points to choose an item from the decision set. Each item in the decision set is assigned a weight that determines its probability of being chosen. Since an item is chosen randomly at each decision point, a different schedule can be obtained if the procedure is performed multiple times.

The approach has the following general structure:

- Set k to 1
- Repeat the following steps until k > n (a fixed value):
    - Develop a schedule q and calculate its objective value.
    - If the objective value of q is better than the objective value of the best schedule found so far, store q as the new best schedule.
    - Increase k by 1.

To develop a single schedule q, the following steps are repeated until all operations have been scheduled or until we are sure that the partial schedule will have an objective value that is at least as great as the objective value of the best schedule found so far:

1. Choose an operation O_{ij} to schedule next:
    a. Determine a decision set D of those operations that may be scheduled as the next operation.
    b. Assign to each operation in D a weight and determine for each operation the probability of being chosen.
    c. Randomly choose an operation from D based on the probabilities of the operations in D.
2. Determine a resource assignment $A_q(O_{ij}, M_q)$ for O_{ij}.
3. Determine a position $W_q(O_{ij}, M_q, A_q)$ for O_{ij} on each resource in $A_q(O_{ij}, M_q)$.
4. Calculate S_{ij} and C_{ij} for operation O_{ij}.
5. Determine the objective value of the partial schedule: The objective value of the partial schedule is equal to $\sum_j T_j$ for all jobs whose operations have all been scheduled.

7.2 Implementation details of the integrated approach

In this section, we specify the details of the version of the integrated approach that we have implemented in PLANWISE.

To create a single schedule, we use a serial generation scheme in combination in RBRS. In Section 6.4, we compared the serial scheme to the parallel scheme. There, we concluded that both schemes are suitable for the general scheduling problem, but that we require less computation time when we use the serial scheme to create a schedule. We therefore choose to use a serial scheme in this approach.

We use the RBRS algorithm to choose items from the different decision sets. To choose operations from the decision set, we assign each operation a weight that is equal to the latest starting time of that operation. We determine the latest starting time of an operation O_{ij} as follows: $lst_{ij} = d_{ij} - p_{ij}$. In this equation, d_{ij} is the internal due date of O_{ij} and p_{ij} is its processing time. As we mentioned in Chapter 5, we want to minimize the total tardiness of the jobs. Therefore, we give the operation with the smallest latest starting time the highest regret value. By doing so we try to ensure that the operations that need to be scheduled earliest are scheduled before the operations that need to be scheduled later in time. Similarly, to choose a resource from the resource decision set, we assign each resource a weight equal to the earliest time that the operation can be scheduled on that resource. In determining this earliest time we also consider any changeover time that might need to take place between the operation and its resource predecessor on that resource. By assigning weights this way, we try to find a set of resources on which the operation can be scheduled as soon as possible.

In Section 7.1, we described the general structure of the integrated approach. This is also the structure that we have implemented in PLANWISE. Therefore, in this section we only mention the algorithm that we use to develop a single schedule. A schedule q is developed as follows:

Repeat until all operations have been considered:

1. Choose an operation O_{ij} to schedule next:
    a. Develop an operation decision set consisting of all unscheduled operations that are precedence-feasible.
    b. Assign each operation a weight equal to its latest starting time.
    c. Choose an operation by using RBRS.
2. Determine a resource assignment $A_q(O_{ij}, M_q)$:
    a. For each resource demand U_{ijmk} do:
        i. Assign each resource in R_{ijmk} a weight equal to the earliest time that O_{ij} can be scheduled on that resource.
        ii. Use RBRS to choose a_{ijmk} resources from R_{ijmk}.
3. Determine $W_q(O_{ij}, M_q, A_q)$:
    a. Assign operation O_{ij} the earliest position on its resources.
4. Calculate the starting and completion times for the operation as described in Section 5.2.

7.3 A general description of the hierarchical approach

As we mentioned in Section 6.2, we solve the assignment problem and the sequencing problem separately in the hierarchical approach. The approach uses a two way communication scheme: The schedules that we obtain when solving the assignment problem serve as input when solving the sequencing problem and vice versa.

We use local search algorithms to solve both subproblems. To solve a subproblem, we perform a number of iterations. In each iteration, we choose a schedule from the neighbourhood of the current solution and we move to this solution if a set of criteria are met. When we solve the assignment problem, we create an assignment neighbourhood consisting of schedules with a different resource assignment A_q than the current solution. When we solve the sequencing problem, we create a sequencing neighbourhood consisting of schedules with a different sequence W_q than the current solution.

Both neighbourhoods consist of complete schedules. In other words: The operations in each solution in the neighbourhoods have been assigned to a set of resources and they have obtained starting and completion times. Thus, when we create a solution q in an assignment neighbourhood, we do not only specify the resource assignment A_q, but we also determine a sequence W_q and, hence, starting and completion times for all operations in q. We choose to create neighbourhoods consisting of complete schedules for a number of reasons: First, we determine the quality of a neighbour by determining the tardiness of that neighbour. As we mentioned in Section 6.2, however, all operations in a schedule must have starting and completion times in order to determine the tardiness of the schedule. We must thus create a complete schedule in order to determine the tardiness of that schedule. Second, we increase the flexibility of the scheduling approach by creating complete schedules in both neighbourhoods: Since the neighbourhoods consist of complete schedules, we are able to solve each subproblem separately. As a result, we can easily adjust the scheduling approach to special cases of the scheduling problem that only contain an assignment or a sequencing problem. For instance, we discussed equipment scheduling problems in Section 4.3. When we solve an equipment scheduling problem, we only need to assign operations to resources.

When we use the hierarchical approach, we can now skip the solving of the sequencing problem.

As we mentioned in Section 6.2, we predominantly focus on determining an assignment A_q when we solve the assignment problem. We therefore wish to limit the amount of calculation time needed to create a schedule for a given assignment. Therefore, we use a simple procedure $f_{assign}(A_q)$ to determine a sequence W_q for A_q. Once we have solved the assignment problem, we focus on determining a good sequence for the most promising assignments that we have found. We then use a more elaborate procedure $f_{seq}(A_q)$ to solve the sequencing problem for those assignments.

The general structure of the approach is as follows:

- Start with an initial schedule. This becomes the initial current schedule q.
- **REPEAT** steps 1 through 4:
  1. Solve the assignment problem:
     a) Create an assignment neighbourhood $N_{assign}(q)$ of the current schedule. We create the schedules in $N_{assign}(q)$ by choosing a set O' of operations and determining an assignment $A_{q'}(O_{ij}, M_{q'})$ for each operation O_{ij} in O' that differs from the assignment in the current schedule.
     b) Accept one neighbour from $N_{assign}(q)$ according to a set of criteria. This neighbour becomes the new current schedule q. Store the neighbour in a set $Q_{assignment}$ if certain criteria are met.
  2. If a stop criterion is met, proceed to solving the sequencing step (step 3). Otherwise, resolve the assignment problem (step 1) with the new current schedule.
  3. For each schedule $q_{rr} \in Q_{assignment}$ do
     a) Solve the sequencing problem with procedure $f_{seq}(A_{q,})$:
        i. Create a sequencing neighbourhood $N_{seq}(q_{rr})$. We create the schedules in $N_{seq}(q_{r,})$ by choosing a set O' of operations and determining a set of positions $W_{q'}(O_{ij}, M_{q'}, A_{q'})$ for each operation O_{ij} in O' that differs from the current schedule.
        ii. Accept one neighbour from $N_{seq}(q_{r,})$ according to a set of criteria. This neighbour becomes the new current schedule q. Store the neighbour as schedule $\bar{q}$ if it is the best schedule found so far in step 3.
     b) If a stop criterion is met, proceed to the next schedule in $Q_{assignment}$. Otherwise, resolve the sequencing problem with the new current schedule.
  4. The best schedule $\bar{q}$ found when solving the sequencing problem becomes the new current schedule.

  UNTIL a criterion is met to exit the hierarchical approach

The possible moves between the different parts of the approach are highlighted in Figure 7.1:

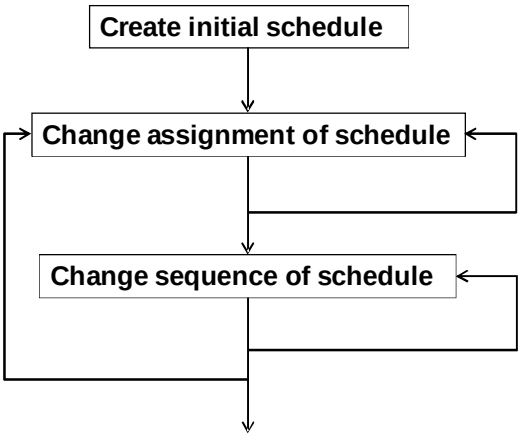

Figure 7.1: Possible moves between the different parts of the hierarchical approach

In the hierarchical approach, the set $Q_{assignment}$ contains the schedules that have the most promising assignments. This set may contain more than one schedule. In other words: We may choose to solve the sequencing problem for more than one schedule. We make this choice based on the ideas behind the two-stage approach in the beam search algorithm: When we solve the assignment problem, we may find multiple schedules that have promising assignments. If, however, we only choose to solve the sequencing problem for the schedule with the most promising assignment, we might miss a good schedule in the solution space simply because this schedule has a different assignment. On the other hand, we cannot solve the sequencing problem for each schedule that we create when solving the assignment problem, because this requires too much calculation time. Therefore, we choose a subset of the schedules that we have created when solving the assignment problem and we solve the sequencing problem for each schedule in the subset.

We use the following basic procedure to create a schedule q' in $N_{assign}(q)$ (i.e. an assignment neighbour of solution q):

1. Choose the set of operations O' that will obtain new assignments.
2. Determine an assignment $A_{q'}$:
    a) For each operation $O_{ij} \in O'$ do
        i. Determine a new assignment $A_{q'}(O_{ij}, M_{q'})$ that differs from $A_q(O_{ij}, M_q)$.
3. Use procedure $f_{assign}(A_{q'})$ to determine a new sequence $W_{q'}$
4. Calculate starting and completion times for all operations.

To create a schedule q' in $N_{seq}(q)$ (i.e. an sequencing neighbour of solution q), we use the following basic procedure:

1. Choose a set of operations O' that will obtain new positions on their respective resources.
2. Determine a sequence $W_{q'}$:
    a) For each operation $O_{ij} \in O'$ do
        i. Determine a new set of positions $W_{q'}(O_{ij}, M_{q'}A_{q'})$ for O_{ij} on the set of resources in $A_{q'}(O_{ij}, M_{q'})$.

3. Calculate starting and completion times for all operations.

7.4 Implementation details of the hierarchical approach

In the section, we specify which versions of the hierarchical approach we have actually implemented in PLANWISE. We divide this section into three parts: First, we describe some preliminaries. Subsequently, we describe how we create the assignment and sequencing neighbourhoods. Finally, we describe the different versions of the approach that we have implemented.

Preliminaries

In this section, we describe some preliminaries when we create the assignment and sequencing neighbourhoods. We mention which kinds of operations we consider for rescheduling (either by assignment or by sequencing) and some focus points and assumptions when creating a neighbour.

In Section 5.3, we defined the bottleneck operations of a schedule. There, we mentioned that the bottleneck operations determine the tardiness of a schedule: To reduce the tardiness of the schedule, we thus need to reschedule at least one bottleneck operation. Therefore, when we create an assignment or sequencing neighbourhood, we focus on reassigning or resequencing the bottleneck operations in the schedule.

We focus on creating neighbours by making small adjustments to the current solution. In our assignment neighbourhoods, for instance, we focus on reassigning one operation at a time. We focus on making small adjustments, because we expect to require less computation time when we make small adjustments compared to when we make large adjustments. For instance, if we create a neighbour by reassigning multiple operations to different resources, we can expect the approach to take more time than if we were to reassign only one operation to different resources.

As we mentioned in Section 7.3, the assignment and sequencing neighbourhoods that we use contain complete schedules. Therefore, we require a procedure to determine a sequence W_q when we create a neighbour. When determining positions for operations, we must ensure that we obtain a feasible schedule. This is especially the case for operations that require multiple resources: If two operations are assigned to the same set of resources, we must ensure that O_{ij} is always scheduled before operation O_{hl} on all resources to which both operations are assigned. Otherwise, we create a situation in which operation O_{ij} has a position before operation O_{hl} on resource 1, whereas O_{ij} has a position after O_{hl} on resource 2. As a result, neither operation can obtain a starting (and completion) time, because the one operation cannot be scheduled until the other operation is scheduled and vice versa.

To ensure that we always create a feasible schedule, we calculate a temporary starting time for each operation and we subsequently assign positions to each operation based on this starting time: If operation O_{ij} has a lower temporary starting time than operation O_{hl}, then O_{ij} obtains a lower position than O_{hl}. Since the temporary starting time is not dependent on the resource, we ensure that operation O_{ij} obtains a lower position than O_{hl} on all resources to which both operations are assigned. After we have assigned positions to the operations, we can determine the actual starting and completion times for the operations by using the procedure as described in Section 5.2.

We take the following general steps when determining a temporary starting time for an operation O_{ij}:

- We first determine the position for O_{ij} on a subset S of its resources according to a specific set of criteria.
- Subsequently, we determine a temporary starting time for O_{ij} by only considering the job predecessors of O_{ij} and the operations scheduled before O_{ij} on the resources in S.

Once we have determined a temporary starting time for O_{ij}, we insert O_{ij} on its other resources as follows: On each resource, we look for the last operation with a lower starting time than the temporary starting time of O_{ij} and insert O_{ij} directly behind this last operation.

When we describe the procedures to create the different neighbourhoods, we specify in more detail how we determine the temporary starting time.

The creation of neighbours

In the implemented versions of the hierarchical approach, we use two different neighbourhood structures when solving the assignment problem and one neighbourhood structure when solving the sequencing problem. We refer to the first neighbourhood structure for the assignment problem as $N_{assign}{}^{1}(q)$ and to the second structure as $N_{assign}{}^{2}(q)$. We refer to the neighbourhood structure for the sequencing problem as $N_{seq}(q)$.

As we mentioned in the preliminaries, we focus on reassigning and resequencing the bottleneck operations in a schedule. Therefore, we create the different neighbourhoods by specifying one neighbour for each bottleneck operation that meets a set of criteria. In other words: We have a set S of bottleneck operations that meet certain criteria. For each operation O_{ij} in this set S we create one neighbour by rescheduling O_{ij} (either by reassignment to other resources or by resequencing). Therefore, the set S determines the size of a neighbourhood. Note that we might be able to create multiple neighbours in which we reassign O_{ij}. For instance, it might be possible to reassign O_{ij} to different sets of resources. However, to limit the size of a neighbourhood, we choose to create only one neighbour for each operation.

When creating a neighbour, we use RBRS at different decision moments to choose among different possible options. For instance, we create assignment neighbours by reassigning an operation to a new set of resources. We use RBRS to choose the new resources to which we assign the operation.

We now discuss in detail the procedures we have used to create the different neighbourhood structures:

Procedure to create neighbourhood $N_{assign}{}^{1}(q)$

The neighbourhood $N_{assign}{}^{1}(q)$ contains the schedules where one bottleneck operation O_{ij} is removed from one resource and is reassigned to a new resource to replace the resource removed. Subsequently, O_{ij} is given new positions on all resources to which it is assigned. The bottleneck operation O_{ij} must have a critical resource predecessor or a critical resource successor. The main advantage of this neighbourhood structure is that it is easy to create:

To create a neighbour, we only need to choose one operation to reassign and we only need to reassign this operation to one new resource.

We restrict the operation O_{ij} to those bottleneck operations with either a critical resource predecessor or a critical resource successor for the following reasons:

- If O_{ij} has a critical resource predecessor, then it might be scheduled earlier in time if it is reassigned to a different resource. To schedule O_{ij} earlier in time, O_{ij} must be removed from the resource on which it has the critical resource predecessor.
- If O_{ij} has a critical resource successor, then this successor might be scheduled earlier in time if O_{ij} is reassigned to a different resource. To schedule the critical resource successor earlier in time, O_{ij} must be removed from the resource on which it has the successor.

We define the *bottleneck resource demand* as that resource demand that is satisfied by the resource on which the bottleneck operation has the critical resource predecessor or successor.

There may be multiple suitable resources to which we can reassign O_{ij}. As we mentioned before, we use RBRS to choose a new resource among the suitable resources. We assign each suitable resource a weight equal to the amount of free capacity that the resource has over the planning horizon. The higher the amount of free capacity a resource has, the greater the regret value given to that resource. We assign higher regret values to resources with more free capacity, because we think that, in general, it should be easier to schedule an operation on time on a resource with a lot of free capacity available compared to a resource with little free capacity available[5]. We calculate the free capacity of a resource as follows: The free capacity is the total uptime of the resource over the planning horizon minus the processing times of the operations scheduled on the resource during the planning horizon. Here, the amount of uptime is equal to the length of the planning horizon minus the periods of preemptive and non-preemptive downtime during the planning horizon. We calculate the free capacity over the entire planning horizon for simplicity reasons: The total uptime of a resource is fixed for a scheduling problem. Therefore, we only need to determine the sum of the processing times of the operations scheduled on the resource to determine the free capacity. The disadvantage, however, of calculating the free capacity over the total planning horizon is that we do not consider the interval in which we actually wish to schedule the operation. As a result, we may choose a resource that has a lot of free capacity over the planning horizon, but that has very little free capacity in the interval in which we actually wish to schedule the operation.

After choosing a new resource to assign O_{ij} to, we need to determine positions for O_{ij} on all resources to which it is assigned. In other words: We not only determine a position for O_{ij} on the new resource, but we also determine new positions for O_{ij} on the other resources to which the operation is assigned. We determine new positions for O_{ij} on all resources to increase the probability of obtaining a good schedule: Once we have determined a position

[5] Note however, that we do not necessarily obtain a better starting time for an operation if we assign that operation to a resource B with more free capacity compared to resource A with less capacity. This may be the case if the resource with more free capacity also has a number of non-preemptive downtimes. In such a case, it may be difficult to schedule an operation on time, because we are not allowed to schedule the operation over the non-preemptive downtimes.

for O_{ij} on its new resource, it might no longer be optimal to give O_{ij} the same positions on the remaining resources as before. Therefore, we also determine whether the positions of O_{ij} on the remaining resources need to be adjusted. To increase the probability of obtaining a neighbour with a better objective value than the current solution, we resequence the operation such that the operation obtains an earlier starting time in the neighbour than in the current solution. This is especially important for those operations that directly determine the tardiness of the schedule: These operations must be scheduled earlier in time to reduce the schedule's tardiness. As we mentioned in the preliminaries, however, we wish to obtain neighbours by making minimal changes to the current solution. Therefore, we do not wish to give the operation a much earlier starting time than it has in the current solution. For this reason, we choose to give the operation the latest position on each resource that still enables the operation to start earlier than in the current solution.

We use the following procedure to create a schedule q' in $N_{assign}{}^{1}(q)$:

1. Pick one bottleneck operation O_{ij} with either a critical resource predecessor or a critical resource successor.
2. Find a new resource assignment $A_{q'}(O_{ij}, M_{q'})$ for O_{ij}:
    a) Find a resource demand U_{ijmk} of O_{ij} that can be satisfied by a different set of resources than the set to which O_{ij} is currently assigned for that resource demand. Preferably, a bottleneck resource demand should be chosen. If there is no resource demand that can be satisfied by different resources, we do not consider the neighbour any further. In that case, we do not include the neighbour in $N_{assign}{}^{1}(q)$.
    b) Remove O_{ij} from one resource currently used to satisfy U_{ijmk} and replace this resource with a resource from R_{ijmk} not currently assigned to O_{ij}. If U_{ijmk} is a bottleneck resource demand, O_{ij} must be removed from the resource on which it has the critical resource predecessor or successor. Otherwise, O_{ij} must be removed from the resource with the least amount of free capacity. RBRS is used to choose the new resource to which O_{ij} is assigned.
3. Determine new positions $W_{q'}(O_{ij}, M_{q'}, A_{q'})$ for O_{ij} on the resources in $A_{q'}(O_{ij}, M_{q'})$:
    a) Find the latest operation O_{hl} scheduled on a resource in $A_{q'}(O_{ij}, M_{q'})$ that has a smaller completion time in q than the starting time of O_{ij} in q. Insert O_{ij} directly behind O_{hl} on all resources to which both operations are assigned.
    b) Determine positions for O_{ij} on the other resources in $A_{q'}(O_{ij}, M_{q'})$. The job predecessors and new resource predecessor of O_{ij} must be considered when determining these positions:
        i. Determine the earliest starting time S_{ij}^{temp} for O_{ij} in q' if we only consider the job predecessors of O_{ij} and the new resource predecessor O_{hl}.
        ii. Insert O_{ij} on each remaining resource directly behind the latest operation with a smaller starting time in q than S_{ij}^{temp}.

4. Calculate starting and completion times for all operations in the schedule.

We now use the following example to demonstrate the procedure:

We have a schedule that includes operations belonging to six different jobs. In this example, operation O_{12} is the only tardy operation. The schedule is given in Figure 7.2:

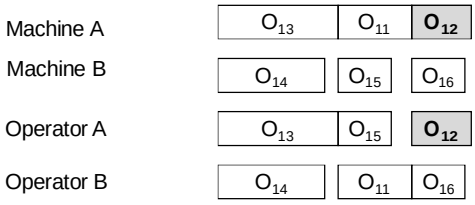

Figure 7.2: An example schedule

In this schedule, $COS(O_{12})$ is the set $\{O_{13}, O_{11}\}$. We assume that all operations may start at time zero.

We obtain an assignment neighbour q' as follows:

1. *The bottleneck operations (i.e. the set $\{O_{12}, O_{13}, O_{11}\}$) either have a critical resource predecessor or a critical resource successor. Therefore, they are all suitable candidates for reassignment. In this case, we choose operation O_{12} for reassignment.*
2. *Determination of $A_{q'}$:*
   a) *Operation O_{12} has two resource demands: A machine and an operator. Both resource demands can be satisfied by a different resource. Since the machine is the bottleneck resource demand, we reassign O_{12} to a new machine.*
   b) *There is only one suitable candidate for the new resource: Machine B. We thus remove O_{12} from machine A and we reassign O_{12} to machine B.*
3. *Inserting O_{12} on its resources:*
   a) *In this instance, operation O_{hl} is O_{15}: O_{15} is the last resource with a completion time that is smaller than the starting time of O_{12} in q. We thus insert O_{12} behind O_{15} on both resources.*
   b) *We have assigned O_{12} a position on both machine B and operator A. Since O_{12} is not assigned to other resources, we are done.*
4. *Determine starting and completion times for all operations as discussed in Section 5.2.*

In Figure 7.3, we can see the resulting assignment neighbour:

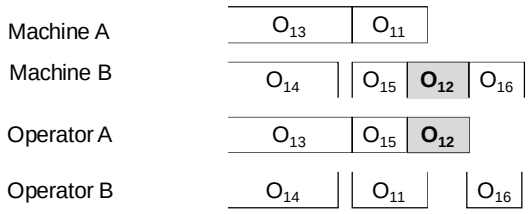

Figure 7.3: The neighbour obtained with the procedure

Procedure to create neighbourhood $N_{assign}{}^{2}(q)$

The neighbourhood $N_{assign}{}^{2}(q)$ contains the schedules where one bottleneck operation O_{ij} is removed for each resource demand from one resource and is reassigned to a new resource for each resource that has been removed. Subsequently, O_{ij} is given new positions on all resources to which it is assigned. As in $N_{assign}{}^{1}(q)$, the bottleneck operation O_{ij} must have a critical resource predecessor or critical resource successor. With this neighbourhood we try to remove an operation from all resources on which it has a critical resource predecessor: We expect that an operation has for each resource demand at most one resource on which the operation has a critical resource predecessor. Therefore, by reassigning the operation to a new resource for each resource demand, we ensure that the operation is removed from all resources on which it has a critical resource predecessor. In neighbourhood $N_{assign}{}^{1}(q)$, in contrast, it is not possible to remove an operation from all resources on which it has a critical resource predecessor or successor, because the operation is removed from only one resource.

We choose to reassign the operation to a new resource for each resource demand instead of only the bottleneck demands of the operation. We do so for simplicity reasons: Now, we do not need to first determine which resource demands contain a bottleneck resource. Instead, we try to reassign the operation to a new resource for each resource demand. We then ensure that we have also reassigned the operation to a new resource for all bottleneck resource demands.

As in neighbourhood $N_{assign}{}^{1}(q)$, there may be multiple suitable resources to which we can assign O_{ij} to replace a resource from which O_{ij} has been removed. We again use RBRS to choose a resource among the suitable resources. As in $N_{assign}{}^{1}(q)$, we give each suitable resource a weight equal to its free capacity over the planning horizon.

We use the following procedure to create a schedule q' in $N_{assign}{}^{2}(q)$:

1. Pick one critical operation O_{ij} with either a critical resource predecessor or a critical resource successor.
2. Find a new resource assignment $A_{q'}(O_{ij}, M_{q'})$ for O_{ij}:
    a) Repeat the following for each resource demand U_{ijmk} of O_{ij} that can be satisfied by a different set of resources than the set assigned to O_{ij} at this time (i.e. the resource was not assigned to O_{ij} in q and is not in the set $A_{q'}(O_{ij}, M_{q'})$):
        - Remove O_{ij} from one resource currently used to satisfy U_{ijmk} and replace this resource with a resource from R_{ijmk} not assigned to O_{ij} at present. If U_{ijmk} is a bottleneck resource demand, O_{ij} must be removed from the resource on which it has the critical resource predecessor or successor. Otherwise, O_{ij} must be removed from the resource with the least amount of free capacity. RBRS is used to choose the new resource to which O_{ij} is assigned.

b) If O_{ij} does not have a resource demand that can be satisfied by different resources, we do not consider the neighbour any further. In that case, we do not include the neighbour in $N_{assign}{}^{2}(q)$.

3. We use the same procedure $f_{assign}(A_{q'})$ to determine $W_{q'}$ as when we create a neighbour for $N_{assign}{}^{1}(q)$.
4. Calculate starting and completion times for all operations in the schedule.

We now use a new example to demonstrate the procedure to create a neighbour for $N_{assign}{}^{2}(q)$:

In this example, we again have a schedule containing operations from six different jobs. This example is slightly different from the example in Figure 7.2. Figure 7.4 presents the new example schedule:

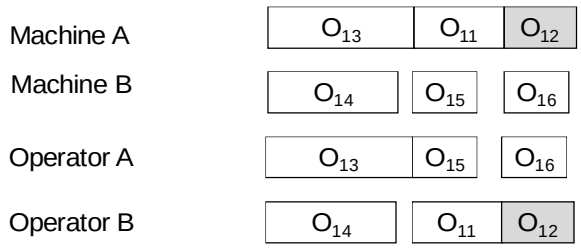

Figure 7.4: An example schedule

As in the example in Figure 7.2, O_{12} is the tardy operation. $COS(O_{12})$ is also the same, namely the set $\{O_{13}, O_{11}\}$. In addition, each operation may again start at time zero.

In this case, we obtain a neighbour as follows:

1. *As in Figure 7.2, we have three suitable operations for reassignment: O_{12}, O_{11}, and O_{13}.We again focus on reassigning operation O_{12}.*
2. *Reassignment of O_{12} to new resources:*
   a) *Operation O_{12} again has two resource demands: A demand for a machine and a demand for an operator. Both demands can be satisfied by a different resource.*
   b) *For both resource demands there is only one suitable candidate. We remove O_{12} from the old resources and we reassign O_{12} to machine B and operator A.*
3. *Inserting O_{12} on its resources:*
   a) *Again, operation O_{hl} is O_{15}. We insert O_{12} behind O_{15} on machine B and operator A.*
   b) *Since we have determined a position for O_{12} on both resources, we are done.*
4. *Determine starting and completion times for all operations as discussed in Section 5.2.*

In Figure 7.5, we see the resulting assignment neighbour:

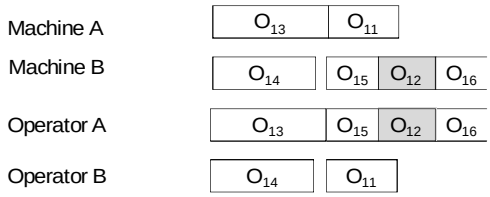

Figure 7.5: The neighbour obtained with the procedure

Procedure to create neighbourhood $N_{seq}(q)$

We create schedules in $N_{seq}(q)$ by choosing a bottleneck operation O_{ij} and swapping O_{ij} with an operation O_{hl} that is assigned to a resource on which O_{ij} has a critical resource predecessor and that is scheduled earlier than O_{ij} on this resource. After swapping O_{hl} and O_{ij} on the resources on which both operations are scheduled, each operation is also given new positions on the other resources to which the operation is assigned. The bottleneck operation O_{ij} has exactly one critical predecessor and this predecessor must be a critical resource predecessor. The advantage of this neighbourhood structure is its simplicity: Now, we only need to consider the influence of two operations on a schedule. The more operations we resequence at a time, the more effort we must put into finding a feasible schedule. This is especially the case for the general scheduling problem, where operations may require multiple resources to be performed: As we mentioned in the preliminaries, we must ensure that if two operations are scheduled on the same resources, that the one operation is always scheduled before the other operation on the common resources. Otherwise, we do not obtain a feasible schedule. When we need to resequence many different (multi-resource) operations, we must check for several operations whether they have feasible positions on their resources. This can be time-consuming.

We only consider operations that have exactly one critical resource predecessor. If O_{ij} does not have a critical resource predecessor, then we cannot decrease the starting time of O_{ij} by swapping O_{ij} with an operation that has been scheduled before O_{ij}. As a result, we do not choose operations that have critical job predecessors or operations that have no critical predecessors. In addition, we do not consider operations that have multiple critical resource predecessors. If we have an operation with multiple critical resource predecessors, we must swap the operation with each predecessor for the operation to obtain an earlier starting time. To limit the number of swaps that need to be performed, we limit the number of critical resource predecessors that an operation may have. We, however, do not expect that the number of possibilities for O_{ij} is drastically reduced if we limit O_{ij} to those operations with only one critical resource predecessor: We assume that the probability is small that an operation can have two resource predecessors that finish at the exact same time.

We swap O_{ij} with an operation O_{hl} that is sequenced before O_{ij} to ensure that O_{ij} (i.e. the bottleneck operation) obtains an earlier starting time. As a result, we might decrease the schedule's tardiness. We limit the possibilities for operation O_{hl} to those operations with a completion time in the current schedule q that is greater than the internal release date of O_{ij} in q. The internal release date of O_{ij} specifies the minimum starting time that the operation

can have based on the job predecessors of O_{ij}. Therefore, if we swap O_{ij} with an operation that has a completion time before the internal release date of O_{ij}, we unnecessarily give the operation a later starting time: O_{ij} cannot start before its internal release date, so it cannot be (partially) scheduled where the operation used to be scheduled.

If we can choose from multiple operations for O_{hl}, we use RBRS to choose among these operations. We assign each operation a weight equal to the starting time of the operation in q. In this case, an operation will obtain a higher regret value if it has a higher starting time. As a result, the critical resource predecessor of the operation will obtain the highest probability of being chosen. We choose to give higher probabilities to operations that are scheduled in proximity of operation O_{ij}, because we think that a swap is less disruptive if it takes place between two operations that are scheduled in proximity of each other compared to two operations that are scheduled further away from each other. If, for instance, we choose to swap non-adjacent operations, we can expect that the operations scheduled between these two operations will also obtain different starting and completion times. In contrast, if we schedule two adjacent operations, we expect less other operations to be influenced by the swap.

We use the following procedure to create a neighbour q' in $N_{seq}(q)$:

1. Pick one bottleneck operation O_{ij} that has one critical resource predecessor.
2. Pick an operation O_{hl} to swap O_{ij} with:
    a) Develop a decision set D for O_{hl} consisting of operations that are scheduled on the same resource as O_{ij} and its critical resource predecessor and whose completion times are greater than the internal release date of O_{ij}.
    b) Assign each operation in D a weight equal to its starting time in q and pick an operation from D by using RBRS.
3. Determine $W_{q'}$:
    a) Swap operations O_{ij} and O_{hl} on the set S of common resources: O_{ij} obtains the position of O_{hl} and O_{hl} obtains the position of O_{ij}.
    b) Determine new positions for O_{ij} on the remaining resources to which O_{ij} is assigned. In determining these positions, the job predecessors and new resource predecessors[6] of O_{ij} on the resources in S must be considered:
        - Determine the earliest starting time S_{ij}^{temp} for O_{ij} if we consider the internal release date of O_{ij} and the new resource predecessors of O_{ij}.
        - Insert O_{ij} on each resource R_r ($R_r \in A_{q'}(O_{ij}, M_{q'}) - S$) directly behind the latest operation with a smaller starting time in the current schedule q than S_{ij}^{temp}.
    c) Determine new positions for O_{hl} on the remaining resources to which O_{hl} is assigned. This happens in a similar way to O_{ij}: S_{hl}^{temp} is now equal to the

[6] The new resource predecessor of O_{ij} on resource R_r ($R_r \in S$) is the operation that was scheduled before O_{hl} on R_r in the current schedule q.

earliest starting time for O_{hl} if we consider the internal release date of O_{hl} and the new resource predecessors of O_{hl} (i.e. the resource predecessors of O_{ij} in q). Based on this starting time, O_{hl} obtains new positions on its remaining resources in the same way as O_{ij}.

4. Calculate starting and completion times for all operations in the schedule.

We use the example in Figure 7.6 to demonstrate how the procedure works:

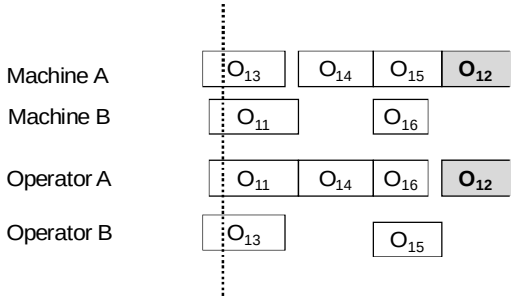

Figure 7.6: An example schedule

In Figure 7.6, O_{12} is our tardy operation. $COS(O_{12})$ is the set $\{O_{11}, O_{14}, O_{15}\}$. The dotted line in the figure indicates the internal release date of O_{12}.

With this schedule, we can obtain multiple different neighbours. We therefore illustrate how we can use the procedure to determine two different neighbours.

We obtain the first neighbour as follows:

1. *In this example, we have four bottleneck operations: O_{11}, O_{14}, O_{15}, and O_{12}. Of these operations, we can choose operations O_{14}, O_{15}, and O_{12} for resequencing. We choose to resequence O_{12}.*
2. *Determine O_{hl}:*
   a) *Operation O_{12} has one resource in common with its critical resource predecessor: Machine A. Our decision set thus consists of all operations scheduled before O_{12} on machine A with a completion time after the internal release date of O_{12}. In this case, the decision set is the set $\{O_{13}, O_{14}, O_{15}\}$.*
   b) *We choose to swap O_{12} with O_{13}. Note that O_{13} does not belong to $COS(O_{12})$.*
3. *Determine $W_{q'}$:*
   a) *We swap operations O_{12} and O_{13} on machine A, which is the only common resource. Operation O_{12} is scheduled as the first operation and O_{13} is scheduled behind O_{15}.*
   b) *We need to determine a new position for O_{12} on operator A. S_{12}^{temp} is equal to the internal release date of O_{12}. Since operation O_{11} has a smaller starting time in q than the internal release date of O_{12}, we schedule O_{12} between O_{11} and O_{14} on this resource.*

c) *We need to determine a new position for O_{13} on operator B. S_{13}^{temp} is equal to the completion time of O_{15} in q. On operator B, operation O_{13} is therefore scheduled behind operation O_{15}.*

4. *Determine starting and completion times for all operations as discussed in Section 5.2.*

We see the resulting neighbour in Figure 7.7:

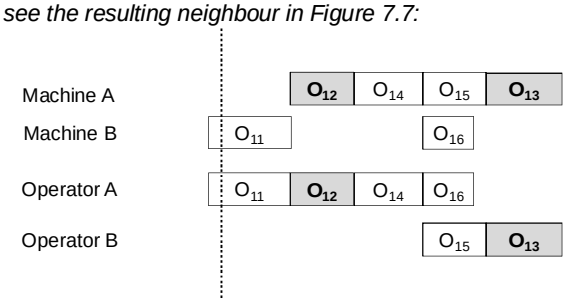

Figure 7.7: A possible sequencing neighbour.

We obtain a second neighbour as follows:

1. *We choose O_{15} as operation O_{ij}.*
2. *The choice for operation O_{hl}:*
   a) *Operation O_{15} also has machine A as the common resource with its critical predecessor O_{14}. Assuming that operation O_{15} has the same internal release date as operation O_{12}, the decision set is the set $\{O_{13}, O_{14}\}$.*
   b) *We choose to swap O_{15} with its critical resource predecessor O_{14}.*
3. *Determine $W_{q'}$:*
   a) *We swap operations O_{15} and O_{14} on machine A: Operation O_{15} gets the position behind O_{13} and O_{14} is scheduled between operations O_{15} and O_{12}.*
   b) *We determine a new position for O_{15} on operator B: S_{15}^{temp} is equal to the completion time of O_{13}.Therefore, O_{15} is scheduled behind O_{13} on operator B.*
   c) *We determine a new position for O_{14} on operator A: S_{14}^{temp} is equal to the new completion time of O_{15}. The latest operation with a smaller starting time in q than S_{14}^{temp} is operation O_{11}. Therefore, we schedule O_{14} between O_{11} and O_{16}.*
4. *Determine starting and completion times for all operations as discussed in Section 5.2.*

In this case, we obtain the neighbour in Figure 7.8:

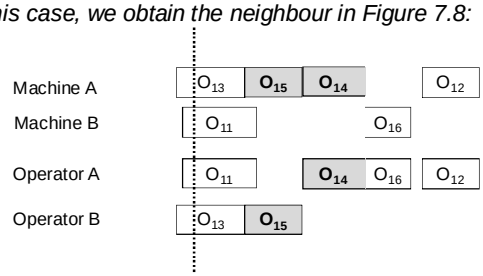

Figure 7.8: A second sequencing neighbour.

Note that our assignment neighbourhoods are not suitable when we use the scheduling approach to solve a scheduling problem with routing constraints. If a problem contains

routing constraints, we might need an assignment neighbourhood where we reassign multiple operations in one neighbour to different resources. The assignment neighbourhoods that we use, however, only reassign one operation to new resources. Consider the example in Figure 7.9:

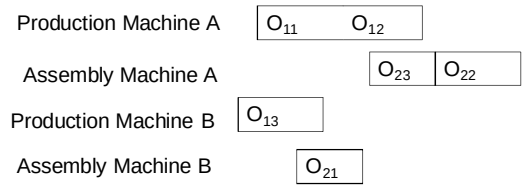

Figure 7.9: An example schedule

In the example, there is a connection from production machine A to assembly machine A. This connection is also present between the machines of type B. It is however forbidden to schedule an operation on production machine A and its job successor on assembly machine B. Similarly, the set {Production machine A, Assembly machine B} is also a forbidden set. As we can see, we can obtain a better schedule by reassigning operation O_{12} to production machine B and operation O_{22} to assembly machine B. However, we cannot reach this solution, because we obtain an infeasible solution by only reassigning one operation at a time.

Implemented versions of the hierarchical approach

We have created four versions of the hierarchical approach. In this section, we describe each version. First, we describe the general idea behind a version. Subsequently, we give details on each version.

As we previously mentioned, we have created two different assignment neighbourhoods: $N_{assign}{}^{1}(q)$ and $N_{assign}{}^{2}(q)$. In versions 1 and 2 of the hierarchical approach, we only use assignment neighbourhood $N_{assign}{}^{1}(q)$ when solving the assignment problem. The remaining versions of the approach we test with both neighbourhoods: We first perform a number of tests with assignment neighbourhood $N_{assign}{}^{1}(q)$ and then we perform the same tests with assignment neighbourhood $N_{assign}{}^{2}(q)$. We do so to compare the different neighbourhoods to each other and determine which neighbourhood is best for the scheduling instances that we test.

When describing the different versions of the hierarchical approach, we use a different general structure than the one we described in Section 7.3. We do so for easier explanation of the different versions.

Hierarchical approach version 1

In version 1 of the hierarchical approach, we solve both the assignment and the sequencing problem by using the iterative improvement algorithm. We solve both subproblems in the same way: In each iteration, we create a neighbourhood of the current solution and we move to the solution in the neighbourhood with the lowest tardiness value if this solution has a lower tardiness than the current solution. We continue in this way until we cannot find a neighbour with a lower tardiness value than the current solution.

After we solve the assignment problem, we use the best schedule found so far as the initial current solution when solving the sequencing problem. Once we have solved the sequencing

problem for the given assignment, we exit the hierarchical approach. This version of the hierarchical approach thus uses a one-way communication scheme: We do not use the best solution found in the sequencing problem to resolve the assignment problem.

We use the integrated approach to obtain an initial solution. The main advantage of using the integrated approach is that the approach enables us to easily create different initial solutions by varying bias parameter α that we use in the RBRS algorithm and the number of iterations that we perform of the approach.

With this version of the hierarchical approach we hope to get an impression of the influence of the different subproblems on the objective value of a schedule. In other words, we hope to determine whether the method to solve the assignment problem or the method to solve the sequencing problem has most influence on the objective value of a schedule. Then, we can determine whether we should put more effort in finding a schedule with a good assignment or a schedule with a good sequence.

In this version, the hierarchical approach works as follows:

- Create an initial schedule by using the integrated approach. This schedule becomes the current schedule q.
- **PERFORM** the steps of the hierarchical approach once:
  - 1 + 2. Solve the assignment problem:
    - a) Repeat the following steps until no assignment neighbour can be found with a lower tardiness value than the current schedule q:
      - i). Create assignment neighbourhood $N_{assign}{}^{1}(q)$ of q.
      - ii). Choose the neighbour with the lowest tardiness value. Accept this neighbour if it has a lower tardiness than q. The neighbour then becomes the new current schedule.
  - 3 + 4. Solve the sequencing problem for the best schedule found so far:
    - a) Repeat the following steps until no sequencing neighbour can be found with a lower tardiness value than q:
      - i). Create sequencing neighbourhood $N_{seq}(q)$ of q.
      - ii). Choose the neighbour with the lowest tardiness. Accept this neighbour if it has a lower tardiness than q. The neighbour then becomes the new current schedule.

Hierarchical approach version 2

In version 2 of the hierarchical approach, we again solve both subproblems by using iterative improvement. However, we now solve the subproblems in a different way: We solve the assignment problem by moving to the first neighbour in the assignment neighbourhood that has a lower tardiness than the current solution. Once we have moved to a new neighbour (or we do not have a neighbour with a lower tardiness than q), we are done: We then exit the method for solving the assignment problem. We solve the sequencing problem in a similar way.

After we have solved one subproblem, we use the best schedule found so far as input to solve the next subproblem. In other words: After we have solved the assignment problem (i.e. we have either found a better assignment neighbour or we have not been able to find any assignment neighbour that was better than the current solution), we use the best schedule found so far as the initial solution when solving the sequencing problem and vice versa. We continue to iterate between the assignment step and the sequencing step until we are no longer able to find either an assignment neighbour or a sequencing neighbour with a

better solution value than the current solution. Note that we use a two-way scheme in this version of the hierarchical approach: Once we have solved the sequencing problem, we use the best solution found so far as the new initial solution and we resolve the assignment problem for this initial solution.

With this version of the hierarchical approach we can determine whether we obtain better results by using a two-way communication scheme as opposed to a one-way scheme. To do so, we compare the schedules obtained with version 1 of the hierarchical approach with the schedules obtained with this version.

The approach works as follows:

- Create an initial schedule q by using the integrated approach.
- **REPEAT** the following steps of the hierarchical approach:

  1 + 2. Solve the assignment problem:

  a) Create assignment neighbourhood $N_{assign}^{1}(q)$.

  b) Choose the first neighbour in $N_{assign}^{1}(q)$ with a lower tardiness value than q. This neighbour becomes the new current solution.

  3 + 4. Solve the sequencing problem for the best schedule found so far:

  a) Create sequencing neighbourhood $N_{seq}(q)$.

  b) Choose the first neighbour in $N_{seq}(q)$ with a lower tardiness than q. This neighbour becomes the new current solution.

  UNTIL we cannot find a neighbour in either $N_{assign}^{1}(q)$ or $N_{seq}(q)$ with a lower tardiness than the current solution

Hierarchical approach version 3

In versions 1 and 2 of the hierarchical approach, we only consider a neighbour to be interesting if the neighbour has a lower tardiness value than the current schedule. As we mentioned in Chapter 6, a disadvantage of only considering neighbours that are better than the current schedule is that we can remain stuck in local optima. In this version of the hierarchical approach we therefore use an algorithm that enables us to escape local optima when we solve the assignment problem. When we solve the sequencing problem, however, we still only consider a neighbour if the neighbour is better than the current schedule.

In this version, we solve the assignment problem by using the simulated annealing (SA) algorithm. As we mentioned in Section 6.1, this algorithm enables us to escape local optima. We prefer this algorithm over the tabu search (TS) algorithm, because SA does not require us to create an entire neighbourhood for the current schedule: In SA, we only need to create one neighbour at a time and we compare this neighbour to the current schedule. In TS, in contrast, we need to create all schedules in the neighbourhood of the current schedule before we can compare the neighbours to the current schedule.

As we previously mentioned, when we solve the sequencing problem we still only consider a neighbour if the neighbour has a better solution value than the current schedule. We do so, because we think that we can reach a greater part of the solution space by reassigning operations to different resources than by resequencing operations on their resources: We assume that we can only create a limited number of different sequencing neighbours, because each bottleneck operation has a limited number of operations with which it can be swapped. If the number of resources is large, in contrast, we expect that we can create several different assignment neighbours where a bottleneck operation is assigned to different resources. In addition, we also determine a new sequence when we create an assignment neighbour. Therefore, we choose to use the method for solving the assignment problem as

the main method to examine the solution space. We then use the method for solving the sequencing problem to determine if we can further improve the schedules we have found when solving the assignment problem.

Once we stop the search for schedules with a different assignment than the current solution, we take the best found solution as starting point for the sequencing phase. Note that the last considered assignment neighbour may be much worse than the best solution found. We choose to solve the sequencing problem for the best solution found so far, because we only accept a neighbour in the sequencing phase if this neighbour has a lower tardiness than the current solution. Therefore, we only solve the sequencing problem for the best solution found so far to see if this scheduled can be improved further in the sequencing step.

In this version of the approach, we again use a two-way communication scheme: We use the best schedule we have found when solving the sequencing problem as the initial current schedule when solving the assignment problem. However, in this version of the hierarchical approach we only iterate once from the method for solving the sequencing problem to the method for solving the assignment problem. In other words: Once we exit the method to solve the sequencing problem for a second time, we exit the hierarchical approach. We thus solve each subproblem twice in this version of the hierarchical approach. We choose to solve the subproblems twice so we are still able to determine a new assignment for the schedule after having solved the sequencing problem the first time. To limit the calculation time of the approach, we do not use a more extensive communication scheme (i.e. a scheme where more frequent iteration takes place between the two substeps).

The approach works as follows:

- Create an initial schedule q by using the integrated approach.
- **REPEAT** the following steps twice:
  - 1 + 2. Solve the assignment problem by using the Simulated Annealing algorithm with q as initial solution.
  - 3 + 4. Solve the sequencing problem for the best solution found so far: Repeat the following steps until no sequencing neighbour can be found with a better tardiness than the current solution.
    a) Create sequencing neighbourhood $N_{seq}(q)$.
    b) Choose the first neighbour in $N_{seq}(q)$ with a lower tardiness value than q. This neighbour becomes the new current solution q.

Hierarchical approach version 4

In version 4 of the hierarchical approach, we choose to solve both the assignment and the sequencing problem by using the SA algorithm. As a result, we now also have the ability to escape local optima when solving the sequencing problem.

With this version, we can determine whether better results can be obtained if we also accept worse solutions than the current solution when we solve the sequencing problem. To determine this, we compare the results obtained with version 3 to those obtained with this version.

When we solve the sequencing problem in this version, we choose to store the best schedule we have found so far. This schedule is the one that we use as the initial current schedule when we solve the assignment problem. As in version 3, we choose to exit the hierarchical approach once we exit for the second time the method for solving the sequencing problem.

We now describe how the approach works. Since this version of the hierarchical approach is very similar to version 3, we only specify those steps that differ from version 3 of the hierarchical approach.

The approach is as follows:

- We create an initial solution with the integrated approach. This initial solution becomes the current schedule q.
- **REPEAT** the following steps twice
  - 1 + 2. As in version 3, we use Simulated Annealing to solve the assignment problem. Again, we use q as the initial solution.
  - 3 + 4. We use Simulated Annealing to solve the sequencing problem. The best solution found so far serves as the new initial solution q.

8. Evaluation of the scheduling approaches

In this section, we describe the results of the tests that we have performed with the different scheduling approaches. In Section 8.1, we first describe the scheduling instances on which we have tested the approaches. In Section 8.2, we subsequently give the parameters that we have used to test the different scheduling approaches. Next, we discuss the results of the tests. In Section 8.3 we first discuss the performance of the integrated approach with respect to tardiness minimization. Subsequently, we discuss the performance of the hierarchical approach in Section 8.4. In Section 8.5 we discuss the performance of the approaches on calculation time and other performance indicators than total tardiness. Finally, we give some conclusions based on the test results in Section 8.6.

Before we proceed with the remainder of the chapter, we first discuss a number of assumptions that we made when implementing the approaches. These assumptions are:

1. We assumed that all resources have the same calendar. As a result, all resources have the same intervals of preemptive downtime. This assumption makes it easier to determine starting and completion times for multi-resource operations: We can be sure that an operation can be scheduled at a certain time t on all resources in a set S if that operation can be scheduled on each individual resource in S at time t. If each resource has a different calendar, however, the calculation becomes more time-consuming. Then, it becomes possible that an operation can be scheduled on each resource in S at time t, but that it cannot be scheduled on al resources in S simultaneously at that time[7]. In that case, we thus need to consider all resources in S simultaneously to determine if an operation can be scheduled at a certain time.
2. We assumed that the job predecessors of fixed operations are fixed as well. If we do not make this assumption, we might introduce deadlines in the schedule. As a result, we might not be able to find a feasible schedule, because we are not able to schedule job predecessors of the fixed operation before the starting time of the fixed operation.

8.1. Tested scheduling instances

In this section, we describe the scheduling instances on which we tested the different approaches. We have tested the approaches on 10 different instances.

We created the instances in the following way: At present, ORTEC has a scheduling instance that it uses to demonstrate PLANWISE to potential customers. This instance consists of 25 jobs that need to be scheduled. Each job consists of a set of operations, where the number of operations per job varies between 1 and 5. When a job consists of 5 operations, the operations need to be performed in the sequence given in Figure 8.1:

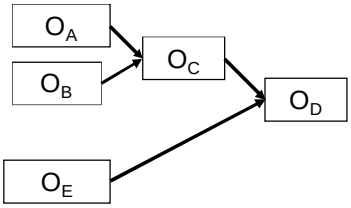

Figure 8.1: The sequence of operations when a job consists of 5 operations.

As we can see in Figure 8.1, certain operations have two job predecessors. The operations of the remaining jobs in the schedule need to be performed in a simple chain, e.g. for 3 operations the sequence is as given in Figure 8.2:

[7] In Figure 6.1, we present an example to demonstrate this issue.

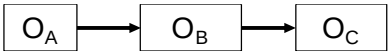

Figure 8.2: The sequence of operations when a job has 3 operations.

Based on this instance, we have created 4 instances consisting of 50 jobs, 4 instances consisting of 100 jobs, and 1 instance consisting of 150 jobs. For each instance, we create a set of jobs and resources in the following way:

1. We start with the set of jobs and resources as in the ORTEC instance with 25 jobs. This set of jobs is the initial set of jobs for the instance. Similarly, this set of resources is the initial resource set.
2. We then expand the initial set of jobs: By using a random generator, we create an additional set K' of jobs by making copies of a subset of the jobs in the initial set. We subsequently adjust each job in K' such that the job no longer resembles the job in the initial set from which it was copied. There are two ways in which we can adjust the jobs in K': First, each job in K' obtains a different release and due date. In addition, a subset of the jobs in K' also consists of a different set of operations: The processing times of the operations are adjusted or the set of operations belonging to the job is adjusted.
3. For a subset of the instances, we also expand the initial set of resources. We do so by creating additional resources of a specific type. For example, in instances with more jobs we might specify that the jobs can be performed on five machines of type X instead of the four machines of type X included in the initial set. Note that the additional resources need not be exactly the same as the resources in the initial set. For instance, if we have a machine of type X in the initial set that can process operations in two different speeds, we may include a new machine of type X that can only process operations in one of those speeds.

As a result of these steps, the sets of jobs and resources are specific for each instance. In addition, each instance has its own specific set of constraints. The following constraints are instance-specific:

1. Precedence relations: Each operation has a set of job predecessors and job successors. However, the set of job predecessors and job successors of each operation is instance-specific.
2. Resource-eligibility constraints: In all instances, the set R_{ijmk} specifies the suitable resources for resource demand U_{ijmk} of operation O_{ij}. However, the resources included in the set R_{ijmk} differ per instance. For example, resource R_r may be suitable to satisfy U_{ijmk} in instance A, while it is not suitable for U_{ijmk} in instance B.
3. Downtimes: Resources in all instances contain periods of preemptive downtime. However, the intervals of downtime differ per instance. In addition, certain instances also contain resources on which there are intervals of non-preemptive downtime.
4. Changeovers: In instance 100-1, no changeovers apply between any two adjacent operations. In the other instances, a changeover time C_{ijhlr} applies if operation O_{hl} is scheduled after operation O_{ij} on resource R_r.

Table 8.1 gives the details on each instance that we have created. In this table, we focus on the differences between the different instances. Therefore, we do not specify the restrictions that apply for all instances.

Instance	# jobs	# operations per job	# resources	Average # of resources for operation	Non-preemptive downtimes	Changeovers
25	25	1-5	25	2	-	X
50-1	50	1-5	25	2	-	X
50-2	50	1-7	27	2	-	X
50-3	50	1-5	28	2	-	X
50-4	50	1-5	28	2	X	X
100-1	100	1-7	34	2	-	-
100-2	100	1-5	38	2	-	X
100-3	100	1-8	30	2	X	X
100-4	100	1-5	28	2	-	X
150-1	150	1-5	44	2	X	X

Table 8.1: Details on each instance

In Table 8.1, the column 'Average # of resources for operation' specifies the average number of resources required to perform an operation. The columns 'non-preemptive downtimes' and 'changeovers' specify whether an instance contains non-preemptive downtimes or changeover times respectively: If an instance has an 'X', the constraint applies for that instance, otherwise it does not. Note that instances 50-2, 100-1 and 100-3 contain jobs with 7 or 8 operations. The operations of these jobs are performed in a network (e.g. certain operations in these jobs have multiple job predecessors).

We consider the created instances to different sufficiently for the purposes of this assignment: First, we have created instances with different numbers of jobs. In addition, the set of jobs also differs among the instances with the same number of jobs: The number of operations per job varies for the different instances as well as the processing times of the operations themselves. The resource requirements per operation also differ per instance. Finally, the set of constraints also differ per instance.

With these instances we are able to test whether the different scheduling approaches are suitable to solve production scheduling and task scheduling problems. The created instances are production scheduling instances: With the exception of instance 100-1, all instances have changeovers that need to take place between operations O_{ij} and O_{hl} if those operations are scheduled after each other on a specific resource. Thus, by testing the approaches on these instances, we can obtain an impression on the suitability of the approaches for production scheduling. As we mentioned before, task scheduling is a special case of production scheduling. As a result, we can also obtain an impression on the suitability of the approaches for task scheduling.

As we mentioned in Section 4.4, we do not focus on creating a scheduling approach for the equipment scheduling environment, because this environment differs in a number of ways from the production scheduling environment Since we do not focus on equipment scheduling, we did not create an equipment scheduling instance. As a result, we cannot test how the scheduling approaches perform on an equipment scheduling problem. In addition, we are not able to test how well the approaches can handle fixed operations: At present, we are not able to create scheduling problems in PLANWISE that contain fixed operations.

8.2 The parameters used for each approach

As we mentioned in Chapter 7, we created an integrated scheduling approach based on RBRS and different versions of a hierarchical scheduling approach. In this section, we specify the parameters that we use to test the different versions. We first discuss the

parameters for the integrated approach. Subsequently, we discuss the parameters for the hierarchical approach.

Parameters for the integrated approach

For the integrated approach, we use the RBRS algorithm. We therefore need to determine how many schedules we create (i.e. the number of iterations performed) and which values we will use for α. Kolisch and Drexl (1996) refer to the number of iterations performed as the sample size. Therefore, in the remainder of this document we also use this term.

The sample size required and the values for α are closely related: Kolisch and Drexl mention that the best value for α is a function of the sample size: If the sample size is 1, we obtain the best results if we use a large α value. As we mentioned in Section 6.1, a large α value leads to a deterministic choice for an item in the decision set, because the items with the highest regret values are also given the highest probabilities of being chosen. Therefore, the items that seem most promising are chosen in each step. Once the sample size increases, however, it becomes more beneficial to use smaller values for α. As mentioned in Section 6.1, the choice for an item in the decision set becomes more random if we use smaller values of α. Therefore, we are able to reach more solutions in the solution space if we use a small value of α compared to a large value of α. Thus, if we are able to perform many iterations, we should use a small value of α to obtain many different solutions.

We thus need to determine an α scheme that specifies which α value is most suitable for different sample size values. Kolisch and Drexl determined this scheme by solving each instance that they created with different combinations of sample size and α values. Table 8.2 specifies the different values for the sample size and α that the authors used:

Parameter	Value
Sample size	1, 5, 10, 50, 100
α	∞, 3, 2, 1, 0

Table 8.2: The values used by Kolisch and Drexl (1996) for sample size and α value.

The authors solved the different instances with all combinations of sample size and α value.

For this assignment, we use the same approach as Kolisch and Drexl. However, we test a more extensive set of combinations of sample sizes and α values than those that Kolisch and Drexl use. Table 8.3 specifies the different values that we test for each parameter. We also test the integrated approach with each combination of sample size and α value.

Parameter	Value
Sample size	1, 5, 10, 25, 50, 100, 250, 500
α	150, 100, 50, 25, 10, 5, 4, 3, 2, 1, 0

Table 8.3: The values tested for the parameters of the integrated approach.

As we can see in Table 8.3, we vary the values of α from 0 to 150. By using a value of 0, we emulate that items are chosen completely randomly from the decision set. We use a maximum α value of 150 to create solutions almost deterministically: When we use a α value of 150, only the items with the highest regret value obtain a probability of being chosen, whereas the remaining items obtain a probability close to 0 of being chosen. For instance, if there are two items with the highest regret value, then those items each have a probability close to 0.5 of being chosen, whereas the remaining items have a probability close to 0 of being chosen.

We vary the sample size from 1 to 500. We choose larger sample sizes than Kolisch and Drexl, because we focus on solving practical scheduling problems. We expect the solution

space for practical problems to be larger than for the scheduling problems discussed in literature. Therefore, we use larger sample sizes to ensure that we can reach a larger part of the solution space when we use small values for α.

To limit the number of tests that we need to perform, we test the different combinations on three instances. Based on these tests, we create an α scheme that we apply to the remaining instances when we use the integrated approach.

Parameters for the hierarchical approach

For the hierarchical approach we need to specify which parameter values we use for the integrated approach to obtain an initial solution. In addition, we need to specify which parameter values we use for the Simulated Annealing (SA) algorithm. In this section, we specify these different values.

In Section 7.4 we mentioned that we use the integrated approach to create an initial schedule for the hierarchical approach. To use the integrated approach, we need to specify the number of iterations that we perform and the values for α that we use. We choose to create multiple initial solutions so we are able to determine what influence, if any, the initial solution has on the performance of the hierarchical approach. There are several options to obtain different initial solutions: If we use a low value for α (for example, an α of 0), we expect to obtain a different solution each time we perform a new iteration of the integrated approach. In addition, we can also obtain different solutions by using a different sample size to create each solution (again for small α values). Finally, we can vary the value for α that we use: When we use a high value for α (for example, an α of 100), we expect the approach to choose the item with the highest regret from the decision set. When we use a low α, in contrast, the choice for an item is more random. Therefore, we expect to obtain a different solution when we use a low α value compared to a high α value. In this assignment, we use different values for α to obtain different initial solutions. To test the different instances, we use three different initial solutions. We use an α of 2, an α of 25, and an α of 100 to obtain the different solutions. We create each solution by performing one iteration of the integrated approach.

For SA, we need to determine the initial temperature, the decrease factor, the Markov Chain Length, and the end temperature. When determining the values for these parameters we look for a combination of parameters that gives good results while limiting the calculation time of the hierarchical approach. Therefore, we performed some initial tests on three of the created instances. Table 8.4 specifies for each parameter the different values that we have used during testing:

Parameter	**Values**
Initial temperature	10, 50, 100, 150
End temperature	0.5
Decrease factor	0.85, 0.9, 0.95, 0.99
Markov Chain Length	20, 30, 50

Table 8.4: The tested values of the Simulated Annealing parameters

We did not test all possible combinations of parameter values. For example, when we performed tests with an initial temperature of 100 or 150, we used a decrease factor of 0.85 and 0.9 and a Markov Chain Length of 20. We chose not to test all combinations to limit the amount of time spent on the initial tests. Therefore, we did not perform the tests that would require a lot of calculation time (i.e. the tests with high values for both initial temperature and decrease factor).

When we use an initial temperature of 150, the approach initially accepts 86% of the neighbours on average. We consider this to be a high acceptance ratio: Many neighbours are thus accepted initially. As a result, we can easily reach different areas in the solution space at the start of the approach. Similarly, with a temperature of 0.5 we accept very few, if any, neighbours: If a neighbour has one more day of tardiness than the current solution, that neighbour only has a probability of 13% of being chosen. If the difference in tardiness becomes two or greater, this probability decreases to 2% or less. At the end of the approach, we thus predominantly accept neighbours with a tardiness that is equal to, or smaller than, the current solution.

When we compared the results of the approach with different combinations of parameter values, we noticed that the calculation time increases as the values of the parameters increase. However, when we used high values for the parameters, the approach performs similar to, or even worse than, an approach with low parameter values. Therefore, we choose to use an initial temperature of 10 and a Markov Chain Length of 20 when testing the hierarchical approach on the remaining instances. For the decrease factor, however, we have not been able to specify which value gives the best results: Contrary to the other parameters, we sometimes obtained better results with a high decrease factor compared to a low decrease factor. Therefore, we use two values for the decrease factor when testing the different instances: 0.9 and 0.95. Table 8.5 specifies the parameter values that we use when we solve the remaining seven instances with the hierarchical approach.

Parameter	**Values**
Initial temperature	10
End temperature	0.5
Decrease factor	0.9, 0.95
Markov chain length	20

Table 8.5: The values of the Simulated Annealing parameters used for the hierarchical approach

As we mentioned in Section 7.4, we use SA to solve both the assignment and the sequencing problem in version 4 of the hierarchical approach. We have chosen to use the same parameter settings when solving both the assignment and the sequencing problem.

In Table 8.6, we specify the different versions of the hierarchical approach that we test. In addition, we specify which parameter values we test the versions with. Note that we only specify the α values that we use to create an initial solution and the decrease factor used. We do not specify the values for the other simulated annealing parameters, because these parameters only take on one value.

Approach version	**Algorithm used for assignment**	**Algorithm used for sequencing**	**Number of iterations**	**Decr. factor**	**α**
HA v1	Iterative improvement	Iterative improvement	1	N/A	2
HA v2	Iterative improvement	Iterative improvement	∞	N/A	2
HA v3-1	SA	Iterative improvement	2	0.95	2, 25, 100
HA v3-2	SA	Iterative improvement	2	0.9	2, 25, 100
HA v4-1	SA	SA	2	0.95	2, 25, 100
HA v4-2	SA	SA	2	0.9	2, 25, 100

Table 8.6: The different versions of the hierarchical approach

In Table 8.6, the column ‘Algorithm used for assignment’ specifies whether we use iterative improvement or simulated annealing to solve the assignment problem. Similarly, the column

'Algorithm used for sequencing' specifies which algorithm is used to solve the sequencing problem. The column 'Number of iterations' specifies how often the two subproblems are solved. In HA v1, for example, we solve both subproblems once (i.e. we use a one-way communication approach). In HA v2, in contrast, we keep iterating from one substep to the other until we are not able to find a neighbour with a better solution value than the current solution.

As we can see in Table 8.6, we only test versions 1 and 2 with the initial solution that we create by using an α of 2. The choice to use an α of 2 for these two versions is a random one: In versions 1 and 2 of the hierarchical approach we are less interested in the best schedule that the approach is able to find. For instance, with version 1 of the hierarchical approach we wish to investigate whether solving the assignment problem or solving the sequencing problem has most influence on the performance of the approach. With version 2 of the approach we wish to investigate whether it is beneficial to use a two-way communication scheme instead of a one-way scheme. In both versions we are thus predominantly interested in the amount by which we reduce the tardiness of the schedule and not in the actual tardiness value of the best schedule that we have found. Therefore, we can use any initial solution to test these versions of the approach.

As we can see in Table 8.6, we test versions 3 and 4 of the hierarchical approach with different decrease factors and different initial solutions. We thus test these two versions with six different parameter combinations. For clarity, we specify version 3 as version 3-1 when we test this version with a decrease factor of 0.95. Similarly, we refer to version 3 as version 3-2 when we use a decrease factor of 0.9.

In all versions of the hierarchical approach we use an α of 100 when we need to choose an item from the decision set in the methods that we use to create an assignment or sequencing neighbour.

8.3 Tardiness results for the integrated approach

In this section, we present the results obtained when using the integrated approach. As we mentioned in the previous section, we need an α scheme when we use to integrated approach to test the different instances. Therefore, we first give the results of the tests that we have performed to determine the α scheme. Then, we give the results on the performance of the integrated approach with the α scheme that we determined.

As we mentioned in the previous section, we use a similar approach as Kolisch and Drexl (1996) to determine the α scheme. Table 8.3 specifies the different values of sample size and α values that we test. As we mentioned in Section 7.2, we use a serial scheme to create decision sets. We subsequently assign to each operation in the decision set a weight that is equal to the latest starting time (LST) of that operation. Tables 8.7a and 8.7b specify the test results for instances 50-2 and 100-4 respectively. In the tables we furthermore specify the objective value of the best solution found (expressed in days).

	Sample size							
	1	**5**	**10**	**25**	**50**	**100**	**250**	**500**
Alpha = 150	199	118	118	118	111	105	105	102
100	151	151	144	117	117	109	109	109
50	192	149	145	128	128	124	124	124
25	213	199	159	158	153	137	137	137
10	275	186	171	171	154	154	136	136
5	192	192	181	181	143	143	143	143
4	218	218	213	159	159	150	150	140
3	198	198	180	174	174	155	155	155
2	204	204	204	179	179	149	149	149
1	242	242	242	195	195	193	193	173
0	426	356	356	351	332	285	283	283
Min. Value	**151**	**118**	**118**	**117**	**111**	**105**	**105**	**102**

	Sample size							
	1	**5**	**10**	**25**	**50**	**100**	**250**	**500**
Alpha = 150	338	241	205	152	152	144	141	132
100	339	235	155	155	155	154	146	113
50	220	171	171	171	160	145	141	131
25	270	212	212	169	169	163	163	140
10	289	282	198	198	198	198	198	198
5	462	409	343	343	343	323	285	285
4	435	435	435	426	362	362	306	306
3	686	588	455	455	446	446	363	363
2	666	585	585	443	443	443	443	431
1	830	657	578	560	560	560	450	449
0	1174	1078	965	902	808	808	803	665
Min. Value	**220**	**171**	**155**	**152**	**152**	**144**	**141**	**113**

Tables 8.7a and 8.7b: Test results for instance 50-2 and 100-4.

The results in Tables 8.7a and 8.7b are unexpected. As we mentioned in the previous section, we expected lower values for α to perform better than higher values for α when we increased the sample size. However, in Tables 8.7a and 8.7b we notice that the higher α values still perform the best for large sample sizes. We also notice that the quality of the solutions still varies greatly when we use a high value for α: Even when we use an α of 150 the objective values of the schedules in instance 100-4 differ greatly when we perform 500 iterations compared to 1 iteration. This is strange, because with such a high α we expected to obtain the same schedule in each iteration.

The unexpected results are caused by the specific data of the jobs that need to be scheduled in the different instances. For example, in instance 50-2 we can divide the jobs into two sets. The jobs belonging to the same set all have release and due dates that are close to each other. The operations of these jobs thus need to take place at approximately the same moment in time. The two sets, however, need to take place at different moments in time. In other words: The release date of a job in set B is much later in time than that of a job in set A. Figure 8.3 gives a simple representation of the instance. In the figure, the grey blocks indicate the jobs belonging to the different sets. The length of the block indicates the interval in which the job must be scheduled.

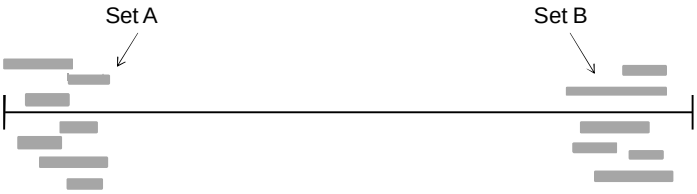

Figure 8.3: A simple representation of the jobs in instance 50-2.

As a result of the job sets in the instances, the serial scheme and LST rule no longer work as expected: As we mentioned in Section 6.1, the serial scheme does not consider the time in which operations need to be scheduled. Therefore, an operation decision set in instance 50-2 will contain operations from jobs belonging to both set A and set B. In addition, each operation in the decision set is given a weight equal to its latest starting time. As a result of these factors, the operations in set A will have very low weights while those of set B have very high weights. Because of the great difference in weights between the operations in the two sets, the scheduling approach is not able to distinguish well among the operations in the decision set that have the highest priority (i.e. the operations in set A): Because of the great difference in weights, the operations in set A all obtain high regret values. As a result, all operations in set A obtain a high similar probability of being chosen, even when we use a high value for α. In general, we obtain this problem when the difference in weights between the operations with high priority and those with low priority is large: If in this case, the scheduling approach cannot distinguish well among the operations with the highest priorities. Consider the following example:

We have a decision set consisting of three operations. Table 8.8 specifies two different scenarios with respect to the regret values of the operations:

Operation	Regrets in scenario 1	Regrets in scenario 2
1	10	300
2	9	299
3	0	0

Table 8.8: Two different scenario's with respect to regret values of operations in a decision set.

If we used an α of 100 in scenario 1, operation 1 would obtain a probability of 0.999 of being chosen. However, if we used the same α in scenario 2, operation 1 would only obtain a probability of 0.58 of being chosen, while operation 2 would obtain a probability of 0.42.

Since the weight of the operations in the decision set has a lot of influence on the ability of the approach to distinguish among the operations with the highest priorities, we performed the same set of tests again with a different priority rule to assign weights to operations. We gave each operation O_{ij} the following weight: $w_{ij} = d_{ij} - r_{ij} - p_{ij}$. Here, d_{ij} specifies the internal due date of the operation and r_{ij} specifies the internal release date. With this priority rule, we determine the amount of time flexibility that we have to schedule an operation. The time flexibility of O_{ij} determines how important it is to schedule O_{ij} as the next operation: If O_{ij} has a lot of time flexibility, then we have a large interval in which we can schedule O_{ij} such that O_{ij} finishes before its internal due date. In that case O_{ij} does not have a high priority to be scheduled next. Note that this priority rule closely resembles the minimum slack rule. The minimum slack rule, however, determines the weight at a particular moment in time. With the time flexibility rule, we expect the difference in weights between the operations in the decision set to become smaller compared to the LST rule. In addition, with this rule we still focus on reducing the tardiness of a schedule: We have more possibilities to schedule an operation with a lot of time flexibility on time than an operation with little time flexibility. Therefore, the operations with the lowest values for time flexibility obtain the highest regret values. In this way, we can try to schedule those operations on time or with as little tardiness as possible.

We perform the tests with the new priority rule on the same set of instances. Tables 8.9a and Tables 8.9b give the results for instance 50-2 and 100-4 respectively.

	Sample size							
	1	**5**	**10**	**25**	**50**	**100**	**250**	**500**
Alpha = 150	179	176	168	149	149	147	147	145
100	183	164	159	159	159	142	140	139
50	176	171	171	144	144	144	139	139
25	178	177	169	148	148	144	142	142
10	190	182	157	157	149	148	145	133
5	232	192	179	172	168	165	165	141
4	209	165	165	163	163	163	163	152
3	252	196	195	195	169	157	157	157
2	291	193	188	188	187	179	167	165
1	233	233	219	205	205	193	167	167
0	426	356	356	351	332	285	283	283
Min. Value	**176**	**164**	**157**	**144**	**144**	**142**	**139**	**133**

	Sample size							
	1	**5**	**10**	**25**	**50**	**100**	**250**	**500**
Alpha = 150	437	437	437	414	414	414	352	352
100	622	483	467	422	396	396	396	370
50	536	468	463	365	365	365	365	365
25	640	418	418	418	407	374	374	356
10	443	443	443	443	443	441	419	409
5	529	517	429	429	429	429	410	410
4	567	567	502	489	347	347	347	347
3	715	556	494	494	494	406	406	406
2	700	469	469	469	469	469	448	447
1	884	701	701	594	568	529	529	482
0	1174	1078	965	902	808	808	803	665
Min. Value	437	418	418	365	347	347	347	347

Tables 8.9a and 8.9b the results of the tests with the time flexibility rule

As we can see in Tables 8.9a and 8.9b, the results resemble the expectations more closely: The optimal value for α becomes smaller as we increase the sample size. However, we are still not able to create a deterministic schedule when we use a high value for α. In addition, when we compare Tables 8.9a and 8.9b to Tables 8.7a and 8.7b, we notice that the approach performed much better when we used the LST rule.

The poorer performance of the approach when we use the time flexibility rule is caused by the fact that the scheduling approach no longer considers the internal due date of the operations when determining which operation needs to be scheduled first: Now, the least flexible operation is given the highest probability of being chosen even if that operation needs to be scheduled at a late moment in time. As a result, it may become more difficult to schedule an earlier operation O_{ij} on time, because this operation must now finish before the starting time of the previously scheduled operation.

From the previous tests we can conclude that we must consider the due date of the operations in some way when we create an operation decision set. However, when we use the serial scheme in combination with simple priority rules, we do not obtain expected results: We then obtain operation decision sets where the difference in regret values between operations with the highest weights and those with the lowest weights is very large. As a result, the scheduling approach is not able to distinguish well between the operations with the highest regret values (i.e. the operations with the highest priority of being scheduled as the next operation).

To solve this problem, we have two main possibilities:

1. We can adjust the scheme by which we determine which operations are included in the decision set. For example, instead of including all operations whose job predecessors have been scheduled, we can choose to include only the subset of operations that are most promising (i.e. the subset with the highest priorities).
2. We choose weights for the operations in the decision set such that the approach is able to distinguish more clearly among the operations with the highest priorities.

In this assignment, we use the following procedure: We still use the serial scheme to create the decision set and we give each operation a weight equal to its latest starting time. We then adjust the weights for each operation such that the approach is able to distinguish more clearly among the most promising operations. We adjust the weights as follows:

1. We first determine the subset S of operations in the decision set that have the highest priorities (e.g. the operations with the lowest LST values). We include an operation in S if that operation has a weight that is equal to or lower than a specific threshold weight $w_{threshold}$.
2. We now determine the actual weights for the operations in the decision set as follows: The operations in S maintain their original weights. The remaining operations in the decision set obtain the threshold weight as their new weight.

Once we have assigned new weights to the operations in the decision set, we proceed in the usual way: We determine regret values and probabilities for each operation and we subsequently choose an operation from the decision set.

With this scheme, each operation in the decision set still has the probability of being chosen. However, the largest weight in the decision set is now smaller than before. As a result, operations obtain smaller regret values and the approach is able to distinguish better between the operations with the largest regrets. Consider the following example:

We have a decision set consisting of five operations. Table 8.10a gives the weights of the operations when we use certain priority rule. Table 8.10b gives the adjusted weights for the operations when we use a threshold weight of 1. We use an α of 100 to determine the probabilities for the operations.

Operation	Weight	Regret	Probability
1	0,1	99,9	0,41
2	0,4	99,6	0,31
3	0,5	99,5	0,28
4	99	1	0,00
5	100	0	0,00

Operation	Adj. Weight	Regret	Probability
1	0,1	0,9	1,00
2	0,4	0,6	0,00
3	0,5	0,5	0,00
4	1	0	0,00
5	1	0	0,00

Tables 8.10a and 8.10b: Two example decision sets

As we can see in Table 8.10a, there is a large difference between the smallest and largest weight in the decision set. As a result, the approach is not able to distinguish well among the operations with the lowest weights, even when we use a high α value. In Table 8.10b, in contrast, the difference between the smallest and largest weight in the set has become much smaller. As a result, the regret values of the operations are much smaller than in Table 8.10a. As we can see in Table 8.10b, operation 1 now has a rounded probability of 1 of being chosen.

We have tested the new procedure on instance 50-2, because in this instance there is a large difference in weights between the operations in set A and those in set B. We use the LST rule to determine initial weights for the operations. We then apply the new procedure as follows:

1. For each operation in the decision set D, we determine both the weight and the normalized weight. The normalized weight is equal to the weight of the operation when it is normalized on a scale from 0 to 10. As a result, the operation with the highest weight obtains a normalized weight of 10 and the operation with the lowest weight obtains a normalized weight of 0.
2. We determine the set S as follows: Each operation with a normalized weight of 0.3 or less, is included in S. If S consists of less than 5 operations, we add the operations with the lowest normalized weights to S until S consists of 5 operations.
3. The operations in S retain their original weights. The remaining operations (i.e. the set $D-S$) obtain the same weight as the operation in $D-S$ with the smallest weight.

We have set the threshold weight to include an operation in set S at 0.3. We set the threshold weight at 0.3, because the set of operations with a normalized weight of 0.3 or less has a distribution that most closely resembles a uniform distribution. In other words: The normalized weights of those operations are distributed relatively evenly over the interval [0, 0.3]. The advantage of a uniform distribution of the operation weights is that the integrated approach is able to distinguish more clearly among the different operations in the decision set. We have also specified that the set S should consist of at least 5 operations. We make this choice to ensure that S contains multiple operations that have original weights. We limit the threshold to 5, however, to avoid creating a decision set that still contains several disruptive operations. We consider these thresholds to be suitable for our testing purposes. In the future, however, further research is required to determine good values for the thresholds.

Once we have assigned new weights to the operations, we again normalize the weights on a scale from 0 to 10. Now, all operations in $D-S$ obtain a weight of 10, while the other operations obtain a weight between 0 and 10. We choose to normalize the weights to ensure that the weights always remain within the interval [0,10] irrespective of the instance that we are considering or the priority rule that we use. As a result, we further reduce the influence of the weights themselves on the decision set: For example, we may have two decision sets consisting of five operations. The weights of the operations in these decision sets are as follows [1, 2, 3, 4, 5] and [10, 20, 30, 40, 50]. As we can see, in both decision sets the weights are uniformly distributed over their respective intervals. However, the regret values that the operations receive will still differ per decision set. Now, if the weights are normalized, the operations in the decision sets obtain the same normalized weights and, thus, the same regret values. Once we have normalized the weights of the operations, we proceed to determine regret values for the operations and so forth.

Table 8.11 specifies the results of the tests with the new approach:

	Sample size							
	1	5	10	25	50	100	250	500
Alpha = 150	136	126	116	113	111	111	106	106
100	136	128	127	113	112	109	107	106
50	137	126	124	111	110	106	106	106
25	138	105	105	105	105	105	105	105
10	134	119	107	107	104	104	95	95
5	156	132	132	117	107	107	103	96
4	153	129	128	117	115	106	98	98
3	177	134	134	120	120	118	115	110
2	191	169	152	130	124	120	120	120
1	297	249	207	185	185	178	178	157
0	415	371	363	324	283	283	283	283
Min. Value	**134**	**105**	**105**	**105**	**104**	**104**	**95**	**95**

Table 8.11: Results obtained for instance 50-2 when using the new approach.

As we can see in Table 8.11, the results resemble the expectations even closer than in the previous tests: The optimal α value decreases as the sample size increases. We still do not find a deterministic solution when we use high values for α, but the range of objective values reached is more limited than in the previous tests. In addition, this scheme performs better than the previous schemes: For most combinations of sample size and α value the approach performs much better than in the first test. The best schedule that we find now has an objective value of 95 compared to a value for 102 in the first test. In addition, we require much less iterations to obtain a good solution: When we use an α of 10 or 25, we already find good solutions when we perform 10 to 25 iterations. In the first test, in contrast, we reach good solutions after performing 50 to 100 iterations.

Based on the results in Table 8.11, we have tested the other instances with the same procedure and the following α scheme: We perform 1 iteration with an α of 100, 25 iterations with an α of 25, and 500 iterations with an α of 10. We compared these results with the results that we obtain when we do not use the new procedure to assign weights to operations. We performed three additional sets of tests for the instances: 500 iterations, 1000 iterations, and 5000 iterations. The iterations were equally spread over five α values: 100, 50, 25, 10, and 2. As we have seen in previous tests, we can still reach several solutions in the solution space with high α values. Therefore, we spread the iterations equally over the α values to ensure that we perform several iterations with each α value.

We find the results of the tests in Table 8.12. In this table, we have four columns with results. The column '526' specifies the results with the new procedure and the α scheme with 526 iterations. The remaining columns specify the results of the other three tests. The number specifies the total number of iterations performed.

Run	Instance	Integrated approach			
		526	500	1000	5000
1	25	23	23	23	23
2	50_1	148	122	117	117
3	50_2	99	109	109	109
4	50_3	230	232	218	218
5	50_4	287	271	271	266
6	100_1	501	535	535	532
7	100_2	425	457	457	457
8	100_3	1664	1645	1543	1584
9	100_4	126	130	127	113
10	150	963	1001	970	962

Table 8.12: Comparison of the approach with the new procedure to approaches without the procedure

As we can see in Table 8.12, the approach with the new procedure seems very promising: For certain instances the approach even finds better results than when we perform 5000 iterations with equal α values. However, we can also see that the approach performs poorly for certain instances (such as instances 50-1 and 100-3). A good area for future research is thus to test the procedure further to determine how it can be fine-tuned to the scheduling instance. Further research should also be done to determine whether there are any alternative procedures that can be used (e.g. a procedure where we only include the most promising operations in the decision set).

8.4 Tardiness results for the hierarchical approach

In this section, we present the results when solving the scheduling instances with the hierarchical approach. Table 8.13 presents the objective values of the best solutions found with each version of the hierarchical approach. The table also contains the objective values of the best solutions found with the integrated approach.

Run	Instance	Assignment neighbourhood	Best solution						
			IA	HA 1	HA 2	HA 3_1	HA 3_2	HA 4_1	HA 4_2
1	25	N1	23	43	43	23	23	23	23
		N2				25	23	23	25
2	50_1	N1	117	152	156	109	118	106	105
		N2				108	107	107	104
3	50_2	N1	99	152	139	95	92	90	94
		N2				78	100	92	97
4	50_3	N1	218	317	295	163	159	157	158
		N2				136	170	157	178
5	50_4	N1	266	443	401	252	267	238	233
		N2				242	248	241	234
6	100_1	N1	501	676	724	417	365	369	371
		N2				394	423	480	459
7	100_2	N1	425	622	519	400	419	395	366
		N2				405	409	373	386
8	100_3	N1	1543	1869	1755	1380	1426	1335	1435
		N2				1404	1434	1496	1496
9	100_4	N1	113	481	446	195	150	156	194
		N2				165	202	220	167
10	150	N1	962	1834	1636	1038	1040	1051	1081
		N2				1139	1072	1108	1120

Table 8.13: The best solutions found with the different approaches

In Table 8.13, N1 refers to assignment neighbourhood $N_{assign}^{1}(q)$ and N2 refers to neighbourhood $N_{assign}^{2}(q)$. All values are expressed in days.

Based on Table 8.13 we are able to make a number of conclusions. First, versions 3 and 4 of the hierarchical approach often perform better than the integrated approach: These versions manage to find better solutions than the integrated approach in 7 of the 10 instances. In some instances, the difference between the best solution found with the integrated approach and the one found with the hierarchical approach is even quite large. For instance 100_4 and 150 (i.e. the two instances where the integrated approach has the better performance) we compared the best schedule found with the hierarchical approach to the one found with the integrated approach. When comparing the two schedules, we noticed that the assignments $A(q)$ of the schedules are quite similar. In the solution obtained with the hierarchical approach, however, the maximum tardiness is much larger than in the solution with the

integrated approach. Based on these observations, we can conclude that the difference in schedules is caused by the way that operations are sequenced on their resources. Because the hierarchical solution has a much larger maximum tardiness than the integrated solution, we think that the hierarchical approach schedules a set of jobs very late in time so the operations of other jobs can be scheduled on time.

In Table 8.13, we also see that versions 1 and 2 of the hierarchical approach perform the poorest. This result is not surprising: As we mentioned before, we used an α of 2 to create an initial solution for these versions of the hierarchical approach. Compared to an α of 100, an item in the decision set is chosen randomly when we use an α of 2. As a result, this initial solution had a poor objective value. In addition to a poor initial solution, the two versions of the hierarchical approach only accept neighbours if those neighbours have a better objective value than the current solution. As a result, these versions are not able to escape local optima. Therefore, these versions will probably only find neighbours that are in close proximity to the initial solution. Since we use a relatively poor initial solution, we can thus also expect our best solution to be relatively poor. Because we have only tested versions 1 and 2 of the hierarchical approach with a poor initial solution, it is not possible to compare the performance of versions 1 and 2 to other versions of the hierarchical approach and the integrated approach based on the data in Table 8.13. We have chosen to still include the data for these two versions in the table to obtain an overview of the best schedules found with all versions of the hierarchical approach.

As we just mentioned, we are not able to determine how well versions 1 and 2 of the hierarchical approach perform compared to versions 3 and 4 based on the results of Table 8.13: When we used versions 3 and 4 we often found the best solutions when we used an α of 25 or 100 to create an initial solution. As we just mentioned, however, we only tested versions 1 and 2 with the initial solution that we obtained when we used an α of 2. To be able to compare the different versions to each other, we also compare the performance of the different versions when we used the same initial solution. Figure 8.4 gives the results of this comparison for four of the instances tested. For each version of the hierarchical approach we used an α of 2 to create an initial solution. In addition, each version used assignment neighbourhood $N_{assign}{}^{1}(q)$ when solving the assignment problem.

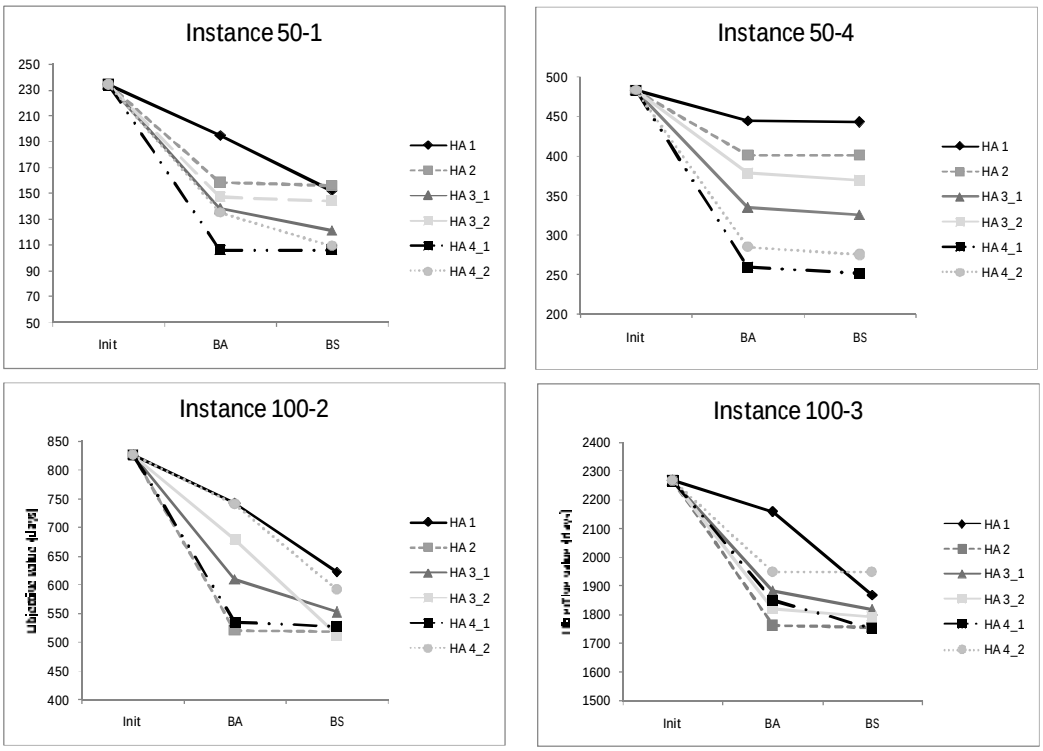

Figure 8.4: Comparison of the different versions of the hierarchical approach when using the same initial solution and assignment neighbourhood.

In Figure 8.4, 'Init' refers to the value of the initial solution, 'BA' refers to the best solution found by reassigning operations to different resources, and 'BS' refers to the best end solution found.

As we can see in Figure 8.4, version 1 of the hierarchical approach still performs poorly when compared to the other versions of the approach. Version 2, however, performs relatively well: In two of the four instances we manage to find an end solution that is comparable to those found with versions 3 and 4 of the hierarchical approach. This observation also applies to the other instances that we have tested. The main difference between version 2 and the other versions is that version 2 has a more extensive communication scheme than the other versions: In version 2, we iterate between reassignment and resequencing until we cannot find either an assignment neighbour or a sequencing neighbour with a better objective value than the initial solution. In other versions, we either use a one-way scheme or we only iterate once from the sequencing procedure to the assignment procedure. An interesting area for future research is thus to iterate more frequently between reassignment and resequencing.

In Figure 8.4 we can also see that versions 3 and 4 of the hierarchical approach generally have the best performance. From this we can conclude that it is beneficial to use SA to solve the assignment problem. However, we are not able to specify clearly which version works best: In these instances, we find the best end solution with version 4-1. However, we also find good solutions with versions 3-1 and 4-2.

We also investigate which assignment neighbourhood gives the best results. When we look at the results in Table 8.13, we notice that the best neighbourhood is dependent on the tested instance: For some instances, we obtain the best results with $N_{assign}{}^{1}(q)$, whereas $N_{assign}{}^{2}(q)$ leads to the best results in other instances.

Finally, we make some conclusions on the influence of the initial solution on the performance of the hierarchical approach. To determine the influence of a particular initial solution, we compare the tardiness of the initial solution to the performance of the hierarchical approach when that initial solution was used. We measure the performance of the hierarchical approach by determining the average tardiness over the best solutions found with the different versions of the hierarchical approach. Figure 8.5 presents the results obtained for four of the tested instances:

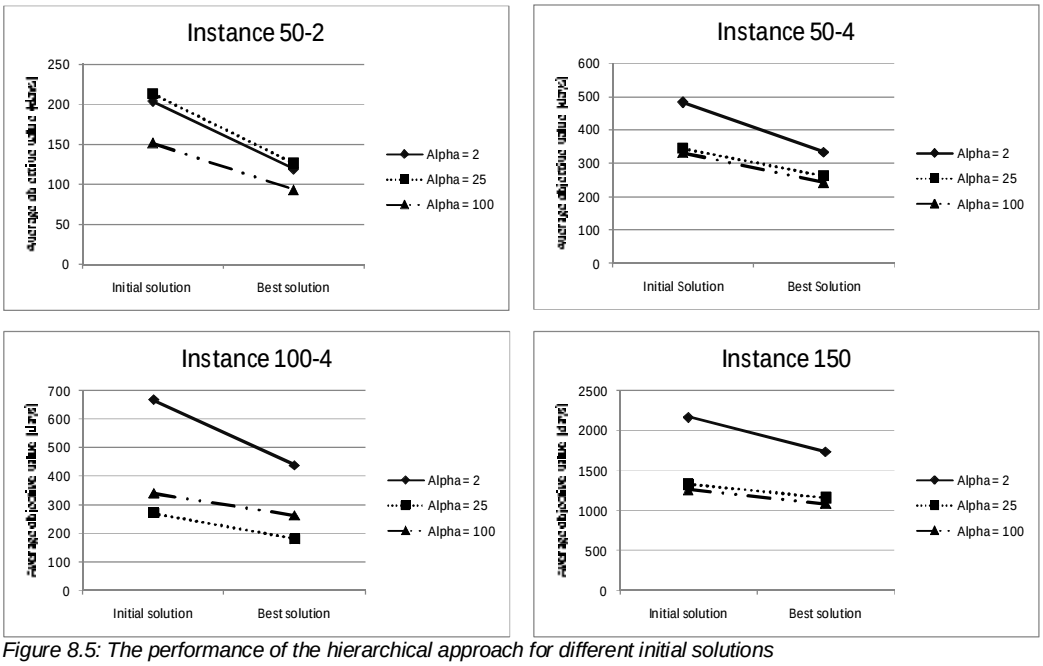

Figure 8.5: The performance of the hierarchical approach for different initial solutions

In Figure 8.5, we can see that the best solution found by the approach is very dependent on the quality of the initial solution: The better the initial solution, the better the end solution found. When we consider the slopes of the lines in the figure, we see that the approach manages to find a greater improvement when we use a poor initial solution compared to a good initial solution. However, the different versions of the approach are not able to examine the entire solution space and find a good end solution irrespective of the initial solution. For the current versions of the hierarchical approach we can thus conclude that it is important to create a good initial solution in order to find a good end solution.

Based on the information above, we can thus conclude that further research needs to be done to determine the influence of the initial solution and the communication scheme on the performance of the hierarchical approach. Therefore, we have performed some additional tests with the hierarchical approach on a subset of the instances. In these tests, we vary the initial solution used and the amount of times that we iterate between the assignment and the sequencing step.

As we have done before, we use the integrated approach to create initial solutions. In this case, however, we perform several iterations to create an initial solution. We create two initial solutions in the following way:

1. We perform 100 iterations of the integrated approach. We divide the iterations equally over the following five α values: 100, 50, 25, 10, and 2.
2. We perform 500 iterations of the integrated approach: Again, we divide the iterations equally over the same five α values as the scheme with 100 iterations.

As in version 3-1, we solve the assignment problem with simulated annealing and the sequencing problem with iterative improvement. Similarly, we used a decrease factor of 0.95. We choose this version of the approach so we have the possibility to consider a large part of the solution space when we solve the assignment problem. As we have seen in previous tests, the hierarchical approach does not perform much better when we also use simulated annealing to solve the sequencing problem. Since simulated annealing requires more calculation time than iterative improvement (calculation times are discussed in the next section), we use iterative improvement in the sequencing step.

To determine whether more frequent iterating between the assignment and sequencing step results in better schedules, we used two different communication schemes: In one scheme, we only iterate once from the sequencing problem to the assignment problem (i.e. we use the same scheme as version 3-1). In the other scheme, we iterate between assignment and sequencing until we have solved both subproblems four times. We use assignment neighbourhood N_{assign}^{2} when we solve the assignment problem.

We perform the tests on three instances: instance 50-4, instance 100-4, and instance 150. We find the results of the tests in Table 8.14. In the column 'α scheme' we specify the number of iterations performed to obtain an initial solution. Again, we express the results in days:

Instance	α scheme	Both subproblems are solved twice			Both subproblems are solved four times		
		Init.	BA	BS	Init.	BA	BS
50-4	100	298	267	266	298	266	263
	500	289	225	225	289	225	218
100-4	100	157	153	126	157	153	118
	500	124	117	109	124	117	107
150	100	994	992	965	994	964	960
	500	973	973	858	973	973	858

Table 8.14: Initial tests with a better initial solution and more frequent iterations between substeps

From Table 8.14, we can see that the hierarchical approach manages to find improvements when we use relatively good initial solutions. Now, the best solution with the hierarchical approach is also better than the best solution with the integrated approach for instances 100-4 and 150 (note that the best solution of the hierarchical approach is also better than that of the integrated approach for instance 50-4. However, in previous tests we have already found solutions with a lower tardiness value than the best solution found with the integrated approach).

In the table we can also see that we generally obtain better solutions when we perform each subproblem four times. However, compared to the communication scheme in which we perform each subproblem twice, the improvements are relatively minor.

The results of these initial tests seem promising. Further testing is still required, however, to determine a good initial solution and communication scheme.

8.5 The performance of the approaches on other factors

In this section, we focus on the performance of the approaches with respect to calculation time. In addition, we determine how the hierarchical approach influences other performance indicators when it focuses on minimizing the total tardiness.

Comparison of the approaches with respect to time

During the different runs, we stored information on the calculation time of the different approaches. In Table 8.15, we can see the results of each approach with respect to calculation time. The values are average values over all runs done during testing. We specify the calculation times in minutes.

Run	Instance	Integrated approach	Hierarchical approach					
		526 iterations	HA 1	HA 2	HA 3-1	HA 3-2	HA 4-1	HA 4-2
1	25	<1	<1	<1	<1	<1	<1	<1
2	50_1	2	1	1	5	2	9	6
3	50_2	2	1	1	6	2	9	5
4	50_3	2	1	1	4	2	7	4
5	50_4	3	1	1	4	2	7	4
6	100_1	4	14	8	13	10	16	8
7	100_2	8	11	8	20	13	30	22
8	100_3	7	29	30	30	23	34	16
9	100_4	5	10	8	11	6	17	10
10	150	15	40	42	36	27	45	21

Table 8.15: Calculation times of the different approaches

In Table 8.15 we have only specified the calculation times when we performed 526 iterations of the integrated approach. The calculation time for the integrated approach increases in a linear fashion when the number of iterations increases.

Based on the information from Table 8.15 we can conclude that the integrated approach is roughly comparable to version 3-1 of the hierarchical approach with respect to time when we perform 1250 iterations of the integrated approach. As we have seen in the previous section, the hierarchical approach presently performs better than the integrated approach with respect to tardiness minimization. However, we have not yet determined how the integrated approach performs when we use the new procedure for creating decision sets and we perform more than 526 iterations. It may thus be the case that the integrated approach performs better than the hierarchical approach when both approaches require the same amount of calculation time. Further testing of the two approaches is thus required in the future to truly determine which approach works best.

When we compare versions three and four of the hierarchal approach with respect to time, we notice that version 3-2 often has a much lower calculation time than 3-1. We make a similar observation when we compare versions 4-2 and 4-1 of the approach. This result does not surprise us: Since versions 3-2 and 4-2 use a lower decrease factor than versions 3-1 and 4-1, we are able to solve the assignment problem (and the sequencing problem in version 4) more quickly. However, when we compare the performance of the versions to each other with respect to the objective value, we see that the versions with a lower decrease factor do not perform much worse than the versions with a higher decrease factor. We can make similar conclusions when we compare version 3-1 with version 4-1 and version 3-2 with version 4-2.

Finally, we can also conclude that improvements must be made to the speeds of the different approaches. The instances that we have tested contain few jobs. For these instances, we already require a lot of calculation time. This calculation time increases when we solve practical scheduling problems that contain more jobs.

The performance of the hierarchical approach on other performance indicators

In the general scheduling problem we focus on minimizing the tardiness of the jobs in the schedule. However, we are also interested in the performance of the hierarchical approach with respect to other performance indicators. We can then determine how other indicators are affected when we try to reduce the tardiness of the schedule.

We focus on four additional kinds of performance indicators: The number of tardy jobs, the maximum tardiness found, the total amount of time spent on changeovers, and the number of changeovers performed. We chose not to focus on the way the total workload was balanced over the different resources, because information on the workload was not readily available.

We determine how versions 3 and 4 of the hierarchical approach perform on the different indicators. To do so, we relate the tardiness of the schedules that we have created with these versions to the values that the schedules have on the indicator. We create a scatter plot for each indicator in which we specify this data. For each instance we have created 26 schedules with either version 3 or 4. Therefore, each scatter plot contains 26 points.

We compare the tardiness performance of the approach to the other indicators for four instances (two with 50 jobs, one with 100 jobs, and one with 150 jobs). Figure 8.6 shows the scatter plots of the different indicators for instance 100-2:

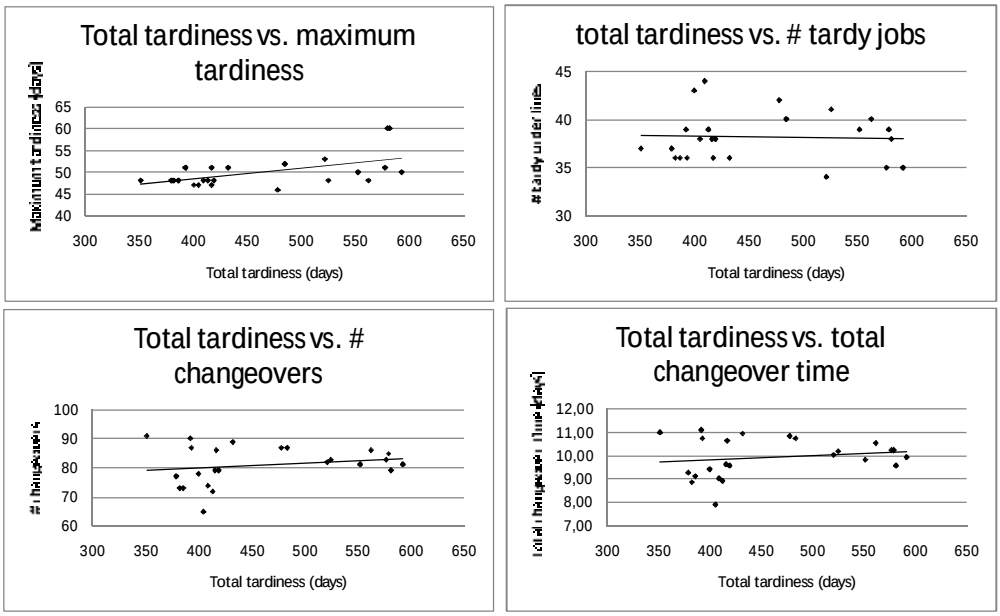

Figure 8.6: Performance of the hierarchical approach on other indicators for instance 100-2

When we compare tardiness to the other performance indicators, we find the strongest relation between the total tardiness and the maximum tardiness: When we calculate the correlation coefficient, we always find a positive relation between the total tardiness and the maximum tardiness. However, the correlation coefficient varies greatly between different instances: The minimum correlation value is 0.15, while the maximum value is 0.95. In three of the four instances, however, the correlation between total and maximum tardiness is at least 0.5. Therefore, we conclude that our versions of the hierarchical approach also tend to reduce the maximum tardiness of a schedule when we reduce the total tardiness of a schedule.

For the remaining indicators we cannot find such a relation: For certain instances, we manage to find a positive relation between two indicators. For other instances, however, we manage to find no relation or even a negative relation between the same two indicators. Based on these results we conclude that there is no clear effect on these indicators when we reduce the total tardiness of a schedule.

8.6 Conclusions based on the test results

In the previous section, we compared the different approaches on their performance with respect to tardiness minimization, calculation time, and other indicators. In this section, we draw some general conclusions based on these results.

In Section 8.3, we performed tests to determine a suitable α scheme for the integrated approach. To determine this scheme, we used a similar approach to Kolisch and Drexl (1996): We tested different combinations of sample size and α values to determine which α value is most suitable for a specific sample size. In general, the most suitable value for α should decrease as the sample size increases. In the tests, however, this did not always occur.

An important conclusion that we are able to make based on the test results is that we should adjust the decision sets that we create and the weights that we give to the items in the decision set to the scheduling instance that we are solving. This conclusion is important, because it has not been mentioned in scheduling literature before. Contrary to the scheduling instances that are tested in scheduling literature, the instances that we have tested contain certain characteristics that influence the regret values given to the items in the decision set and, hence, the probabilities of the items of being chosen. For example, we have scheduling instances in which the jobs are divided over different clusters that need to take place at approximately the same time. These clusters affect the weights given to operations in the decision set and make it more difficult for the scheduling approach to distinguish among the operations from jobs belonging to the same cluster.

We have described a general procedure in Section 8.3 that considers the scheduling instance when creating operation decision sets. This procedure adjusts the weights of the operations with relatively low priority such that the scheduling approach is able to distinguish well between the high priority operations. Initial tests with the procedure seem promising, but further research is still required to fine tune the procedure to the different scheduling instances.

In addition to conclusions with respect to the integrated approach, we are also able to make a number of conclusions for the hierarchical approach. First, the hierarchical approach performs well compared to the integrated approach: As we have seen in Section 8.4, we often find better solutions with the hierarchical approach than with the integrated approach.

Of the different versions of the hierarchical approach, we obtained the best results with the versions in which we used the simulated annealing algorithm to solve the assignment problem. However, we also noticed that version 2 of the hierarchical approach often works well compared to versions 3 and 4 of the hierarchical approach. We attribute the good performance of version 2 to the extensive two-way communication scheme used in that version. Therefore, we can conclude that it is important to iterate often between the assignment and sequencing.

Based on the test results we can also conclude that the initial solution we use in the hierarchical approach is of great influence on the performance of the approach with respect to tardiness: The better the initial solution, the better the performance of the approach. Further testing needs to be done, however, to determine a good initial solution for the hierarchical approach. Testing must also be done to determine a good communication scheme.

When we compare versions 3 and 4 of the hierarchical approach to each other, we notice that the different versions give comparable results with respect to tardiness minimization. However, when we compare the versions on calculation time, we can definitely see differences: The versions in which we used a lower decrease factor required a lot less time than the methods in which we used a higher decrease factor. In addition, versions 4-1 and 4-2, where we used simulated annealing in the sequencing step, required more time than versions 3-1 and 3-2, where we only accepted better neighbours in the sequencing step. Based on these results, we can conclude that we do not require an extensive version of the hierarchical approach to obtain good results. Finally, based on the calculation times required by the different approaches, we can conclude that the calculation time of both the integrated and the hierarchical approach need to be improved if ORTEC wishes to use the approaches to solve practical scheduling problems.

The final conclusion that we wish to make in this section is with respect to the scheduling instances that we have tested. With the instances that we have created, we have certainly been able to test whether the approaches are able to handle specific constraints. In addition, we think that we have obtained a good impression of the ability of the different approaches to minimize the tardiness of a schedule: When creating the instances, we have taken a number of measures to ensure that the instances differ sufficiently for us to make robust conclusions based on the test results. However, we have not been able to test the approaches on any practical scheduling instances (i.e. the scheduling instances found at (potential) PLANWISE customers). We expect that practical scheduling instances will be different from the instances that we have tested. For example, we expect the jobs in practical instances to have different characteristics than the jobs in the instances that we have tested. Therefore, the approaches should also be tested on practical scheduling instances in the future to truly determine how suitable the approaches are for solving practical scheduling problems.

9. Conclusions and recommendations

Based on our research, we are able to make a number of conclusions and give a number of recommendations for future research. In this chapter, we present these conclusions and recommendations. First, in Section 9.1, we make a number of conclusions on the scheduling approaches that we have developed and implemented in the optimizer in PLANWISE. Here, we discuss how well the approaches meet the requirements stated in Chapter 1 and, more specifically, how suitable the approaches are for solving the general scheduling problem of Chapter 5. Subsequently, we present some recommendations for future research in Section 9.2.

9.1 Conclusions

As mentioned in Chapter 1, ORTEC has two main requirements for a scheduling approach:

1. The approach should be sufficiently generic and it should be able to handle different practical scheduling situations.
2. The approach should be able to handle manual adjustments made by users.

The first step in our research was to translate these requirements into a general scheduling problem that the approach would need to solve. For this purpose, we considered the scheduling situations at different PLANWISE customers. Subsequently, we have created a scheduling problem in which we have included the most relevant constraints found in practice. We consider the scheduling problem that we have created to be a generic scheduling problem: The scheduling problem is based on the production scheduling environment, because this environment has the most extensive set of constraints. Since task scheduling problems are special cases of production scheduling problems, we are thus also able to model task scheduling problems with the general scheduling problem. With the scheduling problem that we have created, we are thus able to model production and task scheduling problems, which are the problems that occur most frequently. In addition, we also incorporate the second requirement in the scheduling problem by specifying a set O^{fixed} of fixed operations. As we have mentioned in Chapter 5, the operations in O^{fixed} may not be rescheduled in any way: The assignment $A_q(O_{ij}, M_q)$ for each operation O_{ij} in O^{fixed} is fixed and may not be adjusted. In addition, the starting and completion times of O_{ij} are also fixed. Thus, by specifying that manually scheduled operations belong to the set O^{fixed}, we ensure that manual adjustments made by the user are not changed by the scheduling approach.

Based on the literature study into suitable scheduling approaches for the general scheduling problem, we concluded that a hierarchical scheduling approach was most suitable for the general scheduling problem, because hierarchical approaches can be adjusted more easily to different scheduling situations than integrated approaches. We also concluded that a two-way communication scheme was more beneficial than a one-way scheme. Finally, we concluded that the assignment and sequencing subproblems of the general scheduling problem can best be solved by using either local search algorithms, specifically SA and TS, or the RBRS algorithm, because these algorithms are generic: They are able to solve scheduling problems with different kinds of constraints and objective functions. Second, these algorithms are flexible: They can easily be adjusted to incorporate new constraints or different objective functions. Also, the algorithms are able to reach large parts of the solution space. As a result, they can consider several solutions in the solution space and subsequently determine which solution is best.

Based on these conclusions, we developed two kinds of scheduling approaches: An integrated approach based on RBRS and a hierarchical approach based on local search methods. Both approaches are suitable to solve the general scheduling problem. However, we made a number of simplifying assumptions when creating the approaches:

1. We assumed that each operation is performed in a default mode. As a result, the scheduling approaches are not able to adjust the mode in which an operation is performed and hence the approaches are not able solve scheduling problems where operations have resource-dependent processing times.
2. We assumed that all resources work according to the same calendar. As a result, all resources have the same intervals of preemptive downtime.
3. We assumed that job predecessors of fixed operations are also fixed. We made this assumption to ensure that no deadlines are introduced in the schedule.
4. We assumed that routing constraints did not apply when creating the hierarchical approach. As a result we were able to use assignment neighbourhoods in which only one operation is reassigned to new resources.

We tested the approaches on 10 scheduling instances. With these scheduling instances, we were able to test whether the approaches could handle the following constraints: Precedence relations, capacity constraints, preemptive and non-preemptive availabilities, and setup-times. In addition, the approaches should be able to handle fixed operations. However, we have not been able to test this.

In Table 9.1, we summarize the previous observations. The table contains the same set of constraints as Table 4.1. For each constraint we specify whether the constraint is incorporated in the general scheduling problem, whether the constraint can be solved by the scheduling approaches, and whether we have tested the scheduling approaches on instances that include the constraint (column ‘tested’). Finally, when needed, we make some additional comments with respect to the constraint. We focus on the constraints included in the general scheduling problem when we determine whether the approach can handle the constraint and whether the constraint has been tested.

Constraint	Included in scheduling problem	Included in scheduling approaches	Tested	Additional comments
Resource capacity	X	X	X	We assume that resources are unique.
Resource eligibility	X	X	X	
Precedence relations	X	X	X	We only consider F-S precedence relations.
Multiple resources	X	X	X	
Transportation times	X	X		If transportation time is fixed between two operations, it can be specified as a minimal time lag in a F-S precedence relation
Use of the same resources				
Personnel constraints				
Resource availability	X	X	X	
Changeover times	X	X	X	We assume that changeoves do not require resources to be performed.
Materials				
Eligibility between resources				
Transfer batches				
Shared resources				
Combining and splitting different jobs				
Waiting time between operations	X	X		We model waiting time as a minimal time lag in a F-S precedence relation.
Routing constraints	X	*		The integrated approach is able to handle routing constraints, the hierarchical approach is not.
Use of different resources for two operations				
Resource dependent processing times	X			When creating the scheduling approaches, we assumed that operations are performed in a default mode.
Product carriers				
Fixed operations	X	X		

Table 9.1: A summary of constraints included in the scheduling problem

In Table 9.1, the 'X' specifies that a constraint is included in the scheduling problem and so forth. The table also contains a '*' to specify that the integrated approach is able to handle routing constraints.

As we can see in the table, there are still some activities that need to be done in order for us to use the scheduling approaches to solve the scheduling problem. At present, the scheduling approaches are not able to handle all constraints specified in the scheduling problem. If we thus wish to solve the scheduling problem with the scheduling approaches, we must ensure that the approaches can handle the remaining constraints that they cannot handle at present. Subsequently, the approaches must be tested on instances that contain those constraints. In addition, there are some constraints for which we have not been able to test whether the approaches can truly handle those constraints. In Section 9.2, we specify in

more detail which activities need to be done and the amount of time and effort that the activities require.

In Chapter 8, we tested the approaches on the scheduling instances that we have created. Based on the results, we were able to make a number of conclusions. First, it is important to tune the scheduling approaches to the scheduling instance being solved. As we have seen in Section 8.3, the scheduling instances had a lot of influence on the operation decision sets created and thus the operations chosen from the decision set. We also noticed this to a lesser extent in the hierarchical approach, where the scheduling instances influenced which assigntment neighbourhood led to the best results. Second, when comparing the performance of the hierarchical approach to the integrated approach, we were able to conclude that the hierarchical approach generally gives better results than the integrated approach. When comparing the different versions of the hierarchical approach to each other, we were able to conclude that a number of factors influenced the performance of the approach, such as the quality of the initial solution, the use of simulated annealing to solve the assignment problem, and extensive iterating between the reassignment and resequencing. Initial testing on these factors seemed promising but further research is still required to determine how the factors influence the performance of the approach. Further research is also required to determine how the integrated approach needs to be tuned to a specific scheduling instance. Finally, the calculation time of both approaches must also improve if ORTEC wishes to use the approaches to solve practical scheduling problems.

The conclusions that we have made on the performance of the different approaches is based on the tests that we have done on the scheduling instances that we have created. As we have mentioned in Section 8.1, we have taken a number of measures to ensure that the instances were sufficiently different for us to make robust conclusions. Therefore, we think that the test results give a good impression of the ability of the approaches to reduce a schedule's tardiness and the factors that influence the performance of the approaches. However, we expect that the tested instances differ in certain ways from practical scheduling instances. Therefore, further tests need to be done on practical instances before we can truly determine how the approaches perform with respect tardiness minimization.

9.2 Recommendations

We have developed two generic scheduling approaches and we have performed a number of tests to compare these two approaches to each other. However, there are still a number of areas where further research can be done. As we have seen in the previous section, we still need to perform further research to ensure that the scheduling approaches are able to solve the generic scheduling problem. We also require additional research to improve the performance of the scheduling approaches with respect to tardiness minimization and computation time. Finally, we should consider possibilities to use the scheduling approaches for more complex scheduling problems and capacity planning purposes.

We divide this section into four parts: Research to ensure that the scheduling approaches are able to solve the general scheduling problem, research to improve the performance of the approaches, research to adjust the scheduling approaches to more complex scheduling problems, and research into the possibilities to use scheduling approaches for capacity planning.

Ensuring that the scheduling approaches are able to solve the general scheduling problem

The most important activities that need to be done in the short term are to ensure that the scheduling approaches are able to solve the general scheduling problem. As we mentioned in the previous section, there are a number of constraints in the general scheduling problem that the approaches are not able to handle. In addition, we have not been able to incorporate all constraints that the scheduling approaches can handle in the scheduling instances that we tested. As a result, we cannot be sure that the approaches are truly able to handle these constraints.

The following activities must be performed in order for the scheduling approaches to be able to handle the scheduling problem:

1. A procedure must be created for both the integrated and the hierarchical approach that can assign or reassign an operation to a different mode than its default mode. This procedure must be able to choose a different mode and, subsequently, assign the operation to a set of resources to satisfy the resource demands of the new mode.
2. For both types of approaches, we need to look at possibilities to incorporate resources that use different kinds of calendars.
3. For the hierarchical approach, we need to create an assignment neighbourhood where multiple operations are reassigned to new resources. We require such a neighbourhood to handle routing constraints.

We expect the inclusion of different calendars for resources to require most effort: When resources have different calendars, we must consider all resources in a set simultaneously to determine whether an operation can be scheduled on that set at a specific time. In contrast, when all resources have the same calendar, we can consider each resource in the set individually. We thus need to make a lot of adjustments to the procedure that we use to calculate starting and completion times for operations. In addition, since we need to consider a set of resources simultaneously, we can expect the calculation time of the approach to increase. In comparison, the other two constraints can be implemented in a short period of time.

In addition to adjusting the approaches to solve the scheduling problem, we must also test the approaches to ensure that they are truly able to handle all constraints. For most constraints, we simply need to test the approaches on a scheduling instance that has that constraint. For other constraints, such as fixed operations, some adjustments must be made to PLANWISE before we can create scheduling problems that contain the constraint. Overall, we expect that testing can be done with little time and effort required.

Finally, the approaches also need to be tested on actual scheduling problems of PLANWISE customers to determine whether the approaches are able to solve ‘real’ scheduling problems. We recommend performing these tests once the approaches are able to handle the general scheduling problem. In addition, we recommend improving the speed of the approaches before solving practical scheduling instances.

Improving the performance of the scheduling approaches

In addition to ensuring that the scheduling approaches can solve the general scheduling problem, research must also be done on the short term to improve the performance of the scheduling approaches with respect to tardiness minimization and calculation time.

Research must be performed to determine how to adjust the approaches to the specific scheduling instance that needs to be solved. This is certainly the case for the integrated approach: As we have seen in Section 8.3, the scheduling instances had a lot of influence on the operation decision sets that were created and thus the operation that was chosen from

the decision sets. In Section 8.3 we described a general procedure to create operation decision sets based on the set of jobs in the scheduling problem. Initial tests with this approach seemed promising, but further research is required to fine tune the procedure to the scheduling instance. The research can be done very easily: We can use the instances that we have created to perform the tests.

There are also possibilities to improve the performance of the hierarchical approach. A simple way to improve the performance of the hierarchical approach is to create a new assignment neighbourhood containing schedules in which an operation is only reassigned to new resources to replace those resources on which the operation has a critical resource predecessor or successor. We can further improve the performance of the approach by investigating the influence of the initial solution and the communication scheme used on the performance of the approach. Initial tests showed that the performance of the approach improves when we use a good initial solution and when we iterate frequently between reassignment and resequencing. This research will require more time than the creation of a new assignment neighbourhood, but it is not difficult to do: We can easily create new versions of the hierarchical approach in which we use a good initial solution and an extensive communication scheme. We can then test these new versions on the different scheduling instances that we have created. Finally, we recommend that some research be done on the priority rules used in the hierarchical approach. When we need to choose a new resource to assign an operation to, for example, we give higher priority to those resources that have a lot of free capacity over the planning horizon. As we mentioned in Section 7.4, however, it would be more beneficial to consider the free capacity of the resource in the interval in which we wish to schedule the operation.

Finally, it is also beneficial to consider possibilities to improve the speed of the approaches. Certain procedures in the approaches can be performed more efficiently. A good area for research is the procedure that determines the actual starting and completion times of operations: For instance, we use an inefficient procedure to determine whether a changeover is necessary and the amount of changeover time required. Some time will be required to perform research in this area: First, we must determine which procedures work inefficiently. Subsequently, we need to create more efficient procedures and implement them in PLANWISE.

Scheduling approaches for more complex scheduling problems

We have not been able to consider all constraints and objectives that are relevant for (potential) ORTEC customers. An interesting area for future research is thus to adjust the scheduling approaches so they are able to handle scheduling problems with more constraints and different kinds of objective functions.

The most important constraints to focus on are:

1. Precedence relations: We have chosen to focus on F-S relations in the general scheduling problem, because it is the most common kind of precedence relation found at PLANWISE customers. Customers, however, also experience other kinds of precedence relations between the operations of a job. For instance, there are PLANWISE customers who have S-S relations between some of their operations. The scheduling approaches should therefore also be able to handle at least S-S relations and preferably also F-F and S-F relations. For each type of precedence relation, the approaches should be able to handle minimal time lags. If resources do not have any periods of downtime, this extension will be easy to implement: In this case, S-S, F-F and S-F relations can be translated into F-S relations. For instance, if O_{hj} needs to start either at the same time as, or after, O_{ij} then we can translate this S-S relation into the following F-S relation: $F_{ij} + lag_{ihj}^{fs\min} \leq S_{hj}$. In this equation, $lag_{ihj}^{fs\min}$ is

equal to the negative processing time of O_{ij}. If resources have periods of downtime, however, we cannot translate the other precedence relations into F-S relations. In this case, it will require more effort to implement the other precedence relations: First, it must be possible to specify these kinds of precedence relations in PLANWISE. At the moment, this is not possible. In addition, the scheduling approaches must be adjusted to handle these kinds of relations.

2. Product carriers: Several (potential) PLANWISE customers have scheduling situations in which they work with product carriers. Therefore, it will be beneficial if the scheduling approaches are able to handle these kinds of resources. Product carriers are used to process all operations of a job. The need for these resources is dependent on the schedule: When a job requires a product carrier, this carrier must be available for the entire duration of the job, even when no processing takes place. In other words: When a product carrier is assigned to a job, the carrier remains occupied from the moment the first operation of the job is scheduled until the final operation of the job is completed. An example of a product carrier is a maintenance dock for ships: When maintenance needs to be done to a ship, the ship is kept in a maintenance dock for the entire duration of the maintenance. We expect that it will be easier to implement this constraint in the integrated approach than in the hierarchical approach: In the integrated approach, we can claim a product carrier for a job as soon as the first operation of that job has been scheduled. Additional constraints must then be specified to ensure that the product carrier remains occupied until the last operation of the job has been scheduled. In the hierarchical approach, in contrast, it will be more difficult to implement the constraint: When we try to reschedule the operations of a job, it may occur that the job as a whole obtains a new starting and completion time. Checks must then be present in the approach to determine whether the product carrier is still available for that job in the new time interval in which the job is scheduled. We therefore expect that it will be difficult to create feasible neighbours of a solution.

Another interesting area for future research is to adjust the approaches so they are able to solve equipment scheduling problems. We expect this to be possible with very little adjustments to the scheduling approaches: The most important issue is that we need to use a different objective function (i.e. the number of tardy jobs) to evaluate schedules. In addition, we might require different priority rules to choose items from the decision sets. For example, since we now focus on minimizing the number of tardy jobs, we might give higher priorities to the operations of a job that is partially scheduled over the operations of a job that has not been scheduled yet. However, we can easily adjust the structures of the approaches for equipment scheduling problems: In the hierarchical approaches, for instance, we simply skip the solving of the sequencing problem.

Using the approaches for capacity planning

As mentioned in Section 2.1, ORTEC would like to use the scheduling approaches in an optimizer for capacity planning. A suitable area for future research is thus to investigate how suitable the approaches are for this purpose and which aspects of the approaches need to be adjusted. An important aspect of a scheduling approach for capacity planning is that it should specify which resources are bottleneck resources in the event that the approach is not able to find a feasible schedule. In addition, in this case the approach should suggest alternative schedules where operations are scheduled on a larger number of bottleneck resources. Of the different recommendations, this recommendation should be considered last: First, the scheduling approaches need to be able to solve the general scheduling problem. In addition, it is necessary to first reduce the calculation time of the approach: A scheduling approach for capacity planning must be able to solve multiple scheduling problems. For instance, if the approach cannot find a feasible schedule with the initial set of resources, it must solve a new scheduling problem that contains more units of the bottleneck

resource. Therefore, the approach may not require a lot of time to solve a single scheduling problem. We also think that it will take some time and effort to adjust the approaches so they are able to solve capacity planning problems: A set of criteria need to be implemented in the approaches that specify which resource should be considered the bottleneck resource. In addition, the scheduling approach must be able to create a new scheduling problem containing more units of the bottleneck resource.

References

- Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science,* 34:391-401
- Baker, K. (1974). *Introduction to sequencing and scheduling.* New York: John Wiley & Sons
- Bedworth, D.D., and Bailey, J.E. (1982). *Integrated Production Control Systems - Management, Analysis, Design*. New York: John Wiley & Sons
- Boctor, F (1990). Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research,* 49: pp. 3-13
- Boctor, F(1994). Heuristics for scheduling projects with resource restrictions and several resource duration modes. *International journal of Production Research,*
- Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research,* 41: 157-183
- Brucker, P. (1998). *Scheduling algorithms.* Berlin: Springer – Verlag.
- Brucker, P. and Thiele, O. (1996). A branch and bound method for the general-shop problem with sequence dependent setup-times*. OR Spektrum,* 18: 145-161.
- Dauzère-Pérès, S. et al. (1998). Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research,* 107: pp. 289 – 305
- Fattahi, P. et al. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing* 18, pp. 331–342
- Glover, F. (1986). Future path for integer programming and links to artificial intelligence, *Computers & Operations Research,* 13: 533 – 549
- Graham, R.L. et al. (1979). Optimizer and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics,*
- Guldemond, T.A. (2007). *Analyse van de functionele voorwaarden en mogelijkheden m.b.t. een planautomaat voor PLANWISE 2.5.* (in Dutch, not publicly available)
- Hans, E.W. et al. (2007). A hierarchical approach to multi-project planning under uncertainty. *Omega 35 (5)*, pp. 563-577
- Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: The University of Michigan Press.
- Hurink, J.L. *Introduction to Local Search.* Obtained through private communication.
- Kelley, J.E., Jr. (1963), "The critical-path method: Resources planning and scheduling", in: J.F. Muth and G.L. Thompson (Eds.), Industrial Scheduling, Prentice-Hall, New Jersey, pp. 347-365.

- Kolisch, R. (1995). *Project Scheduling under Resource Constraints: Efficient Heuristics for Several Problem Classes.* Heidelberg: Physica-Verlag

- Kolisch, R. and Drexl, A. (1996). Adaptive search for solving hard project scheduling problems. *Naval Research Logistics,* Vol. 43 pp. 23 – 40

- Kolisch, R. and Drexl, A. (1997). Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions,* 29: pp. 987-999

- Kolisch, R. and Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega,* 29: pp. 249 – 272

- Van Laarhoven, P.J.M and Aarts, E.H.L (1987). *Simulated Annealing: theory and applications* Dordrecht, Holland: D. Reidel Publishing Company.

- Lawler, E.L. et al. (1982). Recent developments in deterministic sequencing and scheduling: A survey. In M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinnooy Kan, editors, *Deterministic and Stochastic Scheduling,* pages 35-73. NATO Advanced Study and Research Institute, D. Reidel Publishing Company, Dordrecht, The Netherlands.

- Lin, S. And Kernighan, W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research,* Vol. 21, No. 2, pp. 498-516.

- Papadimitriou, C. H. and Steiglitz, K. (1982) Combinatorial optimization: Algorithms and complexity, NewJersey: Prentice Hall.

- Reid, R.D. and Sanders, N.R. (2002). *Operations Management.* New York: John Wiley & Sons

- Reyck, B. de and Herroelen, W. (1999). The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research,* 119: pp. 538 – 556

- Pinedo, M. and Chao, X. (1999). *Operations scheduling with applications in manufacturing and services.* Irwin/McGraw-Hill

- Schutten, J.M.J. (1996). *Shop Floor Scheduling with Setup Times: Efficiency versus Leadtime Performance.* Ph.D. thesis, University of Twente

- Zribi, N. et al. (2007). Assignment and Scheduling in Flexible Job-Shops by Hierarchical Optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews* vol. 37, no. 4, july 2007