Chapter 5
Deterministic Scheduling Approaches

In this chapter, we describe deterministic scheduling approaches for equipment (i.e., single machines), work centers, and full wafer fab situations. We start with simulation-based scheduling approaches that are somewhere between dispatching and scheduling. Then, we continue by presenting scheduling approaches for single machines. We especially focus on the case of a single batch machine with incompatible job families and the TWT objective. Furthermore, we study the problem of scheduling jobs on a single cluster tool with $C_{\max}$ objective. Then, we present scheduling approaches for work centers, i.e., for parallel machines. We first study scheduling problems for parallel machines with sequence-dependent setup times and then cover parallel batch machines with incompatible job families and ready times of the jobs. After that, we present work that deals with the treatment of secondary resources. Then, we discuss multiple orders per job (MOJ) scheduling problems. We extend these work center approaches to the full wafer fab situation using the concept of disjunctive graphs. We discuss a modified shifting bottleneck heuristic for wafer fabs, and we also describe a distributed variant of the shifting bottleneck heuristic. Next, we extend the shifting bottleneck heuristic to multicriteria situations. We study the performance of these approaches in a rolling horizon setting using simulation models of wafer fabs.

5.1 Motivation and Definitions

Scheduling is defined as the process of allocation of scarce resources over time (Brucker [34]). The goal of scheduling is to optimize one or more objectives in a decision-making process. As already discussed in Sect. 2.3, scheduling is between order release and dispatching in the PPC hierarchy and therefore an important part of the production control layer.

The two major categories in scheduling are deterministic and stochastic scheduling (cf. Pinedo [240]). Deterministic scheduling is characterized by processing times, setup times, due dates, ready times, and weights that are known in advance. They are assumed not to be influenced by uncertainty. In contrast, stochastic scheduling problems do not assume the existence of deterministic values for processing times, set-up times, or other quantities that are used within the scheduling model. The deterministic values are replaced by corresponding probability distributions. While stochastic scheduling is academically quite interesting, we focus on deterministic scheduling approaches in this monograph.

Deterministic scheduling problems can be further differentiated into static problems where all jobs to be scheduled are available at time $t = 0$ and dynamic scheduling problems that relax this condition. Jobs are ready at different points in time t in the dynamic situation.

In order to classify deterministic scheduling problems, the $\alpha|\beta|\gamma$ notation scheme from the scheduling literature [107, 240] is used within this monograph. The α field describes the machine environment (e.g., single machine or parallel machines), the β field is used for specification of the process restrictions (e.g., ready times or sequence-dependent setup times), and the γ field denotes the performance measure(s) of interest (e.g., $C_{\max}$ or TWT).

The machine environment covered by the α field is determined by the hierarchy provided by the BS (see Sect. 2.2.1). The building blocks of the machinery of a wafer fab are work centers (cf. Sect. 2.2.2). Single machines are the atomic units of a work center. For a single machine, the α field is given by the symbol 1. Single machine scheduling problems are interesting in their own right because they can form subproblems in decomposition schemes for parallel machine or job shop scheduling problems. Often, the machines of a single work center can be modeled as parallel machines. Identical parallel machines will be denoted by Pm, where m specifies the number of machines in the machine group. Because of the different ages of the machines, we often have to deal with unrelated parallel machines, i.e., the situation where the processing time depends on both the job and the machine. For this, we will use the notation Rm. As described in Sect. 2.2.2, each work area consists of several work centers. Therefore, the processing of jobs in a work area can be organized in a flexible flow shop or a flexible job shop manner, and we use the notation FFm and FJm, respectively. A simple job shop, i.e., one machine of each type at each work center, will be denoted by Jm. A wafer fab consists of several work areas. Therefore, we can treat wafer fabs from a scheduling point of view as a complex flexible job shop. A complex job shop as introduced in Sect. 2.2.3 is a job shop that has additional process restrictions that mimic complexities inherent in wafer fabs.

We continue with a discussion of the β field. Important process restrictions are sequence-dependent setup times (s_{jk}) and batching. A batch is defined as a group of jobs that have to be processed jointly (cf. Brucker et al. [36]). The completion time of a batch is determined as the completion time of the last job in the batch. A batch scheduling problem consists in grouping the jobs on each machine into batches and in scheduling these batches. Two types of scheduling problems related to batching are considered. The first type is called a serial batching problem (s-batch). In this situation, the processing time of a batch is the sum of the processing times of all jobs that form the batch. We differentiate between batching with job availability and batching with batch availability for s-batching. Each job of the batch can be further processed (at the next work center) after its completion time in the first situation. In the case of batch availability, the jobs of the batch can be further processed only when all jobs have completed the batching operation. The second type is parallel batching, for short p-batching (p-batch). In this case, the processing time of the batch is given by the maximum processing time of jobs contained in the batch (cf. Potts and Kovalyov [247] and Mathirajan and Sivakumar [176] for recent surveys related to batching in general and to batching for semiconductor manufacturing, respectively). On the entire wafer fab level, we have to take the reentrant process flows (recrc) into account when making scheduling decisions. Furthermore, in some cases, machine dedications (M_j), also called machine eligibility restrictions, and auxiliary resources (aux) should be modeled. Time constraints (tc) for the process steps also occur as specific process restrictions.

Cluster tools also lead to specific process restrictions. For cluster tools, there exist two types of scheduling problems:

- The scheduling of the wafer movements inside a cluster tool, which is similar to job shop scheduling with transportation and blocking
- The sequencing of the jobs waiting to be processed in front of a cluster tool, which leads to a machine scheduling problem with sequence-dependent processing times caused by different load port recipe combinations for cluster tools

We call the scheduling in the first situation internal scheduling and the one in the latter situation external scheduling. The sequence-dependent processing times are denoted by lrc.

As already described in Sect. 2.2.3, 300-mm manufacturers often have the need and the incentive to group small orders from different customers into one or more FOUPs to form production jobs. These jobs have to be scheduled on the various types of machine groups in the wafer fab and processed together. This class of integrated job formation and scheduling problems are called MOJ scheduling problems.

We start our discussion of the γ field by describing performance measures for the entire wafer fab. The most important among them are TP, CT, and various on-time delivery performance measures. The following measures can be derived for scheduling problems. Large values for TP are a result of minimizing makespan. Minimizing the total flow time measure and its weighted counterpart leads to small values for CT. Typical on-time delivery measures are $L_{\max}$, NTJ, WNTJ, or TWT (cf. Sect. 3.3.1). Performance measures for work areas or even work centers are derived starting from top-level wafer fab-wide performance measures. We summarize the three discussed dimensions for deterministic scheduling problems found in semiconductor manufacturing in Fig. 5.1.

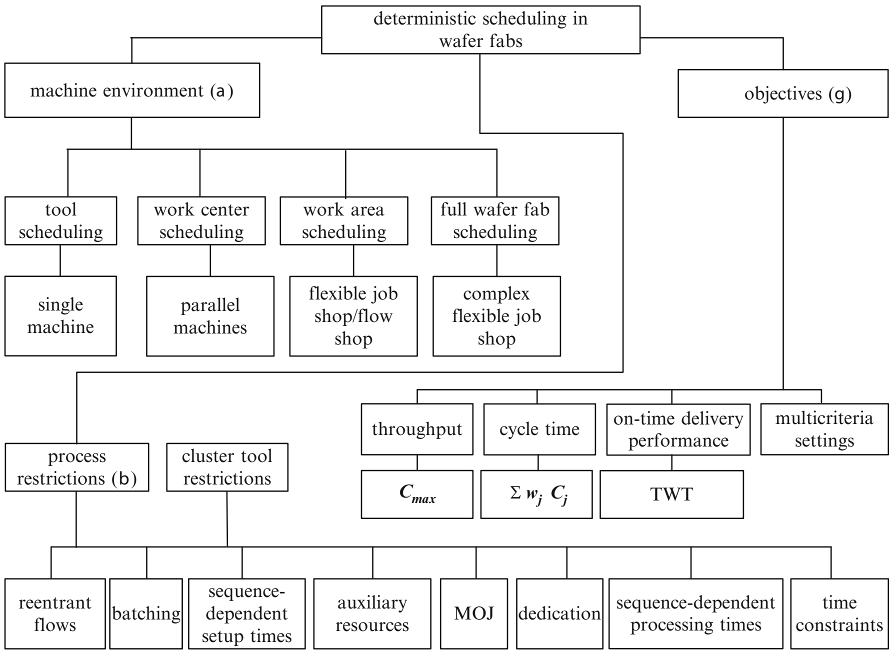

Figure 5.1: Deterministic scheduling in wafer fabs

In today's semiconductor manufacturing, dispatching is typically used in wafer fabs (see Pfund et al. [234]). Scheduling solutions are not very popular. However, with the recent increases in computing efficiency, scheduling methods have become more attractive. Because of the NP-hardness of most of the scheduling problems found in wafer fabs, we typically use heuristics to solve them. Therefore, a large portion of work presented in this chapter is devoted to efficient heuristics. Most of them decompose the overall scheduling problem into smaller, more tractable subproblems.

5.2 Simulation-Based Scheduling

Simulation-based scheduling means that simulation is used as a decision-making tool to determine schedules with a horizon ranging from several hours to a day (see Sect. 3.2.8). Dispatching rules that are already part of the simulation engine are used to allocate jobs to machines. This approach is conceptually similar to list scheduling that is frequently considered in scheduling as a straightforward way to determine schedules (see Pinedo [240]). When a machine becomes available in the simulation, all jobs that are waiting for processing are sorted according to the index of a certain dispatching rule (see Chap. 4). Then the job with the highest or the smallest value for this index is selected to start with processing on this machine. Based on this scheme, the assignment and the sequencing of jobs observed in the simulation are used to produce a control instruction *mc* in the original CS that is used to influence the BS. Major parts of a simulation-based scheduling system are the following:

1. A simulation engine that contains several dispatching rules for next job selection within an appropriate (up-to-date) simulation model
2. A graphical user interface (GUI) that produces Gantt charts based on the results of a simulation run taken from the event files
3. An interface to the operational systems on the shop floor that develops a dispatch list from the Gantt chart

Simulation-based scheduling relies to a large extent upon the capability to produce simulation models that represent the BP and the BS in a very detailed manner. A snapshot of the BS and BP (WIP and machine status) is taken before the determination of schedules, and a dynamic model is created based on this snapshot. Clearly, automated or at least semi-automated simulation model generation abilities based on data in the OS like the MES are necessary in order to run a simulation-based scheduling system. Usually, all stochastic effects like machine breakdowns are turned off because of the small time horizon. An appropriate model initialization is a nontrivial issue in simulation-based scheduling because estimating the near-time transient performance of the system as it evolves from its current state is challenging (see Davis [60]).

Simulation-based scheduling in its simplest form uses a set of different dispatching rules. Each of the resulting schedules is evaluated by a single performance measure like TP or TWT, and the schedule with the largest or smallest performance measure value, respectively, is selected as a control instruction for the BP. In some situations, even an additional manual inspection of the schedules by the production control staff is possible to check the fulfillment of soft constraints and to include human experience in the schedule selection. Job swaps are possible to manually adjust the schedules. This is a typical Leitstand or MES functionality (see Adelsberger and Kanet [2]).

The selection of a final schedule as a set of production control instructions can also be based on several performance measures. This approach is taken by Sivakumar [284]. This approach is reasonable because it is known that different performance measures can be in conflict and determining a trade-off between them is necessary. We consider the case of l different performance measures. Each performance measure a of interest is represented by a priority $X_{ajm}^{t} \in [0,1]$ for processing job j on machine m at time t. We obtain for the weighted priority of job j on machine m at time t:

$$P_{jm}^{t} := \sum_{a=1}^{l} w_a X_{ajm}^{t}, \tag{5.1}$$

where $w_a \geq 0$ is the weight of performance measure a and $\sum_{a=1}^{l} w_a = 1$ is valid. The weights are selected in [284] based on the importance of the different performance measures, which of course is subjective.

The selection of the w_a can be based on the desirability function approach described in Sect. 4.7.3 to construct blended dispatching rules. In an off-line phase, designed experiments with different weight vectors $(w_1, \ldots, w_l)$ are necessary to find an appropriate meta-model. Based on this meta-model, the desirability function can be optimized and appropriate weights obtained.

There are several simulation-based scheduling systems described in the semiconductor manufacturing literature, most of them applied in the back-end stage. Among them we refer to [213, 246, 284, 285]. The combination of simulation-based scheduling with simulation-based optimization is shown in Werner et al. [321] and Weigert et al. [317]. In these cases, instead of using only a single simulation run, after the generation of an initial schedule based on simulation and dispatching rules, a multiphase algorithm is applied to improve the schedule using metaheuristics and simulation.

5.3 Equipment Scheduling

In this section, we discuss scheduling problems for single machines and parallel machines in a single work center. Parallel machines are the building blocks of wafer fabs. Therefore, we may apply scheduling approaches to schedule jobs in front of a single work center. Scheduling problems for work centers appear rather naturally as a result of decomposition heuristics for flexible job shops. Therefore, it makes sense to deal with scheduling algorithms for single machines and also for work centers.

5.3.1 Scheduling Jobs on a Single Batch Machine

We model diffusion and oxidation operations as batch-processing machines with incompatible job families. The performance measure to be minimized is TWT. The batch problem for a single machine is more complex than the single machine problem $1||\sum w_j T_j$ that is NP-hard by Lawler [151]; therefore, we propose heuristic approaches that lead to good solutions.

The assumptions involved in scheduling a single batch machine with incompatible jobs families to minimize TWT include:

1. Jobs of the same family have the same processing times.
2. All the jobs are available at time $t = 0$.
3. Once a machine is started, it cannot be interrupted, i.e., no preemption is allowed.

The following notation is used throughout this section:

1. Jobs fall into different incompatible families that cannot be processed together. There exist f such families. The family of a batch B_s is given by $F(B_s) \in \{1, \ldots, f\}$.

2. There are n_i jobs of family i to be scheduled. We have $\sum_{i=1}^{f} n_i = n$, i.e., there are n jobs that have to be scheduled.
3. Job j of family i is represented as ij.
4. The priority weight for job j of family i is represented as w_{ij}.
5. The due date of job i of family j is represented as d_{ij}.
6. The processing time of jobs in family i is represented as p_i.
7. The batch machine capacity is B jobs. The number of jobs within a batch B_s is denoted by $|B_s|$. Clearly $|B_s| \leq B$ holds.
8. Batch k of family i is represented by B_{ki}.
9. The completion time of job j of family i is denoted by C_{ij}.
10. The weighted tardiness of job ij is represented as $w_{ij}T_j = w_{ij}(C_{ij} - d_{ij})^+$, where we use the notation $x^+ := \max(x, 0)$.

Using the $\alpha|\beta|\gamma$ notation, this problem can be represented as:

$$1|\text{p-batch}, \text{incompatible}|\text{TWT}, \tag{5.2}$$

where p-batch refers to parallel batching and incompatible to the case of incompatible job families. Note that research for problem (5.2) was initiated by the study of the scheduling problem $1|\text{p-batch}, \text{incompatible}|TT$ by Mehta and Uzsoy [180].

We start by proving some simple structural properties of optimal schedules. The first property is due to Uzsoy [303].

Proposition 5.1 *There exists an optimal schedule for problem (5.2) that does not contain partially full batches except possibly the last batch of each family to be processed in the schedule.*

In a next step, we prove that in an optimal schedule there are precedence relationships between the jobs of the same family that can be characterized by job attributes.

Proposition 5.2 *There is an optimal schedule for problem (5.2), where job ij appears before job ik, if the following two conditions are both satisfied:*

$$w_{ij} \geq w_{ik}, \tag{5.3}$$

$$d_{ij} \leq d_{ik}. \tag{5.4}$$

Furthermore, if jobs ij and ik are both tardy in a schedule S and in a schedule S' that is obtained from S by exchanging ij and ik, then the two jobs have to be scheduled by nonincreasing weight order in S.

Proof. The proof is similar to the proof of Proposition 2 in [180]. We consider an optimal schedule S_1 of the form described in Proposition 5.1 and assume that conditions (5.3) and (5.4) are fulfilled. Let job ij be scheduled in batch B_{si} and job ik in batch B_{ti} that is processed before B_{si}. Now we consider a schedule S_2 where the jobs ij and ik are swapped and the remaining jobs are the same. We denote the completion time of B_{si} in S_1 by C_s and the corresponding completion time of B_{ti} by C_t. Because all jobs of one family have the same processing time, C_s and C_t will be the same in S_2 and no job different from ij and ik will be affected by the swap operation. We define

$$\Delta\mathrm{WT}(S_1) := w_{ij}(C_s - d_{ij})^+ + w_{ik}(C_t - d_{ik})^+, \tag{5.5}$$

$$\Delta\mathrm{WT}(S_2) := w_{ik}(C_s - d_{ik})^+ + w_{ij}(C_t - d_{ij})^+, \tag{5.6}$$

where the two quantities are the weighted tardiness values of the two jobs before and after the swap, respectively. We will show that $\Delta\mathrm{WT}(S_2) \leq \Delta\mathrm{WT}(S_1)$, i.e., the tardiness of S_2 is not greater than the tardiness of S_1. We use the identity

$$(x+y)^+ - x^+ = y - \min\{y, (-x)^+\} \tag{5.7}$$

that holds for all $x, y \in \mathbb{R}$ and $y \geq 0$. We set $\varepsilon := C_s - C_t > 0$ and can estimate

$$\begin{aligned}
&\Delta\mathrm{WT}(S_1) - \Delta\mathrm{WT}(S_2)\\
&\quad = w_{ik}\left[(C_t - d_{ik})^+ - (C_t + \varepsilon - d_{ik})^+\right] - w_{ij}\left[(C_t - d_{ij})^+ - (C_t + \varepsilon - d_{ij})^+\right]\\
&\quad = w_{ij}\left[\varepsilon - \min\left\{\varepsilon, (d_{ij} - C_t)^+\right\}\right] - w_{ik}\left[\varepsilon - \min\left\{\varepsilon, (d_{ik} - C_t)^+\right\}\right]\\
&\quad \geq w_{ik}\left\{\left[\varepsilon - \min\left\{\varepsilon, (d_{ij} - C_t)^+\right\}\right] - \left[\varepsilon - \min\left\{\varepsilon, (d_{ik} - C_t)^+\right\}\right]\right\} \geq 0
\end{aligned}$$

because of $w_{ik} \leq w_{ij}$, $d_{ik} \geq d_{ij}$, and $\varepsilon > 0$. The schedule S_1 is optimal. Therefore, we have $\Delta\mathrm{WT}(S_1) = \Delta\mathrm{WT}(S_2)$, and we can swap the two jobs without changing the TWT value of the schedule before and after the swap.

Next, we assume that both ij and ik are tardy in S and S' and that ij is scheduled before ik. Let again job ij be scheduled in batch B_{si} and job ik in batch B_{ti} that is processed after B_{si} in S. In this case, we obtain

$$\begin{aligned}
\Delta\mathrm{WT}(S) - \Delta\mathrm{WT}(S') &= w_{ij}(C_s - d_{ij}) + w_{ik}(C_t - d_{ik}) - w_{ik}(C_s - d_{ik}) - w_{ij}(C_t - d_{ij})\\
&= (w_{ik} - w_{ij})(C_t - C_s).
\end{aligned}$$

Because $C_t > C_s$, $\Delta\mathrm{WT}(S) \geq \Delta\mathrm{WT}(S')$ is only valid when

$$w_{ik} \geq w_{ij} \tag{5.8}$$

holds. Therefore, changing the order of job ij and job ik is only beneficial from a TWT point of view when ik has a larger weight than ij. □

Proposition 5.2 shows that it may be beneficial to change the content of batches of the same family. This property can be used to design efficient neighborhood search approaches. Further swapping rules for jobs across batches of the same family similar to inequality (5.8) in Proposition 5.2 can be found in Devpura et al. [67].

We continue by presenting a MIP formulation for problem (5.2). Because of Proposition 5.1, we know the number of batches in an optimal schedule.

Denote this number by n_b. We use the following indices in the MIP formulation:

$b = 1,\ldots,n_b$: batch index
$s = 1,\ldots,f$: family index
$j = 1,\ldots,n$: job index

The following parameters will be used within the model:

B : maximum batch size
d_j : due date of job j
e_{js} : $\begin{cases} 1, \text{ if job } j \text{ belongs to family } s \\ 0, \text{ otherwise} \end{cases}$
M : large number
p_s : processing time of family s
w_j : weight of job j

The following decision variables are necessary:

C_b : completion time of the bth batch
X_{jb} : $\begin{cases} 1, \text{ if job } j \text{ is assigned to the } b\text{th batch} \\ 0, \text{ otherwise} \end{cases}$
Y_{bs} : $\begin{cases} 1, \text{ if batch } b \text{ belongs to family } s \\ 0, \text{ otherwise} \end{cases}$
T_j : tardiness of job j

The scheduling problem (5.2) may be formulated as follows:

$$\min \sum_{j=1}^{n} w_j T_j \tag{5.9}$$

subject to

$$\sum_{b=1}^{n_b} X_{jb} = 1, \quad j = 1,\ldots,n, \tag{5.10}$$

$$\sum_{j=1}^{n} X_{jb} \le B, \quad b = 1,\ldots,n_b, \tag{5.11}$$

$$\sum_{s=1}^{f} Y_{bs} = 1, \quad b = 1,\ldots,n_b, \tag{5.12}$$

$$e_{js} X_{jb} \le Y_{bs}, \quad j = 1,\ldots,n,\ b = 1,\ldots,n_b, \tag{5.13}$$

$$p_s Y_{1s} \le C_1, \quad s = 1,\ldots,f, \tag{5.14}$$

$$C_{b-1} + \sum_{s=1}^{f} p_s Y_{bs} \le C_b, \quad b = 2,\ldots,n_b, \tag{5.15}$$

$$(C_b - d_j) - M(1 - X_{jb}) \le T_j,\ j = 1,\ldots,n,\ b = 1,\ldots,n_b, \tag{5.16}$$

$$C_b, T_j \ge 0, X_{jb}, Y_{bs} \in \{0,1\}, \quad j = 1,\ldots,n,\ b = 1,\ldots,n_b,\ s = 1,\ldots,f. \tag{5.17}$$

The objective (5.9) intends to minimize the TWT value. Constraints (5.10) ensure that each job is assigned to a batch and constraints (5.11) do not allow more than B jobs to be assigned to the same batch. With constraints (5.12), we make sure that each batch belongs to a single job family, and constraints (5.13) ensure that the families of the jobs assigned to a batch match the family of the batch. Using constraints (5.14), the completion time of the first batch on the machine is computed, whereas constraints (5.15) ensure the correct completion times for all subsequent batches. Finally, constraints (5.16) express the tardiness for each job, and in Eq. (5.17), the nonnegativity and binary constraints are denoted. A similar formulation for minimizing $C_{\max}$ can be found in Klemmt et al. [145].

Note that the MIP formulation (5.9)–(5.17) is able to solve problem instances up to 20 jobs and two families optimally within 2 h of computing time using CPLEX 10.0. Therefore, we can use the MIP to assess the solution quality of heuristics for small-size problem instances and to test whether the implementation of the heuristics is correct or not.

Because of the NP-hardness of the problem, we decompose the problem into two phases. We form batches in the first phase. After this, we sequence these batches in the second phase. We use the ATC index

$$I_{ij}(t) := \frac{w_{ij}}{p_i} \exp\left\{ -\frac{(d_{ij} - p_i - t)^+}{\kappa \bar{p}} \right\} \tag{5.18}$$

introduced in Chap. 4 to sequence the jobs. Note that the processing time p_i depends on the family in expression (5.18). When a batch B_s of family i is formed, it can be assessed using the BATC index from Chap. 4. Therefore, we have

$$I_{B_s}(t) := \sum_{ij \in B_s} I_{ij}(t), \tag{5.19}$$

i.e., we sum up the ATC indices of the jobs that form the batch. For forming and some initial sequencing of the batches, we use the following algorithm.
Algorithm ATC-BATC

1. Sort the jobs within each family according to the ATC dispatching rule, i.e., according to nonincreasing $I_{ij}(0)$ values.
2. For each family, starting from the first job of each sequence obtained in step 1, form full batches as long as this is possible. Let $k_i := \left\lceil \frac{n_i}{B} \right\rceil$ be the number of batches that are formed of family i. Set $l_i = 0, i = 1, \ldots, f$. Initialize $t = 0$.
3. Let the current partial schedule contain l_i batches of family i. Denote the set of unscheduled batches by $\Phi(l_1, \ldots, l_f)$. Note that $\Phi(l_1, \ldots, l_f)$ contains $k_i - l_i$ batches of family i. Calculate the $I_{B_s}(t)$ index for each $B_s \in \Phi(l_1, \ldots, l_f)$. Schedule the batch B_s with the highest BATC index. Let B_s be a batch of family s. Update $l_s := l_s + 1$ and $t := t + p_s$.
4. When all batches are scheduled, then stop. Otherwise, go to step 3.

The κ values used in the indices (5.18) and (5.19) are determined by a grid search approach, i.e., we use $\kappa = 0.1h,\ h = 1, \ldots, 50$ to determine schedules by the ATC-BATC algorithm. Then we pick the schedule that leads to the smallest TWT value. We denote the corresponding κ value by κ_{best}. For a fixed κ, we use for abbreviation the notation ATC-BATC(κ). Note that a similar heuristic can be based on the EDD dispatching rule (see Perez et al. [232]). In that heuristic, jobs are sequenced in nondecreasing due date order. Then, batches are formed per family starting with the jobs with the smallest due date. After the batch formation, we sequence the batches by taking the batch that contains the job that has the smallest due date. The resulting scheme is called EDD-EDD. However, ATC-BATC outperforms this heuristic.

The initial sequencing of batches obtained by ATC-BATC can be further improved by dynamic programming and a decomposition heuristic. We continue with describing the two heuristics in detail.

We start with the dynamic programming formulation that minimizes TWT given a set of formed batches. Let $N := \sum_{i=1}^{f} \left\lceil \frac{n_i}{B} \right\rceil$ be the number of batches formed by the ATC-BATC rule. The set of all the N batches is denoted by $\mathbf{\Phi}$. Let $f(J)$ denote the minimum TWT value of a partial schedule containing the batches of the set $J \subseteq \mathbf{\Phi}$. We describe the boundary condition, the optimal value that is the TWT value for the optimal sequence, and the recursion relation. The boundary condition is used to initialize the recursion relation. We obtain the following:

Boundary condition: $f(\emptyset) := 0$,
Optimal value: $f(\mathbf{\Phi})$,
Recursive relation:

$$f(J) := \min_{B_s \in J} \left\{ f(J - \{B_s\}) + \sum_{k \in B_s} w_k \left(\sum_{B_r \in J} p_{F(B_r)} - d_k \right)^{+} \right\}. \qquad (5.20)$$

We use the abbreviation DP for this dynamic programming procedure. The idea behind the DP is straightforward. The optimal sequence for a subset of batches is determined in each iteration assuming that this subset goes first. This is carried out for each subset of batches of size l. Using the recursive relation, this is extended to each subset of size $l+1$. Each of the $l+1$ batches is a candidate to be appended. It is not necessary to know the sequence of the batches for the subsets of size l; knowing the contribution of the l batches to the objective is enough. The optimal sequence of batches can be determined by a simple backtracking procedure after $f(\mathbf{\Phi})$ is determined. Because the DP considers all subsets of $\mathbf{\Phi}$, it requires an amount of computation time that is $O(2^N)$. Hence, the DP is appropriate only for a small number of batches.

Note that a similar dynamic programming formulation is presented for the problem $1|\text{p-batch}, \text{incompatible}|TT$ in [180]. However, because of more structural insights into this problem with TT objective, the number of possible batches to be considered at a partial schedule is smaller.

Because of the large computational burden of the DP, we also consider a heuristic that decomposes the sequencing problem for batches into a series of smaller sequencing problems that can be solved optimally. This type of heuristic was proposed by Chambers et al. [43] for the non-batching case and later applied by Mehta and Uzsoy [180] for solving $1|\text{p-batch},\text{incompatible}|TT$. The procedure can be described as follows.

Algorithm Decomposition Heuristic (DH)

1. Determine an initial sequence of batches using ATC-BATC(κ_{best}). Denote this schedule by S. Set the values of the parameters λ, α, and iter. The parameter λ is the number of batches that are sequenced optimally. The first α batches for the resulting optimal sequence are fixed, and finally, iter is the total number of iterations.
2. Let $B_{[i]}$ denote the batch in the ith position of S. Furthermore, let $N := \sum_{j=1}^{f} \lceil \frac{n_j}{B} \rceil$ be the number of batches to be scheduled. We initialize $k := \min(\lambda, N)$. Set $\tilde{S} = \emptyset$, where $\tilde{S}$ represents the final schedule to be constructed. Initialize $j = 0$ and $\widehat{i} = 1$. Finally, let $P := \{B_{[1]}, \ldots, B_{[k]}\}$ be the set of batches of the initial subproblem.
3. Determine an optimal sequence of the batches of P by using complete enumeration. Denote the batch in the ith position of this sequence by $\beta_{[i]}$.
4. We define $y := \min(|P|, \alpha)$. Fix the batches $\beta_{[1]}, \ldots, \beta_{[y]}$ in positions $j+1, \ldots, j+y$ in $\tilde{S}$. Update $j := j+y$. Set $P := (P - \{\beta_{[1]}, \ldots, \beta_{[y]}\}) \bigcup \{B_{[j+1]}, \ldots, B_{[x]}\}$, where $x := \min(j+\alpha, N)$.
5. When $j < N$, then go to step 3; otherwise, go to step 6.
6. When $\hat{i} < \text{iter}$ and the TWT value of $\tilde{S}$ is smaller than the TWT value of S, then set $S := \tilde{S}$ and go to step 2 and update $\widehat{i} = \widehat{i} + 1$; otherwise, stop. The last found schedule is the resulting one.

We use the notation DH($\lambda, \alpha, \text{iter}$) for this heuristic. Usually, we use the setting $\lambda = 5$, $\alpha = 2$, and finally $\text{iter} = 15$. In this case, 5! schedules have to be evaluated within each step of DH, which is feasible from a computational point of view. So far, we change entire sequences. But in the spirit of Proposition 5.2, it may also be beneficial to swap jobs across batches of the same job family.

We continue by presenting some results of computational experiments. The experiments are performed with respect to a different number of jobs per family and a different maximum batch size. The processing times of the jobs for each family are given in Table 5.1. Note that this setting mimics the situation in wafer fabs, where process steps on batch machines tend to be long, but at the same time, because jobs on different stages compete for processing, short processing times are possible. We expect that the performance of the heuristics also depends on the due date setting. Therefore, we select due dates according to the following distribution:

$$d_{ij} \sim U\left(\mu\left(1 - \frac{R}{2}\right), \mu\left(1 + \frac{R}{2}\right)\right), \tag{5.21}$$

where R is called the range parameter. The quantity μ within the distribution in expression (5.21) is given by $\mu = (1-T)\widehat{C}_{\max}$, where T is the expected percentage of tardy jobs and $\widehat{C}_{\max}$ is an estimator for $C_{\max}$ given by $\widehat{C}_{\max} := \frac{nE(P)}{B}$. We denote by $E(P)$ the expected processing time. A larger value of T leads to a smaller mean due date and hence to a larger TWT value, and the larger the value of R, the larger the range and variance of the due dates. In total, we consider 320 problem instances.

Table 5.1: Design of experiments for problem (5.2)

Factor	Level	Count
f	4	1
n_i	30, 40, 50, 60	4
B	4, 8	2
p_i	2 with a probability of 0.2 4 with a probability of 0.2 10 with a probability of 0.3 16 with a probability of 0.2 20 with a probability of 0.1	1
w_{ij}	$\sim U(0,1)$	1
d_{ij}	$\sim U\left(\mu\left(1-\frac{R}{2}\right), \mu\left(1+\frac{R}{2}\right)\right)$ $T = 0.3, 0.6,\ R = 0.5, 2.5$	4
	Total factor combinations	32
	Number of problem instances per combination	10
	Total number of problem instances	320

The problem instances are solved using ATC-BATC, ATC-BATC-DH, ATC-BATC-DP, EDD-EDD, and finally EDD-EDD-DH. The first two entries in the notation triplet refer to the scheme that is used to form the batches and determine an initial sequence, whereas the third entry is related to the scheme to improve the initial schedule. Among the different heuristics, we use schedules obtained by ATC-BATC-DH as a reference, because ATC-BATC-DH outperforms the remaining heuristics with respect to the combination of solution quality and computational efforts.

In Table 5.2, we present some computational results. We show the TWT values obtained by the heuristics relative to the TWT values of the ATC-BATC-DH heuristic. Instead of comparing all problem instances individually, the instances are grouped according to factor levels such as the number of jobs per family and maximum batch size. For example, results for a maximum batch size of 4 imply that all other factors have been varied, but B has been kept constant at 4. Due to the fact that sometimes the TWT value is zero, we sum up the TWT values of all the instances for that factor level and divide by the sum of the TWT values obtained by ATC-BATC-DH. In the case where the batches are formed and also sequenced with the EDD dispatching rule, i.e., the EDD-EDD heuristic, the performance is 51 % worse than that of the ATC-BATC-DH heuristic across all the problem instances. When the

Table 5.2: Computational results for problem (5.2)

Compare	EDD-EDD	ATC-BATC	EDD-EDD-DH	ATC-BATC-DP
n_i				
30	1.5035	1.3640	1.1234	0.9912
40	1.5402	1.4332	1.1033	0.9945
50	1.5169	1.4107	1.0923	0.9837
60	1.4784	1.3826	1.0877	0.9777
B				
4	1.4995	1.4133	1.0805	0.9778
8	1.5175	1.3681	1.1300	0.9976
% of tardy jobs				
$T = 30\,\%$	1.4149	1.2838	1.0968	0.9896
$T = 60\,\%$	1.5359	1.4375	1.0965	0.9824
Due date range				
$R = 0.5$	1.5520	1.4397	1.1275	0.9733
$R = 2.5$	1.4405	1.3416	1.0537	0.9994
Overall	1.5053	1.3987	1.0966	0.9842

heuristic EDD-EDD-DH is used, the overall result obtained is just 10 % worse than that of the ATC-BATC-DH. On the other hand, if the method to form batches is a little more sophisticated than just arranging jobs by due dates (i.e., using the ATC rule, but the final sequence is also obtained by the BATC approach), the result is 40 % worse than that of ATC-BATC-DH. The ATC-BATC-DH performance is similar to that of ATC-BATC-DP. In fact, ATC-BATC-DP performs only slightly more than 1 % better than the ATC-BATC-DH on average. In general, the heuristics using the EDD dispatching rule to form batches improve compared to ATC-BATC-DH when n_i increases. However, the performance decreases as B increases. On the other hand, those heuristics forming batches with the ATC rule perform about the same when n_i increases and have mixed performance when B increases. Increasing T does not affect the results much for any of the heuristics except for the EDD-EDD and ATC-BATC heuristics. In general, as expected, all the heuristics perform better when R is large.

Additional computational results for a different number of families can be found in [232]. Note that for $f = 5$ families and 60 jobs per family, the DP heuristic is very time-consuming and cannot be used any longer to improve the initial schedule. A discussion of different scheduling problems with batch machines in semiconductor manufacturing is contained in the survey by Mathirajan and Sivakumar [176].

5.3.2 Scheduling Jobs on a Single Cluster Tool

The external scheduling of cluster tools, i.e., the job sequencing for this type of equipment, is challenging because the CT of wafers—and therefore also the processing time of the jobs in a cluster tool—depends on the wafer recipes used, cluster tool control and architecture, wafer waiting times, and the sequencing of the jobs (see Dümmler [75]). The following assumptions are made for the scheduling of jobs on a single cluster tool:

1. There are n jobs to be scheduled.
2. All jobs are available at time $t = 0$.
3. The cluster tool has two load locks.
4. Once a cluster tool chamber is started, it cannot be interrupted, i.e., no preemption is allowed.

It is well known that TP maximization can be achieved by $C_{\max}$ minimization. Therefore, we consider the performance measure $C_{\max}$. Using the $\alpha|\beta|\gamma$ notation, the scheduling problem can be described as

$$1|\text{lrc}|C_{\max}. \tag{5.22}$$

Problem (5.22) is NP-hard because the Hamiltonian cycle problem can be reduced to it (see Bianco et al. [26]). This scheduling problem is rarely discussed in the literature except by Oechsner and Rose [218, 219]. Bianco et al. [26] propose several dominance criteria and lower bounding schemes for the more general problem $1|r_j,\text{lrc}|C_{\max}$.

Dümmler [75] considers two main approaches to deal with this complex environment. In the first approach, a detailed deterministic simulation model of the cluster tool is used to evaluate the wafer CT values for job sequences in the scheduling algorithm. A cluster tool simulation engine is described by Dümmler [76]. In a second approach, wafer cycle time approximations are used to construct partial schedules and the detailed model of the cluster tool is only used to determine the $C_{\max}$ value of the final schedules. However, the CT approximations are also found using the simulation model of the cluster tool doing preprocessing before the scheduling decisions are made. In this monograph, we describe only the second approach in detail.

In the following, we present a beam search algorithm proposed by Oechsner and Rose [218, 219] to solve problem (5.22). To introduce the beam search algorithm, we first have to model the scheduling problem by a branching tree as introduced in Sect. 3.2.2. Each node of the tree represents a partial schedule. The root is the sequence with no job scheduled on a distinct position, i.e., (*,*,*,*), with * being a placeholder for a position in the sequence. Each child is a partial schedule with one job scheduled in the first position of the sequence, and for each child this job is different, i.e., we have (1,*,*,*), (2,*,*,*), etc. At each consecutive level, one more job is put into the sequence, until full sequences/schedules are reached. These are the leaves of the branching tree. This means that on level i, one of the jobs still to be scheduled is selected for position i of the schedule. With a finite set of jobs, the number of children per node decreases by one on each level, since fewer jobs remain to be scheduled. We show an example of such a partial branching tree including four jobs in Fig. 5.2.

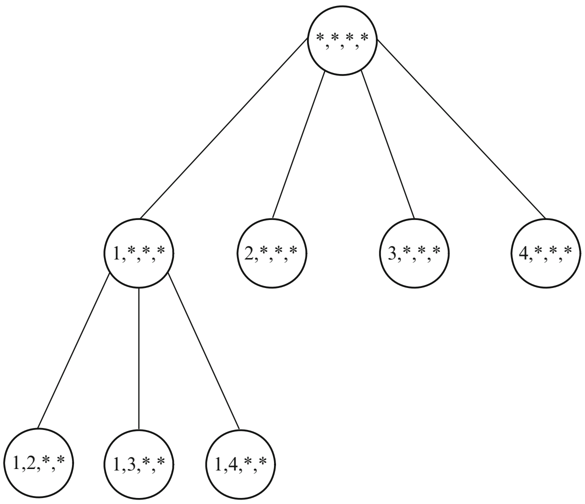

Figure 5.2: Partial branching tree with four different jobs [219]

When we have to schedule jobs belonging to a small number of job types, then the initial number of children per node is given by the number of job types because only the job type is important for the $C_{\max}$ objective. On the other hand, the number of nodes decreases only if all jobs of one job type are scheduled. Each different job type has a specific recipe. A wafer recipe determines the internal flow of the wafers within the cluster tool, i.e., the sequence of chambers to be visited (see the description in Sect. 2.2.3).

Beam search is a heuristic variant of the branch-and-bound approach (see Sect. 3.2.2). Consider a scheduling problem represented as a tree as shown in Fig. 5.2. For a large set of jobs, this branching tree becomes large because of the large number of children of each node on higher levels. Branch-and-bound aims to eliminate some of the children of a node by evaluating each node and comparing the resulting value with a lower bound. If this value is larger, the node and all of its children are discarded. Thus, fewer nodes have to be considered on the next level. However, it is common for many nodes to remain that need to be evaluated. While a branch-and-bound algorithm determines optimal solutions, it can be very time-consuming when the discrete optimization problem is NP-hard. Since problem (5.22) is NP-hard, branch-and-bound is not possible when schedules have to be obtained within a reasonable amount of time.

The aim of a beam search algorithm is to limit the number of nodes that have to be evaluated on each level of the branching tree. With each step, a certain set of nodes is selected based on an evaluation function, and the remaining nodes are discarded. Only the non-discarded nodes are expanded, keeping the size of the branching tree relatively small. The number of these nodes is called the beam width β of the search. When the branching tree has been expanded fully, this means that on each level except the first, there are at most β nodes. This results in the same number of job sequences that have to be simulated at each level. Thus, the algorithm is faster than branch-and-bound, but the optimality of the resulting schedule can no longer be guaranteed. However, if the selection process for the nodes is appropriate, a near optimal (and sometimes the optimal) solution can still be obtained. On the other hand, if the evaluation is too time-consuming, the speed advantage will be lost.

We continue by describing an appropriate evaluation function used to prune the partial branching tree. To evaluate a certain node of the branching tree, we have to decide how well it fits with the last scheduled job of the partial schedule.

Because we have only two load locks, not every job that is in the partial schedule is of interest when we insert a job after the last job in the partial schedule at time t. For most jobs of the partial schedule, the inequality $C_j < t$ is valid, and these jobs are therefore not of interest at time t. We only need to look at the jobs that are being serviced in the cluster tool at t, because only those jobs will have an effect on the C_j of the other jobs. The situation is shown in Fig. 5.3 adopted from [219].

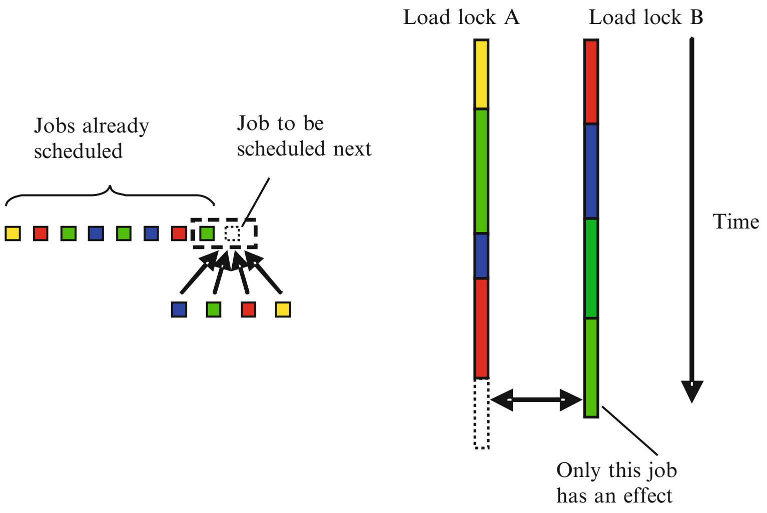

Figure 5.3: Example for including a new job into the partial schedule

We now describe two evaluation functions for the comparison of job combinations. For the first function, the slowdown factor of all possible two-job combinations is computed. The slowdown factor of two jobs i and j is defined as follows:

$$\mathrm{sdf}(i,j) := \frac{\mathrm{CT}(i,j)}{\mathrm{CT}(i)}, \tag{5.23}$$

where $\mathrm{CT}(i,j)$ is the mean CT of a wafer of job i when it is serviced at the same time as job j. $\mathrm{CT}(i)$ is the mean cycle time of a wafer of job i with only this job being processed in the cluster tool. Since $\mathrm{sdf}(i,j)$ is much higher if two jobs i and j do not fit together well, i.e., when they compete for a resource, it is a good indicator for evaluating job combinations.

The corresponding $\mathrm{CT}(i)$ values are obtained by short off-line simulations completed before starting the scheduling process and the computed sdf values are stored for fast access by the beam search algorithm.

The resulting algorithm can be summarized as follows where we assume for simplicity reasons that the jobs belong to different job types and that $\beta < n$ holds.
Algorithm Beam Search with Two-Job Slowdown

1. Create the root of the branching tree BT. Initialize the level l by $l = 1$. Denote by Φ_k the set of unscheduled jobs associated with node k at the first level and set initially $\Phi_k := \{1,\ldots,n\}$, $k = 1,\ldots,n$.
2. Form a partial schedule for each node by setting $j_{[1]} := k$ for $k = 1,\ldots,n$. Assign the resulting partial schedules to the nodes of level $l = 1$. Set $l := l+1$ and update the corresponding Φ_k accordingly.
3. For each node at level l, consider the job $j_{[l]}$, i.e., the last job of the partial schedule. If for the corresponding set $\Phi_k \neq \emptyset$, consider all the pairs of the form $(j_{[l]}, j_{[l+1]})$, where $j_{[l+1]} \in \Phi_k$, and sort them by nondecreasing $\mathrm{sdf}(j_{[l]}, j_{[l+1]})$ values; otherwise, go to step 4. Use the first $\min(\sum|\Phi_k|,\beta)$ pairs for all nodes of level l to further expand the corresponding nodes of BT. Set $l := l+1$ and update the corresponding Φ_k accordingly. Repeat this step.
4. Simulate the schedules associated with each leaf using the cluster tool simulator to determine $C_{\max}$ and select the schedule with the smallest $C_{\max}$ value.

We abbreviate this algorithm BS-SLD-2. Note that step 4 is necessary because BS-SLD-2 improves the schedules just locally as only pairs of jobs are considered. It is also clear that BS-SLD-2 tends to be computationally expensive when β becomes large.

We refine BS-SLD-2 by taking more than two jobs into account and call it BS-SLD-3. We assume that at least two jobs are part of the partial schedule and call the last two jobs $j-1$ and j, respectively, according to their position in the partial schedule. We consider the case that the two jobs are interleaved and that job j is the last job scheduled, but has a small CT value compared to the CT value of job $j-1$. This means that $C_{j-1} > C_j$ is valid. Of course, the job to be scheduled next has to be compatible with job $j-1$ and not with job j.

In this case, we try to find a job j^* such that $sld(j-1, j^*)$ is as small as possible. To decide which method to use, we look at the CT values of job $j-1$ and job j. If

$$\mathrm{CT}(j-1) \geq \mathrm{thresh}\ \mathrm{CT}(j), \tag{5.24}$$

there is a high probability that $C_j < C_{j-1}$ holds, and we choose the refined method that considers $j-1$ instead of j. When condition (5.24) is not true, we use the method from BS-SLD-2 that computes the sdf value for the last job scheduled. Therefore, we have to change step 3 of BS-SLD-2. The setting $\mathrm{thresh} = 4$ is chosen based on extensive experimentation and might be changed for different problem instances.

Computational results are presented in [218, 219]. The two heuristics provide solutions with a $C_{\max}$ value close to the optimal or best-known $C_{\max}$. However, in case of larger values for n, their performance degrades. As expected, BS-SLD-3 outperforms BS-SLD-2, but the difference is small.

An alternative evaluation function is described in [219]. The method is based on recipe comparison. The basic idea is to select only job pairs with a large number of alternative chambers for each process step to avoid situations where wafers of the two jobs compete for scarce resources. BS-SLD-3 is outperformed by the BS-type algorithm that uses the latter evaluation function.

We also note that Dümmler [75, 76] suggests a GA to tackle a generalization of the problem discussed in this section, namely $Pm|lrc|C_{\max}$. The GA simultaneously assigns the jobs to cluster tools and sequences them. All the chromosomes of an iteration are assessed using the cluster tool simulation engine.

5.3.3 Scheduling Jobs on Parallel Machines with Sequence-Dependent Setup Times

We consider parallel machine scheduling problems with sequence-dependent setups in this section as can be found in the ion implantation work area in a wafer fab (cf. Sect. 2.2.3). A job will enter the ion implantation work center when it completes its previous process step. Each implant process step needs a potentially different processing time and carries two types of information: product type and the specie being implanted. The setup time between different product types is often neglected because it is considerably less than the setup time between different species. Boron, difluoroborane, phosphorous, and arsenic are typical species. When the product with a certain specie is completed on the machine, a setup is required if the next job processed in the same machine has a different specie.

The following assumptions are made for this scheduling problem:

1. All the jobs j are available at time $r_j \geq 0$.

2. Once a machine is started, it cannot be interrupted, i.e., no preemption is allowed.
3. The parallel machines are identical.
4. There are sequence-dependent setup times.

Using the $\alpha|\beta|\gamma$ notation, the considered class of scheduling problems can be described as follows:

$$\text{Pm}|r_j, s_{jk}|\text{PM}_i, i = 1,2,3, \tag{5.25}$$

where s_{jk} represents the setup time that occurs when job k is processed next and the current setup state is appropriate for job j. We use the three performance measures $\text{PM}_1 := C_{\max}$, $\text{PM}_2 := \text{TWC}$, and $\text{PM}_3 := \text{TWT}$. These three performance measures represent the indices used for the performance of most manufacturing systems, as discussed in Sect. 3.3.1. Each of the three scheduling problems from Eq. (5.25) is NP-hard, even if $s_{jk} = 0$ is assumed (see Brucker [34]). Therefore, we have to look for efficient heuristics.

We start by presenting a MIP formulation for problem (5.25) with $C_{\max}$ objective. We use the following indices in the MIP formulation:

$j = 0,\ldots,n$: job index.

Note that job $j = 0$ is a dummy job that is used on each machine. We have $p_0 = r_0 = 0$ and $s_{0j} = 0,\ j = 1,\ldots,n$. The following parameters will be used within the model:

r_j : ready time of job j
p_j : processing time of job j
s_{jk} : sequence-dependent setup time when processing job k immediately after job j
M : large number

The following decision variables are necessary:

X_{ij} : $\begin{cases} 1, \text{ if job } i \text{ immediately precedes job } j \text{ on the same machine} \\ 0, \text{ otherwise} \end{cases}$
C_j : completion time of job j on machine i
$C_{\max}$: makespan

The scheduling problem can be formulated as follows:

$$\min\ C_{\max} \tag{5.26}$$

subject to:

$$\sum_{i=0}^{n} X_{ij} = 1, \quad j = 1,\ldots,n, \tag{5.27}$$

$$\sum_{j \neq i} X_{ij} \leq 1, \quad i = 1,\ldots,n, \tag{5.28}$$

$$\sum_{k=1}^{n} X_{0k} \leq m, \tag{5.29}$$

$$X_{ij} \leq \sum_{\substack{k=0\\k\neq i,k\neq j}}^{n} X_{ki}, \quad i=1,\dots,n,\ j=1,\dots,n,\ i\neq j, \tag{5.30}$$

$$C_j + s_{jk} + p_k \leq C_k + M(1-X_{jk}), \quad j=0,\dots,n,\ k=1,\dots,n,\ j\neq k, \tag{5.31}$$

$$C_0 = 0, \tag{5.32}$$

$$C_j \geq r_j + \sum_{k=0}^{n} \left(p_j + s_{kj} + (r_k + p_k - r_j)^{+}\right) X_{kj}, \quad j=1,\dots,n, \tag{5.33}$$

$$C_j \leq C_{\max}, \quad j=1,\dots,n, \tag{5.34}$$

$$X_{ij} \in \{0,1\}, \quad i=0,\dots,n,\ j=1,\dots,n,\ i\neq j. \tag{5.35}$$

The objective is to minimize the makespan. Constraints (5.27) make sure that each job is assigned to exactly one machine and has exactly one predecessor. The dummy job 0 is used within $X_{0k}, k \in \{1,\dots,n\}$. We have $r_0 = p_0 = 0$ and $s_{0j} = 0,\ j = 1,\dots,n$. Constraints (5.28) ensure that each job except the dummy job has at most one successor. The number of successors of the dummy jobs is at most m because of constraint (5.29). When a given job i is processed on a machine, then a predecessor has to exist on the same machine. This is modeled by constraints (5.30). When a job k is assigned to a machine immediately after job j, i.e., when $X_{jk} = 1$, its completion time has to be greater than the sum of the completion time of job j, the setup time between j and k, and the processing time of k. This is expressed by constraints (5.31). In case of $X_{jk} = 0$, constraints (5.31) are also fulfilled because of the big M. Constraint (5.32) sets the completion time of the dummy job to zero. Constraints (5.33) indicate that the completion time of a job is always larger than its ready time plus the processing time and the setup time. The makespan is not smaller than any of the completion times of the jobs. This is modeled by constraints (5.34). Finally, constraints (5.35) model the fact that the main decision variables X_{ij} only take binary values. The MIP formulation (5.26)–(5.34) has been used to solve problem instances up to $m = 2$ and $n = 18$ optimally. Therefore, we can use the formulation to assess whether heuristics are correctly implemented or not. Note that similar formulations can be derived for the two remaining performance measures.

Next, we describe some appropriate heuristics. A GA (cf. the description in Sect. 3.2.6) is used for this parallel machine scheduling problem for several reasons. First, a GA is not only flexible in dealing with additional processing restrictions of problems such as w_j, d_j, r_j, or p_j of the jobs, but can also easily handle different objectives without changing the complete evaluation algorithm or technique. We will see that the GAs for the different scheduling problems from expression (5.25) are very similar. Second, a GA can provide feasible solutions in each generation while the feasible solutions of MIP formulations are greatly dependent on the nature of the problems. Third, a GA produces a population of solutions. Knowing a set of feasible solutions is desirable because this offers flexibility in case of machine breakdowns when the schedule will be implemented in the BS.

A GA hybridized with dispatching rules is used to solve problem (5.25). The GA is used to assign jobs to individual machines, and each machine is scheduled according to a single machine dispatching rule that will be described later. The sequencing of the jobs assigned to an individual machine is necessary because we have to evaluate each chromosome using one of the performance measures $\mathrm{PM}_i, i = 1,2,3$. After all the single machine schedules are determined, the schedule of parallel machines is completed, and the performance measure is returned by the GA and is used in creating the solutions in the next generation. We will use the abbreviation HGA for the hybridized GA throughout the rest of this section. The main architecture of the HGA is shown in Fig. 5.4.

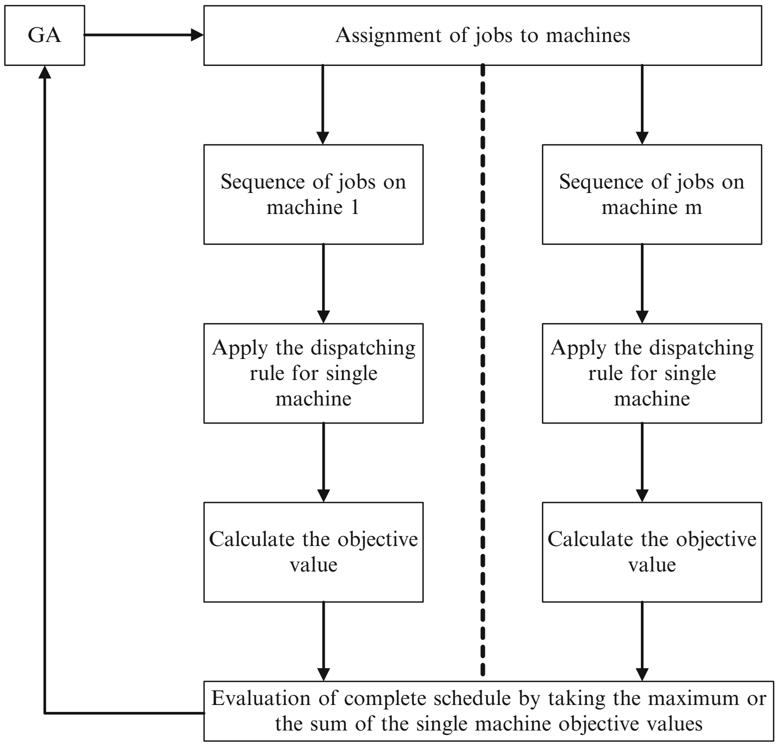
GA
Assignment of jobs to machines
Sequence of jobs on machine 1
Sequence of jobs on machine m
Apply the dispatching rule for single machine
Apply the dispatching rule for single machine
Calculate the objective value
Calculate the objective value
Evaluation of complete schedule by taking the maximum or the sum of the single machine objective values

Figure 5.4: Hybridization of a GA to solve parallel machine scheduling problems

The following steps describe how the HGA approach works, i.e., we discuss the encoding scheme, the initialization scheme, selection, crossover, mutation operations, and finally the evaluation and termination scheme.

1. Encoding: We use a job-based representation. In the case of n different jobs, the following representation is used for machine assignment:
$$c := (m_1, m_2, \ldots, m_{n-1}, m_n), \tag{5.36}$$
where we denote by m_j the machine that is used for processing job j. For example, $c := (1, 10, 4, 4, 4, 4)$ means that job 1 will be processed on machine 1, job 2 on machine 10, and the remaining jobs on machine 4. We call each of these representations a single chromosome. HGA maintains a population of these chromosomes.
2. Initialization: Each chromosome, represented by the array in expression (5.36), is initialized by randomly assigning an equally likely integer from the set $\{1, \ldots, m\}$ to each of the n jobs.
3. Parents selection: Selection is an operation to select chromosomes according to their performance, for the purpose of generating new offspring. For selection, the roulette wheel technique is used (see Goldberg [103] and Michalewicz [183]). The probability of selection of a certain chromosome is proportional to its fitness. Hence, the chromosomes with better performance are given a better chance to survive in the next generation. The fitness is calculated using linear scaling. The fitness function of the HGA is provided by one of the expressions $1/\mathrm{PM}_i, i = 1, 2, 3$.
4. Crossover: A one-point crossover (cf. Goldberg [103] or Michalewicz [183]) is utilized in which two parent chromosomes are selected randomly according to a predefined crossover probability p_c from the set obtained in step 3. Furthermore, a crossover point is also randomly chosen to divide each of the two parents. We denote the first chromosome by $c_1 := (m_{11}, m_{12}, \ldots, m_{1,n-1}, m_{1n})$ and the second chromosome by $c_2 := (m_{21}, m_{22}, \ldots, m_{2,n-1}, m_{2n})$. After performing a one-point crossover with a crossover point at position s, we obtain the two resulting chromosomes given by $c_3 := (m_{21}, \ldots, m_{1s}, \ldots, m_{1n})$ and $c_4 := (m_{11}, \ldots, m_{2s}, \ldots, m_{2n})$.
5. Mutation: According to a pre-defined mutation probability p_m for each gene of the resulting offspring, it is decided whether or not to change the machine assignment. A selected gene is randomly changed to a different machine number by a flip operator. For example, the operator might select entry two in the array $(2, 1, 3, 3, 1, 2)$ and 2 as the randomly generated number to be inserted. Thus, the new string will be $(2, 2, 3, 3, 1, 2)$. The p_m value is typically one percent of the size of the array (see Levine [160]), because a change of the entry in an array often means a dramatic change of searching direction. A proper value of p_m helps the algorithm maintain diversity while searching. On the other hand, an improper p_m value will either damage the strings if it is too large or lead to premature convergence if it is too small.

6. Elitism: A steady state genetic algorithm with overlapping populations is used. The worst elements of the preceding population are replaced. This is called an elitist strategy. Rudolph [272] proved that this strategy helps the algorithms converge to the optimum. The number of replaced elements depends on the given replacement probability.
7. Evaluation: The three objectives $C_{\max}$, TWC, and TWT are evaluated separately.
8. Termination: The HGA is controlled by a prescribed number of evaluations, which is equal to the population size times the number of generations.

The HGA requires the calculation of the objective function for each element of the population. Therefore, a sequence of the jobs has to be determined for each single machine. Dispatching rules are used to determine a sequence of the jobs. The FIFO, EDD, SPT, WSPT, LPT, and finally the HVF (highest weight) dispatching rules with priority index (4.7) (cf. Sect. 4.2) are outperformed by two dispatching rules that take sequence-dependent setup times into account.

The first dispatching rule among them is the setup avoidance dispatching rule LSC with priority index (4.11) (cf. Sect. 4.2). A tie between jobs for a given machine will be broken according to one of six secondary dispatching rules mentioned above. In the case of FIFO with multiple jobs ready at the same time, EDD will be used to break the tie.

The second rule is the ATCS dispatching rule. It was proposed by Lee and Pinedo [159] to find schedules with a small TWT value for the scheduling problem $1|s_{jk}|$TWT. The index of job j at time t when job l has completed its processing on the machine is calculated by

$$I_{lj}(t) := \frac{w_j}{p_j}\exp\left\{-\frac{(d_j - p_j - t)^+}{\kappa_1 \bar{p}}\right\}\exp\left\{-\frac{s_{lj}}{\kappa_2 \bar{s}}\right\}, \tag{5.37}$$

as introduced in Sect. 4.3.2. The resulting schedules will potentially be improved by using a swapping technique. We consider swaps between pairwise adjacent jobs. Because we use only one pass, we use the notation one pass adjacent (ADJ) swap. To have a fair comparison with the LSC dispatching rule, ADJ is also applied to LSC. We also combine the LSC dispatching rule with the SPT rule as a secondary dispatching rule.

The algorithm HGA can be summarized as follows.
Algorithm HGA

1. Create an initial population as described above. Calculate the fitness values of all chromosomes of the initial population by solving the resultant single machine scheduling problems using the LSC and ATCS dispatching rules, respectively. Improve these single machine schedules by ADJ. Set $\text{iter} = 1$.
2. Select appropriate parent chromosomes to find a new population.

3. Perform crossover and mutation operations applied to the chromosomes chosen in step 2 to produce offspring.
4. Calculate the fitness values of all chromosomes of the population by solving the resultant single machine scheduling problems using the LSC and ATCS dispatching rules. Improve these single machine schedules by ADJ. Update the current best chromosome.
5. Select a new generation based on the results in steps 3 and 4, i.e., replace only a certain part of the chromosomes of the current population by offspring. Set $\text{iter} = \text{iter} + 1$.
6. When $\text{iter} > \text{iter}_{\max}$ then stop; otherwise, go to step 2.

A population size of 8, $p_m = 0.01$, and $p_c = 0.6$ is selected for HGA. The number of generations was 300. We run HGA ten times with different seeds to obtain statistically significant results. A grid search was used to select appropriate values for the scaling parameters (κ_1, κ_2) in index (5.37).

Next, we describe the design of experiments used. We expect that the performance of HGA depends on the range of the weights, the range of the due dates, the range of the ready times, and the ratios of average processing times to average setup times. A total of 36 cases with 30 random problem instances for each factor combination are generated, resulting in 1,080 problem instances. The setup time between jobs of different species is a $U(3k, 7k)$-distributed random variable where $k = 2, 6, 10$ to model three setup levels, respectively. The ratio represents the impact of the magnitude of the setup time to the total processing times. The processing times are random variables with a probability distribution $U(-9, 9)$ plus 50, 30, and 10 for each of the three levels. This results in ratios of average processing times to average setup times equal to 5, 1, and 1/5, respectively. The resulting design of experiments is summarized in Table 5.3.

Table 5.3: Design of experiments for problem (5.25)

Factor	Level	Count
w_j	Narrow $\sim U(1, 10)$ Wide $\sim U(1, 20)$	2
r_j	High load: $r_j \equiv 0$ Moderate load: 50 % $r_j \equiv 0$ and 50 % $r_j \sim U(0, 720)$ Low load: $r_j \sim U(0, 720)$	3
d_j	Narrow: $r_j + z_j \sum_{j=1}^{n} p_j$, where $z_j \sim U(-1, 2)$ Wide: $r_j + z_j \sum_{j=1}^{n} p_j$, where $z_j \sim U(-2, 4)$	2
$\bar{p}/\bar{s}$	High: 50/10 with $p - 50 \sim U(-9, 9)$, $s/2 \sim U(3, 7)$ Moderate: 30/30 with $p - 30 \sim U(-9, 9)$, $s/6 \sim U(3, 7)$ Low: 10/50 with $p - 10 \sim U(-9, 9)$, $s/10 \sim U(3, 7)$	3
	Total factor combinations	36
	Number of problem instances per combination	30
	Total number of problem instances	1,080

Because we know that ATC-type dispatching rules do not perform well for wide due dates and widespread ready times (cf. Balasubramanian et al. [18]), HGA is hybridized with LSC in this situation. This setting is also used for the $C_{\max}$ objective. In the remaining cases, HGA is hybridized with the ATCS dispatching rule.

The relative distance (RD) percentage is used to evaluate each heuristic, and it represents how far the performance of the current method is away from the best performance of all heuristics under consideration. ARD is the average RD from all instances of each factor combination. A low ARD value indicates better overall performance, and a heuristic with ARD of 0 % is the best rule for all the problem instances considered. We obtain

$$\mathrm{RD}(H) := (\mathrm{PM}(H) - \mathrm{BP})/\mathrm{BP}, \tag{5.38}$$

where $\mathrm{PM}(H)$ is the performance measure value of heuristic H and BP is the best-known performance measure value over all studied heuristics, i.e., the list scheduling approaches including all the dispatching rules and the HGA. The corresponding computational results are shown in Table 5.4.

Table 5.4: Computational results for problem (5.25)

Compare	$C_{\max}$			TWC			TWT		
	HGA	ATCS	LSC	HGA	ATCS	LSC	HGA	ATCS	LSC
d_j									
Narrow	1.0	14.5	1.3	0.0	19.9	13.4	1.0	160.5	57.1
Wide	0.9	14.4	1.3	0.0	20.6	13.1	0.2	113.2	41.2
r_j									
High load	2.7	12.2	0.2	0.0	28.7	22.5	1.4	38.4	20.7
Moderate load	0.0	9.8	1.5	0.0	18.9	11.4	0.4	106.5	45.1
Low load	0.2	21.4	2.1	0.0	13.0	5.9	0.0	265.5	81.6
$\bar{p}/\bar{s}$									
5	0.2	6.8	0.5	0.0	8.4	18.9	0.0	15.9	32.9
1	0.8	19.3	1.4	0.0	18.6	10.9	0.4	61.7	16.4
1/5	1.9	17.2	1.9	0.0	33.7	10.0	1.4	332.9	98.1

We show the RD values. Instead of comparing all problem instances individually, the instances are grouped according to factor levels. For the performance measures of TWC and TWT, the HGA produces the best solutions across all the problem instances. For the problems of minimizing $C_{\max}$, HGA performs slightly worse than LSC when all the jobs are ready at the beginning. These results indicate that as the setup times become more important in the total processing time and the jobs are all ready at the beginning, LSC takes advantage of grouping the jobs with the least setup times together to reduce the $C_{\max}$ value while the HGA suffers from being trapped in a local minimum. For the objective of TWC, the smaller the percentage of jobs that are ready at the beginning, the less improvement HGA has in comparison with ATCS and LSC. The HGA performs extremely well for problem instances with a low processing time and setup time ratio and moderate and low load levels when the goal is to minimize TWT. The improvement of at least 100 % indicates that HGA is very practical since jobs will arrive at the work center continuously in many real-world cases. More computational results can be found in [88].

Note that the GA approach taken in HGA is generalized to a four-phase scheduling framework for parallel machines proposed by Mönch [190]. This framework is applied to scheduling jobs on photolithography steppers in [189]. The framework is implemented based on the object-oriented C++ framework GAlib (cf. Wall [315]) that supports the chromosome representations, the genetic operators used, and the scheme of the GA. Furthermore, a multi-population GA to solve multiobjective scheduling problems for parallel machines is proposed by Cochran et al. [53]. This approach avoids the separate consideration of $\mathrm{PM}_i, i = 1,2,3$.

5.3.4 Scheduling Jobs with Ready Times on Parallel Batch Machines

In this section, we extend the single machine batch scheduling problem discussed in Sect. 5.3.1 to a more real-world-like setting. We model diffusion and oxidation operations as scheduling problems for parallel batch-processing machines with incompatible job families. The performance measure to be minimized is TWT.

The assumptions involved in the scheduling of parallel batch-processing machines with incompatible jobs families and unequal ready times of the jobs to minimize TWT include:

1. Jobs of the same family have the same processing times on all machines.
2. The batch-processing machines are unrelated. That means that the machines have a machine-specific maximum batch size.
3. Once a machine is started, it cannot be interrupted, i.e., no preemption is allowed.
4. We assume unequal ready times of the jobs.
5. Dedications/qualifications are used on the batch machines, i.e., only a certain set of incompatible job families is allowed on each machine, mainly because of quality concerns.

We use the same notation as in Sect. 5.3.1. The only additional notation is with respect to ready times, unrelated parallel machines, and machine dedications:

1. The ready time of job j of family i is represented as r_{ij}. When the family information is not important, we use the notation r_j to refer to the ready time of job j of the n jobs.

2. The maximum batch size at machine $i = 1, \dots, m$ is represented as $B(i)$. Clearly, we have $|B_s| \le B(i)$ for each batch B_s that is processed on machine i.
3. The family dedication of machine i is denoted by D_i. We have $D_i \subseteq \{1, \dots, f\}$. Because each job belongs to a family, we can derive from D_i the set of jobs that are allowed to be processed on machine i and also a set of machines that can process a certain job j. The latter set is denoted by M_j to conform with the notation introduced in Sect. 5.1.

Using the $\alpha|\beta|\gamma$ notation, the scheduling problem can be represented as

$$\mathrm{Rm}|r_j, \text{p-batch}, \text{incompatible}, M_j|\mathrm{TWT}, \tag{5.39}$$

where M_j refers to dedications for job j. Note that because of the unequal ready times of the jobs, it is sometimes advantageous to form non-full batches, while in other situations it is a better strategy to wait for future job arrivals to increase the fullness of the batch.

We start by presenting a MIP for problem (5.39). This formulation is similar to the MIP (5.9)–(5.17). The following index sets will be used:

$b = 1, \dots, b_i$	: index for batches on machine i,
$s = 1, \dots, f$	: family index
$j = 1, \dots, n$	: job index
$i = 1, \dots, m$	: machine index
$J_i, i = 1, \dots, m$	: set of all jobs that are allowed on machine i
$M_j, j = 1, \dots, n$	: set of all machines that are allowed to process job j

The following parameters will be used within the model:

$B(i)$	: maximum batch size on machine i
d_j	: due date of job j
e_{js}	: $\begin{cases} 1, \text{ if job } j \text{ belongs to family } s \\ 0, \text{ otherwise} \end{cases}$
M	: large number
p_s	: processing time of family s
w_j	: weight of job j
r_j	: ready time of job j
D_i	: set of all families that can be processed on machine i

The following decision variables are necessary:

S_{bi}	: starting time of the bth batch on machine i
C_j	: completion time of job j
X_{jbi}	: $\begin{cases} 1, \text{ if job } j \text{ is assigned to the } b\text{th batch on machine } i \\ 0, \text{ otherwise} \end{cases}$

Y_{bis} : $\begin{cases} 1, \text{ if batch } b \text{ on machine } i \text{ belongs to family } s \\ 0, \text{ otherwise} \end{cases}$

T_j : tardiness of job j

The scheduling problem (5.39) may be formulated as follows:

$$\min \sum_{j=1}^{n} w_j T_j \tag{5.40}$$

subject to

$$\sum_{i\in M_j} \sum_{b=1}^{b_i} X_{jbi} = 1, \quad j = 1,\ldots,n, \tag{5.41}$$

$$\sum_{j\in J_i} X_{jbi} \le B(i), b = 1,\ldots,b_i, i = 1,\ldots,m, \tag{5.42}$$

$$\sum_{s\in D_i} Y_{bis} = 1, b = 1,\ldots,b_i, i = 1,\ldots,m, \tag{5.43}$$

$$e_{js} X_{jbi} \le Y_{bis}, j = 1,\ldots,n,\ i \in M_j, b = 1,\ldots,b_i, \tag{5.44}$$

$$X_{jbi} r_j \le S_{bi}, j = 1,\ldots,n,\ i \in M_j, b = 1,\ldots,b_i, \tag{5.45}$$

$$S_{bi} + \sum_{s\in D_i} p_s e_{js} X_{jbi} \le S_{b+1,i},\ j = 1,\ldots,n, i \in M_j, b = 1,\ldots,b_i - 1, \tag{5.46}$$

$$S_{bi} + \sum_{s\in D_i} p_s e_{js} \le C_j + M(1 - X_{jbi}),$$

$$j = 1,\ldots,n,\ i \in M_j,\ b = 1,\ldots,b_i, \tag{5.47}$$

$$C_j - d_j \le T_j,\ j = 1,\ldots,n, \tag{5.48}$$

$$C_j, T_j, S_{bi} \ge 0, X_{jbi}, Y_{bis} \in \{0,1\}, j = 1,\ldots,n, b = 1,\ldots,b_i, i = 1,\ldots,m. \tag{5.49}$$

The objective (5.40) intends to minimize the TWT value. Constraints (5.41) ensure that each job is assigned to only one batch, and constraints (5.42) do not allow more than $B(i)$ jobs to be assigned to the same batch on machine i. With constraints (5.43), we make sure that each batch belongs to a single job family, and constraints (5.44) ensure that the families of the jobs assigned to a batch match the family of the batch. Using constraints (5.45), the start time of batch b is related to the ready times of the jobs that form batch b, whereas constraints (5.46) ensure the correct start times for all subsequent batches. Constraints (5.47) make sure that the completion time of each job is not smaller than the sum of the start time of its batch and the processing time of the batch. Finally, constraints (5.48) express the tardiness for each job, and expression (5.49) represents nonnegativity and binary constraints. The MIP (5.40)–(5.49) has been solved to optimality for $m = 2$ and up to $n = 18$ in reasonable time. Hence, we have to look for efficient heuristics.

Time window decomposition approaches can be used to solve scheduling problems where not all $r_{ij} = 0$. This approach was originally proposed for the problem $\mathrm{Pm}|r_j, s_{jk}|L_{\max}$ by Ovacik and Uzsoy [222]. We consider at time t

only those jobs that already wait or will be ready within a given time window Δt for processing on the machines of a work center. The set of jobs can be described as follows:

$$J(i,t,\Delta t) := \left\{ ij | r_{ij} \le t + \Delta t \right\}. \tag{5.50}$$

Because $|J(i,t,\Delta t)|$ can be large, we might reduce the set of jobs by using appropriate dispatching rules and consider only thresh jobs with a high priority index value in a second step. We consider the reduced job set

$$\widetilde{J}(i,t,\Delta t,\text{thresh}) := \{ ij | ij \in J(i,t,\Delta t) \text{ and } \text{pos}(ij) \le \text{thresh} \}, \tag{5.51}$$

where we use the ATC index (5.18) for evaluating the jobs of $J(i,t,\Delta t)$ and we denote by $\text{pos}(ij)$ the position of job ij with respect to this index. We consider all job combinations on the set $\widetilde{J}(i,t,\Delta t,\text{thresh})$ to form a batch for machine k. We set $c := |\widetilde{J}(i,t,\Delta t,\text{thresh})|$ for abbreviation. If $c \ge B(k)$, then we have to consider

$$\binom{c}{B(k)} + \binom{c}{B(k)-1} + \ldots + \binom{c}{1} \tag{5.52}$$

different batches for the next batch to be formed. Obviously, the computational effort for evaluating all the job combinations depends strongly on the choice of the parameters Δt and thresh. We will use the following index to assess a certain batch B_s of family i on machine k:

$$I_{B_s}(t) := \sum_{j=1}^{|B_s|} \left(\frac{w_{ij}}{p_i} \right) \exp\left(- \frac{(d_{ij} - p_i - t + (r_{B_s} - t)^+)^+}{\kappa \overline{p}} \right) \frac{|B_s|}{B(k)}, \tag{5.53}$$

where $r_{B_s} := \max_{ij \in B_s}(r_{ij})$ is the maximum ready time of the jobs in the batch. This index is proposed by Mönch et al. [203]. It is called the BATC-II index. Note that the $|B_s|/B(k)$ part of the index in Eq. (5.53) is related to the fullness of batches on machine k. The batch with the highest BATC-II index will be selected next for processing on machine k, so fuller batches are preferable. Finally, we move the time window into the future in a rolling horizon manner.

The time window decomposition heuristic (TWDH) can be described in a more algorithmic manner as follows.
Algorithm BATC-II-TWDH

1. Denote the set of all jobs by J_n. Initialize $i := 1$ to iterate over all families.
2. Determine the next available machine k with availability time $t_a(k)$ and initialize $t := t_a(k)$.
3. Determine the job $ij \in J_n$ with minimal r_{ij}. When $t < r_{ij}$, then set $t := r_{ij}$.
4. Consider a time window of length Δt with left endpoint t. Determine the sets $J(i,t,\Delta t)$ and $\widetilde{J}(i,t,\Delta t,\text{thresh})$.
5. Consider all feasible combinations of jobs to form batches based on jobs from the set $\widetilde{J}(i,t,\Delta t,\text{thresh})$. Find the batch B_s with largest index (5.53) among the batches formed in this step.

6. Update $i := i+1$. When $i > f$, then go to step 7; otherwise, set $t := t_a(k)$ and go to step 3.
7. Select the batch with the largest index found in step 5. This batch is chosen to be selected on machine k. Let B_s be the chosen batch and i the family of this batch. Update $J_n := J_n - \{ij | ij \in B_s\}$ and set $t_a(k) := \max\{t_a(k), r_{B_s}\} + p_i$.
8. When $|J_n| > 0$, then set $i := 1$ and go to step 2, otherwise stop.

Note that BATC-II-TWDH is used for several κ values within the indices (5.18) and (5.53) from a grid over the interval $(0.0, 6.5]$. Neural networks and inductive decision trees from machine learning (cf. Sect. 3.2.10) are proposed by Mönch et al. [205] to automate this step for BATC-II-TWDH. In Klemmt et al. [146], a MIP approach is used to carry out steps 3–7 of BATC-II-TWDH, i.e., solving a problem instance for problem (5.39) leads to solving a sequence of MIPs.

BATC-II-TWDH can be used to find good initial solutions for neighborhood search approaches. We continue by describing a VNS-type heuristic for problem (5.39). VNS is a neighborhood search-based metaheuristic (cf. the description in Sect. 3.2.6).

The VNS algorithm designed for the batching problem operates on the final solution representation, i.e., each job is assigned to a batch and each batch is assigned to a certain position on a machine. The proposed VNS algorithm can be summarized as follows.
Algorithm VNS for Batch Scheduling
Initialization

1. Define K different neighborhood structures N_k.
2. Generate an initial solution x using BATC-II-TWDH.
3. Set $k = 1$.

Loop

1. Repeat until stopping criterion is met.
2. Shaking: Choose randomly $x' \in N_k(x)$, where $N_k(x)$ is a neighborhood of x that is based on the neighborhood structure N_k (see the description in Sect. 3.2.6).
3. Local search: Improve x' by the batch local search (BLS) method (described below).
4. Acceptance decision: If x' is better than x, then $x := x'$ and $k := 1$; otherwise, set $k := (k \bmod K) + 1$.

The proposed BLS algorithm used for the VNS scheme consists of two different phases. During the first phase, the workload of the machines is balanced. If the last batch of the machine with the maximum completion time starts later than the completion time of another machine that is suitable for that batch, the batch is moved to that machine. This step is repeated until no batch

can be moved. During the second phase of the local search, jobs and entire batches are exchanged. The BLS algorithm can be summarized as follows.
Algorithm BLS

1. Phase 1: Balance the workload across machines.
2. Phase 2: Apply the following steps iteratively as long as improvements are obtained:
    (a) Job insert: Remove a job from a batch and insert it into another batch.
    (b) Job swap: Swap two jobs of different batches from the same family.
    (c) Batch swap: Swap two batches of the same family.

The design of the neighborhood structure for the shaking step of the VNS considers manipulations of whole batches across different machines. We define five classes of neighborhood structures:

- *splitBatch*(l): Randomly select a batch and split it into two batches. One remains in the current position, the other one is inserted into the sequence on a different machine. Repeat this step l times.
- *moveBatch*(l): Randomly select a batch from a machine and remove it. Insert it in a random position on a randomly selected machine. Repeat this step l times.
- *moveSeq*(l): Randomly select a position on a machine. Remove a sequence of at most l (l or all remaining batches) and insert this partial sequence on another machine at a randomly selected position.
- *swapBatch*(l): Randomly select two batches from different machines that are both capable of handling those batches and exchange their positions. Repeat this step l times.
- *swapSeq*(l): Randomly select two positions on different machines and exchange the batch sequences starting from that position of at most length l (l or all remaining batches).

Note that in each of these neighborhoods only splits, moves, and swaps are considered that result in a feasible solution, i.e., restrictions of job families and batch sizes are considered. The neighborhoods are applied in the order given above. They are parameterized with different l values. We use $K = 15$ and $l = 2, 3, 5$.

We expect that the solution quality depends on the number of incompatible job families, the number of jobs, the number of machines, and the release and due date settings. The ready times of the jobs are taken as instances of a $U(0, \hat{C}_{\max})$-distributed random variable, where the quantity $\hat{C}_{\max}$ is an estimate of the makespan and is determined as follows:

$$\hat{C}_{\max} := n\bar{p} \Big/ \left(0.75\ u \sum_{k=1}^{m} B(k) \right). \tag{5.54}$$

The quantity $\bar{p}$ denotes the average processing time and u the utilization of the machines. The factor 0.75 mimics the average fullness of a batch. The corresponding design of experiments is summarized in Table 5.5.

Table 5.5: Design of experiments for problem (5.39)

Factor	Level	Count
f	4, 6, 8	3
m	3, 4, 5	3
n_i	20, 30, 40	3
$B(k)$	$B(1)=3$, $B(2)=4$, $B(3)=6$, $B(4)=4$, $B(5)=2$	1
p_i	2 with $p=0.2$, 4 with $p=0.2$, 10 with $p=0.3$, 16 with $p=0.2$, 20 with $p=0.1$	1
w_{ij}	0.3 with $p=0.75$, 0.6 with $p=0.2$, 1 with $p=0.05$	1
r_{ij}	$\sim U(0,\hat{C}_{\max})$	1
d_{ij}	$\sim U\left(r_{ij}-4\bar{p}, r_{ij}+4\bar{p}\right)$	1
D_i	$D_1=\{1,\ldots,6\}$, $D_2=\{1,3,7,8\}$, $D_3=\{1,4,6,7\}$, $D_4=\{1,4,5,8\}$, $D_5=\{1,3,4\}$	1
u	0.7, 0.8, 0.9	3
	Total factor combinations	81
	Number of problem instances per combination	2
	Total number of problem instances	162

We compare the performance of the BATC-II-TWDH with the performance of the VNS scheme. The corresponding computational results are shown in Table 5.6. The results in the third row correspond to BATC-II-TWDH. We use a time window of size $\Delta t = \bar{p}/4$. It can be shown that a too small or a too large time window is not beneficial. A similar behavior is observed for the algorithm NACH (cf. Sect. 4.6.2). The computing time for the VNS scheme is 60 s. For each problem instance, three replications with different seeds are performed, and the average TWT value is used for comparison. We show the TWT values for the VNS approach relative to the TWT values found by BATC-II-TWDH. All problem instances are grouped according to different factor levels. For example, $m=3$ means in Table 5.6 that the average TWT value for all problem instances is taken, where $m=3$ is valid. We can see that a larger number of machines and incompatible job families makes the scheduling problem harder to solve. We see from Table 5.6 that VNS clearly outperforms BATC-II-TWDH.

More computational results for problem (5.39) can be found in [146]. GAs similar to HGA from Sect. 5.3.3 are discussed for problem $\mathrm{Pm}|\text{p-batch}, \text{incompatible}|\mathrm{TWT}$ by Balasubramanian et al. [18] and for $\mathrm{Pm}|r_j, \text{p-batch}, \text{incompatible}|\mathrm{TWT}$ by Mönch et al. [203]. Either jobs are assigned to machines via the GA or batches are formed first by BATC-type dispatching rules and then these batches are assigned to the machines by the corresponding GA. The second approach from [203] is extended to the following bicriteria problem $\mathrm{Pm}|r_j, \text{p-batch}, \text{incompatible}|\mathrm{TWT}, C_{\max}$ by Reichelt and Mönch [259].

Table 5.6: Computational results for problem (5.39)

Compare	m			n_i			f		
	3	4	5	20	30	40	4	6	8
TWDH	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
VNS	0.89	0.92	0.93	0.92	0.91	0.93	0.91	0.89	0.94

A different GA for $\mathrm{Pm}|r_j,\text{p-batch},\text{incompatible}|\mathrm{TWT}$ is proposed by Chiang et al. [48] for the problem studied in [203].

5.3.5 Scheduling Problems for Parallel Machines with Auxiliary Resources

Auxiliary resources in wafer fabs are typically related to steppers in the photolithography work area (cf. Sect. 2.2.2). As ICs are built by repeatedly constructing layers with desired properties on the surface of the wafers, stepper processing depends on both the layer of the wafer that is associated with the current process step of the job and the correct reticle being available at the same time. The auxiliary resource problem arises because every layer of each product can require its own unique reticle, as reticle requirements are typically both product- and layer-dependent. Considering the fact that a reticle must be on the machine for the duration of processing, the relatively small number of reticles present in a wafer fab further complicates the parallel machine scheduling problem associated with steppers.

In this section, we start by discussing a simplified model problem. The assumptions for scheduling jobs on steppers within this model problem are the following:

1. The parallel machines are identical.
2. Once a machine is started, it cannot be interrupted, i.e., no preemption is allowed.
3. We assume unequal ready times of the jobs.
4. An appropriate reticle is necessary to process a job on a machine.

We consider a TWC objective. Note that this measure takes different weights of the jobs into account. In the context of stepper scheduling, it might be preferable to schedule jobs next with many already completed layers. Therefore, individual job weights can be derived based on this information. On the other hand, a small TWC value leads to a small CT value of the jobs. Using the $\alpha|\beta|\gamma$ notation, the scheduling problem can be represented as

$$\mathrm{Pm}|r_j,\mathrm{aux}|\mathrm{TWC}. \tag{5.55}$$

Problem (5.55) is NP-hard because the problem $1|r_j|\text{TC}$ is NP-hard (see Brucker [34]). Hence, we have to look for efficient heuristics.

In the following, we present a MIP formulation for problem (5.55) that is due to Cakici and Mason [42]. It is similar to the formulation given for problem (5.25). We use the following indices and index sets in the corresponding MIP model:

$i = 1,\dots,m$	: machine index
$j = 0,\dots,n$	: job index
$l \in L$	: set of layers
J_l	: set of jobs that require layer $l \in L$ for processing

Note that the job $j = 0$ is a dummy job that is used on each machine. We have $p_0 = r_0 = 0$. The following parameters will be used within the model:

r_j	: ready time of job j
p_j	: processing time of job j
w_j	: weight of job j
M	: large number

The following decision variables are used within the MIP:

$$X_{ij} : \begin{cases} 1, \text{ if job } i \text{ immediately precedes job } j \text{ on the same machine} \\ 0, \text{ otherwise} \end{cases}$$

C_j : completion time of job j

$$e_{ij} : \begin{cases} 1, \text{ if job } i \in J_l \text{ is completed before job } j \in J_l,\ i \neq j \text{ starts its} \\ \quad \text{processing} \\ 0, \text{ otherwise} \end{cases}$$

The scheduling problem can be formulated as follows:

$$\min \sum_{j=1}^{n} w_j C_j \tag{5.56}$$

subject to:

$$\sum_{i=0}^{n} X_{ij} = 1, \quad j = 1,\dots,n, \tag{5.57}$$

$$\sum_{j \neq i} X_{ij} \leq 1, \quad i = 1,\dots,n, \tag{5.58}$$

$$\sum_{k=1}^{n} X_{0k} \leq m, \tag{5.59}$$

$$X_{ij} \leq \sum_{\substack{k=0 \\ k\neq i, k\neq j}}^{n} X_{ki}, \quad i = 1,\dots,n,\ j = 1,\dots,n,\ i \neq j, \tag{5.60}$$

$$C_j + p_k \leq C_k + M(1 - X_{jk}), \quad j = 0,\dots,n,\ k = 1,\dots,n,\ j \neq k, \tag{5.61}$$

$$C_0 = 0, \tag{5.62}$$

$$C_j \geq r_j + \sum_{k=0}^{n} (p_j + (r_k + p_k - r_j)^+) X_{kj}, \quad j = 1, \ldots, n, \tag{5.63}$$

$$C_j + p_k e_{jk} \leq C_k + M(1 - e_{jk}),$$
$$j = 1, \ldots, n,\ k = 1, \ldots, n,\ j,k \in J_l,\ l \in L,\ j \neq k, \tag{5.64}$$

$$1 \leq e_{ij} + e_{ji},\ i = 1, \ldots, n,\ j = 1, \ldots, n,\ i,j \in J_l,\ l \in L,\ i \neq j, \tag{5.65}$$

$$X_{ij} \in \{0,1\},\ i = 0, \ldots, n,\ j = 1, \ldots, n,\ i \neq j, \tag{5.66}$$

$$e_{ij} \in \{0,1\},\ i = 1, \ldots, n,\ j = 1, \ldots, n,\ i,j \in J_l,\ l \in L,\ i \neq j. \tag{5.67}$$

The objective is to minimize the TWC performance measure. Constraints (5.57) ensure that each job is assigned to exactly one machine and has exactly one predecessor. The dummy job 0 is used within $X_{0k}, k \in \{1, \ldots, n\}$. Constraints (5.58) model that each job except for the dummy job has at maximum one successor. The number of successors of the dummy jobs is at most m because of constraint (5.59). When a given job i is processed on a machine, then a predecessor must exist on the same machine. This is modeled by constraints (5.60). When a job k is assigned to a machine immediately after job j, i.e., when $X_{jk} = 1$, its completion time has to be greater than the sum of the completion time of j and the processing time of k. This is expressed by constraints (5.61). In case of $X_{jk} = 0$, constraints (5.61) are also fulfilled because of the big M. Constraint (5.62) sets the completion time of the dummy job to zero. Constraints (5.63) ensure that the completion time of a job is always larger than the sum of its ready time and processing time. Constraints (5.64) and (5.65) ensure that if two jobs require the same reticle, one of the jobs has to complete its processing before the other job starts its processing. Finally, constraints (5.66) and constraints (5.67) model the fact that the main decision variables X_{ij} and e_{ij} only take on binary values.

The MIP formulation (5.56)–(5.67) has been used to solve problem instances up to $m = 2$ and $n = 15$ and up two six layers within reasonable time optimally. Therefore, again we can use the MIP formulation (5.56)–(5.67) to assess whether heuristics are correctly implemented and to get some insights into the potential performance of the proposed heuristics.

Next, we describe a heuristic for problem (5.55) that is proposed in [42]. The heuristic consists of two phases. In the first phase, the heuristic determines a feasible solution using dispatching rules within a list scheduling approach that takes the auxiliary resources into account. Then, in a second step, this solution is improved by tabu search. The algorithm of the first phase is summarized as follows.
Algorithm Construction Heuristic (CH)

1. Determine the current values of t and φ, where t is the current time and φ represents a machine that becomes idle at time t. The quantity Φ is the set of unscheduled jobs, and Θ is the set of candidate jobs. Initially, set $t = 0$, $\varphi = 1$, $\Phi := \{1, \ldots, n\}$, and finally $\Theta := \Phi$.
2. For each $j \in \Theta$ determine the index

$$I_j(t) := \begin{cases} 0, \text{ if } t < r_j \\ w_j/p_j, \text{ otherwise} \end{cases} . \tag{5.68}$$

If for all $j \in \Theta$ the condition $t < r_j$ is valid, then use the index

$$I_j(t) := w_j/(p_j + r_j - t) \tag{5.69}$$

instead of index (5.68). The job k with the highest index value will be chosen as a candidate to be assigned to machine φ.

3. When k can be feasibly assigned to φ at $\max(t, r_k)$, i.e., when an appropriate reticle is available, then go to step 4; otherwise, go to step 5.
4. Assign k to φ at time $\max(t, r_k)$. Update $\Phi := \Phi - \{k\}$ and go to step 6.
5. Update $\Theta := \Theta - \{k\}$. When $\Theta = \emptyset$ holds, then go to step 6; otherwise, go to step 2.
6. The value of t has to be updated to the smallest future point of time in which a feasible assignment of a job to a machine is possible, i.e., when a new job is ready for processing or when a machine becomes available. Set $\Theta := \Phi$. If $\Theta \neq \emptyset$, go to step 2; otherwise, stop.

Note that the indices (5.68) and (5.69) are influenced by the fact that the WSPT dispatching rule is optimal for the scheduling problem $1||\mathrm{TWC}$ (cf. Pinedo [240]). The algorithm of the second phase is a tabu search variant. A neighborhood of a feasible solution x of problem (5.55) consists of all pairwise job swaps. After a swap of two jobs, a feasible schedule is obtained by a placement heuristic [42]. The resultant two-phase heuristic is called CH+TS.

Next, we discuss the results from computational experiments. A variety of problem instances are examined to evaluate the efficacy of CH and CH+TS. First, two different levels are considered for the number of layer types, sampling from a discrete uniform distribution $\mathrm{DU}[1, v]$ with $v \in \{3, 6\}$. Experiments are also performed for both two and three machines operating in parallel. For each job, an integer processing time is generated from $\mathrm{DU}[45, 75]$. One-half of the job ready times are generated from $\mathrm{DU}[1, 360]$, while the second half of the jobs have $r_j = 0$. Further, job weights are selected according to $\mathrm{DU}[1, 20]$. The design of experiments is summarized in Table 5.6.

Table 5.6: Design of experiments for problem (5.55)

Factor	Level	Count
m	2, 3	2
n	10, 15	2
p_j	$\sim \mathrm{DU}[45, 75]$	1
w_j	$\sim \mathrm{DU}[1, 20]$	1
r_j	$50\,\% \sim \mathrm{DU}[1, 360]$ and $50\,\%$ $r_j \equiv 0$	1
l	$\sim \mathrm{DU}[1, v]$, $v \in \{2, 3\}$	2
	Total factor combinations	8
	Number of problem instances per combination	10
	Total number of problem instances	80

We compare the TWC values obtained by CH and CH+TS, respectively with the optimal TWC results from the MIP formulation (5.56)–(5.67) by presenting the ratios in Table 5.7.

Table 5.7: Computational results for problem (5.55)

Compare	CH	CH+TS
m		
2	1.012	1.007
3	1.022	1.008
n		
10	1.014	1.004
15	1.021	1.012
l		
1–3	1.021	1.011
1–6	1.014	1.005
Overall	1.017	1.008

We see from Table 5.7 that CH and CH+TS perform well. CH+TS was able to determine the optimal solution for 76 problem instances. CH+TS slightly outperforms CH.

A couple of real-world conditions are not included in problem (5.55). The only additional constraint compared to other parallel machine scheduling problems is the limited number of reticles. In real-world situations, however, many more constraints appear. For example, the steppers often have to be modeled as unrelated parallel machines due to their different generations. Steppers generally have a local reticle stocker that can hold only a certain small number of reticles. Unfortunately, wafer fabs processing a wide variety of products may need to have a large number of different reticles available for use at any given time. Therefore, central stockers located near the stepper work center are sized with sufficient capacity to manage all reticles required in the facility. Because the pattern on each reticle is important in the manufacturing process, moving reticles from one place to another in a wafer fab is not trivial. Loading a reticle onto a machine, called a machine setup, must be done very carefully. In some wafer fabs, every time a reticle is moved, it must be inspected either to ensure that the original pattern is intact or to be sure that there are no unwanted particles of dust or vapor on the surface. Therefore, jobs that require the same reticle are often processed in a consecutive manner on the same machine to avoid frequent reticle changes.

Send-ahead wafers for steppers are also common to perform quality measurements with a single wafer. In this situation, a single wafer out of a certain job is processed instead of the entire job. Because of the required reticle change, a small number of send-ahead wafers is desirable. In order to process a job on a stepper, the job has to be ready, the stepper has to be idle, and finally, the reticle has to be inspected and set up on the idle stepper. In the remainder of this section, we will briefly discuss approaches from the literature that address some of these constraints.

Akçali and Uzsoy [4] studied the shift-scheduling problem of a photolithography station and created a policy that efficiently distributes the workload across all machines for an entire shift. In their capacity allocation routine (CAR), all waiting jobs are categorized based on the process step they await, and all machines are categorized based on how much work has already been allocated to them. With this information, the CAR incorporates operational and auxiliary resource constraints and uses a greedy heuristic to assign the largest group of waiting jobs to the machine with the highest amount of available capacity and preferably, the required reticle. In the second step, a sequencing problem for jobs on each machine is solved to create a detailed schedule for an entire shift. A similar approach is discussed by Klemmt et al. [147].

A network flow model is proposed by Dìaz et al. [68] to efficiently load reticles onto machines based on the jobs that are soon available to be processed. The current location of all reticles and information of jobs that are waiting for processing or that will be ready for processing soon are used as inputs for the model. The objective is to minimize the number of setups and inspections over the next shift.

The assignments of reticles can be used to determine schedules by list scheduling using the following variant of the ATCS dispatching rule:

$$I_{lj}(t) := \frac{w_j}{p_j} \exp\left(-\frac{(d_j - p_j - t)^+}{\kappa_1 \bar{p}}\right) \exp\left(-\frac{s_{lj}}{\kappa_2 \bar{s}} - \frac{\max(r_j - t, a_{lj}, 0)}{\kappa_3 \bar{p}}\right), \tag{5.70}$$

where we denote by a_{lj} the time that is required to make the reticle for job j ready on a machine that has just finished processing job l and κ_3 is a third look-ahead parameter. Index (5.70) has the advantage compared to index (5.68) that the availability of the reticle and the required setups are more directly taken into account. Computational results using a simulation environment similar to that described in Sect. 3.3.2 can be found in [68].

Operational problems related to send-ahead wafers are discussed by Akçali et al. [5]. In this paper, the main performance measure is CT. A GA for scheduling steppers that takes send-ahead wafers and reticle constraints into account is described by Mönch [189]. A combined objective function is taken that considers TP, TWT, and the number of send-ahead wafers.

Finally, we discuss a different example from the back-end area provided by Kempf et al. [137]. A single burn-in oven is considered. The auxiliary resources are given by load boards (cf. Sect. 2.2.2). A burn-in oven is a batch-processing machine. The oven capacity is given by the number of boards that an oven can carry. The size s of a single job is given by the number of boards that are required to process the job. Because the board type required by a job depends on the packaging of the circuits, different jobs may require different load boards. Only a limited number of boards are available for each board type. The concept of incompatible job families as already described for scheduling problems (5.2) and (5.39) can be used to model the fact that certain jobs have to be kept in the oven for the same amount of time. Only jobs of the same family can be batched together. Jobs of the same family may use different types of boards. Using the $\alpha|\beta|\gamma$ notation, the scheduling problem can be modeled as

$$1|\text{p-batch}, \text{incompatible}, s, \text{aux}|\text{TC}, \tag{5.71}$$

where s refers to different sizes of the jobs. Because this problem is NP-hard [137], heuristics are suggested that also take the load board constraint into account. Because of the capacity constraints of the oven, the heuristics are inspired either by full-load strategies or by first-fit decreasing (FFD) heuristics for bin packing (see Dowsland and Dowsland [70]). Furthermore, a heuristic that is based on a relaxation of an IP formulation for problem (5.71) is also discussed.

5.3.6 Multiple Orders per Job Scheduling Problems

In this section, we discuss another class of scheduling problems that can be found in 300-mm wafer fabs. A FOUP is the standard unit of job transfer between machines in a wafer fab. FOUPs are expensive. More importantly, a large number of FOUPs can cause a congested AMHS. Therefore, the number of FOUPs is limited and is often a restriction. Because of the combination of decreased line width and increased area per wafer, fewer wafers are required to fill the orders for ICs than was the case before. Therefore, there is quite often a need to group orders of different customers into one FOUP (cf. the description in Sect. 2.2.3). We call the resulting entity a job to conform with scheduling literature. The jobs have to be scheduled on the different machines within a wafer fab when such a grouping decision for orders is made. Consequently, the resulting class of problems are called MOJ scheduling problems.

We consider a set of orders $O := \{1, \ldots, N\}$. Each order $o \in O$ has a size s_o, measured in number of wafers, and a weight w_o that is used to model the importance of order o. Let K denote the capacity of a FOUP measured in wafers. For simplicity, we assume that $s_o \leq K$ for all $o \in O$. There are F FOUPs available.

The assumptions for scheduling MOJ on a single machine are as follows:

1. All the orders are available at time $t = 0$.
2. The orders are of the same product type, i.e., all the orders can be grouped together.
3. Once a machine is started, it cannot be interrupted, i.e., no preemption is allowed.

4. All the orders in a job are processed together. The completion time of the job determines the completion time of all the orders that form the job.
5. There are two different processing modes. In the lot processing mode, the job processing time is just the wafer processing time ρ. The processing time of job j is given as $p_j = \rho \sum_{o \in j} s_o$ in the single item processing mode.

Using the $\alpha|\beta|\gamma$ notation, the two scheduling problems can be represented as follows. In the lot processing case, we obtain

$$1|\text{moj(lot)}|\text{TWC}, \tag{5.72}$$

where we have $\text{TWC} := \sum_{o \in O} w_o C_o$. We denote by C_o the completion time of o. The corresponding scheduling problem in the single item case can be represented as

$$1|\text{moj(item)}|\text{TWC}. \tag{5.73}$$

It is proved by Mason and Chen [171] that problems (5.72) and (5.73) are NP-hard. Hence, we have to look for efficient heuristics.

We present a MIP formulation from Mason et al. [174] that covers both problems (5.72) and (5.73). We use the following indices and index sets in the corresponding MIP model:

$j = 1, \ldots, F$: job index
$o = 1, \ldots, N$: order index
O : set of all orders

The following parameters will be used within the model:

p_j : processing time of job j in the lot processing case
w_o : weight of order o
s_o : size of order o
K : capacity of the FOUP
M : large number

The following decision variables are used within the MIP:

$$X_{oj} : \begin{cases} 1, & \text{if order } o \text{ is assigned to job } j \\ 0, & \text{otherwise} \end{cases}$$

C_j : completion time of job j
p_j : processing time of job j in the single item processing case
δ_o : completion time of order o

Note that we have $p_j = \rho$ in the lot processing case and $p_j = \rho \sum_{o \in O} s_o X_{oj}$ in the single item case. Therefore, depending on the MOJ environment, p_j is a parameter or a decision variable. Furthermore, without loss of generality, we assume that there is a given sequence of the FOUPs. Now the scheduling problems (5.72) and (5.73) can be formulated as follows:

$$\min \sum_{o \in O} w_o \delta_o \tag{5.74}$$

subject to:

$$\sum_{j=1}^{F} X_{oj} = 1, \quad o = 1,\ldots,N, \tag{5.75}$$

$$\sum_{o\in O} s_o X_{oj} \le K, \quad j = 1,\ldots,F, \tag{5.76}$$

$$C_j - M(1 - X_{oj}) \le \delta_o, \quad j = 1,\ldots,F,\ o = 1,\ldots,N, \tag{5.77}$$

$$p_j \le C_j, \quad j = 1,\ldots,F, \tag{5.78}$$

$$C_j + p_{j+1} \le C_{j+1}, \quad j = 1,\ldots,F-1, \tag{5.79}$$

$$\delta_o \ge 0,\ C_j \ge 0,\ X_{oj} \in \{0,1\}, \quad j = 1,\ldots,F,\ o = 1,\ldots,N. \tag{5.80}$$

The objective is to minimize the TWC performance measure. Constraints (5.75) model that each order is assigned to exactly one job. The capacity restrictions for each FOUP are represented by constraints (5.76). The completion time of o is calculated using constraints (5.77). Constraints (5.78) ensure that each job spends at least its processing time in the system. Constraints (5.79) are used to properly sequence the jobs according to their indices. Finally, constraints (5.80) model the fact that the main decision variables X_{oj} take only binary values and the completion times of orders and jobs are nonnegative. The model is completed by adding constraints

$$p_j = \rho \sum_{o\in O} s_o X_{oj} \tag{5.81}$$

to the model in the single item case. We set $p_j = \rho$ in the lot processing case. Note that problem instances up to 20 orders have been solved to optimality in reasonable time using the MIP model (5.74)–(5.81). Therefore, we continue with the discussion of heuristics to solve large-size problem instances.

We describe a heuristic that is influenced by bin packing algorithms (cf. Dowsland and Dowsland [70] for bin packing-related information). The heuristic consists of a first phase where orders are sequenced by a dispatching rule. Then, the jobs are formed based on a bin packing-type heuristic. The heuristic for the lot processing case can be summarized as follows.
Algorithm MOJ Lot Processing

1. Sort the orders from O with respect to nonincreasing values of the weighted smallest size (WSS) index
$$I_o := w_o/s_o. \tag{5.82}$$
2. Let Φ be the set of orders that are not assigned to a job. Set initially $\Phi := \{1,\ldots,N\}$. Sort Φ according to the order sequence obtained in step 1. The current job is denoted by j_{curr}. Set initially $j_{\text{curr}} := 1$.

3. Form j_{curr} by going through $\mathbf{\Phi}$ starting with the order with the largest WSS index and taking the capacity constraints of the job into account. Update $\mathbf{\Phi}$ when an order is placed in j_{curr}.
4. If $j_{\text{curr}} = F$ or $\mathbf{\Phi} = \emptyset$, then stop; otherwise, set $j_{\text{curr}} := j_{\text{curr}} + 1$. Go to step 3.

Note that using the WSS rule in step 1 is motivated by the fact that the WSPT dispatching rule leads to an optimal schedule for $1||\text{TWC}$. The FFD heuristic is used for the job formation phase. The job formation part of the algorithm tries to form jobs that use all of the available capacity of the FOUPs. Because of the WSS sorting, the jobs formed early in the process contain more orders with small s_o, and therefore they are often more fully loaded. Solutions of problem (5.72) with small TWC value tend to use a small number of FOUPs.

We now describe a different algorithm for problem (5.73) because the processing times of the jobs are determined by the actual schedule.
Algorithm MOJ Single Item Processing

1. Sort the orders from O with respect to nonincreasing values of the WSS index.
2. Let $\mathbf{\Phi}$ be the set of orders that are not assigned to a job. Set initially $\mathbf{\Phi} := \{1, \ldots, N\}$. Sort $\mathbf{\Phi}$ according to the order sequence obtained in step 1. Set initially $F_{\text{curr}} := F$. Let j_{curr} be the current job. Initialize it with $j_{\text{curr}} := F$.
3. Determine the sum of the sizes of the jobs from $\mathbf{\Phi}$. Divide this sum by F_{curr}. This gives the expected average job size for each remaining job.
4. Start assigning orders from $\mathbf{\Phi}$ to j_{curr} starting with the order with smallest WSS index taking the job capacity and the expected average job size from step 3 into account. Update $\mathbf{\Phi}$ when an order is inserted into j_{curr}. Then, update F_{curr}.
5. When $j_{\text{curr}} = 1$ or $\mathbf{\Phi} = \emptyset$, then stop; otherwise, set $j_{\text{curr}} := j_{\text{curr}} - 1$ and go to step 3.

The job formation part of the algorithm tries to balance the sizes of the formed jobs. The job size is calculated as the sum of the sizes of the orders that form the job. Later jobs are being more fully loaded than earlier jobs in order to avoid unnecessary delay times in solutions that have small TWC values. Therefore, we start the job formation with job j_F and assign orders with small WSS index to the later jobs. Solutions of problem (5.73) with small TWC value tend to use as many FOUPs as possible.

The two algorithms decompose the scheduling problems into an order-sequencing subproblem and a job-formation subproblem and solves them in this sequence. Now we take the opposite perspective and decompose the scheduling problems into a job-formation problem and a job-sequencing problem. Based on Proposition 5.3, it turns out that the second subproblem can be solved in a straightforward manner.

Proposition 5.3 *When the jobs are formed in the lot processing environment, then an optimal solution of the job-sequencing subproblem can be obtained by simply sequencing the jobs in nonincreasing order with respect to $\sum_{o\in j} w_o$. Moreover, when the job formation decision is made in the item processing mode, then the optimal job sequence can be determined by sorting the jobs with respect to nonincreasing values of $\sum_{o\in j} w_o / \sum_{o\in j} s_o$.*

Proof. The proof is based on the fact that the WSPT dispatching rule leads to an optimal schedule for 1||TWC. In the lot processing case, all jobs have the same processing time. Hence, instead of considering the WSPT index given by expression (4.8), it is enough to sort the jobs with respect to $\sum_{o\in j} w_o$. Furthermore, we have $p_j = \rho \sum_{o\in j} s_o$ in the single item processing mode. Therefore, it is sufficient to sort the jobs using $\sum_{o\in j} w_o / \sum_{o\in j} s_o$. □

Grouping GAs (GGAs) (see Falkenauer [78]) are proposed by Sobeyko and Mönch [286] to tackle the job formation phase. A GGA uses an encoding scheme based on genes that represent the groups because groups are the meaningful building blocks of a solution. This means for problem (5.72) and problem (5.73) that jobs are the groups to be formed.

We expect that the performance of the algorithms depend on the size of the orders and the FOUP capacity. Therefore, we consider 40 small-size problem instances for both the lot processing and the single item processing mode. The corresponding design of experiments can be found in Table 5.8.

Table 5.8: Design of experiments for problem (5.72) and (5.73)

Factor	Level	Count
N	10	1
ρ	10	1
w_o	$\sim$ DU$[1,15]$	1
s_o	$\sim$ DU$[\frac{v-1}{2}, \frac{3v+1}{2}]$, $v \in \{3,5\}$	2
K	$12\beta+1$, where $\beta \in \{1,2\}$	2
F	$\lceil Nv/(12\beta)\rceil + 1$	1
	Total factor combinations	4
	Number of problem instances per combination	10
	Total number of problem instances	40

We assess the performance of the two heuristics by solving the problem instances to optimality using the MIP formulation (5.74)–(5.81). The ratio $R(H) := \mathrm{TWC}(H)/\mathrm{TWC}(\mathrm{MIP})$ is considered, where we denote by $\mathrm{TWC}(H)$ the TWC value for a problem instance using one of the heuristics and by $\mathrm{TWC}(\mathrm{MIP})$ the corresponding TWC value obtained by the MIP formulation. It turns out that in the lot processing mode, we obtain $R(H)$ values between 1.005 and 1.020, whereas the corresponding values are between 1.010 and 1.020 in the single item processing mode. Note that the GGA from [286] almost always finds the optimal solution for problem instances with $N = 10$.

The methods for this single machine MOJ scheduling problem without ready times of the orders were extended to single machine MOJ scheduling problems with ready times by Qu and Mason [254], to parallel machine situations by Jia and Mason [129], to flow shops by Laub et al. [149], and finally to job shops by Jampani and Mason [126] and by Jampani et al. [127].

5.4 Full Factory Scheduling

In this section, we consider scheduling approaches for full wafer fabs, which can be modeled from a scheduling point of view as complex job shops (cf. Sect. 2.2.3). We discuss the disjunctive graph representation of complex job shop scheduling problems. The shifting bottleneck heuristic (SBH) is introduced and subproblem solution procedures (SSPs) are discussed. Moreover, a distributed variant of this scheduling heuristic is described. Rolling horizon approaches and their simulation-based performance assessment are discussed. Finally, a multicriteria extension is presented.

5.4.1 Motivation and Problem Statement

The assumption for scheduling jobs in complex job shops are:

1. The job shop consists of groups of unrelated parallel machines.
2. Once a machine is started, it cannot be interrupted, i.e., no preemption is allowed.
3. The jobs j have unequal ready times.
4. Each job j has a process flow $O(j) := \{O_{j1}, \dots, O_{jn_j}\}$, where we denote by O_{jk} the kth operation of j. Totally, we have n_j operations.
5. Some of the machines have sequence-dependent setup times.
6. There are batch machines in the job shop. Only jobs of the same job family can be batched together.
7. Reentrant process flows are considered.

Using the $\alpha|\beta|\gamma$ notation scheme, the scheduling problem to be solved can be described as follows:

$$\text{FJm}|\text{p-batch}, \text{incompatible}, s_{jk}, r_j, \text{recrc}|\text{TWT}. \tag{5.83}$$

We use the completion time C_j of job j with respect to the entire wafer fab and the corresponding due date d_j to calculate the TWT value. The scheduling problem (5.83) is NP-hard because the single machine scheduling problem $1||\text{TWT}$ that is known to be NP-hard (see Lawler [151]) is a special case of it. Hence, we have to look for efficient heuristics. In the following, we will study mainly decomposition heuristics. These heuristics are based on the concept of disjunctive graphs (cf. Brucker and Knust [35]) that will be described in Sect. 5.4.2.

Until recently, full factory scheduling methods seemed to be too computationally costly in comparison to dispatching methods. However, with the recent dramatic increase in computer efficiency, full fab scheduling methods have become more competitive.

Decomposition heuristics for complex job shops have been studied very intensively over the last ten years starting with the pioneering work of Uzsoy et al. [305] for a test facility. A good documentation of attempts to solve the problem

$$\text{FJm}|\text{p-batch}, s_{jk}, r_j, \text{recrc}|L_{\max} \tag{5.84}$$

for test facilities can be found in Ovacik and Uzsoy [223]. Decomposition approaches for problem (5.83) were studied for the first time by Mason *et al.* [172]. In this section, we will mainly discuss this approach and some extensions of it. We will also show how a simulation-based performance assessment can be carried out for this type of full factory scheduling approach.

5.4.2 Disjunctive Graph Representation for Job Shop Problems

Job shop scheduling problems can be represented by disjunctive graphs (cf. [1, 35, 223]). Many papers have discussed the use of disjunctive graphs for the scheduling problem $\text{Jm}||C_{\max}$. Therefore, we start describing the basic properties for disjunctive graphs with respect to this problem, but later we will add several extensions needed to adequately model wafer fabs. A disjunctive graph is given by the triple

$$G = (V, E_{\text{c}}, E_{\text{d}}). \tag{5.85}$$

We denote by V the set of nodes of the graph. E_{c} is used for the set of conjunctive arcs, whereas E_{d} denotes the set of disjunctive arcs. We set $E := E_{\text{c}} \cup E_{\text{d}}$. Furthermore, we define an incidence mapping

$$\omega : E \to V \times V. \tag{5.86}$$

A node $v \in V$ represents an operation of a job on a machine. We use the notation $\langle i, j \rangle$ for a $v \in V$ that represents an operation of job j on one of the machines of machine group i. We may have several operations of a job on machines of one machine group due to reentrant flows of the jobs within a wafer fab. In this case, we use the notation $\langle i, ja \rangle, \langle i, jb \rangle, \ldots$ to distinguish between these operations. Furthermore, the node set contains an artificial starting node s and an artificial end node e. We need also an end node $V_j,\ j = 1, \ldots, n$ for each job. The quantity n is the number of considered jobs. The end nodes represent the due dates d_j of each job j.

Conjunctive arcs are defined as directed arcs (u, v) from $u \in V$ to $v \in V$. They indicate a direct precedence relationship between these nodes. This relationship is either based on the process flow of the job or on a scheduling decision. In the first case, u and v belong typically to different machine groups. In the latter case, u and v are associated with the same machine group.

Disjunctive arcs are defined as undirected arcs $e := \{u, v\}$ between nodes $u, v \in V$ associated with operations that are performed on machines of the same machine group. A disjunctive arc e represents the following situations:

1. There is no directed arc between u and v.
2. Alternatively, there exists either the directed arc (u, v) or the directed arc (v, u).

In the first situation, the operations that are associated with u and v are not performed in a consecutive manner on the same machine. There is such a consecutive processing in the second situation. Therefore, disjunctive arcs are used to model scheduling decisions to be made. Some of the disjunctive arcs will be eliminated, and some of them will be changed into conjunctive arcs in the course of the scheduling algorithms. Figure 5.5 shows a disjunctive graph for a job shop that consists of four machines $\{1, 2, 3, 4\}$ and three jobs $\{1, 2, 3\}$. Dashed arcs are used to represent disjunctive arcs. A schedule can be determined by eliminating disjunctive arcs. That means that we have to solve a scheduling problem for each machine group. Then, we replace the disjunctive arcs that are associated with scheduling decisions for each single machine of the machine group by a conjunctive arc chain, and we remove the other arcs.

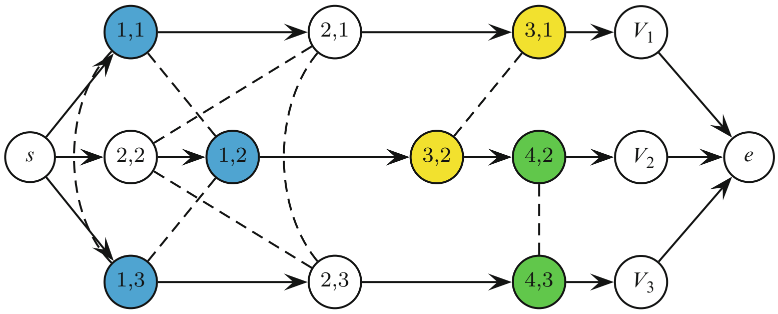

Figure 5.5: Disjunctive graph representation for four jobs on three machines

Such a chain is assigned to each of the machines of the machine groups. An arc chain connects nodes in such a way that the direct precedence relationship among the operations due to the schedule is translated into the graph. Of course, we have to make sure that the resulting graph does not contain any cycle, because otherwise the schedule is not feasible. We show an example of a partial schedule based on the example used for Fig. 5.5 in Fig. 5.6. The schedule for machine 2 is implemented within G. The operation that corresponds to $\langle 2,1\rangle$ is sequenced before the operation that is associated with $\langle 2,3\rangle$. Similarly, the operation associated with $\langle 2,3\rangle$ is sequenced before the operation belonging to $\langle 2,2\rangle$.

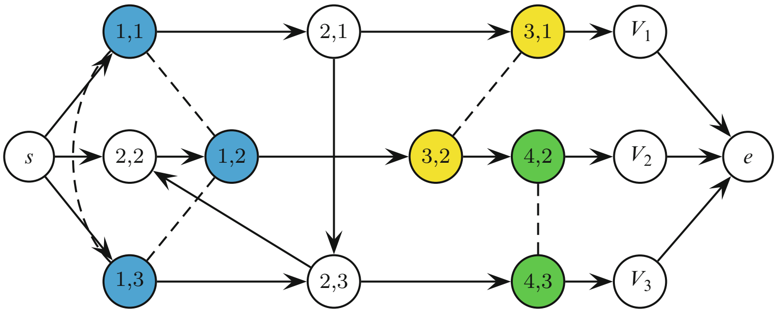

Figure 5.6: Representation of a partial schedule

We complete the description of the main concepts of disjunctive graphs by assigning weights to the arcs. These weights are necessary to represent the time delay caused by the processing of previous operations of the different jobs. We denote the weight of an arc (u,v) as $p(u,v)$. We differentiate between several cases:

- Each conjunctive arc (u,v) with $u \neq s$ and $u \neq V_j$ gets the processing time of the operation that is associated with the node u. Because u is nonartificial, this setting is well defined.
- The ready time of the job that is associated with node u is assigned to arcs of the form (s,u).
- The weight 0 is assigned to arcs of the form (V_j,e).
- Because undirected arcs do not cause any time delay, a weight with value 0 is assigned to them.

Note that all the conjunctive arcs of the form (u,v) with nodes $u \neq s$ have the same weight. Therefore, we simplify the notation and call the corresponding weight $p(u)$.

After introducing weights for the arcs of G we can see the main advantage of modeling job shop scheduling problems by disjunctive graphs. G allows us to evaluate the influence of single scheduling decisions on the entire job shop. In order to do so, we have to determine ready times and due dates for the individual operations of the jobs on the machines of the machine groups by performing longest path calculations. A path in G is defined as a sequence of directed arcs $e_1,\dots,e_k$ such that a sequence of nodes $v_0,\dots,v_k$ exists, such that $\omega(e_j) = (v_{j-1},v_j)$. The length of a path is given as the sum of the weights of the arcs that form the path. Shortest and also longest paths can be efficiently calculated using Dijkstra's algorithm (cf. Brucker and Knust [35]). This type of calculation is also used in the critical path method in project scheduling.

The ready time of an operation associated with node u is denoted by $r(u)$. It is the minimum release time of the operation associated with u when we assume that all operations associated with nodes that precede u are started at the ready times that corresponds to these nodes. This is the longest path from s to u. The following recursion is used to set the ready times for a node $v \in V - \{s\}$:

Initial condition:

$$r(s) := 0, \tag{5.87}$$

Recursive relation:

$$r(v) := \max\{r(u) + p(u,v) | v \in V, (u,v) \in E_c\} \tag{5.88}$$

if the operation that is associated with v has to be performed on a machine of an unscheduled machine group. If u is scheduled on a machine with availability time r_a, then we have to set

$$r(v) := \max\{r_a, r(v)\}. \tag{5.89}$$

Based on the ready times, we are able to introduce completion times. The completion time $c(u)$ of an node $u \in V - \{e,s\}$ is defined by

$$c(u) := r(u) + p(u). \tag{5.90}$$

For e, we define furthermore

$$c(e) := r(e). \tag{5.91}$$

The due date of the operation that is associated with node $u \in V - \{e,s\}$ can be determined by the following recursion:

Initial condition:

$$d(e) := r(e), \tag{5.92}$$

Recursive relation:

$$d(u) := \min\{d(v) - p(v) | v \in V, (u,v) \in E_c\}. \tag{5.93}$$

By solving the recursion (5.92), (5.93), it can be shown that $d(u) - p(u)$ for $u \in V - \{s,e\}$ is the difference between $c(e)$ and the length of the longest path from u to e. The length of the longest path represents the sum of the processing times and the waiting times of the remaining operations of the job that is associated with u. Therefore, $d(u) - p(u)$ can be considered as the latest time when the operation associated with node u has to start to avoid an increased value of $C_{\max}$ of the jobs.

Sophisticated production conditions like parallel machines, sequence-dependent setup times, batch machines, and reentrant flows are important in complex job shops. Therefore, we describe how these characteristics can be modeled by extensions of the disjunctive graph G for $\mathrm{Jm}||C_{\max}$ following [172, 225].

Parallel machines are included in a natural way in G. Only those nodes are connected after a scheduling decision that represent jobs to be processed on a specific machine. We obtain arc chains as already described by the implementation of scheduling decisions within G.

Sequence-dependent setup times can be incorporated into G only after the scheduling decisions for the jobs on a machine group are made. Therefore, we increase the weight of the corresponding arc by the setup time. The initial setup of a machine is included into the weights of the outgoing arcs of the node that corresponds to the first scheduled operation on the machine.

An appropriate modeling of batch machines is more sophisticated [172, 223]. A scheduling decision for a batch machine includes three types of decisions (cf. the description in Sect. 5.3):

1. Batching decisions: which jobs should form a certain batch
2. Assignment decisions: which batch should be assigned to a certain machine
3. Sequencing decisions: in which sequence should batches be processed on a given machine

After solving the scheduling problem for machine groups that contain batch machines, artificial batch nodes are added to G for the formed batches. The predecessors of the artificial batch nodes are those nodes that represent the jobs that form the batch. They are called batched nodes. We denote the set of all batched nodes by V_{b}. For each $v \in V_{\mathrm{b}}$, the corresponding artificial batch node is denoted by v_{b}. The weight 0 is used for the incoming arcs of an artificial batch node because the processing time is represented by the outgoing arcs. The outgoing arcs of the artificial batch node connect the batch node with the successor nodes according to the process flows of the jobs that form the batch. Because we consider batching with incompatible families, all jobs within a batch have the same processing time. Therefore, it is reasonable that the weight of each outgoing arc of an artificial batch node is set to be the processing time of the batch.

In Fig. 5.7, we depict a disjunctive graph where machine 2 is a batch machine. The jobs represented by node $\langle 2,2\rangle$ and node $\langle 2,3\rangle$ form a batch. Rectangles with rounded angles are used to represent batches. Furthermore, we consider a second batch consisting only of one job that corresponds to node $\langle 2,1\rangle$. The second batch is sequenced immediately after the first batch.

The ready time determination scheme is also valid in graphs with artificial batch nodes. The due date determination scheme (5.92), (5.93) can be also applied to calculate the due dates of artificial batch nodes because the weights of the outgoing arcs are defined as in the non-batching case. However, the recursion equation (5.93) has to be modified for predecessors of batched nodes. We obtain

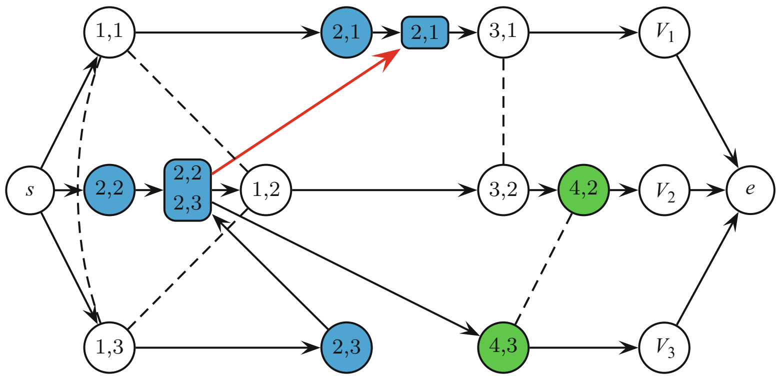

Figure 5.7: Representation of the batches $\{\langle 2,2\rangle, \langle 2,3\rangle\}$ and $\{\langle 2,1\rangle\}$ in G

$$\tilde{d}(u) := \min\{d(v) - p(v) | v \in V - V_b, (u,v) \in E_c\}. \quad (5.94)$$

We can derive $d(u)$ based on $\tilde{d}(u)$ as follows:

$$d(u) := \min\left\{\tilde{d}(u), \min\{d(v) - p(v_b) | v \in V_b, (u,v) \in E_c,\}\right\}. \quad (5.95)$$

This modification is necessary because $p(v) = 0$ for $v \in V_b$. Hence, Eq. (5.94) is not reasonable in this situation.

Finally, we have to consider the modeling of reentrant process flows. They are modeled in such a way that the corresponding nodes are not connected by disjunctive arcs. In this situation, the nodes are already connected by conjunctive arcs according to the process flow of the job.

As in the case of batch machines, new process restrictions can often be modeled by introducing additional nodes and arcs into G. For example, the integrated scheduling of manufacturing and transportation operations for complex job shops with AMHS is tackled in such a way for job shops by Knust [148] and by Driessel and Mönch [73] for complex job shops.

The disjunctive graph model can be used to find MIP formulations for job shop scheduling problems. We illustrate this modeling approach by means of the problem $Jm||C_{\max}$ (see Brucker and Knust [35]). The following parameters will be used within the model:

p_{ij} : processing time of the operation that is associated with node $\langle i,j\rangle$
M : large number

The following decision variables are used within the MIP:

S_{ij} : starting time of the operation that is associated with node $\langle i,j\rangle$

$$y_{\langle i,j\rangle,\langle i,l\rangle} : \begin{cases} 1, \text{ if the operation associated with } \langle i,j\rangle \text{ is scheduled before } \langle i,l\rangle \\ 0, \text{ otherwise} \end{cases}$$

$C_{\max}$: makespan of the schedule

Let N be the set of nodes that are associated with operations. By defining set A as the pairs of operations constrained by process flow relations and E_i the set of operations to be performed on machine i, the scheduling problem $\mathrm{Jm}||C_{\max}$ can be formulated as follows:

$$\min\ C_{\max} \tag{5.96}$$

subject to:

$$S_{ij} + p_{ij} \le S_{kj}, \quad (\langle i,j\rangle, \langle k,j\rangle) \in A, \tag{5.97}$$

$$S_{ij} + p_{ij} \le C_{\max}, \quad \langle i,j\rangle \in N, \tag{5.98}$$

$$S_{ij} + p_{ij} - M\left(1 - y_{\langle i,j\rangle,\langle i,l\rangle}\right) \le S_{il}, \quad \langle i,j\rangle, \langle i,l\rangle \in E_i,\ i = 1,\ldots,m, \tag{5.99}$$

$$S_{il} + p_{il} - My_{\langle i,j\rangle,\langle i,l\rangle} \le S_{ij}, \quad \langle i,j\rangle, \langle i,l\rangle \in E_i,\ i = 1,\ldots,m, \tag{5.100}$$

$$0 \le S_{ij}, \quad \langle i,j\rangle \in N, \tag{5.101}$$

$$y_{\langle i,j\rangle,\langle i,l\rangle} \in \{0,1\},\ \langle i,j\rangle, \langle i,l\rangle \in E_i,\ i = 1,\ldots,m. \tag{5.102}$$

The objective (5.96) is to minimize the $C_{\max}$ performance measure. Constraints (5.97) model the fact that an operation cannot be started earlier than the sum of the start time of the predecessor operation and the corresponding processing time of this operation. Note that these constraints represent the conjunctive arcs. Constraints (5.98) ensure that $C_{\max}$ is equal to the largest completion time of the jobs. Constraints (5.99) and (5.100) model the fact that only one of the disjunctive arcs can be selected for two operations that have to be processed on the same machine. Finally, the constraints (5.101) lead to non-negative start times of the operations, while constraints (5.102) enforce binary values for the $y_{\langle i,j\rangle,\langle i,l\rangle}$.

An extension of this MIP to problem (5.83) can be found in Mason et al. [175]. In documented computational experiments, only problem instances up to eight jobs have been tackled using the resulting MIP. This is another indication for the need to design appropriate heuristics.

5.4.3 Decomposition Approach

The disjunctive graph modeling approach introduced in Sect. 5.4.2 forms the base for heuristics based on decomposition or on neighborhood search approaches for $\mathrm{Jm}||C_{\max}$ (cf. Brucker and Knust [35] for a description of such techniques). Tabu search approaches are very competitive for $\mathrm{Jm}||C_{\max}$ (see Nowicki and Smutnicki [215]) mainly because neighborhood structures can be designed that allow for fast evaluation of the $C_{\max}$ objective; however, this is not true for complex job shops with the TWT objective. Therefore, we describe decomposition heuristics based on the SBH proposed originally by Adams et al. [1] for $\mathrm{Jm}||C_{\max}$.

The SBH decomposes the overall scheduling problem into a series of smaller scheduling problems: one for each machine group. The resulting scheduling problems are typically smaller. The corresponding scheduling decisions lead to new conjunctive arcs within G. We have to deal with the following two problems:

1. Subproblem identification step: In this step, we have to identify appropriate constraints for the subproblems to model the interaction between the subproblems.
2. Composition step: We have to determine an appropriate sequence to solve the subproblems identified in the first step. Furthermore, the solutions of the subproblems have to be combined to obtain a solution of the overall scheduling problem.

We start by describing the resulting subproblems. Ready times and due dates for the jobs to be processed on individual machine groups can be determined by using the two basic recursion schemes (5.87), (5.88) and (5.92), (5.93). We use this data to formulate subproblems. Because of the production restrictions in wafer fabs (cf. Sects. 2.2.2 and 2.2.3), we have to solve problems of type

$$\mathrm{Pm}|\text{p-batch}, \text{incompatible}, s_{jk}, r_j|\mathrm{TWT}. \tag{5.103}$$

It is well known that the subproblem formulation based on ready times and due dates is not sufficient to determine schedules that can be combined to form feasible schedules for the overall scheduling problem (see Dauzère-Pérès and Lassere [59]). This is caused by the fact that scheduling decisions on other machines require a certain amount of time to elapse between the start times of operations on a given machine. Therefore, the resulting constraints are called delayed precedence constraints.

To illustrate this concept, we consider a disjunctive graph for three jobs that have to be processed on two machines similar to Ovacik and Uzsoy [223]. The data of this problem instance are shown in Table 5.10.

Table 5.10: Problem instance for a disjunctive graph with cycle

Job	Process flow (machines)	Processing time
1	1	$p_{11} = 5$
2	1–2	$p_{12} = 1, p_{22} = 1$
3	2–1	$p_{13} = 1, p_{23} = 1$

Figure 5.8 depicts the disjunctive graph where machine 2 is scheduled. The operation that is associated with node $\langle 2,3\rangle$ is processed after the operation that belongs to node $\langle 2,2\rangle$. Using longest path calculations, we determine that $r_{11} = 0$, $r_{12} = 0$, and finally $r_{13} = 3$. Taking only these ready times into account, it is clear that it is possible to schedule the operations associated with the nodes $\langle 1,1\rangle$, $\langle 1,3\rangle$, and $\langle 1,2\rangle$ in this sequence on machine 1. The corresponding

start times of the operations are 0, 5, and 6. The disjunctive graph now contains a cycle that consists of the nodes $\langle 1,2\rangle$, $\langle 2,2\rangle$, $\langle 2,3\rangle$, and finally $\langle 1,3\rangle$.

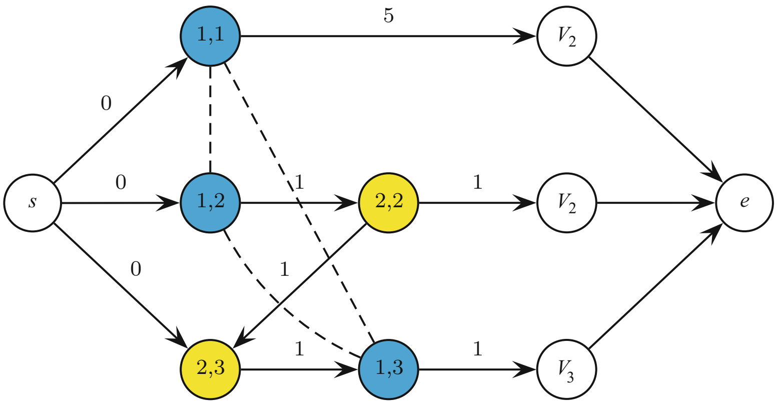

Figure 5.8: Disjunctive graph for three jobs and two machines

We can see from Fig. 5.9 that there is a path from $\langle 1,2\rangle$, $\langle 2,2\rangle$ and $\langle 2,3\rangle$ to $\langle 1,3\rangle$. Therefore, we conclude that the operation associated with $\langle 1,2\rangle$ has to be performed before the operation that corresponds to $\langle 1,3\rangle$. This precedence relation is a consequence of the scheduling decision for machine 2. When we formulate the subproblem for machine 1, we take only precedence relations into account that are followed from the process flows of the jobs. Because we do not respect the precedence relation between $\langle 1,2\rangle$ and $\langle 1,3\rangle$, we determine a schedule that is not feasible. The operation associated with $\langle 1,3\rangle$ can only start when the operations that belong to $\langle 1,2\rangle$, $\langle 2,2\rangle$, and $\langle 2,3\rangle$ are completed. It follows that the earliest start time for the operation associated with $\langle 1,3\rangle$ is two time units after the completion time of the operation corresponding to $\langle 1,2\rangle$.

We denote the precedence relations among the operations of the jobs that have to be taken into account during scheduling decisions by prec. We have to respect such delayed precedence relations to make sure that no cycles are created in the disjunctive graph during the implementation of subproblem-related schedules within the disjunctive graph. We have to replace the subproblem (5.103) by

$$\text{Pm}|\text{p-batch}, \text{incompatible}, s_{jk}, r_j, \text{prec}|\text{TWT}. \tag{5.104}$$

Delayed precedence relations are determined by a topological sorting of the nodes of G using an efficient depth-first search. We refer to Sect. 5.4.4 for details on this approach and its implementation. The algorithm of Pabst [225] is used within our SBH implementation.

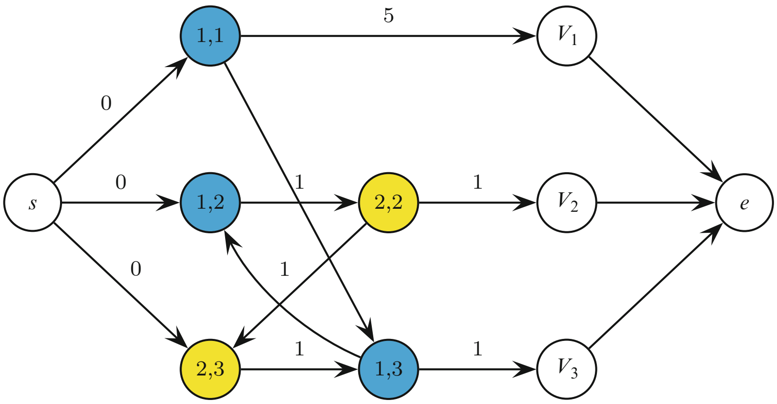

Figure 5.9: Disjunctive graph containing a cycle

The performance of decomposition approaches is influenced by the sequence in which the subproblems are solved and also by the solution approaches called SSPs for the scheduling problem found in Eq. (5.104).

Now we are able to present the SBH and its modification for scheduling entire wafer fabs. The SBH can be formulated as follows [1, 172, 223].
Algorithm SBH

1. We denote by M the set of machine groups that have to be scheduled. Furthermore, we choose the notation M_0 for the set of machine groups that are already scheduled. Initially, set $M_0 := \emptyset$. Generate an initial disjunctive graph G based on the process flows of the jobs. Determine ready times and due dates for all operations associated with nodes in G based on longest path calculations using the two recursions (5.87), (5.88) and (5.92), (5.93).
2. Identify and solve subproblems for each machine group $i \in M - M_0$ taking delayed precedence constraints into account.
3. Determine the most critical machine group $k \in M - M_0$ with respect to a certain criticality measure.
4. Implement the schedule determined in step 2 for the machine group $k \in M - M_0$ into the disjunctive graph, i.e., update G. Set $M_0 := M_0 \cup \{k\}$.
5. (Optional) Reoptimize the determined schedule for each machine group $m \in M_0 - \{k\}$ by considering the newly added disjunctive arcs from step 4 for machine group k.
6. The algorithm terminates if $M = M_0$. Otherwise, determine ready times and due dates for operations based on the nodes of G and go to step 2.

Note that the criticality measure used in step 3 determines in which sequence the machine group schedules are incorporated within G. For determining the most critical machine group, different approaches are possible. One option is to consider the machine group that is associated with the subproblem that has the largest TWT value within an iteration of the SBH as the most critical machine group.

While this simple approach is intuitively appealing, it turns out that more sophisticated methods are more effective. It is suggested by Pinedo and Chao [241] to use the change of the completion times after the implementation of the schedule for a machine group relative to the graph in the previous iteration as a measure for machine criticality. The criticality measure CM for machine group i is given by

$$\mathrm{CM}(i) := \sum_{j=1}^{n} w_j (C_j'' - C_j') \exp\left\{ -(d_j - C_j'')^+ / K \right\}, \tag{5.105}$$

where C_j' denotes the completion time of job j determined during the previous iteration of the heuristic. C_j'' is the new completion time of j, assuming that the schedule proposed in the current iteration for machine group i is implemented within G. Finally, K is a scaling parameter. Obviously, it holds $C_j' \le C_j''$, as the inclusion of additional precedence constraints never decreases the completion times of the jobs. Therefore, $\mathrm{CM}(i)$ is a measure of how much the proposed schedule for machine group i will potentially increase the TWT value of all jobs in the job shop.

Furthermore, it is possible to exploit more global information like, for example, knowledge of planned and dynamic bottlenecks during decision-making (see Uzsoy and Wang [304]). Various criticality measures like the total machine load (TML) and the average remaining operations to completion (AROC) are studied in Aytuk et al. [15]. A blended index based on these criticality indices to determine the most critical machine group within each iteration of the SBH is proposed by Mönch and Zimmermann [198]. Finally, GAs (see Dorndorf and Pesch [69]) and inductive decision trees (see Osisek and Aytuk [221]) have been used to find an appropriate sequence in which the solutions of subproblems are implemented within G; however, these techniques tend to cause a very large computational burden.

Next, we describe the reoptimization carried out in step 5 in more detail. The reoptimization procedure reschedules the machine groups scheduled earlier taking G into account. G is the disjunctive graph that already contains the schedule of the last bottleneck machine group. Then, based on the TWT value of the new and the old schedule, it is decided whether the old schedule for a machine group is replaced by the new one. Even a small number of reoptimization steps increases the quality of the schedules considerably. The reoptimization algorithm can be described in a more formal way as follows.
Algorithm SBH Reoptimization

1. Set $r := 0$, where r is the number of the completed reoptimization steps. Let $r_{\max}$ be the maximum number of reoptimization steps. Calculate the TWT value that is associated with G. Assume that M_0 machine groups are already scheduled. Let k be the machine group that is scheduled

in the current iteration of the SBH. Set $\mathsf{imp} := \mathsf{true}$ where imp states whether an improvement with respect to TWT was found or not within one reoptimization step.
2. Repeat steps 3 to 9 until $r_{\max}$ is reached or $\mathsf{imp} = \mathsf{false}$.
3. Set $\mathsf{imp} := \mathsf{false}$.
4. For all $i \in M_0 - \{k\}$, repeat the following steps.
5. Store the schedule for machine group i and remove the schedule for i from G, i.e., insert disjunctive arcs for the corresponding conjunctive arcs.
6. Determine ready times and due dates for all operations associated with nodes in G based on longest path calculations using the two recursions (5.87), (5.88) and (5.92), (5.93). Identify and solve the subproblem for machine group i taking delayed precedence constraints into account.
7. Implement the schedule determined in step 6 within G. Determine ready times for all operations associated with nodes in G. Calculate completion times based on these ready times and determine the TWT value that corresponds to G.
8. If the new TWT value is smaller than the old one, then update the smallest TWT value found so far. In this case, update $\mathsf{imp} := \mathsf{true}$; otherwise, replace the new schedule for machine group i by the old one stored in step 5. Go to step 4.
9. Update $r := r + 1$ and go to step 2.

Usually, two or three reoptimization steps are sufficient. The machine groups can be considered in the order in which they are introduced into the solution. There are SBH variants possible where reoptimization is only carried out after Step 6 in the SBH algorithm, i.e., when all machine groups are scheduled (cf. Ovacik and Uzsoy [223] for refined reoptimization strategies). Real options analysis is proposed by Yeung and Mason [327] to value reoptimization options in the SBH.

So far we assumed that G contains all the operations that have to be performed on all the machine groups. Because this might cause a large computational burden, several attempts were made to reduce the number of nodes in the scheduling graph by considering only specific heavily loaded machine groups explicitly. This results in reduced process flows that contain only operations that belong to these heavily loaded machine groups. The lightly loaded machine groups are represented by an infinite capacity resource that is scheduled using a due-date-oriented dispatching rule like EDD. Experiments with this approach for job shops that also include machine breakdowns are presented by Upasani and Uzsoy [301] and by Upasani et al. [302].

5.4.4 Subproblem Solution Procedures

We next describe in more detail how we can ensure that the delayed precedence relations are respected at the SSP level. The following availability condition from Pabst [225] for each operation associated with a node at the time of its positioning in the schedule is necessary and sufficient for the avoidance of cycles in G: Either all operations that are associated with predecessors of the node are already scheduled or every operation associated with an unscheduled predecessor will not be scheduled on the same machine as the operation that belongs to the considered node. This statement can be proven by induction over the number of scheduled machine groups (see Pabst [225]).

For the design of appropriate SSPs, the following statement is important. If the previous availability condition holds for all nodes at the scheduling time of the associated operations, then for each node of an SSP it holds that either all direct predecessors are already scheduled or every operation associated with an unscheduled predecessor will not be scheduled later on the same machine like the operation associated with the considered node.

Moreover, as proved by Pabst [225], the next condition is necessary to avoid errors in the calculation of the availability times of the nodes and sufficient for the avoidance of cycles in G. For each node at the time of the positioning of the associated operation in the schedule, it holds that operations associated with direct predecessors are already scheduled and completed. Following Pabst [225], we define a direct precedence relation between two nodes belonging to the same subproblem if each path connecting the related nodes passes only nodes that belong to other subproblems.

In order to provide the required information for the solution of the subproblems, it is necessary to use an algorithm that determines all direct predecessors for a given set of jobs. A coupling with the calculation of the ready times of the operations of the jobs is useful. The depth-first search algorithm already mentioned in Sect. 5.4.3 is used.

SSPs are often provided by list scheduling algorithms that allow for taking the delayed precedence constraints into account. We briefly sketch an SSP for problem (5.104). Because the algorithm is a generalization of the BATC-II-TWDH algorithm described in Sect. 5.3.4, we describe only the main differences that are caused by the delayed precedence relations. Steps 1–4 are the same with the exception that the jobs to be scheduled on the machine group are determined using G. In step 5, only such job combinations are considered that do not contain jobs that are related by a precedence constraint to each other. Furthermore, only jobs with predecessors that are already included in a batch are considered. We denote the time, where all predecessors are scheduled and completed with $r_{\text{succ},i}$ for batch B_s of family i selected in step 7. In contrast to BATC-II-TWDH, in the presence of the delayed precedence relations, we have to advance the availability time of the machine k to $t_a(k) := \max(t_a(k), r_{B_s}, r_{\text{succ},i}) + p_i + s_{B_s}$, where s_{B_s} is the setup time that occurs before processing of batch B_s. We denote this SSP by SSP(BATC-II-TWDH). Note that this SSP can also be used to solve the scheduling problem

$$\text{Pm}|r_j, \text{prec}|\text{TWT}. \tag{5.106}$$

In this case, we set thresh $:= \infty$, Δt can be large, and the assessment of job combinations is obsolete.

An SSP based on GAs is studied by Mönch et al. [206]. Note that SSP(BATC-II-TWDH) is outperformed by the GA-based SSP in many situations. Furthermore, an SSP based on list scheduling approaches using variants of the ATCS dispatching rule (cf. Sect. 4.3) and VNS for the problem

$$\mathrm{Pm}|r_j, s_{jk}, \mathrm{prec}|\mathrm{TWT} \tag{5.107}$$

is proposed by Driessel and Mönch [72]. Note that in principle, a time window approach as for SSP(BATC-II-TWDH) also can be used to solve this problem.

5.4.5 Simulation-Based Performance Assessment

Several assessment efforts for static problem instances are described in the literature (cf. [64, 65, 223, 242] among others). But because we apply a deterministic scheduling approach to a stochastic BS and BP, there is a need to use the SBH in a rolling horizon setting. This approach allows one to take current information of the BS and the BP into account.

We use the software architecture described in Sect. 3.3.2 to carry out experiments in this section. The center point of this architecture is a data layer in the memory of the computer that contains all the information to construct the disjunctive graph and make the scheduling decisions. The data layer is between a simulation model that emulates the manufacturing process of interest and the scheduling application for the SBH. It acts as a mirror of the manufacturing process. It contains job information, process flows, and machines.

Calculated schedules are submitted to the simulation engine AutoSched AP in order to use the information of the schedules in a dispatching-based manner. The architecture allows for rolling horizon-type scheduling as well as for event-driven rescheduling activities. The different products are represented by separate ASCII files of the simulation model. An object-oriented database is used to store results and the disjunctive graph in order to reduce initialization efforts. The architecture is depicted in Fig. 5.10. Note that in contrast to the situation in Fig. 3.6, there is no need for a forecast module, a demand generation module, or a PS. The most important part of the CS is the SBH. The dispatcher is given by a dispatching rule that is used to select the next job to be processed on a machine with respect to the schedule determined by the SBH.

Next, we describe the design of experiments. We consider the full MIMAC 1 model (see Fowler and Robinson [83]). It contains over 200 machines that are organized into over 80 machine groups (cf. also Sect. 4.4). The model contains two technologies with 210 and 245 steps, respectively. We denote these two technologies by P1 and P2. In order to generate random process flows, process flows P1 and P2 are divided into 16 subprocess flows. The

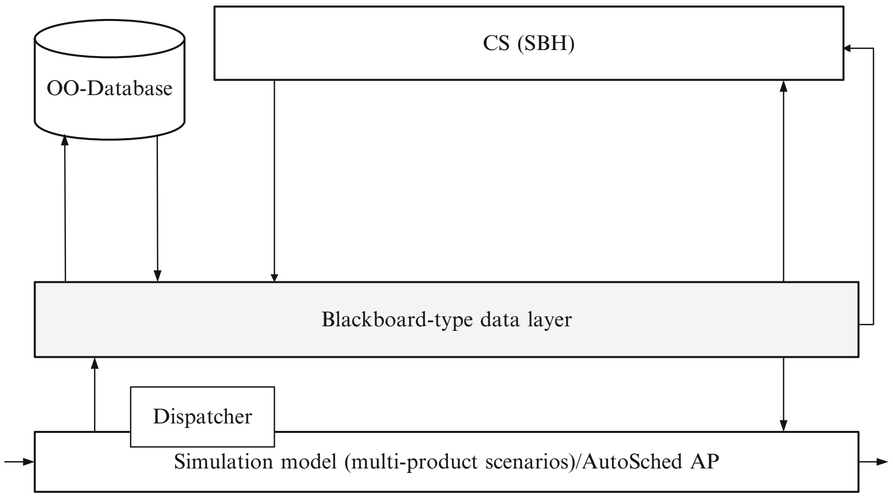
CS (SBH)
OO-Database
Blackboard-type data layer
Dispatcher
Simulation model (multi-product scenarios)/AutoSched AP

Figure 5.10: Simulation architecture for a performance assessment of the SBH

segmentation into subprocess flows takes the layer structure of wafers into account, i.e., performing the process steps of one subprocess flow leads to the manufacturing of a certain number of layers. We cover approximately the same number of layers within the corresponding segments of process flows P1 and P2. For each segment of the new process flows, we randomly choose one subprocess flow either from P1 or from P2. Following this approach, we are able to create a large number of different process flows. We consider 2, 4, 8, and 16 different products in our simulation experiments. Jobs representing different products form different families and cannot be batched together. Therefore, they influence the dynamics of the wafer fab to a large extent. We set the due date of job j by

$$d_j := r_j + \mathrm{FF} \sum_{k=1}^{n_j} p_{jk}, \tag{5.108}$$

where the abbreviation FF is used for the flow factor. The allocated amount of waiting time that can be spent by job j is therefore given by $(\mathrm{FF}-1)\sum_{k=1}^{n_j} p_{jk}$. We use $\mathrm{FF} = 1.4$, i.e., tight due dates, in our simulation experiments. The weights of the jobs are selected according to the discrete distribution

$$D := \begin{cases} w_j = 1,\ p_1 = 0.5 \\ w_j = 5,\ p_2 = 0.35. \\ w_j = 10,\ p_3 = 0.15 \end{cases} \tag{5.109}$$

Furthermore, we use a high load of the wafer fab. To reduce the simulation time, we start each simulation with a WIP distribution of the jobs. We perform deterministic simulation runs for 50 days. We apply the SBH every 2 h with a scheduling horizon of 2 h, i.e., we have $\tau_\Delta = 2h$ and $\tau_{ah} = 0h$ (cf. Sect. 2.3 for this notation). The current state of the BS and BP is taken into account every 2 h for determining a new schedule.

We are interested in the TP, the CT, and the AWT value. We use AWT instead of TWT because AWT takes also the number of completed jobs into account. Therefore, AWT is a fairer measure for comparison in situations where the TP value, expressed as the number of completed jobs, is different for the SBH and a FIFO-based dispatching scheme. In order to avoid difficulties with absolute values, we use relative values denoted by REL. The value for REL is defined as the ratio of the performance measure value obtained by the SBH or the HVF-EDD dispatching rule and the corresponding performance measure value for the wafer fab that is controlled using the FIFO dispatching rule. The HVF-EDD dispatching rule uses first HVF (highest weight) (cf. Sect. 4.2.1) and breaks ties by applying EDD. This dispatching rule is an example of a multilevel rule (cf. Sect. 4.1). It is selected for comparison with the SBH because it outperforms several dispatching rules in the present situation (cf. Mönch and Zimmermann [200] for those dispatching rules). The simulation results are presented in Table 5.11.

Table 5.11: Computational results for the SBH and HVF-EDD

Number of products	Rel(AWT) SBH	Rel(AWT) HVF-EDD	Rel(TP) SBH	Rel(TP) HVF-EDD	Rel(CT) SBH	Rel(CT) HVF-EDD
2	0.4588	0.4734	0.9384	0.9788	0.9994	0.9165
4	0.2499	0.1373	0.9973	0.9892	0.9185	0.8297
8	0.1673	0.2113	1.0567	0.9845	0.8412	0.8004
16	0.2276	0.7131	1.0068	0.9002	0.9625	1.0168

We can conclude from Table 5.11 that we obtain decreasing AWT values for an increasing number of products compared to the FIFO strategy. At the same time, we can observe that the relative CT value is decreased with an increasing number of products in some situations. We find some TP losses for two and four products. TP reductions are the price for AWT reductions in the case of large-scale wafer fabs, at least for a small number of products. The SBH clearly outperforms the HVF-EDD dispatching rule in many situations with respect to the three measures.

Many more computational results can be found for a small number of products in Mönch et al. [206] for the TWT measure, in Sourirajan and Uzsoy [288] for the $L_{\max}$ measure, and for the multiproduct case with TWT as performance measure in Mönch and Zimmermann [200].

We have to deal with the question of how to take the feedback from the BS and from the BP into account in a rolling horizon setting. In our experiments, we periodically determine a schedule from scratch. Dispatching rules are used to implement this schedule, i.e., when a machine becomes free, the job or batch is selected based on priorities obtained from the schedule rather than implementing the schedule rigidly. A similar approach is taken by Barua et al. [23].

It is also possible to repair an existing schedule when machine breakdowns occur. A repair can also be performed periodically. Different rescheduling strategies for a SBH for scheduling wafer fabs are proposed by Mason et al. [173].

5.4.6 Distributed Shifting Bottleneck Heuristic

Even computationally efficient implementations of the SBH (see Blazewicz et al. [30]) for complex job shops are time-consuming for moderate scheduling horizons. In the situation of a larger scheduling horizon, for example, $h = 2$ days, the number of nodes of G to be considered grows tremendously; hence, the solution of the scheduling problem requires very large computational efforts in terms of memory and computation time. On the other hand, considering a very small scheduling horizon leads to the problem of a proper internal due date setting that is often difficult and can lead to performance deterioration.

A conceptual framework for solving scheduling problems via decomposition into subproblems that are larger than the machine groups in the original shifting bottleneck approach is described by Brandimarte et al. [32]. But a concrete decomposition and solution scheme and results of computational experiments are not presented. Based on an appropriate physical decomposition of the BS into work areas (cf. the description in Sect. 2.2.2), we present a two-level hierarchical approach following Mönch and Driessel [193]. We use a simple job planning approach in order to assign internal ready times $r_j^{(k)}$ and internal due dates $d_j^{(k)}$ to each job j for each work area k on the top level. Based on these internal ready times and due dates, we apply the SBH to each work area in order to come up with detailed schedules for the jobs within the work areas on the base level. The overall scheme is called distributed SBH (DSBH).

For describing the DSBH, we start at the top level. This level is based on an aggregated model. As proposed by Habenicht and Mönch [112], we aggregate consecutive process steps of a process flow into macro operations. We denote the macro operation l of job j by mo_{jl}. Assume, that mo_{jl} is formed by the $s+1$ consecutive process steps $O_{jr}, O_{j,r+1}, \ldots, O_{j,r+s}$, where the rth process step is the generating one. In an analogous way to the processing time for a process step, we define the processing time of the macro operation mo_{jl} by

$$p(\mathrm{mo}_{jl}) := p_{jr} + p_{j,r+1} + \ldots + p_{j,r+s}. \tag{5.110}$$

Each macro operation is related to a specific work area. We assume that we can determine a unique process step among the macro operation forming process steps that has to be performed on a machine group with a high utilization. Hence, the behavior of a macro operation is basically influenced by the machine group of this process step. We consider aggregated capacities for the machines of a certain machine group that take batching characteristics and machine breakdowns into account. We assign start dates and due dates to each single macro operation based on an infinite capacity approach (ICA). The ICA algorithm can be summarized as follows.
Algorithm ICA

1. Adjust d_j for each job j in such a way that $\sum p(\mathrm{mo}_{jl}) \leq d_j - t$ is valid, where we denote by t the current time.
2. Determine the quantity $h_j := (d_j - t)/\sum p(\mathrm{mo}_{jl})$ at time t. If $h_j \geq 1$, the remaining time until d_j is distributed equally among the remaining macro operations of job j. The sum of the possible waiting times for the operations that form the macro operation mo_{jl} is $(h_j - 1)p(\mathrm{mo}_{jl})$ to meet the due date d_j.
3. Calculate ready times $r(\mathrm{mo}_{jl})$ for operation mo_{jl} in a recursive manner via the expression $r(\mathrm{mo}_{j,l+1}) := r(\mathrm{mo}_{jl}) + h_j p(\mathrm{mo}_{jl})$ for $l \geq l_0 + 1$, where we assume that the macro operation mo_{jl_0} is the current one with a given $r(\mathrm{mo}_{jl_0})$.
4. The due date $d(\mathrm{mo}_{j,l-1})$ of macro operation $\mathrm{mo}_{j,l-1}$ is given by $r(\mathrm{mo}_{jl})$ for $l \geq l_0 + 1$.

We apply the ICA approach in a rolling horizon manner every $p_{\max}\tau_\Delta$ time units taking the current state of the BS and the BP into account. We call τ_Δ the scheduling interval of the base level, and $p_{\max}$ is a positive integer. ICA provides the internal ready times $r_j^{(k)}$ and due dates $d_j^{(k)}$ with respect to work area k.

The naive DSBH (NDSBH) uses the internal ready times and due dates determined by ICA to calculate detailed schedules for the jobs that have to be processed on the machine groups of a single work area within a certain scheduling horizon. The heuristic can be summarized as follows.
Algorithm NDSBH

1. Determine $r_j^{(k)}$ and $d_j^{(k)}$ of the jobs for each work area k using ICA.
2. Determine schedules for each single work area by using the algorithm SBH for $r_j^{(k)}$ and $d_j^{(k)}$ from step 1.

In the case that more than one consecutive macro operation is expected to be performed within the scheduling horizon h, we have to consider the due date of the last macro operation as the due date for the calculation of the TWT.

Furthermore, we have to modify the disjunctive graph G to make sure that the reentrant behavior of the process flows is correctly modeled. Therefore, we add additional disjunctive arcs between nodes that represent operations of the same job belonging to consecutive macro operations. The approach is similar to the treatment of reentrant flows in the SBH. The main differences are the multiple internal ready times and multiple internal due dates with respect to work area k. We show the resulting G for two jobs in Fig. 5.11. Job 1 contains one macro operation, represented by the nodes $\langle 1,1\rangle$, $\langle 2,1\rangle$, and $\langle 3,1\rangle$. Job 2 has two macro operations: one consists of the operations that belong to the nodes $\langle 1,2a\rangle$, $\langle 2,2a\rangle$, and $\langle 3,2a\rangle$, whereas the second one is given by $\langle 1,2b\rangle$, $\langle 2,2b\rangle$, and finally $\langle 3,2b\rangle$. Additional conjunctive arcs are included between nodes $\langle 1,2a\rangle$ and $\langle 1,2b\rangle$, $\langle 2,2a\rangle$ and $\langle 2,2b\rangle$, and $\langle 3,2a\rangle$ and $\langle 3,2b\rangle$. Note that the artificial end nodes V_j contain the $d_j^{(k)}$ of the related macro operations. The ready times of the operations associated with node $\langle 1,1\rangle$, $\langle 1,2a\rangle$, and $\langle 1,2b\rangle$ are also derived from the top level of the hierarchy via the ICA approach.

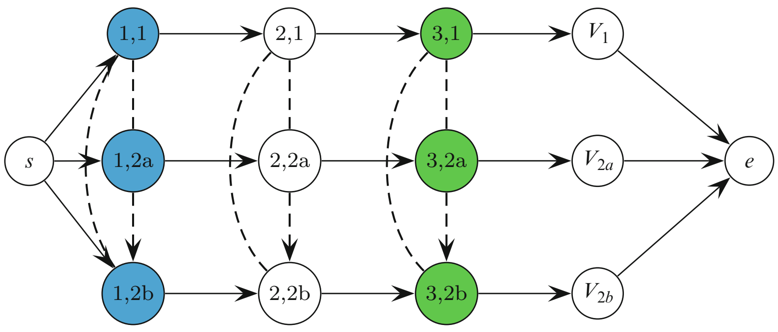

Figure 5.11: Modified graph for macro operations

NDSBH has the drawback that a complete decoupling of the work areas can take place that might result in suboptimal or even infeasible schedules with respect to the overall scheduling problem for the entire job shop. On the other hand, if ICA of the top level determines accurate start dates and end dates with respect to available capacity, it might be sufficient to consider NDSBH.

A second approach considers modified start dates and end dates iteratively. By using this approach, we obtain better schedules for the overall scheduling problem in a step-by-step manner. The suggested procedure is called iterative DSBH (IDSBH). The basic steps can be summarized as follows.

Algorithm IDSBH

1. Let $\mathbf{MB}$ denote the set of all work areas. Denote by $\mathbf{MB}_0$ the set of work areas for which a schedule is already calculated. Initially, set $\mathbf{MB}_0 := \emptyset$.

Determine initial ready dates $r_j^{(k)}$ and due dates $d_j^{(k)}$ of the jobs with respect to each work area k using ICA.

2. Determine schedules for all work areas $\mathbf{MB}-\mathbf{MB}_0$ by using the SBH and the current $r_j^{(k)}$ and $d_j^{(k)}$ calculated in step 1.
3. Determine the most critical work area from $\mathbf{MB}-\mathbf{MB}_0$ by calculating the TWT value of the jobs with respect to the corresponding work area. Denote the most critical work area by k. Update $\mathbf{MB}_0 := \mathbf{MB}_0 \cup \{k\}$.
4. Use the schedule for k in order to determine modified ready times $\tilde{r}_j^{(k)}$ and end due dates $\tilde{d}_j^{(k)}$ for the remaining work areas, i.e., for macro operations of jobs in work areas $\mathbf{MB}-\mathbf{MB}_0$, based on the start and completion times of the jobs in k. Send the modified start dates and end dates to the remaining work areas. Update the current ready times and due dates.
5. Determine schedules for the work areas $m \in \mathbf{MB}-\mathbf{MB}_0$ that take the new ready times and due dates into account.
6. If $\mathbf{MB}=\mathbf{MB}_0$, then stop. Otherwise, go to step 3.

IDSBH eliminates most of the drawbacks of NDSBH. However, if the scheduling horizon is large, scheduling decisions based on modified start dates and end dates might negatively affect the previous scheduling decisions for critical work areas, especially because of multiple macro operations of jobs within one work area. Hence, an adaptation of the schedules, similar to the reoptimization cycles of the shifting bottleneck heuristic is necessary, and this tends to be computationally expensive. A truncated variant of IDSBH may perform only a certain number of iterations instead of doing all iterations in order to reduce the computational burden.

Results of extensive simulation experiments with a similar setting as described in Sect. 5.4.5 can be found in [193]. It turns out that IDSBH outperforms NDSBH with respect to TWT and that CT and TP are similar. The SBH as described in Sect. 5.4.3 performs only slightly better. It is also shown by Mönch et al. [204] that the DSBH-type algorithms lead to smaller computational burden when different computers are used to solve the work-area-related scheduling problems after the decoupling by ICA.

5.4.7 Multicriteria Approach to Solve Large-Scale Job Shop Scheduling Problems

In this section, we extend problem (5.83) to the case of more than one performance measure. This approach is reasonable because usually several performance measures are of interest in wafer fabs. It is demonstrated by Demirkol and Uzsoy [63] that solving problem (5.84) leads to schedules that perform well even for performance measures other than $L_{\max}$. Multicriteria approaches are not often studied for complex job shop scheduling problems, with the notable exception of Pfund et al. [235].

In the following, we assume a scheduling problem with performance measures γ_1 and γ_2 to be minimized. Let S be a feasible schedule and let $\gamma_1(S)$ and $\gamma_2(S)$ be the objective function values for S with respect to the two measures. While our discussion is on bicriteria problems, the ideas can be easily extended to problems with more criteria.

Sometimes the decision maker has a priori information regarding the nature of optimization to be performed. For instance, criteria γ_1 may be of primary importance and γ_2 of secondary interest. In other words, a solution best in γ_2 may be desired among all solutions that are best in γ_1. This is called hierarchical or lexicographic optimization. In other cases, the decision maker may have a composite linear function of the form $F(\gamma_1, \gamma_2) := \alpha\gamma_1 + (1-\alpha)\gamma_2$, $0 \leq \alpha \leq 1$ in mind that needs to be minimized. The weighted sum translates multiple objectives into a single objective value for a proposed schedule.

S is called Pareto optimal or nondominated if there exists no other schedule S' such that $\gamma_1(S') \leq \gamma_1(S)$ and $\gamma_2(S') \leq \gamma_2(S)$ where at least one of the two inequalities is strict. If all Pareto optimal solutions are known, the decision maker can choose the schedule that is most preferred from this set. This approach is called an a posteriori approach and is generally the most difficult and the most computationally expensive. For a comprehensive survey on multicriteria scheduling, we refer the reader to T'kindt and Billaut [298].

We consider a multicriteria scheduling problem in which we combine $C_{\max}$, TC, and TWT into a single aggregation function. Our aggregation function, however, is different from the linear combination of objectives described earlier. We use, instead, a desirability function (cf. the description in Sect. 3.3.1) to aggregate the objectives. We assume the decision maker has a prefixed goal/target value and an upper/worst-case value for every criterion. We also assume the decision maker, just as in the linear combination case, is aware of the priorities on the objectives. We use the desirability function approach already discussed in Sect. 3.3.1. We use $m = 3$, as we represent the desirabilities of $C_{\max}$, TC, and TWT as d_1, d_2, and d_3, respectively. Each desirability value d_i in our experiments will have goal value G_i and maximum (upper) limit U_i.

It is important to link the properties of a solution that is optimal with respect to a desirability function and its connection to the classical multicriteria idea of a Pareto optimal or nondominated solution. From the definition and discussion of the desirability function, it is clear that the optimal solution depends upon the G_i values and U_i values assumed for each criterion i. We have listed below two specific cases with regard to this. While these cases highlight situations where optimality with respect to the desirability function and Pareto optimality may not always match with each other, they also show that these differences are more due to the framing of the problem with respect to upper and lower limits than any inherent issues in the desirability approach.

1. The way the desirability function is structured, any solutions that are over the prespecified upper limit for a minimization problem for any of the criteria are assigned a value of 0. If the upper limit is fairly strict,

i.e., aggressive, this can exclude nondominated solutions that are at the ends of the efficient frontier. But in practice, this simply means that these solutions are not acceptable because they perform poorly for at least one of the criteria. Thus a nondominated solution may not be optimal from a desirability point of view as certain constraints on individual criteria have to be met.

2. We consider now the set of feasible solutions $\bar{S}$ with non-zero desirability values. Thus, for each solution in $\bar{S}$, the criteria values will be strictly less than the upper limits. Let x^* be a dominated solution in $\bar{S}$. Let ND_{x^*} be the set of nondominated solutions that dominate x^*. Also, let $x(i)^*$ represent the value of criterion i for solution x^* and G_i represent the goal value for criterion i. Then, for a given set of desirability weights, x^* is an optimal solution with respect to the desirability function D if there exists no criterion i and no x^{**} in ND_{x^*} such that $x(i)^{**} \le G_i \le x(i)^*$. This situation arises because D does not distinguish between solutions that are below G_i for criterion i because it assigns them a value of 1 even though there are differences in criteria values. This reflects the idea that a satisfaction level for that criterion has been met. We note that while x^* is optimal for the desirability function despite being dominated, there exists a solution(s) in ND_x which is (are) nondominated and also alternately optimal. Also, as a special case, if we were to set G_i to 0 or to a really low value, x^* would no longer be optimal for the desirability function. This leads to our decision for the choice of G_i values in our approach.

The assumptions of the discussed scheduling problem are the same as in Sect. 5.4.1. However, in contrast to the problem discussed there, we have to deal with three different criteria. Using the $\alpha|\beta|\gamma$ notation, the multicriteria scheduling problem to be solved can be represented as

$$\mathrm{FJm}|\text{p-batch}, \text{incompatible}, s_{jk}, r_j, \text{recrc}|C_{\max}, \mathrm{TC}, \mathrm{TWT}. \tag{5.111}$$

Note that the problem (5.111) is NP-hard because when any of the three performance measures are considered separately, the resultant scheduling problem is NP-hard. Therefore, we will use an appropriate modification of the algorithm SBH described in Sect. 5.4.3. At step 2 and step 3 of algorithm SBH, we propose to use the desirability function approach. SSPs are identified and solved in step 2, while the most critical machine group is determined in step 3 using a machine criticality measure (MCM). We will use the abbreviation SSP level and MCM level in the remainder of this section.

At the SSP level, we use the ATCSR dispatching rule (see Sect. 4.3.2 for the corresponding index) to determine schedules for the SSPs. The priority index of the ATCSR dispatching rule for job j is given by

$$I_j(t,l) := \frac{w_j}{p_j} \exp\left(-\frac{(d_j - p_j - \max(r_j,t))^+}{\kappa_1 \bar{p}}\right) \exp\left(-\frac{s_{lj}}{\kappa_2 \bar{s}} - \frac{(r_j - t)^+}{\kappa_3 \bar{p}}\right), \tag{5.112}$$

where $\bar{s}$ is the average setup time and κ_2 and κ_3 are look-ahead parameters for the setup and ready time terms, respectively. The three look-ahead parameters serve as scaling parameters. Varying the look-ahead parameters κ_i and therefore varying the relative importance of the terms in index (5.112) can provide high-quality schedules for the subproblems. It can be easily shown by experimentation that the solutions that will be picked at the SSP level using the desirability function will be the nondominated solutions among the schedules that are generated by varying the look-ahead parameters of the ATCSR dispatching rule. Using ATCSR-based list scheduling in this manner for multicriteria purposes is based on work by Balasubramanian et al. [19] that empirically explores the performance of a composite dispatching rule similar to the ATCSR rule for single machine bicriteria scheduling.

At the SSP level, the look-ahead parameters are varied using a grid search approach in order to generate a wide range of schedules for subsequent consideration by D for the three objectives of interest. We use five different values for each scaling parameter and thus test 125 different combinations. Parameter κ_1 is incremented from 0.1 to 2.1 in steps of 0.4; κ_2 is incremented from 0.1 to 1.1 in steps of 0.2, while κ_3 is incremented from 0.001 to 0.011 in steps of 0.002. A schedule is determined for each combination of the different look-ahead values, i.e., for each triple $(\kappa_1, \kappa_2, \kappa_3)$ from the grid. The values for U_i and G_i are thus set to the worst or best value, respectively, observed for each objective over the different schedules considered at the SSP level. Clearly, the values for U_i and G_i can be fixed, predetermined values, as knowledge of a particular machine group and its performance may be known a priori in a real-world setting, thereby making it easier to decide upon appropriate values for U_i and G_i. In a next step, using these values for U_i and G_i, we calculate the combined desirability D for each schedule from the grid.

Considering the fact that D is the geometric mean of all d_i, we mandate that the schedule with the highest D value over all schedules from the grid will not result in a d_i of zero. Clearly, the schedule that performs worst for one objective may not necessarily perform poorly for the other objectives of interest. Therefore, we assign a desirability value of 0.0001. Letting any $d_i = 0$ for a schedule causes the D value of this schedule to equal 0, which disqualifies this schedule for selection even when its performance for other objectives may be quite good.

We will use the TWT value of an SSP associated with a machine group as the MCM because it turns out that a criticality measure based on index (5.105) does not lend itself easily to the inclusion of multiple objectives using the desirability approach. This is mainly because the index contains completion time differences. Furthermore, the value of the scaling parameter K can dramatically affect which machine group will be selected as the most critical one. We call this approach for determining the most critical machine group MCM-TWT.

Before the desirability function can be used at the MCM level, the U_i and G_i values for each of the three objectives of interest must be determined.

We employ a procedure similar to the one used at the SSP level. The only difference is that at the MCM level, we are interested in identifying and scheduling the machine group with the lowest D value. Since the D value aggregates three different criteria, a low desirability value for a given machine group indicates that it is the most critical with respect to all the criteria under consideration and therefore should be scheduled first.

We consider two different approaches for identifying the critical machine group at the MCM level. First, the impact of each schedule for a machine group on $C_{\max}$, TC, and TWT of the entire complex job shop is assessed when identifying the critical machine group by inserting the schedule of the machine group into G. This approach is abbreviated as global MCM (GMCM). Alternatively, the critical machine group can be identified using only SSP-level performance metrics, i.e., we do not consider the impact of the machine group on the rest of the complex job shop. This approach is called local MCM (LMCM). The goal in the proposition of these two approaches is to test whether a difference is noticeable in the global and local approaches. In a practical setting, if the SBH procedure were to be used for scheduling, the computation time for the LMCM approach for large problem sizes would be less than the time for the GMCM approach. But intuitively, it seems that the GMCM approach reflects the critical machine group more accurately, since it takes into account conflicts between jobs in the entire wafer fab.

Next, we discuss some computational results for the MiniFab model, described in Sect. 3.2.8 and shown in Fig. 3.4. In contrast to Sect. 5.4.5, we consider only static problem instances of the MiniFab model. Each of the problem instances contains 20 jobs. Two products are taken into account, and there are ten jobs of each product. The weights of the jobs are selected as $w_j \sim \mathrm{DU}[1,100]$, and the ready times are zero. The due dates of the jobs j are chosen according to

$$d_j := \mathrm{FF}_j \sum_{k=1}^{n_j} p_{jk}, \tag{5.113}$$

where FF_j is the FF value for job j. The FF values are generated according to $U(\mathrm{FF}_{\min}, \mathrm{FF}_{\min} + \mathrm{FF}_{\max}/2)$, where $\mathrm{FF}_{\min} := \min_j \mathrm{FF}_j$ and $\mathrm{FF}_{\max} := \max_j \mathrm{FF}_j$ and the FF_j are determined by solving a single problem instance using FIFO dispatching at all machine groups.

We generate 20 different problem instances based on the MiniFab model, each with its own unique FF_j and w_j values. We focus on optimizing $C_{\max}$ and TWT, disregarding TC in an attempt to illustrate the difference between the performance of the different approaches. The impact of using desirability functions at the MCM level is small, because the MiniFab model contains only three machine groups. However, using desirability functions at the SSP level produces significantly better results for the MiniFab model-based problem instances. As the SSP-level results are independent of the approach used at the MCM level, the approaches to be compared reduce to using SSPs for pure TWT minimization as described in Sect. 5.4.4 and for SSPs that are based on the ATCSR dispatching rule and grid search to find schedules with large values for D. The former SSP is called SSP-TWT, while the latter one is called SSP-Des for abbreviation.

We compare the results obtained by SBH with results taken from using a deterministic forward simulation with EDD and CR dispatching rules (cf. Sects. 4.2 and 4.3 for a definition of the corresponding priority indices). We show objective space plots for two representative problem instances based on the MiniFab model in Figs. 5.12 and 5.13.

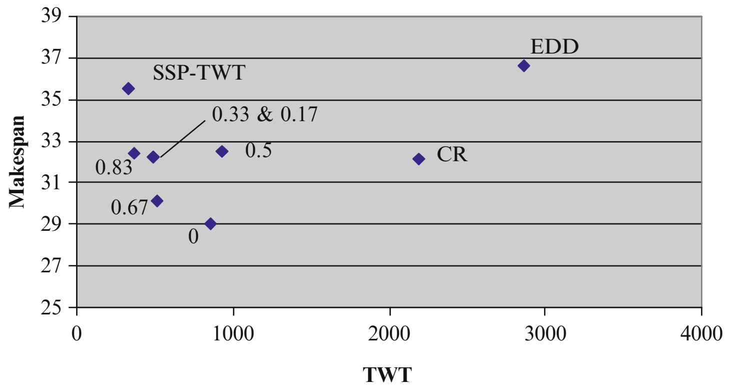

Figure 5.12: Solutions in the objective space: trade-off situation

The numbers adjacent to each point indicate the weight z_3 of the TWT criterion in D. Therefore, all solution points that have been labeled by a fraction between 0 and 1 are those obtained by using the different weight combinations at the SSP level. The weight of $C_{\max}$ is given by $1 - z_3$. In Fig. 5.12, it is possible to see the trade-off between TWT and $C_{\max}$. While the SSP-TWT solution produces the smallest TWT value, its $C_{\max}$ value is not small. The solutions generated by the different weight combinations generate a variety of solutions, which produce slightly worse TWT values, but the solutions are still significantly better than solutions obtained by the CR or EDD rules, and they improve on $C_{\max}$. Figure 5.13 shows a graph where using the desirability approach generates a solution better in both TWT and $C_{\max}$ than the solution found by SSP-TWT.

These two different problem instances shown in Figs. 5.12 and 5.13 roughly classify the nature of solutions for all 20 of the problem instances, i.e., either a trade-off exists between the two objective values or SSP-Des generates a solution better in both objective values.

More results of computational experiments also using the SBH in a rolling horizon setting for the MIMAC 1 model can be found in [235] taking a design

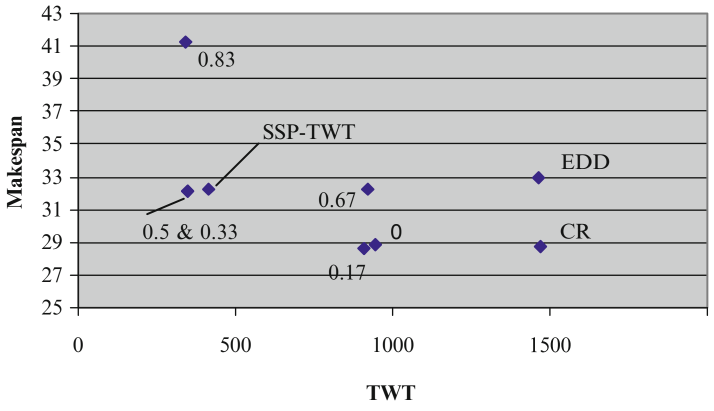

Figure 5.13: SSP-Des solution is better in both objective values

of experiments similar to those in Sect. 5.4.5. The three different performance measures for problem (5.83) are considered. Experimental results suggest that SSP-Des on the SSP level and MCM-TWT on the MCM level is the setting that performs well most often for a wide range of decision-maker priorities, followed closely by the combinations SSP-Des/LMCM and SSP-Des/GMCM. An important conclusion is that the use of the desirability function approach on the SSP level consistently produces superior results.